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 201 учебной группы факультета ВМК МГУ

Мещерякова Алексея Олеговича

гор. Москва

2020 год

Содержание

Подвариант 1

Постановка задачи и её цели

Необходимо написать программу решающую заданную систему линейных алгебраический уравнений Ax = y, с невырожденной матрицей A методом Гаусса и методом Гаусса с выбором главного элемента. Размеры матрицы n x n, n – параметр задачи, задаваемый пользователем.
Матрица задается одним из следующих способов:

1. Матрица A и ее правая часть y задаются во входном файле программы.
2. Элементы матрицы A вычисляются по заданным формулам

Поставленные задачи:

1. Решить заданную СЛАУ методом Гаусса и модифицированным методом Гаусса.
2. Вычислить определитель матрицы
3. Вычислить обратную матрицу
4. Определить число обусловленности матрицы
5. Исследовать вопрос вычислительной устойчивости метода Гаусса при больших значениях параметра n

Описание метода решения

Рассмотрим систему линейных алгебраических уравнения с невырожденной матрицей А = (aij) вида:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n = y_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n = y_2 \\ \cdots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \cdots + a_{nn} \cdot x_n = y_n \end{cases}$$

1. Метод Гаусса. Метод Гаусса разделим на 2 этапа

   (a) Прямой ход: система приводится к треугольному виду.

   (b) Обратный ход: последовательное отыскание неизвестных x1...xn

   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 $a_{11} = 0$, то поменяем местами уравнения с номерами 1 и i, где$a_{i1} \neq 0$)

$$\begin{cases} x_1 + \dfrac{a_{12}}{a_{11}} \cdot x_2 + \cdots + \dfrac{a_{1n}}{a_{11}} \cdot x_n = \dfrac{y_1}{a_{11}} \\ x_1 + \dfrac{a_{22}}{a_{21}} \cdot x_2 + \cdots + \dfrac{a_{2n}}{a_{21}} \cdot x_n = \dfrac{y_2}{a_{21}} \\ \cdots \\ x_1 + \dfrac{a_{n2}}{a_{n1}} \cdot x_2 + \cdots + \dfrac{a_{nn}}{a_{n1}} \cdot x_n = \dfrac{y_n}{a_{n1}} \end{cases}$$

   Такой элемент найдется в силу невырожденности матрицы
   Затем вычитаем получившуюся после перестановки первую строку из остальных строк

$$\begin{cases} x_1 + \dfrac{a_{12}}{a_{11}} \cdot x_2 + \cdots + \dfrac{a_{1n}}{a_{11}} \cdot x_n = \dfrac{y_1}{a_{11}} \\ 0 + \left(\dfrac{a_{22}}{a_{21}} - \dfrac{a_{12}}{a_{11}}\right) \cdot x_2 + \cdots + \left(\dfrac{a_{2n}}{a_{21}} - \dfrac{a_{1n}}{a_{11}}\right) \cdot x_n = \left(\dfrac{y_2}{a_{21}} - \dfrac{y_1}{a_{11}}\right) \\ \cdots \\ 0 + \left(\dfrac{a_{n2}}{a_{n1}} - \dfrac{a_{12}}{a_{11}}\right) \cdot x_2 + \cdots + \left(\dfrac{a_{nn}}{a_{n1}} - \dfrac{a_{1n}}{a_{11}}\right) \cdot x_n = \left(\dfrac{y_n}{a_{n1}} - \dfrac{y_1}{a_{11}}\right) \end{cases}$$

   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

$$\begin{cases} x_1 + a'_{12}\cdot x_2 + a'_{13}\cdot x_3 + \cdots + a'_{1n}\cdot x_n = y'_1 \\ 0 + x_2 + a''_{23}\cdot x_3 + \cdots + a''_{2n}\cdot x_n = y''_2 \\ 0 + 0 + x_3 + \cdots + a'''_{3n}\cdot x_n = y'''_3 \\ \cdots \\ 0 + 0 + 0 + \cdots + x_n = y^{n\prime}_n \end{cases}$$

одной неизвестной

Обратная подстановка предполагает подстановку полученного на предыдущем шаге значения переменной xn в предыдущие уравнения.

2. 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.

3. 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 деление строки на элемент матрицы, определитель матрицы умножается на этот элемент

4. Обратная матрица

   Для нахождения обратной матрицы заметим, что если с единичной матрицей Е провести элементарные преобразования, которыми невырожденная квадратная матрица А приводится к Е, то получится обратная матрица $A^{-}1$.Все преобразования будем проводить с расширенной матрицей А|Е. Приведение матрицы А к единичной заключается в приведении ее сначала к верхнетреугольной, а затем к нижнетреугольной методом из п.1 Гаусса, таким образом главная диагональ будет нормирована.

5. Число обусловленности матрицы вычислим следующим образом:

   $M_A = ||A|| * ||A^{-1}||$, где норма матрицы $||A|| = \max\limits_{1<=j<=n} \sum_{i=1}^{n} |a_{ij}|$

Структура программы и спецификация функций

Программа состоит из 1 модуля - main.c в котором происходит считывание/заполнение матрицы matrix[n][n] в зависимости от выбора пользователя и реализация функци вычисления приближенного решения СЛАУ методом Гаусса и модифицированным методом Гаусса, вычисления определителя, обратной матрицы, числа обусловленности матрицы

1. void gauss(int n, double **sourse_matr, double *x, int mod)

   принимает матрицу коэффициентов линейного уравнения размера $[n]*[n+1]$ и в зависимости от значения mod(1 либо 2), решает уравнение методом Гаусса либо модифицированным методом Гаусса. Записывает вектор ответов в x

2. double deter(int n, double **sourse_matr)

   считает определитель матрцы размера $[n]*[n]$

3. void inverse(int n, double **sourse_matr, double **invrerse_matr)

   записывает в invrersematr обратную матрицу к soursematr

4. double cond(int n, double **sourse_matr)

   вычисляет число обусловленности

5. int main(void)

   основная программа, взаимодействует с пользователем

Оригинальный текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const double EPS = 0.000001;

void gauss(int n, double **sourse_matr, double *x, int mod)//sourse_matr[n][n + 1], x[n]
{
	double matr[n][n + 1];
	for (int i = 0; i < n; i++)
	for (int j = 0; j < n + 1; j++)
	matr[i][j] = sourse_matr[i][j];
	if(mod == 1){
		for (int k = 0; k < n; k++) {
			for (int i = k; i < n; i++) {
				double tmp = matr[i][k];
				if (fabs(tmp) < EPS)
					continue;
				for (int j = 0; j < n + 1; j++)
					matr[i][j] = matr[i][j] / tmp;
				if (i == k)
					continue;
				for (int j = 0; j < n + 1; j++)
					matr[i][j] = matr[i][j] - matr[k][j];
			}
		}
	} else {
		int max_index;
		double max_elem;
		for (int k = 0; k < n; k++) {
			max_elem = fabs(matr[k][k]);
			max_index = k;
			for (int i = k + 1; i < n; i++)
				if (max_elem < fabs(matr[i][k])) {
					max_elem = fabs(matr[i][k]);
					max_index = i;
			}
			if (max_elem < EPS)
				continue;
			for (int j = 0; j < n + 1; j++) {
				double tmp = matr[k][j];
				matr[k][j] = matr[max_index][j];
				matr[max_index][j] = tmp;
			}
			for (int i = k; i < n; i++) {
				double tmp = matr[i][k];
				if (fabs(tmp) < EPS)
					continue;
				for (int j = 0; j < n + 1; j++)
					matr[i][j] = matr[i][j] / tmp;
				if (i == k)
					continue;
				for (int j = 0; j < n + 1; j++)
					matr[i][j] = matr[i][j] - matr[k][j];
```

```
                }
            }
        }
        for (int k = n - 1; k >= 0; k--)
        {
            x[k] = matr[k][n];
            for (int i = 0; i < k; i++)
                matr[i][n] = matr[i][n] - matr[i][k] * x[k];
        }
}


double deter(int n, double **sourse_matr)
{
        double matr[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                matr[i][j] = sourse_matr[i][j];
        double det = 1;
        for (int k = 0; k < n; k++) {
            for (int i = k; i < n; i++) {
                double tmp = matr[i][k];
                if (fabs(tmp) < EPS)
                    continue;
                for (int j = 0; j < n; j++)
                    matr[i][j] = matr[i][j] / tmp;
                det *= tmp;
                if (i == k)
                    continue;
                for (int j = 0; j < n; j++)
                    matr[i][j] = matr[i][j] - matr[k][j];
            }
        }
        for (int i = 0; i < n; i++)
            det *= matr[i][i];
        return det;
}

void inverse(int n, double **sourse_matr, double **invrerse_matr)//invrerse_matr[n][n]
{
        double matr[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                matr[i][j] = sourse_matr[i][j];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (i == j)
                    invrerse_matr[i][j] = 1;
                else
                    invrerse_matr[i][j] = 0;
        for (int k = 0; k < n; k++) {
            for (int i = k; i < n; i++) {
                double tmp = matr[i][k];
                if (fabs(tmp) < EPS)
                    continue;
                for (int j = 0; j < n; j++) {
```

```
                    matr[i][j] = matr[i][j] / tmp;
                    invrerse_matr[i][j] = invrerse_matr[i][j] / tmp;
                }
                if (i == k)
                    continue;
                for (int j = 0; j < n; j++) {
                    matr[i][j] = matr[i][j] - matr[k][j];
                    invrerse_matr[i][j] = invrerse_matr[i][j] - invrerse_matr[k][j];
                }
            }
        }
        for (int k = n - 1; k >= 0; k--) {
            for (int i = n - 1; i >= 0; i--) {
                double tmp = matr[i][k];
                if (fabs(tmp) < EPS)
                    continue;
                for (int j = 0; j < n; j++) {
                    matr[i][j] = matr[i][j] / tmp;
                    invrerse_matr[i][j] = invrerse_matr[i][j] / tmp;
                }
                if (i == k)
                    continue;
                for (int j = 0; j < n; j++) {
                    matr[i][j] = matr[i][j] - matr[k][j];
                    invrerse_matr[i][j] = invrerse_matr[i][j] - invrerse_matr[k][j];
                }
            }
        }
}

double cond(int n, double **sourse_matr)
{
        double cond1 = 0;
        for (int i = 0; i < n; i++) {
            double sum = 0;
            for (int j = 0; j < n; j++)
                sum += fabs(sourse_matr[i][j]);
            if ((i == 0) || sum > cond1)
                cond1 = sum;
        }

        double **invrerse_matr = (double **)calloc(n, sizeof(*invrerse_matr));
        for (int i = 0; i < n; i++) {
            invrerse_matr[i] = (double *)calloc(n, sizeof(**invrerse_matr));
        }
        inverse(n, sourse_matr, invrerse_matr);
        double cond2 = 0;
        for (int i = 0; i < n; i++) {
            double sum = 0;
            for (int j = 0; j < n; j++)
                sum += fabs(invrerse_matr[i][j]);
            if ((i == 0) || sum > cond2)
                cond2 = sum;
        }
        free(invrerse_matr);
        return cond2 * cond1;
```

```
}

int main(void) {
	printf("Size of matr:\n");
	int n;
	scanf("%d", &n);
	double **matr = (double **)calloc(n, sizeof(*matr));
	for (int i = 0; i < n; i++) {
		matr[i] = (double *)calloc(n + 1, sizeof(**matr));
	}

	printf("Choose the option(1 or 2):\n");
	int opt = 0;
	scanf("%d", &opt);
	if (opt == 1) {
		for (int i = 0; i < n; i++){
			for (int j = 0; j < n + 1; j++){
				printf("a[%d][%d] = ", i, j);
				scanf("%lf", &matr[i][j]);
			}
			printf("\n");
		}
	} else if (opt == 2) {
		const double q = 1.001 - 2 * 6 * 0.001;
		double x0;
		n = 100;
		free(matr);
		matr = (double **)calloc(n, sizeof(*matr));
		for (int i = 0; i < n; i++) {
			matr[i] = (double *)calloc(n + 1, sizeof(**matr));
		}
		printf("n  is auto-resized to 100\n");
		printf("x:\n");
		scanf("%lf", &x0);

		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++) {
				if (i != j)
					matr[i][j] = pow(q, i + 1 + j + 1) + 0.1 * (j - i);
				else
					matr[i][j] = pow(q - 1, i + 1 + j + 1);
				printf("%.*lf ", 8, matr[i][j]);
			}
			matr[i][n] = x0 * exp(x0 / (i+1)) * cos(x0 / (i+1));
			printf("| %.*lf\n", 8, matr[i][n]);
		}
		printf("\n");
	} else {
		printf("It's not correct, try again:\n");
		scanf("%d", &opt);
	}
	double *x1 = (double *)calloc(n, sizeof(*x1));
	double *x2 = (double *)calloc(n, sizeof(*x2));
	double det = deter(n, matr);
	double **invrerse_matr = (double **)calloc(n, sizeof(*invrerse_matr));
```

```
	for (int i = 0; i < n; i++) {
		invrerse_matr[i] = (double *)calloc(n, sizeof(**invrerse_matr));
	}
	gauss(n, matr, x1, 1);//Gaussian
	gauss(n, matr, x2, 2);//with pivot selection
	if(det != 0){
		inverse(n, matr, invrerse_matr);
		printf("Inverse matr:\n");
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++)
				printf("%.*lf ", 5 , invrerse_matr[i][j]);
			printf("\n");
		}
	}
	printf("Determinant:\ndet A = %lf\n", det);
	printf("Gaussian method:\n");
	for (int i = 0; i < n; i++)
		printf("x%d = %.*lf ", i + 1, 5, x1[i]);
	printf("\n");
	printf("Gaussian method with pivot selection:\n");
	for (int i = 0; i < n; i++)
		printf("x%d = %.*lf ", i + 1, 5, x2[i]);
	printf("\n");

	for (int i = 0; i < n; i++) {
		free(matr[i]);
		free(invrerse_matr[i]);
	}
	printf("Conditionality number of the matr:\nM = %lf\n", cond(n, matr));

	double sr=0;
	for (int i = 0; i < n; i++)
		sr+=fabs(x2[i]-x1[i]);
	sr/=(double)n;
	printf("\n%f\n",sr);

	free(matr);
	free(x1);
	free(x2);
	free(invrerse_matr);
	return 0;
};
```

Тесты, доказывающие корректность работы программы

Тесты из таблицы 1 пункт 1:

1. для системы уравнений

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

det A = 10.000000
x1 = 0.60000
x2 = 1.00000
x3 = -1.00000
x4 = -0.20000
Обратная матрица:

$$A^{-1} = \begin{pmatrix} -0.60000 & 0.10000 & 0.30000 & -0.20000 \\ 1.00000 & 0.50000 & -0.50000 & -0.00000 \\ -1.00000 & 1.50000 & -0.50000 & 0.00000 \\ -0.80000 & 0.30000 & -0.10000 & 0.40000 \end{pmatrix} \tag{1}$$

Число обусловленности: M = 20.000000
- все совпадает с ответами, полученными с помощью онлайн-сервиса wolframalpha

2. для системы уравнений

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

det A = 0.000000
x1 = -0.83333
x2 = 0.66667
x3 = 0.16667
x4 = -0.33333
Обратная матрица:
не существует, т.к. определитель = 0 Число обусловленности: M = 17.000000
- все совпадает с ответами, полученными с помощью онлайн-сервиса wolframalpha

3. для системы уравнений

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

det A = -189.000000
x1 = 3.00000
x2 = 2.00000
x3 = 1.00000
x4 = 0.00000
Обратная матрица:

$$A^{-1} = \begin{pmatrix} -1.71958 & 1.22222 & -0.65608 & 0.86243 \\ -0.65079 & 0.33333 & -0.26984 & 0.50794 \\ -0.98942 & 0.55556 & -0.40212 & 0.56085 \\ -0.07407 & 0.11111 & -0.18519 & 0.07407 \end{pmatrix} \tag{2}$$

Число обусловленности: M = 17.000000
- все совпадает с ответами, полученными с помощью онлайн-сервиса wolframalpha

Тесты из таблицы 2.2 пункт 6:
Элементы матрицы A вычисляются по формулам

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), & i \neq j, \\ (q_M - 1)^{i+j}, & i = j, \end{cases}$$

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

i-тый элемент вектора f задается по формуле

$$x \cdot \exp\left(\frac{x}{i}\right) \cdot \cos\left(\frac{x}{i}\right)$$

в ходе проведенных тестов было сделано сравнение решений системы линейных алгебраических уравнений, полученных методом Гаусса и методом Гаусса с выбором главного элемента:

1. при значении x=20 и размере матрицы n=60 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 0.011738
2. при значении x=30 и размере матрицы n=60 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 150.282294
3. при значении x=20 и размере матрицы n=80 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 0.038128
4. при значении x=30 и размере матрицы n=80 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 463.847627
5. при значении x=20 и размере матрицы n=90 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 0.073143
6. при значении x=30 и размере матрицы n=80 среднее отклонение метода Гаусса от модифицированного метода Гаусса составляет 1018.070088

Основные выводы

В ходе данной работы была реализованна программа, вычисляющая решения системы линейных алгебраический уравнений $Ax = y$, с невырожденной матрицей A методом Гаусса и методом Гаусса с выбором главного элемента. Как показали тесты, метод Гаусса дает достаточно точные решения для систем с небольшим порядком, но при увеличении параметра n становится неустойчивым. Модифицированный метод Гаусса более устойчив, чем обычный.

Подвариант 2

Постановка задачи и её цели

Цель работы - изучить классические итерационные методы (Зейделя и верхней релаксации), используемые для численного решения систем линейных алгебраических уравнений, а так же изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра.
Нам 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:
где D, $A^{(-)}$- соответственно диагональная и нижняя треугольные матрицы, k -

$$(D + \omega A^{(-)})\frac{x^{k+1} - x^k}{\omega} + Ax^k = f\ ,$$

номер текущей итерации, w - итерационный параметр (при $w = 1$ метод верхней релаксации переходит в метод Зейделя).
Матрица A задается одним из следующих способов:

1. Матрица A и ее правая часть y задаются во входном файле программы.
2. Элементы матрицы A вычисляются по заданным формулам

Поставленные задачи:

1. Решить заданную СЛАУ итерационным методом Зейделя (или более общим методом верхней релаксации)
2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью
3. Изучить скорость сходимости итераций к точному решению задачи (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра
4. Правильность решения СЛАУ подтвердить системой тестов

Описание метода решения

Рассмотрим систему линейных алгебраических уравнения вида:
с невырожденной матрицей $A = a_{ij}$

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n = y_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n = y_2 \\ \dots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \dots + a_{nn} \cdot x_n = y_n \end{cases}$$

1. Решение СЛАУ методом верхней релаксации
Для сходимости метода верхней релаксации необходимо, чтобы матрица А являлась симметричной. Для этого преобразуем матрицу А и ее правую часть следующим образом:
$A^T AX = A^T Y$

   Тогда метод верхней релаксации записывается в виде:

$$(D + \omega A^{(-)}) \frac{x^{k+1} - x^k}{\omega} + Ax^k = f \, ,$$

   где D, $A^{(-)}$- соответственно диагональная и нижняя треугольные матрицы, на которые раскладывается преобраззованная матрица А, k - номер текущей итерации, w - итерационный параметр, $x^{(k+1)}$– приближение, полученное на итерации с номером s, $x^{(k)}$ – приближение, полученное на предыдущей итерации

2. Критерий остановки итерационного процесса
критерием остановки может послужить Евклидома норма
$||Ax^{(k)} - f|| = \sqrt{\sum_{j=0}^{n} (A_j x_j^{(k)} - f_j)^2} < \epsilon$
где $x^{(k)}$ – приближение, полученное на итерации с номером k, ϵ - заданная точность

Структура программы и спецификация функций

Программа состоит из 1 модуля - main.c, в котором происходит считывание/заполнение матрицы matrix[n][n] в зависимости от выбора пользователя и реализация функци вычисления приближенного решения СЛАУ методом верхней релаксации(при параметре $w = 1$ он совмадает с методом Зейделя), функции нормы, функции умножения матриц.

1. long long relaxation(int n, double **summ, double *c, double *x, double w)

   принимает симметричную матрицу коэффициентов линейного уравнения размера $[n] * [n]$ и вектор-столбец решений линейных уравнений c. Вычисляет решения СЛАУ методом верхней релаксации с параметром w. Записывает вектор ответов в x

2. void mul (int m, int n, int q, double **a, double **b, double **c)

   перемножает матрицы a и b, ответ записывается по адресу c

3. void mulvector (int m, int n, double **a, double *b, double *c)

   перемножает матрицу a и вектор-стобец b, ответ записывается по адресу c

4. double norm(int n, double **summ, double *x, double *c)

   вычисляет норму для заданной матрицы, вектора x и вектора ответов

5. int main(void)

   основная программа, взаимодействует с пользователем

Оригинальный текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
const double EPS = 0.000000001;
void mul (int m, int n, int q, double **a, double **b, double **c)//a[m][n], b[n][q], c[m][q]
{
	for(int i = 0; i < m; i++)
		for(int j = 0; j < q; j++){
			c[i][j] = 0;
			for(int k = 0; k < n; k++)
				c[i][j] += a[i][k] * b[k][j];
		}

}

void mulvector (int m, int n, double **a, double *b, double *c)
{
	for(int i = 0; i < m; i++){
		c[i] = 0;
		for(int k = 0; k < n; k++)
			c[i] += a[i][k] * b[k];
	}
}


double norm(int n, double **summ, double *x, double *c)
{
	double sqr_s = 0;
	for (int i = 0; i < n; i++){
		double sum = 0;
		for (int j = 0; j < n; j++){
			sum += summ[i][j] * x[j];
		}
		sqr_s += (sum - c[i]) * (sum - c[i]);
	}
	return sqrt(sqr_s);
}

long long relaxation(int n, double **summ, double *c, double *x, double w)//summ[n][n], c[n], x[n],
{
	for (int i = 0; i < n; i++){
		x[i] = 0;
	}
	long long cnt = 0;
	double *x_prev = (double *)calloc(n, sizeof(*x_prev));
	do{
		cnt++;
		for (int i = 0; i < n; i++)
			x_prev[i] = x[i];
		for (int i = 0; i < n; i++) {
			double sum = 0;
			for (int j = 0; j < i; j++)
				sum += (summ[i][j] * x[j]);
			for (int j = i; j < n; j++)
```

```
                    sum += (summ[i][j] * x_prev[j]);
                if(summ[i][i] != 0)
                    x[i] = w * (c[i] - sum) / summ[i][i] + x_prev[i];
            }
        } while (norm(n, summ, x, c) > EPS);
        free(x_prev);
        return cnt;
}

int main(int argc, const char *argv[]) {
printf("Size of matr:\n");
printf("Parameter w:\n");
int n;
double w;
scanf("%d%lf", &n, &w);
double **matr = (double **)calloc(n, sizeof(*matr));
for (int i = 0; i < n; i++) {
        matr[i] = (double *)calloc(n, sizeof(**matr));
}
double **transp = (double **)calloc(n, sizeof(*transp));
for (int i = 0; i < n; i++) {
        transp[i] = (double *)calloc(n , sizeof(**transp));
}
double *b = (double *)calloc(n, sizeof(*b));
printf("Choose the option(1 or 2):\n");
int opt = 0;
scanf("%d", &opt);
Begin:
if (opt == 1) {
        for (int i = 0; i < n; i++){
                for (int j = 0; j < n; j++){
                        printf("a[%d][%d] = ", i, j);
                        scanf("%lf", &matr[i][j]);
                        transp[j][i] = matr[i][j];
                }
        printf("y[%d] = ", i);
        scanf("%lf", &b[i]);
        printf("\n");
        }
} else if (opt == 2) {
        const double q = 1.001 - 2 * 6 * 0.001;
        double x0;
        n = 100;
        free(matr);
        free(transp);
        free(b);
        matr = (double **)calloc(n, sizeof(*matr));
        for (int i = 0; i < n; i++) {
                matr[i] = (double *)calloc(n, sizeof(**matr));
        }
        transp = (double **)calloc(n, sizeof(*transp));
        for (int i = 0; i < n; i++) {
                transp[i] = (double *)calloc(n, sizeof(**transp));
        }
        b = (double *)calloc(n, sizeof(*b));
        printf("n  is auto-resized to 100\n");
```

```
        printf("x:\n");
        scanf("%lf", &x0);
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        if (i != j)
                                matr[i][j] = pow(q, i + 1 + j + 1) + 0.1 * (j - i);
                        else
                                matr[i][j] = pow(q - 1, i + 1 + j + 1);
                        transp[j][i] = matr[i][j];
                        printf("%.*lf ", 8, matr[i][j]);
                }
                b[i] = x0 * exp(x0 / (i+1)) * cos(x0 / (i+1));
                printf("| %.*lf\n", 8, matr[i][n]);
        }
        printf("\n");

} else {
        printf("It's not correct, try again:\n");
        scanf("%d", &opt);
        goto Begin;
}

double **summ = (double **)calloc(n, sizeof(*summ));
for (int i = 0; i < n; i++) {
        summ[i] = (double *)calloc(n, sizeof(**summ));
}
double *x = (double *)calloc(n, sizeof(*x));
double *c = (double *)calloc(n, sizeof(*c));
mul(n, n, n, transp, matr, summ);
mulvector(n, n, transp, b, c);
long long cnt = relaxation(n, summ, c, x, w);

for (int i = 0; i < n; i++) {
        printf("x%d = %.8lf ", i, x[i]);
        free(matr[i]);
        free(summ[i]);
        free(transp[i]);
}

printf("\niteration count = %lld\n", cnt);
free(matr);
free(x);
free(summ);
free(b);
free(c);
free(transp);
return 0;
}
```

Тесты, доказывающие корректность работы программы

Во всех тестах погрешность $\epsilon = 0.000000001$ Тесты из таблицы 1 пункт 1:

1. для системы уравнений

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

   x1 = 0.60000
   x2 = 1.00000
   x3 = -1.00000
   x4 = -0.20000
   - совпадает с ответом, полученным с помощью онлайн-сервиса wolframalpha
   количество итераций:

   (a) 3645 при $w = 0.5$
   (b) 1622 при $w = 0.8$
   (c) 1098 при $w = 1$
   (d) 1278 при $w = 1.5$
   (e) 6902 при $w = 1.9$

2. для системы уравнений

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

метод расходится

3. для системы уравнений

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

x1 = 3.00000
x2 = 2.00000
x3 = 1.00000
x4 = 0.00000
- совпадает с ответом, полученным с помощью онлайн-сервиса wolframalpha
количество итераций:

(a) 18311 при $w = 0.5$

(b) 9049 при $w = 0.8$

(c) 5950 при $w = 1$

(d) 2159 при $w = 1.5$

(e) 552 при $w = 1.9$

4. для системы уравнений

$$\begin{cases} 2x_1 + 3x_2 + 11x_3 + 5x_4 = 2, \\ x_1 + x_2 + 5x_3 + 2x_4 = 1, \\ 2x_1 + x_2 + 3x_3 + 2x_4 = -3, \\ x_1 + x_2 + 3x_3 + 4x_4 = -3. \end{cases}$$

x1 = -2.00000
x2 = 0.00000
x3 = 1.00000
x4 = -1.00000
- совпадает с ответом, полученным с помощью онлайн-сервиса wolframalpha
количество итераций:

(a) 5518 при $w = 0.5$

(b) 2744 при $w = 0.8$

(c) 1855 при $w = 1$

(d) 466 при $w = 1.5$

(e) 2079 при $w = 1.9$

Основные выводы

В ходе работы была реализованна программа, решающая заданную СЛАУ итерационным методом верхней релаксации. На основании тестов было получено, что скорость сходимости метода верхней релаксации зависит от принятой точности вычислений и итерационного параметра w, причем оптимальное значение итерационного параметра сильно варьируется в зависимости от самой СЛАУ - например, оптимальный параметр для примера 1 находится в области 1, а для примера 3 - приближается к 2.