**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**имени М.В.Ломоносова**

**Факультет вычислительной математики и кибернетики**

# Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

# ЗАДАНИЕ № 1

## ОТЧЕТ

**о выполненном задании**

студента 204 учебной группы факультета ВМК МГУ

Гаухова Владислава Константиновича

гор. Москва

2020 год

# Оглавление

# Подвариант №1

## Постановка задачи

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

## Цели практической работы:

1) Реализовать функции для решения следующих задач:
    1. Решение СЛАУ стандартным методом Гаусса;
    2. Решение СЛАУ методом Гаусса с выбором главного элемента;
    3. Вычисление определителя матрицы;
    4. Вычисление обратной матрицы;
    5. Определить число обусловленности $M_A=\|A\| \times \|A^{-1}\|$
2) Проверить корректность решения СЛАУ методами Гаусса и верхней релаксации на различных тестах с помощью WolframAlpha;
3) Исследовать вопрос вычислительной устойчивости метода Гаусса при больших значениях параметра $n$;

# Описание алгоритмов

## Метод Гаусса

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа – прямой и обратный ход.

### Прямой ход:

В процессе прямого хода матрица системы с помощью элементарных преобразований над строками приводится к верхней треугольной форме. На i-м шаге в строке выбирается первый ненулевой элемент $a_{ii}$ – ведущий, – после чего i-я строка делится на $a_{ii}$. Затем из i+1, … n строк вычитается i-я строка, умноженная на $a_{i+1,i} \dots a_{n,i}$ соответственно. Сложность прямого хода: $Q_1 = \frac{n(n+1)}{2}$ делений и $Q_2 = \frac{n(n^2-1)}{3}$ сложений и умножений.

### Обратный ход:

В процессе обратного хода последовательно определяются все неизвестные, с $x_n$ до $x_1$. При вычислении значения i-ой переменной, необходимо произвести

$n - i$ умножений и вычитаний и одно деление. Сложность обратного хода: $Q_3 = n(n-1)$ сложений и умножений и $Q_4 = n$ делений. Таким образом, общая сложность метода Гаусса: $Q = Q_1 + Q_2 + Q_3 + Q_4 = \frac{n^3}{3} + O(n^2)$.

## Метод Гаусса с выбором главного элемента

В результате округления чисел в методе Гаусса могут возникать ошибки, которые особенно влияют на результат при работе с плохо обусловленными матрицами. Эта проблема решается в методе Гаусса с выбором главного элемента. Его отличие заключается в том, что на каждом шаге прямого хода в качестве ведущего выбирается максимальный из элементов строки. В этом случае в процессе обратного хода роль ошибок округления снижается.

## Определитель матрицы

Результат прямого хода метода Гаусса также можно использовать для нахождения определителя матрицы, который будет вычисляться как $(-1)^n \prod_{i=1}^{n} a_{ii}$, где $n$ – число перестановок столбцов в ходе выделения ведущего элемента.

## Обратная матрица

Для вычисления обратной матрицы используется метод Гаусса-Жордана, в котором рассматривается расширенная матрица $A|E$, $E$ – единичная матрица, равная $A$ по размерам. Левая часть матрицы приводится к единичной; все производимые над ней преобразования выполняются и над правой частью, в результате чего она приводится к виду $A^{-1}$.

## Число обусловленности

Будем считать норму матрицы следующим образом:

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

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

# Тестирование

Тестирование реализованных методов решения систем линейных алгебраических уравнений проводилось на предложенных наборах из варианта 3 приложения 1 и примера 1 варианта 2 приложения 2 путём сравнения полученных решений с корректными решениями, представленных WolframAlpha и библиотекой numpy языка python.

**СЛАУ №1**:

$$\begin{cases} 2x_1 + 5x_2 + 4x_3 + x_4 = 20 \\ x_1 + 3x_2 + 2x_3 + x_4 = 11 \\ 2x_1 + 10x_2 + 9x_3 + 7x_4 = 40 \\ 3x_1 + 8x_2 + 9x_3 + 2x_4 = 37 \end{cases}$$

**Определитель** матрицы коэффициентов:

$$det \begin{bmatrix} 2 & 5 & 4 & 1 \\ 1 & 3 & 2 & 1 \\ 2 & 10 & 9 & 7 \\ 3 & 8 & 9 & 2 \end{bmatrix} = 18 \neq 0$$

**Обратная матрица**:

$$A^{-1} = \begin{bmatrix} 15 & -21 & 2 & -4 \\ -6.667 & 10.333 & -1 & 1.667 \\ -0.333 & -0.333 & 0 & 0.333 \\ 5.667 & -8.333 & 1 & -1.667 \end{bmatrix}$$

**Число обусловленности**: $M = 1176 \gg 1$, следовательно, матрица коэффициентов СЛАУ плохо обусловлена.

**Ожидаемое решение** (Wolfram Alpha)**:** $(1;\ 2; 2; 0)$

**Решение** (метод Гаусса): $(1;\ 2; 2; 0)$

**Решение** (метод Гаусса с выбором главного элемента): $(1;\ 2; 2; 0)$

**СЛАУ №2**:

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

**Определитель** матрицы коэффициентов:

$$det\begin{bmatrix} 6 & 4 & 5 & 2 \\ 3 & 2 & 4 & 1 \\ 3 & 2 & -2 & 1 \\ 9 & 6 & 1 & 3 \end{bmatrix} = 0$$

Определитель равен нулю => СЛАУ вырождена

**СЛАУ №3**:

$$\begin{cases} 2x_1 + x_2 + x_3 = 2 \\ x_1 + 3x_2 + x_3 + x_4 = 5 \\ x_1 + x_2 + 5x_3 = -7 \\ 2x_1 + 3x_2 - 3x_3 - 10x_4 = 14 \end{cases}$$

**Определитель** матрицы коэффициентов:

$$det\begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 5 & 0 \\ 2 & 3 & -3 & -10 \end{bmatrix} = -242 \neq 0$$

**Обратная матрица**:

$$A^{-1} = \begin{bmatrix} 0.652893 & -0.165289 & -0.107438 & -0.016529 \\ -0.219008 & 0.371901 & -0.008264 & 0.037190 \\ -0.086777 & -0.041322 & 0.223140 & -0.004132 \\ 0.090909 & 0.090909 & -0.090909 & -0.090909 \end{bmatrix}$$

**Число обусловленности**: $M = 16.958678$

**Ожидаемое решение** (Wolfram Alpha)**:** (1; 2; -2; 0)

**Решение** (метод Гаусса): $(1;\ 2; -2; 0)$

**Решение** (метод Гаусса с выбором главного элемента): $(1;\ 2; -2; 0)$

**СЛАУ №4**:

$$A_{ij} = \begin{cases} \dfrac{i+j}{m+n}, & i \neq j, \\ n + m^2 + \dfrac{j}{m} + \dfrac{i}{n}, & i = j, \end{cases}$$

При n = 20, m = 8

Элементы вектора f заданы формулой: $\mathrm{f[i]} = 200 + 50 * \mathrm{i}$

**Определитель** матрицы коэффициентов:

$$det[A] = 45032865927535134 * 10^{22} \neq 0$$

**Число обусловленности**: $M = 1.239563$

**Ожидаемое решение** (numpy)**:**
(0.000000; 2.970003; 3.556609 … 11.145108; 11.611302; 12.080616)

**Решение** (метод Гаусса):
(0.000000; 2.970003; 3.556609 … 11.145108; 11.611302; 12.080616)

**Решение** (метод Гаусса с выбором главного элемента):
(0.000000; 2.970003; 3.556609 … 11.145108; 11.611302; 12.080616)

Для данной СЛАУ метод Гаусса оказался устойчив, в силу диагонального преобладания матрицы A при заданных значениях m и n.

# Выводы

При выполнении практической работы был изучен и воплощен метод Гаусса и модифицированный метод Гаусса с выбором главного элемента. К преимуществам метода Гаусса можно отнести относительную простоту его реализации и широкую область применения в задачах, связанных с исследованием матриц, включая нахождение определителя и обратной матрицы. Недостатком методов является их неустойчивость при работе с плохо обусловленными матрицами, что могло отразиться на полученном решении. Модифицированный метод Гаусса демонстрирует большую точность для таких СЛАУ.

# Код программы

## Метод Гаусса

```
1.  void
2.  Gauss(int n, double arr[][n], double f[n], double *res)
3.  {
4.      //Прямой ход
5.      for (int i = 0; i < n; ++i) {
6.          int j = i;
7.          while(arr[j][i] == 0) { //Поиск ненулевого элемента
8.              ++j;
9.          }
10.         if (j != i) {
11.             for (int k = i; k < n; ++k) {
12.                 double t = arr[i][k];
13.                 arr[i][k] = arr[j][k];
14.                 arr[j][k] = t;
15.             }
16.             double t = f[i];
17.             f[i] = f[j];
18.             f[j] = t;
19.         }
20.         for (int k = i + 1; k < n; ++k) {
21.             double k1 = arr[i][i];
22.             double k2 = arr[k][i];
23.             for (int z = 0; z < n; ++z) {
24.                 arr[k][z] -= arr[i][z] * k2 / k1;
25.             }
26.             f[k] -= f[i] * k2 / k1;
27.         }
28.     }
29.     //Обратный ход
30.     for (int i = n - 1; i > 0; --i) {
31.         for (int j = i - 1; j >= 0; --j) {
32.             double k1 = arr[j][i] / arr[i][i];
33.             for (int k = n - 1; k >= 0; --k) {
34.                 arr[j][k] -= k1 * arr[i][k];
35.             }
36.             f[j] -= k1 * f[i];
37.         }
38.     }
```

```
39.    for (int j = 0; j < n; ++j) {
40.        res[j] = f[j] / arr[j][j];
41.    }
42.}
```

**Метод Гаусса с выбором главного элемента**

```
1.  void
2.  Gauss_modif(int n, double arr[][n], double f[n], double *res)
3.  {
4.      //Прямой ход
5.      for (int i = 0; i < n; ++i) {
6.          double max_value = arr[i][i];
7.                  int max_index = i;
8.                  //Поиск максимального значения
9.                  for (int k = i + 1; k < n; ++k) {
10.                     if (fabs(max_value) < fabs(arr[k][i])) {
11.                         max_value = arr[k][i];
12.                         max_index = k;
13.                     }
14.                 }
15.         if (i != max_index) {
16.             for (int j = 0; j < n; ++j) {
17.                 double t = arr[i][j];
18.                 arr[i][j] = arr[max_index][j];
19.                 arr[max_index][j] = t;
20.             }
21.             double t = f[i];
22.             f[i] = f[max_index];
23.             f[max_index] = t;
24.          }
25.         for (int k = i + 1; k < n; ++k) {
26.             double k1 = arr[i][i];
27.             double k2 = arr[k][i];
28.             for (int z = 0; z < n; ++z) {
29.                 arr[k][z] -= arr[i][z] * k2 / k1;
30.             }
31.             f[k] -= f[i] * k2 / k1;
32.         }
33.     }
34.     //Обратный ход
35.     for (int i = n - 1; i > 0; --i) {
```

```
        for (int j = i - 1; j >= 0; --j) {
            double k1 = arr[j][i] / arr[i][i];
            f[j] -= k1 * f[i];
            for (int k = n - 1; k >= 0; --k) {
                arr[j][k] -= k1 * arr[i][k];
            }
        }
    }
    for (int j = 0; j < n; ++j) {
        res[j] = f[j] / arr[j][j];
    }
}
```

**Приведение к треугольному виду(возвращает множитель(1 или -1) для вычисления определителя)**

```
int
triangle_form(int n, double arr[][n]) {
    int sign = 1;
    for (int i = 0; i < n; i++) {
        int j;
        for (j = i; j < n; j++) {
            if (arr[j][i]) {
                break;
            }
        }
        if (j == n || fabs(arr[j][i]) < EPS) {
            fprintf(stderr, "det[arr] = 0\n");
            return 0;
        }
        if (i != j) {
            sign *= -1;
            for (int k = i; k < n; k++) {
                double t = arr[j][k];
                arr[j][k] = arr[i][k];
                arr[i][k] = t;
            }
        }
        for (int k = i + 1; k < n; k++) {
            double k1 = arr[i][i];
            double k2 = arr[k][i];
            if (fabs(k1) < EPS) {
```

```
            return 0;
        }
        if (fabs(k2) < EPS) {
            continue;
        }
        for (int l = 0; l < n; l++) {
            arr[k][l] -= arr[i][l] * k2 / k1;
        }
    }
}
return sign;
}
```

**Вычисление определителя**

```
double matrix_det(int n, double arr[][n])
{
    int sign = triangle_form(n, arr);
    double ans = 1;
    for (int i = 0; i < n; i++) {
        ans *= arr[i][i];
    }
    return ans * sign;
}
```

**Вычисление обратной матрицы**

```
void inverse(int n, double arr[][n], double res[][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            res[i][j] = i == j ? 1 : 0;
        }
    }
    int j = 0;
    int sign = 1;
    for (int i = 0; i < n; i++) {
        for (j = i; j < n; j++) {
            if (arr[j][i]) {
                break;
            }
        }
        if (j == n || fabs(arr[j][i]) < EPS) {
            fprintf(stderr, "det[arr] = 0\n");
```

```
17.             _exit(1);
18.         }
19.         if (i != j) {
20.             sign *= -1;
21.             for (int k = i; k < n; k++) {
22.                 double t = arr[j][k];
23.                 arr[j][k] = arr[i][k];
24.                 arr[i][k] = t;
25.                 t = res[j][k];
26.                 res[j][k] = res[i][k];
27.                 res[i][k] = t;
28.             }
29.         }
30.         for (int k = i + 1; k < n; k++) {
31.             double k1 = arr[i][i];
32.             double k2 = arr[k][i];
33.             if (fabs(k2) < EPS)
34.                 continue;
35.             for (int l = 0; l < n; l++) {
36.                 arr[k][l] -= arr[i][l] * k2 / k1;
37.                 res[k][l] -= res[i][l] * k2 / k1;
38.             }
39.         }
40.     }
41.     for (int i = n - 1; i > 0; i--) {
42.         for (int j = i - 1; j >= 0; j--) {
43.             if (fabs(arr[i][i]) < EPS) {
44.                 fprintf(stderr, "det[arr] = 0\n");
45.                 return 0;
46.             }
47.             double k1 = arr[j][i] / arr[i][i];
48.             for (int k = n - 1; k >= 0; k--) {
49.                 arr[j][k] -= k1 * arr[i][k];
50.                 res[j][k] -= k1 * res[i][k];
51.             }
52.         }
53.     }
54.     for (int i = 0; i < n; i++) {
55.         for (int j = 0; j < n; j++) {
56.             if (fabs(arr[i][i]) < EPS) {
57.                 fprintf(stderr, "det[arr] = 0\n");
```

```
58.                 return 0;
59.             }
60.             res[i][j] /= arr[i][i];
61.         }
62.     }
63. }
```

**Вычисление числа обусловленности**

```
1.  double condition_num(int n, double arr[][n])
2.  {
3.      double norm_matr = 0;
4.      for (int i = 0; i < n; i++) {
5.          double cur_matr = 0;
6.          for (int j = 0; j < n; j++) {
7.              cur_matr += fabs(arr[i][j]);
8.          }
9.          if (cur_matr > norm_matr)
10.             norm_matr = cur_matr;
11.     }
12.     double norm_inv = 0;
13.     double inv[n][n];
14.     inverse(n, arr, inv);
15.     for (int i = 0; i < n; i++) {
16.         double cur_inv = 0;
17.         for (int j = 0; j < n; j++) {
18.             cur_inv += fabs(inv[i][j]);
19.         }
20.         if (cur_inv > norm_inv)
21.             norm_inv = cur_inv;
22.     }
23.     return norm_matr * norm_inv;
24. }
```

**Генерация матрицы для примера 1-2**

```
1.  void matrix_generate_1 (int n, int m, double arr[][n])
2.  {
3.      for (int i = 0; i < n; ++i) {
4.          for (int j = 0; j < n; ++j) {
5.              if (i == j) {
6.                  arr[i][i] = n + m*m + i*(1.0/m + 1.0/n);
7.              } else {
```

```
8.                 arr[i][j] = (i + j)/(n + m);
9.             }
10.        }
11.    }
12.}
```

**Генерация вектора правой части для примера 1-2**

```
1.  void f_genetate_1 (int n, double f[n])
2.  {
3.      for (int i = 1; i <= n; ++i) {
4.          f[i] = 200 + 50 * i;
5.      }
6.  }
```

# Подвариант №2

## 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; 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 $D, A^{(-)}$ - соответственно диагональная и нижняя треугольные матрицы, $k$ - номер текущей итерации, $\omega$ - итерационный параметр (при $\omega=1$ метод верхней релаксации переходит в метод Зейделя).

## Цели практической работы

1) Изучить метод верхней релаксации
2) Реализовать метод верхней релаксации
3) Разработать критерий остановки процесса для заданной точности
4) Изучить скорость сходимости итераций к точному решению задачи

## Описание алгоритмов

### Метод верхней релаксации

Метод верхней релаксации – стационарный итерационный метод решения СЛАУ. В ходе этого метода каждый следующий вектор приближения точного решения вычисляется по формуле:

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

$x^i$– вектор $i$-го приближения решения СЛАУ;

$D$ – матрица, содержащая только диагональные элементы матрицы $A$;

$T_H$ – матрица, содержащая только элементы матрицы $A$ ниже диагонали;

$w$ – итерационный параметр.

В алгоритме программной реализации используется следующая формула для пересчёта вектора приближённого решения:

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

Критерием остановки алгоритма будет выражаться следующий образом:

$$\left\|x^k - x\right\|_2 < \left||A^{-1}|\right| * \left|\left|\varphi^k\right|\right| = \varepsilon$$

Где $\varepsilon > 0$ – некая фиксированная точность, $\varphi^k$ – невязка уравнения.

При достижении предварительно заданного максимального числа итераций программа также заканчивает вычисления.

# Тестирование

Тестирование реализованных методов решения систем линейных алгебраических уравнений проводилось на предложенных наборах из варианта 3 приложения 1 и примера 2 варианта 2 приложения 2 путём сравнения полученных решений с корректными решениями, представленных WolframAlpha и функциями из библиотекой numpy языка python.

**СЛАУ №1**:

$$\begin{cases} 2x_1 + 5x_2 + 4x_3 + x_4 = 20 \\ x_1 + 3x_2 + 2x_3 + x_4 = 11 \\ 2x_1 + 10x_2 + 9x_3 + 7x_4 = 40 \\ 3x_1 + 8x_2 + 9x_3 + 2x_4 = 37 \end{cases}$$

**Определитель** матрицы коэффициентов:

$$det\begin{bmatrix} 2 & 5 & 4 & 1 \\ 1 & 3 & 2 & 1 \\ 2 & 10 & 9 & 7 \\ 3 & 8 & 9 & 2 \end{bmatrix} = 18 \neq 0$$

**Обратная матрица**:

$$A^{-1} = \begin{bmatrix} 15 & -21 & 2 & -4 \\ -6.667 & 10.333 & -1 & 1.667 \\ -0.333 & -0.333 & 0 & 0.333 \\ 5.667 & -8.333 & 1 & -1.667 \end{bmatrix}$$

**Число обусловленности**: $M = 1176 \gg 1$, следовательно, матрица коэффициентов СЛАУ плохо обусловлена.

**Ожидаемое решение**(Wolfram Alpha) **:** $(1;\ 2; 2; 0)$

**Решение** (метод верхней релаксации): выполнено 1000 итераций, решение не найдено.

**СЛАУ №2**:

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

**Определитель** матрицы коэффициентов:

$$det \begin{bmatrix} 6 & 4 & 5 & 2 \\ 3 & 2 & 4 & 1 \\ 3 & 2 & -2 & 1 \\ 9 & 6 & 1 & 3 \end{bmatrix} = 0$$

Определитель равен нулю => СЛАУ вырождена

**СЛАУ №3**:

$$\begin{cases} 2x_1 + x_2 + x_3 = 2 \\ x_1 + 3x_2 + x_3 + x_4 = 5 \\ x_1 + x_2 + 5x_3 = -7 \\ 2x_1 + 3x_2 - 3x_3 - 10x_4 = 14 \end{cases}$$

**Определитель** матрицы коэффициентов:

$$det \begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 5 & 0 \\ 2 & 3 & -3 & -10 \end{bmatrix} = -242 \neq 0$$

**Обратная матрица**:

$$A^{-1} = \begin{bmatrix} 0.652893 & -0.165289 & -0.107438 & -0.016529 \\ -0.219008 & 0.371901 & -0.008264 & 0.037190 \\ -0.086777 & -0.041322 & 0.223140 & -0.004132 \\ 0.090909 & 0.090909 & -0.090909 & -0.090909 \end{bmatrix}$$

**Число обусловленности**: $M = 16.958678$

**Ожидаемое решение** (Wolfram Alpha) **:** (1; 2; -2; 0)

**Решение** (метод верхней релаксации):

45 итераций при w = 0.4 (0.999998 2.000001 -2.000000 -0.000000)

26 итераций при w = 0.6 (0.999999 2.000001 -2.000000 -0.000000)

16 итераций при w = 0.8 (1.000000 2.000000 -2.000000 0.000000)

11 итераций при w = 1.0 (1.000000 2.000000 -2.000000 0.000000)

16 итераций при w = 1.2 (1.000000 2.000000 -2.000000 0.000000)

43 итераций при w = 1.4 (1.000000 2.000000 -2.000000 0.000000)

Решение не найдено за 1000 итераций при w = 1.6

**СЛАУ №4**:

$$A_{ij} = \begin{cases} \dfrac{i+j}{m+n}, & i \neq j, \\ n + m^2 + \dfrac{j}{m} + \dfrac{i}{n}, & i = j, \end{cases}$$

При n = 20, m = 8

Элементы вектора f заданы формулой: f[i] $= 200 + 50 * i$

**Определитель** матрицы коэффициентов:

$$det[A] = 45032865927535134 * 10^{22} \neq 0$$

**Число обусловленности**: $M = 1.239563$

**Ожидаемое решение** (Wolfram Alpha)**:**
$(0.000000;\ 2.970003;\ 3.556609 \ldots\ 11.145108;\ 11.611302;\ 12.080616)$

**Решение** (метод верхней релаксации): (0.000000; 2.970003; 3.556609 … 11.145108; 11.611302; 12.080616)

31 итераций при w = 0.4

18 итераций при w = 0.6

11 итераций при w = 0.8

5 итераций при w = 1.0

12 итераций при w = 1.2

20 итераций при w = 1.4

36 итераций при w = 1.6

**СЛАУ №5**

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

При n = 40, M = 2, $f[i] = \left|x - \frac{n}{10}\right| \cdot i \cdot \sin(x)$

При x = 5:

**Определитель** матрицы коэффициентов:

$$det[A] = 2.254305 \neq 0$$

**Число обусловленности**: $M = 3162.480388$ >> 1

Ожидаемое решение (Wolfram Alpha)

**Решение** (метод верхней релаксации): решение не найдено за 1000 итераций.

Можно заметить, что матрицы СЛАУ №1 и №5 плохо обусловлены, из-за чего решение не было найдено данным методом. Матрицы СЛАУ №3 и №4 имеют хорошую обусловленность матрицы, к тому же матрица №4 симметрична. Результаты сходимости метода при разных w показывают, что для данных СЛАУ оптимальным параметром является $w \approx 1$.

# Выводы

При выполнении практической работы был изучен и воплощен итерационный метод верхней релаксации. При исследовании скорости сходимости данного метода было замечено, что скорость сходимости метода существенно зависит от параметра w. Оптимальный параметр зависит от самих матриц СЛАУ.

# Код программы

### Метод верхней релаксации

```
double *relaxation(int n, double arr[][n], double *f, double w,
        int max_iter, double eps)
{
    double *x = calloc(n, sizeof(double));
    double *tmp = calloc(n, sizeof(double));
    for (int i = 0; i < n; i++) {
        x[i] = 0;
        tmp[i] = 0;
    }
    for (int k = 0; k < max_iter; k++) {
        for (int i = 0; i < n; i++) {
            double sub_1 = 0, sub_2 = 0;
            for (int j = 0; j < i; j++) {
                sub_1 += arr[i][j] * tmp[j];
            }
            for (int j = i; j < n; j++) {
                sub_2 += arr[i][j] * x[j];
            }
            tmp[i] = x[i] + w / arr[i][i] * (f[i] - sub_1 - sub_2);
        }
        double dist = 0;
        for (int i = 0; i < n; i++) {
            dist += (x[i] - tmp[i]) * (x[i] - tmp[i]);
        }
        double inv[n][n];
        double arr_copy[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
```

```
            arr_copy[i][j] = arr[i][j];
        }
    }
    inverse(n, arr_copy, inv);
    double inv_norm = condition_num(n, arr_copy);
    double norm_nev = 0;
    for (int i = 0; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < n; j++) {
            sum += arr[i][j] * tmp[j];
        }
        sum -= f[i];
        norm_nev += sum * sum;
    }
    norm_nev = sqrt(norm_nev);
    if (norm_nev * inv_norm < eps) {
        printf("k = %d\n", k);
        return tmp;
    }
    for (int i = 0; i < n; i++) {
        x[i] = tmp[i];
    }
  }
  return x;
}
```

**Генерация матрицы для примера 1-2**

```
void matrix_generate_1 (int n, int m, double arr[][n])
{
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (i == j) {
        arr[i][i] = n + m*m + i*(1.0/m + 1.0/n);
      } else {
        arr[i][j] = (i + j)/(n + m);
      }
    }
  }
}
```

**Генерация вектора правой части для примера 1-2**

```
7.  void f_genetate_1 (int n, double f[n])
8.  {
9.      for (int i = 1; i <= n; ++i) {
10.         f[i] = 200 + 50 * i;
11.     }
12. }
```

**Генерация матрицы для примера 2-2**

```
1.  void matrix_generate_2(int n, int m, double arr[][n])
2.  {
3.      double q = 1.001 - 2 * m * 0.001;
4.      for (int i = 0; i < n; ++i) {
5.          for (int j = 0; j < n; ++j) {
6.              if (i == j) {
7.                  arr[i][j] = pow(q - 1, i + j);
8.              } else {
9.                  arr[i][j] = pow(q, i + j) + 0.1 * (j - i);
10.             }
11.         }
12.     }
13. }
```

**Генерация вектора правой части для примера 2-2**

```
1.  void f_genetate_2(int n, double x, double f[n])
2.  {
3.      for (int i = 0; i < n; ++i) {
4.          f[i] = fabs(x - n/10.0) * i * sin(x);
5.      }
6.  }
```