МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

«ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1

ОТЧЕТ

о выполненном задании

студента 202 учебной группы факультета ВМК МГУ

Струкова Павла Вячеславовича

гор. Москва

2022 г.

Оглавление

Подвариант 1

Постановка задачи. Цель работы.

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Требуется написать программу, решающую систему линейных алгебраических уравнений заданного размера методом Гаусса и методом Гаусса с выбором главного элемента.

Цель работы – изучить классический метод Гаусса (а также модифицированный метод Гаусса), применяемый для решения системы линейных алгебраических уравнений.

Задачи практической работы:

- Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;
- Вычислить определитель матрицы $det(A)$;
- Вычислить обратную матрицу A^{-1};
- Определить число обусловленности $M_A = \|A\| \times \|A^{-1}\|$;
- Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра n.

Описание метода решения.

Рассмотрим систему линейных алгебраических уравнений вида $Ax = b$ с невырожденной матрицей $A = (a_{ij})$.

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

Метод Гаусса

Разделим метод Гаусса на два этапа:

1. Приводим систему к верхнему треугольному типу
2. Последовательно находим неизвестные переменные

Первый этап называется прямым ходом, второй – обратным.

- Прямой ход:
  - Если первый элемент a_{11} равен 0, то меняем первую строку с некоторой строкой с номером i, у которой $a_{i1} \neq 0$. Она существует, так как иначе был бы нулевой столбец, а значит, нулевой определитель, но это противоречит условию. Разделим первую строку матрицы на a_{11}, вычтем из каждой последующей i-ой строки первую строку, умноженную на a_{i1}. Таким образом останется только одна строка с ненулевым элементом в первом столбце. Повторим эти же шаги для последующих подсистем уравнений. В итоге получим $Ux = \tilde{f}$, где U – верхняя треугольная матрица, $\tilde{f}$ – измененная правая часть.
- Обратный ход:
  - Из $n - 1$ уравнения вычтем n-ое уравнение, умноженное на $u_{n-1,n}$. Из $n - 2$ уравнения вычтем $n - 1$-ое, умноженное на $u_{n-2,n-1}$ и n-ое уравнение, умноженное на $u_{n-2,n}$ и т. д. После $n - 1$ получим систему уравнений $Ex = b$, где E – единичная матрица, $b = (x_1, x_2, \ldots, x_n)^T$ – решение системы.

Метод Гаусса с главным элементом

Модифицированный метод Гаусса отличается от рассмотренного тем, что в начале каждого шага прямого хода в качестве первой строки выбирается та строка, для которой первый коэффициент при x максимальный по модулю. Остальные шаги выполняются аналогично.

Определитель матрицы

Определитель верхней треугольной матрицы равен произведению её диагональных элементов. Для приведения матрицы к верхнетреугольному виду, воспользуемся описанным выше прямым ходом метода Гаусса и будем учитывать, что при делении строки на элемент матрицы, определитель умножается на этот элемент.

Обратная матрица

Для подсчёта обратной матрицы воспользуемся методом Гаусса-Жордана. Если к единичной матрице I применить элементарные преобразования, которыми матрица A приводится к I, то получится обратная матрица A^{-1}. Все преобразования будем проводить с расширенной матрицей A|I. Приведение матрицы к единичному виду происходит путем приведения её к верхней треугольной, а затем к нижней треугольной методом Гаусса для верхней части. Таким образом получим нормированную главную диагональ.

Число обусловленности матрицы

Число обусловленности матрицы $M_A = \|A\| \times \|A^{-1}\|$

считаем по норме: $\|A\| = \max\limits_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$.

Описание программы

Программа позволяет:

1. Вычислить определитель матрицы
2. Найти число обусловленности матрицы $M_A = \|A\| \times \|A^{-1}\|$, с нормой $\|A\| = \max\limits_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$.
3. Найти обратную матрицу
4. Решить систему линейных алгебраических уравнений методом Гаусса и методом Гаусса с выбором главного элемента

Описание функций

1. Функция $\boldsymbol{double}$ $**$ $\boldsymbol{create\_matrix(int\ n)}$ создает квадратную матрицу порядка n.

```
double **create_matrix(int n)
{
    double **matrix = calloc(n, sizeof(*matrix));
    for (int i = 0; i < n; i++) {
        matrix[i] = calloc(n, sizeof(double));
    }

    return matrix;
}
```

2. Функция $\boldsymbol{double}$ $*$ $\boldsymbol{create\_vector(int\ n)}$ создает вектор длины n.

```
double *create_vector(int n)
{
    double *vector = calloc(n, sizeof(*vector));
    return vector;
}
```

3. Функция $\boldsymbol{double}$ $**$ $\boldsymbol{copy\_matrix(double\ **matrix, int\ n)}$ создает копию квадратной матрицы $matrix$ порядка n.

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

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

    return matrix_copy;
}
```

4. Функция $\boldsymbol{int\ find\_row(double\ **matrix, int\ n, int\ row)}$ в матрице $matrix$ находит ненулевой элемент в столбце row, начиная со строки $row + 1$.

```c
int find_row(double **matrix, int n, int row)
{
    for (int i = row + 1; i < n; i++) {
        if (fabs(matrix[i][row]) > EPS) {
            return i;
        }
    }

    return row;
}
```

5. Функции $\boldsymbol{void\ swap\_rows(double\ **r\_1, double\ **r\_2)}$, $\boldsymbol{void\ swap\_elements(double\ *elm\_1, double\ *elm\_2)}$, $\boldsymbol{void\ swap\_column(double\ **matrix, int\ n, int\ clm\_1, int\ clm\_2)}$ меняют местами строки, элементы и столбцы соответственно.

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

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

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

6. Функция $\boldsymbol{void\ strsub(double\ **matrix, int\ n, int\ row\_1, int\ row\_2, double\ k)}$ в матрице из строки row_2 вычитает строку row_1, умноженную на k.

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

7. Функция $\boldsymbol{int\ find\_max\_element(double\ **\ matrix, int\ n, int\ row)}$ находит номер строки, в которой находится максимальный элемент в столбце row среди элементов в строках $row - n$.

```
int find_max_element(double **matrix, int n, int row)
{
    int i_max = row;
    double max = fabs(matrix[row][i_max]);

    for (int i = row + 1; i < n; i++) {
        if (fabs(matrix[row][i]) > max) {
            i_max = i;
            max = fabs(matrix[row][i]);
        }
    }

    return i_max;
}
```

8. Функция $\boldsymbol{void\ fill\_matrix(double\ **\ matrix, double\ *\ vector, int\ n, FILE\ *\ fp)}$ считывает элементы из файла fp и записывает их в матрицу $matrix$ и вектор $vector$.

```
void fill_matrix(double **matrix, double *vector, int n, FILE *fp)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            fscanf(fp, "%lf", &matrix[i][j]);
        }

        fscanf(fp, "%lf", &vector[i]);
    }
}
```

9. Функции $\boldsymbol{double\ **\ create\_np1\_matrix(double\ **\ matrix, double\ *\ vector, int\ n)}$, $\boldsymbol{double\ **\ create\_nx2n\_matrix(double\ **\ matrix, int\ n)}$ создает присоединенную и расширенную матрицы соответственно.

```c
double **create_np1_matrix(double **matrix, double *vector, int n)
{
    double **np1_matrix = copy_matrix(matrix, n);

    for (int i = 0; i < n; i++) {
        np1_matrix[i] = realloc(np1_matrix[i], (n + 1) * sizeof(double));
        np1_matrix[i][n] = vector[i];
    }

    return np1_matrix;
}

double **create_nx2n_matrix(double **matrix, int n)
{
    double **nx2n_matrix = copy_matrix(matrix, n);

    for (int i = 0; i < n; i++) {
        nx2n_matrix[i] = realloc(nx2n_matrix[i], (2 * n) * sizeof(double));
    }

    for (int i = 0; i < n; i++) {
        for (int j = n; j < 2 * n; j++) {
            if (i + n == j) {
                (nx2n_matrix[i][j] = 1);
            } else {
                (nx2n_matrix[i][j] = 0);
            }
        }
    }

    return nx2n_matrix;
}
```

10. Функция $\boldsymbol{double\ **\ inverse\_matrix(double\ **matrix, int\ n)}$ находит обратную матрицу к матрице $matrix$.

```c
double **inverse_matrix(double **matrix, int n)
{
    double **nx2n_matrix = create_nx2n_matrix(matrix, n);

    for (int i = 0; i < n; i++) {
        gauss_direct(nx2n_matrix, n, 2 * n, i);
    }

    for (int i = 0; i < n; i++) {
        gauss_reverse(nx2n_matrix, n, 2 * n, i);
    }

    double **inv_matrix = create_matrix(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            inv_matrix[i][j] = nx2n_matrix[i][j + n];
        }
    }

    free_matrix(nx2n_matrix, n);

    return inv_matrix;
}
```

11. Функция ***void gauss_direct(double ** matrix, int num_row, int num_clm, int row)*** делает одну итерацию прямого хода метода Гаусса.

```
void gauss_direct(double **matrix, int num_row, int num_clm, int row)
{
    if (matrix[row][row] == 0) {
        int c_row = find_row(matrix, num_row, row);
        swap_rows(&matrix[row], &matrix[c_row]);
    }

    double res = matrix[row][row];

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

    for (int i = row + 1; i < num_row; i++) {
        strsub(matrix, num_clm, row, i, matrix[i][row]);
    }
}
```

12. Функция ***void gauss_reverse(double ** matrix, int num_row, int num_clm, int row)*** делает одну итерацию обратного хода метода Гаусса

```
void gauss_reverse(double **matrix, int num_row, int num_clm, int row)
{
    for (int i = row + 1; i < num_row; i++) {
        strsub(matrix, num_clm, i, row, matrix[row][i]);
    }
}
```

13. Функция ***double determinant(double ** matrix, int n)*** находит определитель матрицы $matrix$ порядка n.

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

    for (int i = 0; i < n; i++) {
        int k;
        if (fabs(matrix_copy[i][i]) < EPS) {
            k = find_row(matrix_copy, n, i);

            swap_rows(&matrix_copy[i], &matrix_copy[k]);
            det *= -1;
        }

        for (int j = i + 1; j < n; j++) {
            if (fabs(matrix_copy[i][i]) < EPS) {
                det = 0;
                free_matrix(matrix_copy, n);
                return det;

            }

            strsub(matrix_copy, n, i, j, matrix_copy[j][i] / matrix_copy[i][i]);
        }
    }

    for (int i = 0; i < n; i++) {
        det *= matrix_copy[i][i];
    }

    free_matrix(matrix_copy, n);

    return det;
}
```

14. Функция $\boldsymbol{double * gauss(double ** matrix, double * vector, int n)}$ находит решение СЛАУ методом Гаусса.

```c
double *gauss(double **matrix, double *vector, int n)
{
    double **np1_matrix = create_np1_matrix(matrix, vector, n);

    for (int i = 0; i < n; i++) {
        gauss_direct(np1_matrix, n, n + 1, i);
    }

    for (int i = 0; i < n; i++) {
        gauss_reverse(np1_matrix, n, n + 1, i);
    }

    double *ans = create_vector(n);

    for (int i = 0; i < n; i++) {
        ans[i] = np1_matrix[i][n];
    }

    free_matrix(np1_matrix, n);

    return ans;
}
```

15. Функция $\boldsymbol{double * modified\_gauss(double ** matrix, double * vector, int n)}$ находит решение СЛАУ методом Гаусса с выбором главного элемента.

```c
double *modified_gauss(double **matrix, double *vector, int n)
{
    double **np1_matrix = create_np1_matrix(matrix, vector, n);

    int *order = calloc(n, sizeof(*order));

    for (int i = 0; i < n; i++) {
        order[i] = i;
    }

    for (int k = 0; k < n; k++) {
        if (np1_matrix[k][k] == 0) {
            int c_row = find_row(matrix, n, k);
            swap_rows(&np1_matrix[k], &np1_matrix[c_row]);
        }

        int i_max = find_max_element(np1_matrix, n, k);
        swap_columns(np1_matrix, n, k, i_max);


        int tmp = order[k];
        order[k] = order[i_max];
        order[i_max] = tmp;

        double res = np1_matrix[k][k];

        for (int i = k; i < n + 1; i++) {
            np1_matrix[k][i] /= res;
        }
```

```
        for (int i = k + 1; i < n; i++) {
            strsub(np1_matrix, n + 1, k, i, np1_matrix[i][k]);
        }
    }

    for (int i = 0; i < n; i++) {
        gauss_reverse(np1_matrix, n, n + 1, i);
    }

    double *ans = create_vector(n);

    for (int i = 0; i < n; i++) {
        ans[i] = np1_matrix[order[i]][n];
    }

    free_matrix(np1_matrix, n);
    free(order);

    return ans;
}
```

16. Функция ***double norm(double ** matrix, int n)*** находит норму матрицы.

```
double norm(double **matrix, int n)
{
    double max = 0;
    for (int i = 0; i < n; i++) {
        max += matrix[i][0];
    }

    double cur_max;

    for (int j = 1; j < n; j++) {
        cur_max = 0;
        for (int i = 0; i < n; i++) {
            cur_max += matrix[i][j];
        }

        if (cur_max > max) {
            max = cur_max;
        }
    }

    return max;
}
```

17. Функция ***double condition_num(double ** matrix, int n)*** находит число обусловленности матрицы.

```
double condition_num(double **matrix, int n)
{
    double **inv_matrix = inverse_matrix(matrix, n);
    double ans = norm(matrix, n) * norm(inv_matrix, n);
    free_matrix(inv_matrix, n);
    return ans;
}
```

Тестирование и результаты эксперимента

Тестирование программы проведено с помощью Wolfram Alpha и примеров, представленных в вариантах задания.

Приложение 1–11.

Тест 1.

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

- Определитель: 135
- Число обусловленности: 7.03704
- Обратная матрица:

$$\begin{matrix} 0.533333 & 2.22045e-16 & -0.6 & 0.333333 \\ 0.518519 & -0.222222 & -0.888889 & 0.407407 \\ -0.540741 & -0.111111 & 0.955556 & -0.296296 \\ 0.192593 & -0.111111 & -0.244444 & 0.037037 \end{matrix}$$

- Решение системы:
  x1 = 2
  x2 = 1
  x3 = -3
  x4 = 1

- Решение системы мод. Методом Гаусса:
  x1 = 2
  x2 = 1
  x3 = -3
  x4 = 1

Тест 2.

$$\begin{cases} 2x_1 - 2x_2 + x_3 - x_4 = 1, \\ x_1 + 2x_2 - x_3 + x_4 = 1, \\ 4x_1 - 10x_2 + 5x_3 - 5x_4 = 1, \\ 2x_1 - 14x_2 + 7x_3 - 11x_4 = -1. \end{cases}$$

- Определитель: 0.

Тест 3.

$$\begin{cases} 2x_1 - x_2 + 3x_3 + 4x_4 = 5, \\ 4x_1 - 2x_2 + 5x_3 + 6x_4 = 7, \\ 6x_1 - 3x_2 + 7x_3 + 8x_4 = 9, \\ 8x_1 - 4x_2 + 9x_3 + 10x_4 = 11. \end{cases}$$

- Определитель: 0.

Приложение 2, п2-4.

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

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

$M = 4; n = 100; b_i = n * exp\left(\frac{x}{i}\right) * cos(x).$

- Определитель: 5.47448e-26
- Число обусловленности: 3.22463

Вывод

В ходе решения данной работы мы реализовали прямые методы решения систем линейных алгебраических уравнений – методы Гаусса и Гаусса с выбором главного элемента. Реализовано нахождение определителя матрицы, обратной матрицы, и числа обусловленности матрицы. Корректность решений проверена с при помощи wolfram alpha.

Подвариант 2

Постановка задачи. Цель работы.

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A.

Требуется написать программу численного решения данной системы линейных алгебраических уравнений (n – параметр программы), использующий численный алгоритм итерационного метода верхней релаксации. Итерационный процесс имеет следующий вид:

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

где ω – итерационный параметр, D – диагональная матрица, A – нижняя треугольная матрица.

Рекуррентные формулы:

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

Цель работы – изучить классические итерационные методы (Зейделя и верхней релаксации), используемых для численного решения систем линейных алгебраических уравнений, а также изучения скорости сходимости этих методов в зависимости от выбора итерационного параметра.

Задачи практической работы:

- Решить заданную СЛАУ 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. Провести эксперименты с различными значениями итерационного параметра;
- Правильность решения СЛАУ подтвердить системой тестов.

Условие остановки итерационного метода

Для оценки расстояния между x_k, найденный на k-ой итерации решением и x, решением СЛАУ используется невязка: $\psi_k = Ax_k - f$.

Обозначим $z_k = x_k - x$.

Тогда $\psi_k = Ax_k - f = A(z_k + x) - f = Az_k$.

Таким образом, $\|x_k - x\| = \|z_k\| \leq \|A - 1\| \cdot \|\psi_k\|$.

В качестве условии остановки положим $\|A - 1\| \cdot \|\psi_k\| < \varepsilon$, где ε – заданная точность.

Описание программы

Программа позволяет вычислить решение СЛАУ с невырожденной матрицей A, используя итерационный метод верхней релаксации.

Описание функций.

1. Функция $\boldsymbol{double} * \boldsymbol{create\_matrix}(\boldsymbol{int}\ \boldsymbol{n})$ создает квадратную матрицу порядка n.

```
double **create_matrix(int n)
{
    double **matrix = calloc(n, sizeof(*matrix));
    for (int i = 0; i < n; i++) {
        matrix[i] = calloc(n, sizeof(double));
    }

    return matrix;
}
```

2. Функция $\boldsymbol{double} * \boldsymbol{create\_vector}(\boldsymbol{int}\ \boldsymbol{n})$ создает вектор длины n.

```
double *create_vector(int n)
{
    double *vector = calloc(n, sizeof(*vector));
    return vector;
}
```

3. Функция $\boldsymbol{double} ** \boldsymbol{copy\_matrix}(\boldsymbol{double} ** \boldsymbol{matrix}, \boldsymbol{int}\ \boldsymbol{n})$ создает копию квадратной матрицы $matrix$ порядка n.

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

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

    return matrix_copy;
}
```

4. Функция $\boldsymbol{double}\ **\ \boldsymbol{mul\_matr\_matr}(\boldsymbol{double}\ **\ \boldsymbol{matrix\_1}, \boldsymbol{double}\ **\ \boldsymbol{matrix\_2}, \boldsymbol{int}\ \boldsymbol{n})$ перемножает 2 квадратные матрицы порядка n.

```
double **mul_matr_matr(double **matrix_1, double **matrix_2, int n)
{
    double **mul = create_matrix(n);

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

    return mul;
}
```

5. Функция $\boldsymbol{double}\ *\ \boldsymbol{mul\_matr\_vect}(\boldsymbol{double}\ **\ \boldsymbol{matrix}, \boldsymbol{double}\ *\ \boldsymbol{vector}, \boldsymbol{int}\ \boldsymbol{n})$ находит произведение матрицы и вектора.

```
double *mul_matr_vect(double **matrix, double *vector, int n)
{
    double *mul = create_vector(n);

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

    return mul;
}
```

6. Функция $\boldsymbol{double}\ **\ \boldsymbol{transpose\_matrix}(\boldsymbol{double}\ **\ \boldsymbol{matrix}, \boldsymbol{int}\ \boldsymbol{n})$ находит матрицу транспонированную к данной.

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

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

    return trp;
}
```

7. Функция $\boldsymbol{double\ difference(double\ *vector\_1, double\ *vector\_2, int\ n)}$ находит норму невязки.

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

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

    return sqrt(res);
}
```

8. Функция ***void relaxation(double ** matrix, double * vector, int n, double w)*** находит решение СЛАУ методом верхней релаксации.

```
void relaxation(double **matrix, double *vector, int n, double w)
{
    int cnt = 0;
    double **trp_matrix = transpose_matrix(matrix, n);
    double **mul_matrix = mul_matr_matr(trp_matrix, matrix, n);
    double *vector_change = mul_matr_vect(trp_matrix, vector, n);
    double *vector_ans = create_vector(n);
    double *vector_prev = create_vector(n);

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

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

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

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

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

            for (int j = i; j < n; 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("Число итераций: %d\n\n", cnt);
    printf("Значение w: %g\n\n", w);

    for (int i = 0; i < n; i++) {
        printf("x%d = %lf\n", i + 1, vector_ans[i]);
    }

    free_matrix(trp_matrix, n);
    free_matrix(mul_matrix, n);
    free_vector(vector_change);
    free_vector(vector_prev);
    free_vector(vector_ans);
}
```

Тестирование и результаты эксперимента

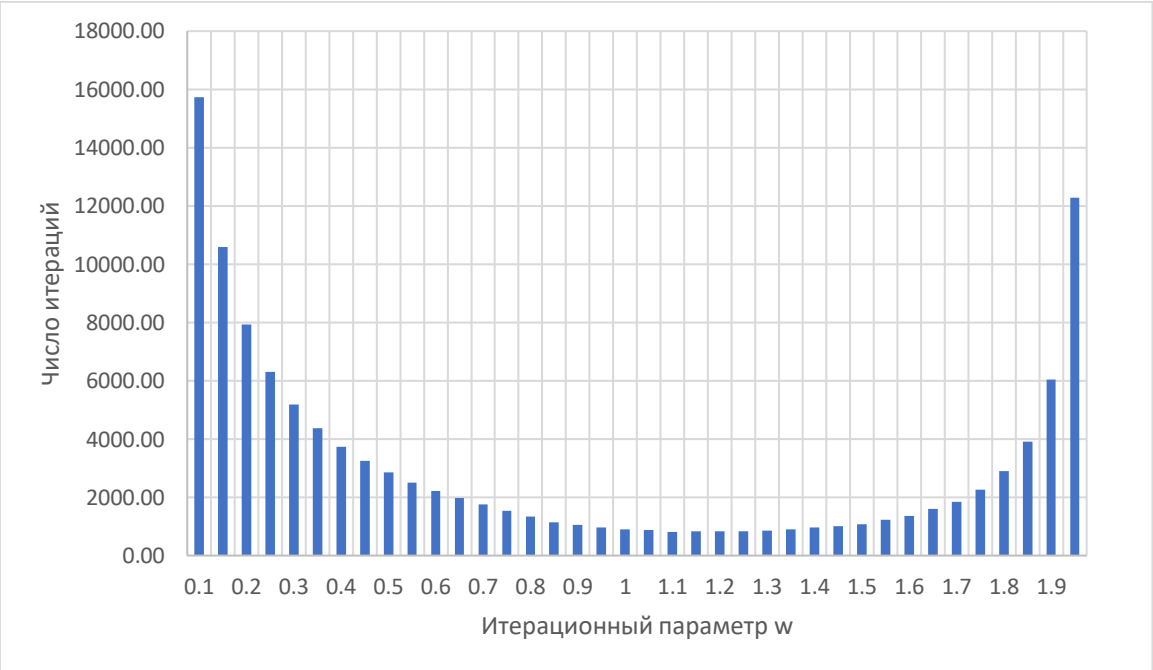
Тестирование программы проведено с помощью Wolfram Alpha и примеров, представленных в вариантах задания.

Приложение 1–11.

Тест 1.

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

- Определитель: 10.
- Решение системы
  x1 = 0.600000
  x2 = 1.000000
  x3 = -1.000000
  x4 = -0.200000



Наилучший показатель при $\omega = 1.1$. Число итераций: 818.

Тест 2.

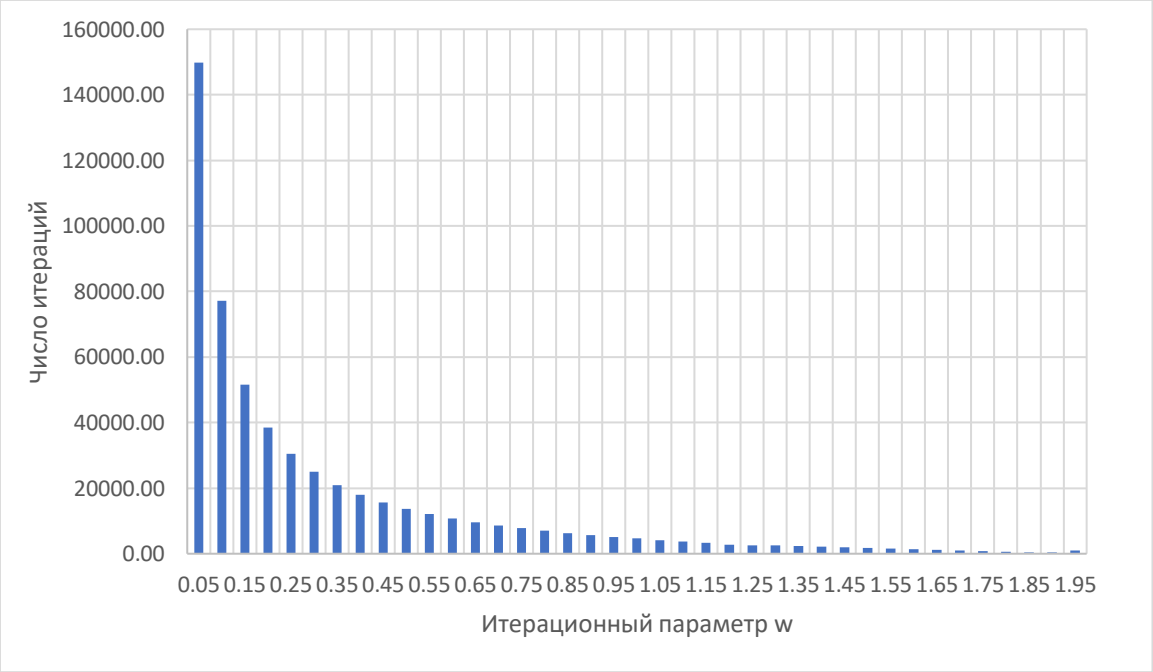
$$\begin{cases} x_1 + x_2 + 3x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 4x_3 - x_4 = 2, \\ 3x_1 + 3x_2 + 5x_3 - 2x_4 = 1, \\ 2x_1 + 2x_2 + 8x_3 - 3x_4 = 2. \end{cases}$$

- Определитель 0.

Тест 3.

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

- Определитель: -189.
- Решение системы:
  x1 = 3.000000
  x2 = 2.000000
  x3 = 1.000000
  x4 = 0.000000



Наилучший показатель при $\omega = 1.85$. Число итераций: 322.

Приложение 2, п2-4.

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

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

$M = 4; n = 100; b_i = n * exp\left(\frac{x}{i}\right) * cos(x).$

- Определитель: 5.47448e-26
- Наилучший показатель при $\omega = 1.9$. Число итераций: 14.

Вывод

В ходе работы изучены метод верхней релаксации и метод Зейделя, используемые для численного решения систем линейных алгебраических уравнений. Разработан критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заранее заданной точностью.

Также изучена скорость сходимости метода верхней релаксации и метода Зейделя в зависимости от выбора итерационного параметра. Проведены эксперименты с различными значениями итерационного параметра.