**Московский Государственный Университет**
**имени М. В. Ломоносова**

# Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ» ЗАДАНИЕ № 1

## ОТЧЁТ
**о выполненном задании**

студента 206 учебной группы факультета ВМК МГУ
Маркова Игоря Валерьевича

г. Москва, 2019 год

**ПРАКТИЧЕСКАЯ РАБОТА № 1**

**Подвариант № 1**

**Решение Систем Линейных алгебраических уравнений методом Гаусса и методом Гаусса с выбором главного элемента**

# Постановка задачи

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

# Цели и задачи практической работы

1. Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного
2. Вычислить определитель матрицы $det(A)$
3. Вычислить обратную матрицу $A^{-1}$
4. Определить число обусловленности $M_A = ||A|| * ||A^{-1}||$
5. Исследовать вопрос вычислительной устойчивости метода Гаусса
6. Правильность решения подтвердить системой тестов

# Описание метода решения

**Алгоритм Гаусса**

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа:

- На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. А именно, среди элементов первого столбца матрицы выбирают ненулевой, перемещают его на крайнее верхнее положение перестановкой строк и вычитают получившуюся после перестановки первую строку из остальных строк, домножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.

- На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

**Поиск главного элемента**

Приступая к первому шагу прямого хода метода Гаусса, рассмотрим элементы $a_{1,j}$ первой строки матрицы *A* и найдем среди них элемент наибольший по модулю. Пусть он имеет номер $j_1$ .Поменяем в системе первый столбец и столбец с номер $j_1$ местами, изменив соответствующим образом нумерацию неизвестных. В результате такой процедуры наибольший по модулю элемент первой строки станет ведущим элементом первого шага $a_{1,1}$ . Благодаря этому элементы $c_{1,j} = \frac{a_{1,j}}{a_{1,1}}$ удовлетворяют неравенству $|c_{1,j}| \leq 1$ , что уменьшает погрешность метода Гаусса. Процедуру выделения наибольшего по модулю элемента в очередной строке и превращения его в ведущий элемент нужно затем повторять во время каждого шага прямого хода метода Гаусса

**Метод Гаусса-Жордана**

Данный метод используется для нахождения обратной матрицы. Метод Гаусса-Жордана полностью повторяет метод Гаусса приведения матрицы *A* к единичной, за исключением того, что преобразования строк, которые применяются к значениям вектора *f*, применяются к единичной матрице, которое при приведении матрицы *A* к единичной и становится обратной матрицей - $A^{-1}$

# Описание программы

В аргументах командной строки при запуске программы первым задается число от 1 до 4, где 1 - найти решения СЛАУ методом Гаусса, 2 - найти определитель матрицы, 3 - найти обратную матрицу, 4 - найти число обусловленности матрицы. Второй аргумент командной строки опциональный и по умолчанию равен 0, где 0 - без использования выбора главного элемента, 1 - с использованием. Отметим, что имеет смысл задавать второй параметр командной строки только, если первый равен 1 или 2. В зависимости от выбранной задачи (1-4) функция $straight\_run$ применяет преобразование из метода Гаусса к вектору f - в случае решения СЛАУ, к единичной матрице - в случае нахождения обратной матрицы и степени обусловленности, ни к чему в случае нахождения определителя.

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

typedef struct linear_system{
   double ** matrix;
   double * f;
   int n;
   int * index_arr;
   int leading_el;
   double ** I;
}linear_system;              //структура, описывающая все компаненты данных задач


void input(linear_system * X, FILE * fp, int task) {  //функция ввода данных из
   fscanf(fp, "%d", &(X->n));                    //файла или из командной строки
   int N = X->n;
   X->matrix = malloc(N * sizeof(double *));
   if (task == 3 || task == 4) {
       X->I = malloc(N * sizeof(double *));
   }
   for (int i = 0; i < N; i++) {
       X->matrix[i] = malloc(N * sizeof(double));
       if (task == 3 || task == 4) {
           X->I[i] = malloc(N * sizeof(double *));
       }
       for (int j = 0; j < N; j++) {
           fscanf(fp, "%lf", &X->matrix[i][j]);
           if (task == 3 || task == 4) {
               X->I[i][j] = (i == j);
           }
       }
   }
   if (task == 1 || task == 5) {
       X->f = malloc(N * sizeof(double));
   }
   X->index_arr = malloc(N * sizeof(int));
   for (int i = 0; i < N; i++) {
       if (task == 1 || task == 5) {
           fscanf(fp, "%lf", &(X->f[i]));
       }
```

```
        X->index_arr[i] = i;
    }
    return;
}

void input_prel(linear_system * X, int task) {  //генерация матрицы из приложения 2
    int n = 20, m = 8;
    double n_d = 20.0, m_d = 8.0;
    X->n = n;
    X->matrix = malloc(n * sizeof(double *));
    if (task == 1 || task == 5) {
        X->f = malloc(n * sizeof(double));
    }
    if (task == 3 || task == 4) {
        X->I = malloc(n * sizeof(double *));
    }
    for (int i = 0;i < n; i++) {
        X->matrix[i] = malloc(n * sizeof(double));
        if (task >= 3) {
            X->I[i] = malloc(n * sizeof(double));
        }
    }
    X->index_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        for (int j = 0;j < n; j++) {
            if (i == j) {
                X->matrix[i][j] = n_d + m_d*m_d + j/m_d + i/n_d;
            } else {
                X->matrix[i][j] = (i + j)/(n_d + m_d);
            }
            if (task >= 3) {
                X->I[i][j] = (i == j);
            }
        }
        if (task == 1) X->f[i] = 200.0 + 50.0 * i;
        X->index_arr[i] = i;
    }
}

FILE * input_selection(linear_system * X) {      //ввод способа получения матрицы
    char choice1;                   //из файла (+путь до файла), из командной строки
    printf("Сгенерировать матрицу из приложения 2 [y/n]: ");
```

```
    scanf("%c", &choice1); //или генерировать из приложения 2
    if(choice1 == 'y') {
        return NULL;
    }
    scanf("%c", &choice1);
    char choice2;
    printf("Считывать матрицу из файла? [y/n]: ");
    scanf("%c", &choice2);

    if (choice2 == 'y') {
        char file_path[PATH_MAX];
        printf("Путь до файла: ");
        scanf("%s", file_path);
        FILE * fp = fopen(file_path, "r");
        return fp;
    }
    return stdin;
}


int get_none_zero_row(linear_system * X, int iter, int task) {  //функция выбора
    int ind_none_zero = 0;            //строки с ненулевым ведущим элементом
                //и последующая перестоновка этой строки с ведущей
    for(int i = iter; i < X->n; i++) {
        if (X->matrix[i][iter] != 0) {
            ind_none_zero = i;
            break;
        }
    }

    double * tmp = X->matrix[iter];
    X->matrix[iter] = X->matrix[ind_none_zero];
    X->matrix[ind_none_zero] = tmp;

    if (task == 3) {
        tmp = X->I[iter];
        X->I[iter] = X->I[ind_none_zero];
        X->I[ind_none_zero] = tmp;
    }

    if (task == 1) {
        double tmp2 = X->f[iter];
```

```
        X->f[iter] = X->f[ind_none_zero];
        X->f[ind_none_zero] = tmp2;
    }

    return !(iter == ind_none_zero);
}



void show_matrix(linear_system * X, int task) {  //вывод матрицы в командную строку
    printf("-----------------\n");
    for (int i = 0; i < X->n; i++) {
        for (int j = 0;j < X->n; j++) {
            printf("%0.2lf ", X->matrix[i][j]);
        }
        if (task == 1 || task == 5) {
            printf("| %0.2lf", X->f[i]);
        } else if (task == 3) {
            printf("| ");
            for (int j = 0;j < X->n; j++) {
                printf("%0.2lf ", X->I[i][j]);
            }
        }
        printf("\n");
    }
    return;
}



int chose_leading_element(linear_system * X, int iter) {//функция для поиска
                                                        //ведущего элемента строки
    double max = fabs(X->matrix[iter][iter]);                  //и последующего
                                                       //переставления его столбца с ведущим
    int max_ind = iter;
    for (int i = iter + 1;i < X->n; i++) {
        if (fabs(X->matrix[iter][i]) > max) {
            max = fabs(X->matrix[iter][i]);
            max_ind = i;
        }
    }
    for (int i = 0; i < X->n; i++) {
        double tmp = X->matrix[i][iter];
        X->matrix[i][iter] = X->matrix[i][max_ind];
```

```
        X->matrix[i][max_ind] = tmp;
    }
    int tmp = X->index_arr[iter];
    X->index_arr[iter] = X->index_arr[max_ind];
    X->index_arr[max_ind] = tmp;
    return !(max_ind == iter);
}



double straight_run(linear_system * X, int task) {  //прямой ход в алгоритме Гаусса
    double det = 1.0;   //приведение к верхней треугольной матрицы с 1 на диагонали
    for (int i = 0; i < X->n; i++) {
        if (task == 3 || !X->leading_el) {
            if (get_none_zero_row(X, i, task)) {
                det = -det;
            }
        }
        if (task != 3 && X->leading_el && chose_leading_element(X, i)) {
            det = -det;
        }
        double leading_a = X->matrix[i][i];
        for(int j = i; j < X->n; j++) {
            X->matrix[i][j] /= leading_a;
        }
        if (task == 1) {
            X->f[i] /= leading_a;
        }
        if (task == 2) {
            det *= leading_a;
        }
        if (task == 3) {
            for (int j = 0; j < X->n; j++) {
                X->I[i][j] /= leading_a;
            }
        }
        for(int k = i + 1; k < X->n; k++) {
            double a = X->matrix[k][i];
            X->matrix[k][i] = 0;
            for(int j = i + 1; j < X->n; j++) {
                X->matrix[k][j] -= a * X->matrix[i][j];
            }
            if (task == 1) {
```

```
                X->f[k] -= a * X->f[i];
            }
            if (task == 3) {
                for (int j = 0; j < X->n; j++) {
                    X->I[k][j] -= a * X->I[i][j];
                }
            }
        }
    }
    return det;
}


double * back_run(linear_system * X, int task) { //обратный ход для решения СЛАУ и
                                                 //нахождения определителя
    double * ans = malloc(X->n * sizeof(double));           //методом Гаусса
    for (int i = X->n - 1; i >= 0; i--) {
        ans[i] = X->f[i];
        for (int j = i + 1; j < X->n; j++) {
            ans[i] -= X->matrix[i][j] * ans[j];
        }
    }
    return ans;
}


void back_run_reverse(linear_system * X, int task) {       //обратный ход для
                                                  //нахождения обратной матрицы
    for (int i = X->n - 1; i >= 1; i--) {
        for (int j = 0; j < i; j++) {
            double a = X->matrix[j][i];
            for (int k = 0; k < X->n; k++) {
                X->I[j][k] -= a * X->I[i][k];
            }
            X->matrix[j][i] = 0;
        }
    }
    return;
}

double MatrixNorm(double ** matrix, int n) {//нахождение бесконечной нормы матрицы
    double max = 0;
```

```
    for (int i = 0; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < n; j++) {
            sum += fabs(matrix[i][j]);
        }
        if (sum > max) {
            max = sum;
        }
    }
    return max;
}


double nevyazka(linear_system * X, double * ans) {//функция подсчета невязки (
                                                   //||A(x_k) - f|| )
    double res = 0;
    for (int i = 0; i < X->n; i++) {
        double sum = 0;
        for (int j = 0;j < X->n; j++) {
            sum += X->matrix[i][j] * ans[j];
        }
        res += (sum - X->f[i]) * (sum - X->f[i]);
    }
    return sqrt(res);
}


double * Relax(linear_system * X, double eps, double om) {//реализация алгоритма
                                                          //релаксации
    double * ans = calloc(X->n, sizeof(double));
    int iter = 0;
    while (nevyazka(X, ans) > eps) {
        for (int i = 0; i < X->n; i++) {
            double sum = 0;
            for (int j = 0; j < X->n; j++) {
                sum += X->matrix[i][j] * ans[j];
            }
            ans[i] = ans[i] + (om / X->matrix[i][i]) * (X->f[i] - sum);
        }
        iter++;
    }
    printf("Количество итераций: %d  ", iter);
```

```
    return ans;
}



int main(int argc, char *argv[]) {
    int task;
    linear_system * X = malloc(sizeof(linear_system));
    sscanf(argv[1], "%d", &task);
    X->leading_el = 0;
    if (task < 3 && argc > 2) {
        sscanf(argv[2], "%d", &(X->leading_el));
    }
    FILE * fp = input_selection(X);
    if (fp == NULL) {
        input_prel(X, task);
    } else {
        input(X, fp, task);
    }
    printf("\nИсходная матрица:\n");
    show_matrix(X, task);
    printf("\n");
    if (task == 1) {
        straight_run(X, task);
        double * ans = back_run(X, task);
        printf("Решение СЛАУ:\n");
        for(int j = 0;j < X->n; j++) {
            for (int i = 0; i < X->n; i++) {
                if (X->index_arr[i] == j) {
                    printf("x%d: %lf\n", j + 1, ans[i]);
                }
            }
        }
    } else if (task == 2) {
        printf("Определитель: %lf\n", straight_run(X, task));
    } else if (task == 3) {
        straight_run(X, task);
        back_run_reverse(X, task);
        printf("A^(-1): \n");
        for (int i = 0; i < X->n; i++) {
            for (int j = 0; j < X->n; j++) {
                printf("%.2lf ", X->I[i][j]);
            }
```

```
            printf("\n");
        }
    } else if (task == 4) {
        double A_norm = MatrixNorm(X->matrix, X->n);
        straight_run(X, 3);
        back_run_reverse(X, 3);
        double A_inv_norm = MatrixNorm(X->I, X->n);
        printf("Степень обусловленности матрицы: %lf\n", A_norm * A_inv_norm);
    } else if (task == 5) {
        double om = 0.1, eps = 0.0001;
        for (int i = 1; i < 20; i++) {
            printf("Омега: %lf ", om * i);
            double * ans = Relax(X, eps, om * i);
            for (int i = 0; i < X->n; i++) {
                printf("x%d: %lf  ", i+1, ans[i]);
            }
            printf("\n");
        }
    }
    for (int i = 0; i < X->n;i++) {
        free(X->matrix[i]);
        if (task == 3 || task == 4) {
            free(X->I[i]);
        }
    }
    free(X->matrix);
    if (task == 3 || task == 4) {
        free(X->I);
    }
    if (task == 1 || task == 5) {
        free(X->f);
    }
    if (task <=2){
        free(X->index_arr);
    }
    free(X);
    fclose(fp);
    return 0;
}
```

# Тесты, доказывающие корректность работы программы

Тестирование проводилось при использовании библиотеки numpy языка программирования python.

**Тест №1**

Исходная матрица:
2 -5 3 1 | 5
3 -7 3 -1 | -1
5 -9 6 2 | 7
4 -6 3 1 | 8
Решение найденное при использовании библиотеки numpy:
(0, -3, -5.333, 6)
Вывод метода Гаусса без использования ведущего элемента:

```
x1: -0.000000
x2: -3.000000
x3: -5.333333
x4: 6.000000
```

Вывод метода Гаусса с использованием ведущего элемента:

```
x1: 0.000000
x2: -3.000000
x3: -5.333333
x4: 6.000000
```

**Тест №2**

Исходная матрица:
4 3 -9 1 | 9
2 5 -8 -1 | 8
2 16 -14 2 | 24
2 3 -5 -11 | 7
Решение найденное при использовании библиотеки numpy:
(3, 2, 1, 0)
Вывод метода Гаусса без использования ведущего элемента:

```
x1: 3.000000
x2: 2.000000
x3: 1.000000
x4: -0.000000
```

Вывод метода Гаусса с использованием ведущего элемента:

```
x1: 3.000000
x2: 2.000000
x3: 1.000000
x4: 0.000000
```

**Тест №3**

Исходная матрица:
12 14 -15 24 | 5
16 18 -22 29 | 8
18 20 -21 32 | 9
10 12 -16 20 | 4
Решение найденное при использовании библиотеки numpy:
(2.22, -1.67, -0.11, 0)
Вывод метода Гаусса без использования ведущего элемента:

```
x1: 2.222222
x2: -1.666667
x3: -0.111111
x4: -0.000000
```

Вывод метода Гаусса с использованием ведущего элемента:

```
x1: 2.222222
x2: -1.666667
x3: -0.111111
x4: 0.000000
```

**Тест №4**

Исходная матрица:
2  1  4  8 | -1
1  3 -6  2 |  3
3 -2  2 -2 |  8
2 -1  2  0 |  4
Решение найденное при использовании библиотеки numpy:
(2, -3, -1.5, 0.5)
Вывод метода Гаусса без использования ведущего элемента:

```
x1: 2.000000
x2: -3.000000
x3: -1.500000
x4: 0.500000
```

Вывод метода Гаусса с использованием ведущего элемента:

```
x1: 2.000000
x2: -3.000000
x3: -1.500000
x4: 0.500000
```

**Тест №5**

Приложение 2

n = 20, m = 8

Решение найденное при использовании библиотеки numpy:

```
array([2.16903211, 2.20456485, 2.24001019, 2.27536843, 2.31063992,
       2.34582495, 2.38092386, 2.41593696, 2.45086455, 2.48570697,
       2.5204645 , 2.55513747, 2.58972619, 2.62423096, 2.65865208,
       2.69298986, 2.7272446 , 2.76141661, 2.79550617, 2.8295136 ])
```

Вывод метода Гаусса с использованием ведущего элемента:

# Вывод

Был изучен метод Гаусса, метод Гаусса с выбором главного элемента для решения СЛАУ и написана программа, реализующая эти методы. Результаты работы совпали с реальными решениями, полученными в Numpy. Было выявлено, что точность метода Гаусса с выбором главного элемента выше, так как в силу использования эвристики он накапливает меньшую погрешность. Помимо нахождения решения СЛАУ, программа дополнительно вычисляет определитель матрицы, её обратную матрицу и число обусловленности, если матрица невырожденная.

**Подвариант №2**

**Итерационные методы решения систем линейных алгебраических уравнений.**
**(на примере методов Зейделя и верхней релаксации)**

# Постановка задачи

Дана система уравнений $Ax = f$ порядка $n * n$ с невырожденной матрицей $A$. Написать программу, численного решения СЛАУ ($n$ - параметр программы), использующий численный алгоритм итерационного метода верхней релаксации:

$$(D + \omega T_H)\frac{(\mathbf{x}_{k+1} - \mathbf{x}_k)}{\omega} + A\mathbf{x}_k = \mathbf{f}.$$

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

# Цели и задачи практической работы

1. Решить заданную СЛАУ итерационным методом верхней релаксации
2. Разработать критерий остановки итерационного процесса, гарантирующий
3. Изучить скорость сходимости итераций к точному решению задачи
4. Правильность решения подтвердить системой тестов

# Описание метода решения

Рассмотрим произвольную квадратную матрицу *A.* Разложим её на сумму трех матриц $A = D + T_н + T_в$ , где *D* - диагональная часть матрицы *A*, которая содержит элементы $a_{ii}$ , стоящие на главной диагонали, $T_н$, $T_в$ - соответственно нижнетреугольная и верхнетреугольная матрицы. Введем параметр омега и запишем рекуррентное соотношение в виде:

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

Которое можно переписать в виде:

Что и задает алгоритм верхней релаксации.

# Описание программы

Первым аргументом командной строки при запуске программы является число - 5 (индекс задачи), 1-4 используется для первого подварианта. После запуска программы, сначала пользователь проходит 3 этапа задания матрицы, затем начинает работать непосредственно алгоритм верхней релаксации, который возвращает значения неизвестных. (Листинг программы см. подвариант 1)

# Критерий остановки

В качестве критерия остановки в данной задаче на каждой итерации алгоритма мы будем оценивать невязку, которая выражается следующей формулой - $||Ax^k - f||$
Алгоритм работает до тех пор, пока невязка не стала меньше некоторого заранее выбранного числа eps (в данной программе оно равно 0.0001)

# Тесты, доказывающие корректность работы программы

**Тест №1**

Исходная матрица:
18 57 55 23 | 242
57 198 188 94 | 829
55 188 182 87 | 795
23 94 87 55 | 385
Решение найденное при использовании библиотеки numpy:
(1, 2, 2, 0)
Решение найденное, используя метод верхней релаксации:

```
Омега: 0.100000 Количество итераций: 1778180  x1: 1.050188  x2: 1.976135  x3: 2.000083  x4: 0.019670
Омега: 0.200000 Количество итераций: 833905  x1: 1.050680  x2: 1.975903  x3: 2.000082  x4: 0.019862
Омега: 0.300000 Количество итераций: 519335  x1: 1.051113  x2: 1.975699  x3: 2.000080  x4: 0.020031
Омега: 0.400000 Количество итераций: 362223  x1: 1.051442  x2: 1.975546  x3: 2.000078  x4: 0.020159
Омега: 0.500000 Количество итераций: 268125  x1: 1.051599  x2: 1.975474  x3: 2.000076  x4: 0.020219
Омега: 0.600000 Количество итераций: 205553  x1: 1.051492  x2: 1.975529  x3: 2.000072  x4: 0.020177
Омега: 0.700000 Количество итераций: 161005  x1: 1.050996  x2: 1.975768  x3: 2.000068  x4: 0.019981
Омега: 0.800000 Количество итераций: 127702  x1: 1.049960  x2: 1.976265  x3: 2.000063  x4: 0.019574
Омега: 0.900000 Количество итераций: 101826  x1: 1.048217  x2: 1.977098  x3: 2.000056  x4: 0.018890
Омега: 1.000000 Количество итераций: 80979  x1: 1.045622  x2: 1.978337  x3: 2.000047  x4: 0.017871
Омега: 1.100000 Количество итераций: 63415  x1: 1.042101  x2: 1.980015  x3: 2.000038  x4: 0.016490
Омега: 1.200000 Количество итераций: 47371  x1: 1.037710  x2: 1.982107  x3: 2.000027  x4: 0.014768
Омега: 1.300000 Количество итераций: 28579  x1: 1.032643  x2: 1.984520  x3: 2.000016  x4: 0.012781
Омега: 1.400000 Количество итераций: 23553  x1: 0.972802  x2: 2.012889  x3: 1.999995  x4: -0.010647
Омега: 1.500000 Количество итераций: 31283  x1: 0.978309  x2: 2.010269  x3: 2.000006  x4: -0.008488
Омега: 1.600000 Количество итераций: 29647  x1: 0.983620  x2: 2.007743  x3: 2.000015  x4: -0.006406
Омега: 1.700000 Количество итераций: 24946  x1: 0.988579  x2: 2.005384  x3: 2.000023  x4: -0.004463
Омега: 1.800000 Количество итераций: 18216  x1: 0.993183  x2: 2.003195  x3: 2.000031  x4: -0.002659
Омега: 1.900000 Количество итераций: 8714  x1: 1.001569  x2: 1.999300  x3: 1.999961  x4: 0.000602
```

**Тест №2**

Исходная матрица:

2 -1  0 | 1

-1 2 -1 | 1

0 -1  2 | 1

Решение найденное при использовании библиотеки numpy:

(1.5, 2, 1.5)

Решение найденное, используя метод верхней релаксации:

```
Омега: 0.100000 Количество итераций: 317  x1: 1.499916  x2: 1.999883  x3: 1.499919
Омега: 0.200000 Количество итераций: 150  x1: 1.499915  x2: 1.999883  x3: 1.499920
Омега: 0.300000 Количество итераций: 95  x1: 1.499920  x2: 1.999892  x3: 1.499928
Омега: 0.400000 Количество итераций: 67  x1: 1.499920  x2: 1.999895  x3: 1.499931
Омега: 0.500000 Количество итераций: 50  x1: 1.499919  x2: 1.999896  x3: 1.499934
Омега: 0.600000 Количество итераций: 39  x1: 1.499926  x2: 1.999908  x3: 1.499943
Омега: 0.700000 Количество итераций: 31  x1: 1.499932  x2: 1.999919  x3: 1.499951
Омега: 0.800000 Количество итераций: 24  x1: 1.499913  x2: 1.999900  x3: 1.499942
Омега: 0.900000 Количество итераций: 19  x1: 1.499912  x2: 1.999905  x3: 1.499948
Омега: 1.000000 Количество итераций: 15  x1: 1.499924  x2: 1.999924  x3: 1.499962
Омега: 1.100000 Количество итераций: 11  x1: 1.499918  x2: 1.999929  x3: 1.499969
Омега: 1.200000 Количество итераций: 8  x1: 1.500019  x2: 2.000012  x3: 1.500002
Омега: 1.300000 Количество итераций: 10  x1: 1.500014  x2: 2.000007  x3: 1.499997
Омега: 1.400000 Количество итераций: 12  x1: 1.500014  x2: 2.000042  x3: 1.500012
Омега: 1.500000 Количество итераций: 15  x1: 1.500025  x2: 2.000048  x3: 1.500046
Омега: 1.600000 Количество итераций: 21  x1: 1.499990  x2: 1.999954  x3: 1.499987
Омега: 1.700000 Количество итераций: 30  x1: 1.499970  x2: 2.000016  x3: 1.500015
Омега: 1.800000 Количество итераций: 47  x1: 1.500060  x2: 2.000039  x3: 1.500012
Омега: 1.900000 Количество итераций: 96  x1: 1.499953  x2: 1.999924  x3: 1.499931
```

**Тест №3**

Исходная матрица:

 10  12   2  -19 | 30

 12  20   0  -27 | 52

  2   0 36   31 | -70

-19 -27 31 101 | -135

Решение найденное при использовании библиотеки numpy:

(1, 2, -2, 0)

Решение найденное, используя метод верхней релаксации:

**Тест №4**

Исходная матрица:
39 49 75 48 | 5
49 68 94 62 |-1
75 94 146 94 | 7
48 62 94 62 | 8

Решение найденное при использовании библиотеки numpy:
(14.333333 -3.888889 -10.629630 9.037037)
Решение найденное, используя метод верхней релаксации:

```
Омега: 0.100000 Количество итераций: 48260  x1: 14.332923  x2: -3.888815  x3: -10.629330  x4: 9.036827
Омега: 0.200000 Количество итераций: 22853  x1: 14.332920  x2: -3.888816  x3: -10.629329  x4: 9.036828
Омега: 0.300000 Количество итераций: 14381  x1: 14.332918  x2: -3.888816  x3: -10.629329  x4: 9.036830
Омега: 0.400000 Количество итераций: 10144  x1: 14.332917  x2: -3.888818  x3: -10.629330  x4: 9.036833
Омега: 0.500000 Количество итераций: 7600  x1: 14.332918  x2: -3.888819  x3: -10.629332  x4: 9.036838
Омега: 0.600000 Количество итераций: 5902  x1: 14.332923  x2: -3.888822  x3: -10.629337  x4: 9.036844
Омега: 0.700000 Количество итераций: 4688  x1: 14.332932  x2: -3.888825  x3: -10.629345  x4: 9.036853
Омега: 0.800000 Количество итераций: 3774  x1: 14.332946  x2: -3.888830  x3: -10.629357  x4: 9.036865
Омега: 0.900000 Количество итераций: 3059  x1: 14.332967  x2: -3.888835  x3: -10.629375  x4: 9.036881
Омега: 1.000000 Количество итераций: 2480  x1: 14.332998  x2: -3.888843  x3: -10.629399  x4: 9.036901
Омега: 1.100000 Количество итераций: 1994  x1: 14.333038  x2: -3.888852  x3: -10.629430  x4: 9.036925
Омега: 1.200000 Количество итераций: 1567  x1: 14.333091  x2: -3.888862  x3: -10.629469  x4: 9.036955
Омега: 1.300000 Количество итераций: 1147  x1: 14.333162  x2: -3.888875  x3: -10.629522  x4: 9.036992
Омега: 1.400000 Количество итераций: 844  x1: 14.333292  x2: -3.888871  x3: -10.629590  x4: 9.036991
Омега: 1.500000 Количество итераций: 1027  x1: 14.333356  x2: -3.888879  x3: -10.629633  x4: 9.037014
Омега: 1.600000 Количество итераций: 1276  x1: 14.333330  x2: -3.888875  x3: -10.629615  x4: 9.037003
Омега: 1.700000 Количество итераций: 1724  x1: 14.333355  x2: -3.888906  x3: -10.629658  x4: 9.037080
Омега: 1.800000 Количество итераций: 2572  x1: 14.333192  x2: -3.888860  x3: -10.629525  x4: 9.036959
Омега: 1.900000 Количество итераций: 5330  x1: 14.333177  x2: -3.888860  x3: -10.629517  x4: 9.036959
```

# Вывод

Был изучен метод верхней релаксации для решения СЛАУ и написана программа, реализующая этот итерационный метод. Было выяснено, что для матриц небольшого размера эффективнее использовать метод Гаусса. Минимальное число итераций происходит при значении итерационного параметра w = 1.2. Если решения не существует, то метод релаксации работает некорректно. То есть, перед его использованием следует исследовать систему на совместность.