Содержание

Подвариант 1

Цель работы

Изучить классический и модифицированный методы Гаусса, применяемые для решения системы линейных алгебраических уравнений (СЛАУ).

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Написать программу, решающую СЛАУ заданного пользователем размера методом Гаусса и методом Гаусса с выбором главного элемента.

Цели и задачи практической работы

- Изучить метод Гаусса для решения СЛАУ
- Реализовать метод Гаусса и метод Гаусса с выбором главного элемента для решения заданной СЛАУ
- Реализовать алгоритм вычисления определителя матрицы A
- Реализовать алгоритм вычисления обратной матрицы A^{-1}
- Реализовать алгоритм вычисления числа обусловленности $M_A = \|A\| \times \|A^{-1}\|$
- Исследовать вопрос вычислительной устойчивости метода Гаусса
- Проверить корректность работы программ системой тестов

Описание метода решения

Рассмотрим СЛАУ вида $Ax = f$, с невырожденной матрицей $A_{n\times n} = (a_{ij})$:

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

Метод Гаусса заключается в нахождении такого вектора x, при котором было бы выполнено равенство $Ax = f$.

Алгоритм можно разбить на 2 этапа: прямой и обратный ход.

- Прямой ход заключается в преобразовании СЛАУ к эквивалентной системе с верхнетреугольной матрицей при выполнении нескольких аналогичных итераций. Рассмотрим на примере первой итерации:
    1. Разделим все члены первого уравнения на $a_{11} \neq 0$, если же $a_{11} = 0$, то находим уравнение, у которого коэффициент при x_1 не равен нулю (т.е. найдем уравнение с номером i, у которого $a_{i1} \neq 0$) и поменяем их местами. Оно обязательно будет существовать, поскольку иначе в матрице A был бы нулевой столбец и, следовательно, ее определитель был бы равен нулю. A по условию матрица A должна быть невырождена.
    2. Затем из всех остальных уравнений с номерами $i = 2, 3, \dots, n$ вычитаем первое уравнение, умноженное на a_{i1}. И таким образом останется только одно уравнение (первое) с ненулевым коэффициентом при x_1:

$$\begin{cases} x_1 + \frac{a_{12}}{a_{11}} x_2 + \dots + \frac{a_{1n}}{a_{11}} \cdot x_n = \frac{f_1}{a_{11}} \\ a'_{22} \cdot x_2 + \dots + a'_{2n} \cdot x_n = f'_2 \\ \dots \\ a'_{n2} \cdot x_2 + \dots + a'_{nn} \cdot x_n = f'_n \end{cases}$$

$$\text{Где } a'_{ij} = a_{ij} - a_{i1} \cdot \frac{a_{1j}}{a_{11}} \text{ и } f'_i = f_i - a_{i1} \cdot \frac{f_1}{a_{11}}$$

Потом необходимо выполнить аналогичные действия начиная со второго уравнения и коэффициента a_{22}. И так далее до последнего уравнения с номером $i = n$. Тогда СЛАУ будет иметь вид:

$$\begin{cases} x_1 + c_{12} \cdot x_2 + \dots + c_{1n} \cdot x_n = y_1 \\ x_2 + \dots + c_{2n} \cdot x_n = y_2 \\ \dots \\ x_n = y_n \end{cases}$$

Где c_{ij} и y_i – коэффициенты, полученные путем указанных преобразований.

- Обратный ход заключается в вычислении решения путем вычитаний уравнений, но теперь с конца. Из предыдущего этапа мы уже получили $x_n = y_n$. Далее вычтем из уравнения с номером $i = n - 1$ последнее уравнение, умноженное на $c_{n-1,n}$, из уравнения с номером $i = n - 2$ последнее уравнение, умноженное на $c_{n-2,n-1}$, и так далее. После $n - 1$ шагов получим систему, в которой ненулевой коэффициент при x_n только у последнего уравнения.

Аналогично проделаем те же операции начиная с уравнения с номером $i = n - 1$ и коэффициентом при x_{n-1}, и так далее до первого уравнения. В результате получим решение СЛАУ:

$$\begin{cases} x_1 = y_1 - c_{12} \cdot x_2 - \ldots - c_{1n} \cdot x_n \\ x_2 = y_2 - c_{23} \cdot x_3 - \ldots - c_{2n} \cdot x_n \\ \qquad\qquad \ldots \\ x_n = y_n \end{cases}$$

Метод Гаусса с выбором главного элемента отличается тем, что во время прямого хода необходимо делить текущее уравнение с номером i не на a_{ii}, а на максимальный коэффициент при всех компонент вектора x. Вследствие этого нужно контролировать порядок элементов вектора x, так как придется менять столбцы матрицы A.

Определитель матрицы

Для вычисления определителя матрицы используем следующий алгоритм. Так как определитель верхнетреугольной матрицы равен произведению ее диагональных элементов, можем воспользоваться прямым обходом из метода Гаусса для приведения матрицы к верхнетреугольному виду и будем учитывать, что при делении строки на число, определитель умножается на это число.

Обратная матрица

Для вычисления обратной матрицы используем метод Гаусса-Жордана, а именно: проведем над единичной матрицей I элементарные преобразования, которыми невырожденная матрица A приводится к единичной. И получим из единичной матрицы I обратную матрицу A^{-1}. Эти элементарные преобразования проводим в расширенной матрице $A|I$. А чтобы привести матрицу A к единичной, сначала приведем ее к верхнетреугольной с помощью прямого обхода из метода Гаусса, а затем к нижнетреугольной методом Гаусса для верхней части. Таким образом получим нормированную главную диагональ.

Число обусловленности

Вычисляем норму матрицы по следующей формуле: $\|A\| = \max\limits_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$

И само число обусловленности: $M_A = \|A\| \times \|A^{-1}\|$

Описание программ

1. Функция create_matrix создает матрицу порядка size

```c
double**
create_matrix(int size)
{
    double **matrix = (double**)malloc(sizeof(double*) * size);

    for (int i = 0; i < size; i++) {
        matrix[i] = (double*)malloc(sizeof(double) * size);
    }

    return matrix;
}
```

2. Функция destroy_matrix освобождает память, занятую матрицей порядка size

```c
void
destroy_matrix(double **matrix, int size)
{
    for (int i = 0; i < size; i++) {
        free(matrix[i]);
    }

    free(matrix);
}
```

3. Функция copy_matrix создает копию матрицы порядка size

```c
double**
copy_matrix(double **matrix, int size)
{
    double **matrix_copy = create_matrix(size);

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix_copy[i][j] = matrix[i][j];
        }
    }

    return matrix_copy;
}
```

4. Функция create_vector создает вектор длины size

```c
double*
create_vector(int size)
{
    double *vector = (double*)malloc(size * sizeof(double));
    return vector;
}
```

5. Функция destroy_vector освобождает память, занятую вектором длины size

```c
void
destroy_vector(double *vector)
{
    free(vector);
}
```

6. Функция create_augmented_matrix создает новую матрицу, склеенную из матрицы коэффициентов A и вектора правой части f.

```c
double**
create_augmented_matrix(double **matrix, double *vector, int size)
{
    double **aug_matrix = copy_matrix(matrix, size);

    for (int i = 0; i < size; i++) {
        aug_matrix[i] = (double*)realloc(aug_matrix[i], (size + 1) * sizeof(double));
        aug_matrix[i][size] = vector[i];
    }

    return aug_matrix;
}
```

7. Функция create_extended_matrix создает расширенную матрицу $A|I$ для matrix

```c
double**
create_extended_matrix(double **matrix, int size)
{
    double **ext_matrix = copy_matrix(matrix, size);

    for (int i = 0; i < size; i++) {
        ext_matrix[i] = (double*)realloc(ext_matrix[i], (2 * size) * sizeof(double));
    }

    for (int i = 0; i < size; i++) {
        for (int j = size; j < 2 * size; j++) {
            (i + size == j) ? (ext_matrix[i][j] = 1) : (ext_matrix[i][j] = 0);
        }
    }

    return ext_matrix;
}
```

8. Функция swap_rows меняет местами строки матрицы

```
void
swap_rows(double **r_1, double** r_2)
{
    double *inv_matrix = *r_1;
    *r_1 = *r_2;
    *r_2 = inv_matrix;
}
```

9. Функция swap_elements меняет местами элементы матрицы

```
void
swap_elements(double *elm_1, double* elm_2)
{
    double inv_matrix = *elm_1;
    *elm_1 = *elm_2;
    *elm_2 = inv_matrix;
}
```

10. Функция swap_columns меняет местами столбцы матрицы

```
void
swap_columns(double **matrix, int size, int clm_1, int clm_2)
{
    for (int i = 0; i < size; i++) {
        swap_elements(&matrix[i][clm_1], &matrix[i][clm_2]);
    }
}
```

11. Функция find_row осуществляет поиск строки матрицы, с ненулевым элементом в данном столбце

```
int
find_row(double **matrix, int size, int row)
{
    for (int i = row + 1; i < size; i++) {
        if (matrix[i][row] != 0) {
            return i;
        }
    }

    return row;
}
```

12. Функция find_main_element осуществляет поиск главного элемент матрицы в данной строке

```c
int
find_main_element(double **matrix, int size, int row)
{
    int ind_main = row;

    for (int i = row + 1; i < size; i++) {
        if (fabs(matrix[row][ind_main]) < fabs(matrix[row][i])) {
            ind_main = i;
        }
    }

    return ind_main;
}
```

13. Функция substruct_rows вычитает из строки row_2 строку row_1, умноженную на коэффициент k

```c
void
substract_rows(double **matrix, int size, int row_1, int row_2, double k)
{
    for (int i = 0; i < size; i++) {
        matrix[row_2][i] -= matrix[row_1][i] * k;
    }
}
```

14. Функция gauss_direct_bypass реализует прямой ход из метода Гаусса

```c
double
gauss_direct_bypass(double** matrix, int row_nmb, int clm_mnb, int row, int find_main, int *order)
{
    if (matrix[row][row] == 0) {
        int choice_row = find_row(matrix, row_nmb, row);
        swap_rows(&matrix[row], &matrix[choice_row]);
    }

    if (find_main == 1) {
        int ind_main = find_main_element(matrix, clm_mnb - 1, row);
        swap_columns(matrix, row_nmb, row, ind_main);
        int inv_matrix = order[row];
        order[row] = order[ind_main];
        order[ind_main] = inv_matrix;
    }

    double res = matrix[row][row];

    for (int i = row; i < clm_mnb; i++) {
        matrix[row][i] /= res;
    }

    for (int i = row + 1; i < row_nmb; i++) {
        substract_rows(matrix, clm_mnb, row, i, matrix[i][row]);
    }

    return res;
}
```

15. Функция gauss_reverse_bypass реализует обратный ход из метода Гаусса

```c
void
gauss_reverse_bypass(double **matrix, int row_nmb, int clm_mnb, int row) {
    for (int i = row + 1; i < row_nmb; i++) {
        substract_rows(matrix, clm_mnb, i, row, matrix[row][i]);
    }
}
```

16. Функция determinant вычисляет определитель матрицы

```c
double
determinant(double **matrix, int size)
{
    double det = 1;
    double **matrix_copy = copy_matrix(matrix, size);

    for (int i = 0; i < size; i++) {
        det *= gauss_direct_bypass(matrix_copy, size, size, i, 0, NULL);
    }

    destroy_matrix(matrix_copy, size);

    return det;
}
```

17. Функция inverse_matrix вычисляет обратную матрицу

```c
double**
inverse_matrix(double **matrix, int size)
{
    if (determinant(matrix, size) == 0) {
        printf("Матрица вырождена\n");
        return NULL;
    }

    double **ext_matrix = create_extended_matrix(matrix, size);

    for (int i = 0; i < size; i++) {
        gauss_direct_bypass(ext_matrix, size, 2 * size, i, 0, NULL);
    }

    for (int i = 0; i < size; i++) {
        gauss_reverse_bypass(ext_matrix, size, 2 * size, i);
    }

    double **inv_matrix = create_matrix(size);

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            inv_matrix[i][j] = ext_matrix[i][j + size];
        }
    }

    destroy_matrix(ext_matrix, size);

    return inv_matrix;
}
```

18. Функция gauss реализует обычный метод Гаусса

```c
double*
gauss(double **matrix, double *vector, int size)
{
    if (determinant(matrix, size) == 0) {
        printf("Матрица вырождена\n");
        return NULL;
    }

    double **aug_matrix = create_augmented_matrix(matrix, vector, size);

    for (int i = 0; i < size; i++) {
        gauss_direct_bypass(aug_matrix, size, size + 1, i, 0, NULL);
    }

    for (int i = 0; i < size; i++) {
        gauss_reverse_bypass(aug_matrix, size, size + 1, i);
    }

    double *ans = create_vector(size);

    for (int i = 0; i < size; i++) {
        ans[i] = aug_matrix[i][size];
    }

    destroy_matrix(aug_matrix, size);

    return ans;
}
```

19. Функция gauss_main_choice реализует метод Гаусса с выбором главного элемента

```c
double*
gauss_main_choice(double **matrix, double *vector, int size)
{
    if (determinant(matrix, size) == 0) {
        printf("Матрица вырождена\n");
        return NULL;
    }

    double **aug_matrix = create_augmented_matrix(matrix, vector, size);
    int* order = (int*)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        order[i] = i;
    }

    for (int i = 0; i < size; i++) {
        gauss_direct_bypass(aug_matrix, size, size + 1, i, 1, order);
    }

    for (int i = 0; i < size; i++) {
        gauss_reverse_bypass(aug_matrix, size, size + 1, i);
    }

    double *ans = create_vector(size);

    for (int i = 0; i < size; i++) {
        ans[i] = aug_matrix[order[i]][size];
    }

    destroy_matrix(aug_matrix, size);

    return ans;
}
```

20. Функция cond_num вычисляет число обусловленности

```
double
cond_num(double** matrix, int size)
{
    double cond_1 = 0, cond_2 = 0;
    double **inv_matrix = inverse_matrix(matrix, size);

    for (int i = 0; i < size; i++) {
        double sum_1 = 0, sum_2 = 0;

        for (int j = 0; j < size; j++) {
            sum_1 += fabs(matrix[i][j]);
            sum_2 += fabs(inv_matrix[i][j]);
        }

        if ((!i) || (sum_1 > cond_1)) {
            cond_1 = sum_1;
        }

        if ((!i) || (sum_2 > cond_2)) {
            cond_2 = sum_2;
        }
    }

    destroy_matrix(inv_matrix, size);

    return (cond_1 * cond_2);
}
```

Тесты и проверка корректности

Корректность работы программы проверена с помощью wolfram и примеров, представленных в вариантах задания.

Тест №1 (приложение 1-7)

$$\begin{cases} 2x_1 - x_2 - 6x_3 + 3x_4 = -1, \\ 7x_1 - 4x_2 + 2x_3 - 15x_4 = -32, \\ x_1 - 2x_2 - 4x_3 + 9x_4 = 5, \\ x_1 - x_2 + 2x_3 - 6x_4 = -8. \end{cases}$$

Результаты работы программы:

- Определитель матрицы коэффициентов A: -252
- Число обусловленности матрицы коэффициентов A: 51.333
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} -0.214286 & 0.380952 & -0.071429 & -1.166667 \\ -0.071429 & 0.238095 & -0.357143 & -1.166667 \\ -0.285714 & 0.119048 & 0.071429 & -0.333333 \\ -0.119048 & 0.063492 & 0.071429 & -0.277778 \end{pmatrix}$$

- Решение системы методом Гаусса*:* $x = (-3, 0, -0.5, 0.666667)$
- Решение системы методом Гаусса с выбором главного элемента*:*

$$x = (-3, 0, -0.5, 0.666667)$$

Результаты, полученные в wolfram:

- Определитель матрицы коэффициентов A: -252
- Число обусловленности матрицы коэффициентов A: $51\frac{1}{3}$
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} -\frac{3}{14} & \frac{8}{21} & -\frac{1}{14} & -\frac{7}{6} \\ -\frac{1}{14} & \frac{5}{21} & -\frac{5}{14} & -\frac{7}{6} \\ -\frac{2}{7} & \frac{5}{42} & \frac{1}{14} & -\frac{1}{3} \\ -\frac{5}{42} & \frac{4}{63} & \frac{1}{14} & -\frac{5}{18} \end{pmatrix}$$

- Решение системы $x = (-3, 0, -\frac{1}{2}, \frac{2}{3})$

Результат прошел проверку на корректность.

Тест №2 (приложение 1-7)

$$\begin{cases} x_1 - 2x_2 + x_3 + x_4 = 0, \\ 2x_1 + x_2 - x_3 - x_4 = 0, \\ 3x_1 - x_2 - 2x_3 + x_4 = 0, \\ 5x_1 + 2x_2 - x_3 + 9x_4 = -10. \end{cases}$$

Результаты работы программы:

- Определитель матрицы коэффициентов A: -118
- Число обусловленности матрицы коэффициентов A: 23.0508
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} 0.279661 & 0.423729 & -0.084746 & 0.025424 \\ -0.161017 & 0.271186 & -0.254237 & 0.076271 \\ 0.466102 & 0.372881 & -0.474576 & 0.042373 \\ -0.067797 & -0.254237 & 0.050847 & 0.084746 \end{pmatrix}$$

- Решение системы методом Гаусса:

$$x = (-0.254237, -0.762712, -0.423729, -0.847458)$$

- Решение системы методом Гаусса с выбором главного элемента*:*

$$x = (-0.254237, -0.762712, -0.423729, -0.847458)$$

Результаты, полученные в wolfram:

- Определитель матрицы коэффициентов A: -118
- Число обусловленности матрицы коэффициентов A: 23.050847
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} \frac{33}{118} & \frac{25}{59} & -\frac{5}{59} & \frac{3}{118} \\ -\frac{19}{118} & \frac{16}{59} & -\frac{15}{59} & -\frac{9}{118} \\ \frac{55}{118} & \frac{22}{59} & -\frac{28}{59} & \frac{5}{118} \\ -\frac{4}{59} & -\frac{15}{63} & \frac{3}{59} & \frac{5}{59} \end{pmatrix}$$

- Решение системы: $x = (-\frac{15}{59}, -\frac{45}{59}, -\frac{25}{59}, -\frac{50}{59})$

Результат прошел проверку на корректность.

Тест №3 (приложение 1-7)

$$\begin{cases} 8x_1 + 6x_2 + 5x_3 + 2x_4 = 21, \\ 3x_1 + 3x_2 + 2x_3 + x_4 = 10, \\ 4x_1 + 2x_2 + 3x_3 + x_4 = 8, \\ 7x_1 + 4x_2 + 5x_3 + 2x_4 = 18. \end{cases}$$

Результаты работы программы:

- Определитель матрицы коэффициентов A: 1
- Число обусловленности матрицы коэффициентов A: 210
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} 1 & -2 & -2 & 1 \\ 0 & 1 & 1 & -1 \\ -1 & 2 & 4 & -2 \\ -1 & 0 & -5 & 4 \end{pmatrix}$$

- Решение системы методом Гаусса*:* $x = (3, 0, -5, 11)$
- Решение системы методом Гаусса с выбором главного элемента*:*

$$x = (3, 0, -5, 11)$$

Результаты, полученные в wolfram:

- Определитель матрицы коэффициентов A: 1
- Число обусловленности матрицы коэффициентов A: 210
- Обратная матрица к матрице коэффициентов A:

$$\begin{pmatrix} 1 & -2 & -2 & 1 \\ 0 & 1 & 1 & -1 \\ -1 & 2 & 4 & -2 \\ -1 & 0 & -5 & 4 \end{pmatrix}$$

- Решение системы методом Гаусса: $x = (3, 0, -5, 11)$
- Решение системы методом Гаусса с выбором главного элемента*:*

$$x = (3, 0, -5, 11)$$

Результат прошел проверку на корректность.

Приложение 2-1

Тест №4

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), & i \neq j, \\ (q_M - 1)^{i+j}, & i = j, \end{cases} \text{ где } q_M = 1.001 - 2 \cdot M \cdot 10^{-3}, M = 1, n = 50$$

$$b_i = n \cdot \mathrm{e}^{\frac{x}{i}} \cdot \cos(x)$$

- Определитель матрицы коэффициентов A: 445.942460
- Число обусловленности матрицы коэффициентов A: 819.922982

Результат прошел проверку на корректность.

Исследование вычислительной устойчивости метода Гаусса

Метод Гаусса является вычислительно неустойчивым для плохо обусловленных матриц. В качестве примера можно привести матрицу Гильберта. При работе с ней метод Гаусса приводит к большим ошибкам, даже при малых порядках матрицы.

Уменьшить данные вычислительные ошибки можно с помощью метода Гаусса с выделением главного элемента. Этот метод является условно устойчивым, поскольку его погрешность не возрастает, так как при делении на максимальный элемент все числа в строке не превосходят единицу.

Далее приведена таблица, в которой сравниваются нормы невязок $\|Ax - f\|$ метода Гаусса и метод Гаусса с выделением главного элемента. Матрицы заполнялись рандомными числами.

Порядок	Метод Гаусса	Метод Гаусса с выделением главного элемента
10	1.234323e-10	1.345743e-14
50	8.438026e-10	8.340309e-14
100	3.385839e-08	2.486504e-12

Вывод

При выполнении практической работы были реализованы функции для вычисления определителя матрицы, обратной матрицы, числа обусловленности, был изучен и реализован метод Гаусса. Также было установлено, что он удобен при решении систем небольших порядков. А для матриц больших порядков и плохо обусловленных матриц рациональнее использовать метод Гаусса с выбором главного элемента, так как он более вычислительно устойчив к подсчёту корней уравнения.

Подвариант 2

Цель работы

Изучить классические итерационные методы Зейделя и верхней релаксации, используемые для численного решения СЛАУ; изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра.

Постановка задачи

Дана система уравнений $Ax=f$ порядка $n\times n$ с невырожденной матрицей A. Написать программу численного решения данной системы линейных алгебраических уравнений (n – параметр программы), использующую численный алгоритм итерационного метода Зейделя: $(D+A^{(-)})(x^{k+1}-x^k)+Ax^k=f$, где $D, A^{(-)}$ - соответственно диагональная и нижняя треугольные матрицы, k - номер текущей итерации. В случае использования итерационного метода верхней релаксации итерационный процесс имеет следующий вид: $(D+\omega A^{(-)})\frac{x^{k+1}-x^k}{\omega}+Ax^k=f$, где ω - итерационный параметр (при $\omega=1$ метод верхней релаксации переходит в метод Зейделя).

Цели и задачи практической работы

- Решить заданную СЛАУ итерационным методом методом верхней релаксации
- Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью
- Изучить скорость сходимости итераций к точному решению задачи в зависимости от итерицонного параметра ω
- Проверить корректность работы программ системой тестов

Описание методов решения

Рассмотрим произвольную самосопряженную положительно определенную квадратную матрицу A и разложим ее в сумму $A = D + T_н + T_в$:

$$A_{n\times n}\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \quad A = A^*, A > 0$$

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} \quad T_{\text{Н}} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix} \quad T_{\text{В}} = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

где D - диагональная, $T_{\text{Н}}$ - нижняя треугольная, $T_{\text{В}}$ - верхняя треугольная матрицы из элементов матрицы A.

Введём параметр $\omega > 0$: $(D + \omega \text{T}_{\text{Н}})\frac{x_{k+1}-x_k}{\omega} + Ax_k = f$

Внесём ω в левую часть: $(\frac{1}{\omega}D + T_{\text{Н}})(x_{k+1} - x_k) + Ax_k = f$

Перегруппируем слагаемые и с учётом $A = D + T_{\text{Н}} + T_{\text{В}}$ получим:

$$(\frac{1}{\omega}D + T_{\text{Н}})x_{k+1} + ((1 - \frac{1}{\omega})D + T_B)x_k = f$$

Теперь рассмотрим условие сходимости метода при $A = A^*, A > 0$:

$A = A^* \Rightarrow T_{\text{Н}}^* = T, T_{\text{В}}^* = \text{T}_{\text{Н}} \Rightarrow (\text{T}_{\text{Н}}x, x) = (T_{\text{Н}}^*x, x) = (T_{\text{В}}x, x)$

Составим матрицу $C = B - \frac{\tau}{2}A$ и получим:

$$C = B - \frac{\tau}{2}A = (1 - \frac{\omega}{2})D + \frac{\omega}{2}(T_H - T_B)$$

Матрица C будет положительно определенной при условии:

$$(Cx, x) = ((B - \frac{\tau}{2}A)x, x) = (1 - \frac{\omega}{2})(Dx, x) + \frac{\omega}{2}((T_H - T_B)x, x) > 0$$

Что означает $(1 - \frac{\omega}{2})(Dx, x) > 0$

И поскольку $A > 0 \Rightarrow \text{D} > 0 \Rightarrow (\text{Dx}, \text{x}) > 0$, то:

$$C = B - \frac{\tau}{2}A > 0 \Leftrightarrow 1 - \frac{\omega}{2} > 0 \Leftrightarrow \omega < 2$$

С учётом $\omega > 0$ получим достаточное условие для сходимости итерационной последовательности метода верхней релаксации $0 < \omega < 2$.

Теперь получим рекурентную формулу для i–й компоненты последовательности. Для этого перейдём от векторной записи к покомпонентной (где $i = 1,\ldots,n$):

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}(f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=1}^{n} a_{ij}x_j^k$$

Наконец, получим условие остановки итерационного процесса. Для оценки расстояние между x_k, найденым на k–й итерации решением, и x, решением СЛАУ, используется невязка: $\psi_k = Ax_k - f$. Обозначим $z_k = x_k - x$. Тогда:

$$\psi_k = Ax_k - f = A(z_k + x) - f = Az_k \qquad \Rightarrow \qquad ||x_k - x|| = ||z_k|| \le ||A^{-1}|| \cdot ||\psi_k||$$

В качестве условия остановки итерационного процесса положим:
$||A^{-1}|| \cdot ||\psi_k|| < \varepsilon$, где ε - заданная точность вычислений.

Описание программ

1. Функция mul_mv перемножает матрицу и вектор

```c
double*
mul_mv(double **matrix, double *vector, int size)
{
    double *mul = create_vector(size);

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            mul[i] += matrix[i][j] * vector[j];
        }
    }

    return mul;
}
```

2. Функция mul_mm перемножает две матрицы

```c
double**
mul_mm(double **matrix_1, double **matrix_2, int size)
{
    double **mul = create_matrix(size);

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                mul[i][j] += matrix_1[i][k] * matrix_2[k][j];
            }
        }
    }

    return mul;
}
```

3. Функция transpose_matrix транспонирует матрицу

```c
double**
transpose_matrix(double **matrix, int size)
{
    double **trp = create_matrix(size);

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            trp[i][j] = matrix[j][i];
        }
    }

    return trp;
}
```

4. Функция difference вычисляет норму невязки решения СЛАУ

```c
double
difference(double *vector_1, double *vector_2, int size)
{
    double res = 0;

    for (int i = 0; i < size; ++i) {
        res += (vector_1[i] - vector_2[i]) * (vector_1[i] - vector_2[i]);
    }

    return sqrt(res);
}
```

5. Функция relaxation вычисляет решение СЛАУ методом верхней релаксации с параметром ω и точностью EPS

```c
void
relaxation(double **matrix, double *vector, int size, double w)
{
    int cnt = 0;
    double **trp_matrix = transpose_matrix(matrix, size);
    double **mul_matrix = mul_mm(trp_matrix, matrix, size);
    double *vector_change = mul_mv(trp_matrix, vector, size);
    double *vector_ans = create_vector(size);
    double *vector_prev = create_vector(size);

    for (int i = 0; i < size; i++) {
        vector_ans[i] = 0;
        vector_prev[i] = 1;
    }

    while (difference(vector_ans, vector_prev, size) > EPS) {
        cnt++;

        for (int i = 0; i < size; i++) {
            vector_prev[i] = vector_ans[i];
        }

        for (int i = 0; i < size; i++) {
            double sum = 0;

            for (int j = 0; j < i; j++) {
                sum += (mul_matrix[i][j] * vector_ans[j]);
            }

            for (int j = i; j < size; j++) {
                sum += (mul_matrix[i][j] * vector_prev[j]);
            }

            vector_ans[i] = w * (vector_change[i] - sum) / mul_matrix[i][i] + vector_prev[i];
        }
    }
    printf("\tЧисло итераций: %d\n", cnt);
    printf("\tЗначение w: %lf\n", w);
    for (int i = 0; i < size; i++) {
        printf("x%d = %lf\n", i + 1, vector_ans[i]);
    }

    destroy_matrix(trp_matrix, size);
    destroy_matrix(mul_matrix, size);
    destroy_vector(vector_change);
    destroy_vector(vector_prev);
    destroy_vector(vector_ans);
}
```

Тесты и проверка корректности

Тест№1

$$\begin{cases} 2x_1 - x_2 - 6x_3 + 3x_4 = -1, \\ 7x_1 - 4x_2 + 2x_3 - 15x_4 = -32, \\ x_1 - 2x_2 - 4x_3 + 9x_4 = 5, \\ x_1 - x_2 + 2x_3 - 6x_4 = -8. \end{cases}$$

Решение системы $x = (-3, 0, -0.5,\ 0.666667)$ совпало с верным с точностью $\varepsilon = 10^{-6}$

Значение ω	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
Количество итераций	6972	3217	1757	1261	957	687	461	246	259

Тест№2

$$\begin{cases} x_1 - 2x_2 + x_3 + x_4 = 0, \\ 2x_1 + x_2 - x_3 - x_4 = 0, \\ 3x_1 - x_2 - 2x_3 + x_4 = 0, \\ 5x_1 + 2x_2 - x_3 + 9x_4 = -10. \end{cases}$$

Решение системы $x = (-0.254237, -0.762712, -0.423729, -0.847458)$ совпало с верным с точностью $\varepsilon = 10^{-6}$

Значение *ω*	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
Количество итераций	973	473	293	197	135	90	47	57	121

Тест№3

$$\begin{cases} 8x_1 + 6x_2 + 5x_3 + 2x_4 = 21, \\ 3x_1 + 3x_2 + 2x_3 + x_4 = 10, \\ 4x_1 + 2x_2 + 3x_3 + x_4 = 8, \\ 7x_1 + 4x_2 + 5x_3 + 2x_4 = 18. \end{cases}$$

Решение системы $x = (3, 0, -5, 11)$ совпало с верным с точностью $\varepsilon = 10^{-6}$

Значение ω	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
Количество итераций	127830	61925	37946	25185	17054	11192	6113	5517	10876

Тест№4

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), & i \neq j, \\ (q_M - 1)^{i+j}, & i = j, \end{cases} \text{ где } q_M = 1.001 - 2 \cdot M \cdot 10^{-3}, M = 1, n = 50$$

$$b_i = n \cdot \mathrm{e}^{\frac{x}{i}} \cdot \cos(x)$$

Наилучшая сходимость при значении параметра: $\omega = 1.4$

Количество итераций при данном $\omega = 1.4$: 8217

Выводы

При выполнении практической работы был изучен и реализован метод верхней релаксации для решения СЛАУ, разработан критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы с заранее заданной точностью, исследована скорость сходимости метода в зависимости от итерационного параметра ω. Из проведенных тестов следует, что скорость сходимости данного метода существенно зависит от итерационного параметра, но при этом не существует оптимального параметра для всех матриц. Оптимальный параметр для скорости сходимости зависит от самих матриц исходных СЛАУ.