Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий Кафедра прикладной математики

Отчет защищен с оценкой

Преподаватель

(подпись)

«\_ » 2023 г.

Отчет

По лабораторной работе №2

по дисциплине «Вычислительные алгоритмы»

Студент группы

ПИ-02 Чередов Р.А.

Преподаватель

Проскурин А.В.

Барнаул 2023

# Задание

* Составить программу для решения системы линейных алгебраических уравнений методами Якоби и Зейделя, Исходные данные — матрица системы уравнений и столбец свободных членов, точность е должны читаться из файла, а результаты расчетов помещаться в файл. Предусмотреть вывод числа итераций, необходимых для получения решения с заданной точностью, нормы невязки. Предусмотреть возможность автоматической генерации матрицы системы уравнений и правой части для тестирования методов на системах большой размерности. Для тестирования программы можно использовать пример из предыдущей работы.
* Исследовать зависимость числа итераций от начального приближения, точности, выбора метода решения.
* Изучить влияние на сходимость величины диагонального преобладания матрицы, то есть величины отношения суммы модулей недиагональных элементов строки к модулю диагонального элемента.
* Подобрать примеры, показывающие, что диагональное преобладание не является необходимым условием сходимости.

# Решение

1. **Код программы**

nclude <random> nclude < ostream> nclude <fstream> nclude < oman p>



// Чтение матрицы из файла

vo d read\_matr x(double\*\* a, nt n, nt m, std:: fstream& np)

{

for ( nt = 0; < n; ++) {

for ( nt j = 0; j < m; j++) { np >> a[ ][j];

}

}

}

// Печать матрицы

vo d pr nt\_matr x(double\*\* a, nt n, nt m)

{

for ( nt = 0; < n; ++) {

for ( nt j = 0; j < m; j++) {

std::cout << std::setw(15) << a[ ][j] << ' ';

}

std::cout << std::endl;

}

}

// Запись матрицы в файл

vo d wr te\_matr x(double\*\* matr x, nt n, nt m, std::ofstream& outp) { for ( nt = 0; < n; ++) {

for ( nt j = 0; j < n + 1; j++) {

outp << std::setw(15) << matr x[ ][j] << ' ';

}

outp << std::endl;

}

}

double add\_form(double\*\* a, nt n, double\* x, nt )

{

double result = 0;

for ( nt j = 0; j < n; j++) {

f (j != ) result -= a[ ][j] \* x[j];

}

return result;

}

// Подсчет ошибки

double get\_error(double\* x, double\* tmp\_x, nt n)

{

double error = fabs(tmp\_x[0] - x[0]); for ( nt = 0; < n; ++) {

double tmp = fabs(tmp\_x[ ] - x[ ]); f (error < tmp) error = tmp;

}

return error;

}

// Метод Якоби

bool jacob (double\*\* a, nt n, double\* x, nt max\_k, double eps, nt& k)

{

bool rashod = false;

double\* tmp\_x = new double[n]; double error = INFINITY;

k = 0;

wh le (error > eps && k < max\_k) { k++;

for ( nt = 0; < n; ++) {

tmp\_x[ ] = (a[ ][n] + add\_form(a, n, x, )) / a[ ][ ];

}

error = get\_error(x, tmp\_x, n);

for ( nt = 0; < n; ++) x[ ] = tmp\_x[ ]; f (error > 1000000) {

rashod = true; break;

}

}

delete[] tmp\_x;

f (k == max\_k || rashod) return false;

return true;

}

// Метод Зейделя

bool ze del(double\*\* a, nt n, double\* x, nt max\_k, double eps, nt& k)

{

bool rashod = false;

double\* tmp\_x = new double[n]; double\* x\_copy = new double[n]; double error = INFINITY;

k = 0;

for ( nt = 0; < n; ++) x\_copy[ ] = x[ ]; wh le (error > eps && k < max\_k) {

k++;

for ( nt = 0; < n; ++) {

tmp\_x[ ] = (a[ ][n] + add\_form(a, n, x\_copy, )) / a[ ][ ]; x\_copy[ ] = tmp\_x[ ];

}

error = get\_error(x, tmp\_x, n);

for ( nt = 0; < n; ++) x[ ] = tmp\_x[ ]; f (error > 1000000) {

rashod = true; break;

}

}

delete[] tmp\_x; delete[] x\_copy;

f (k == max\_k || rashod) return false;

return true;

}

// Метод Якоби с выводом промежуточных результатов и записью в файл

bool jacob (double\*\* a, nt n, double\* x, nt max\_k, double eps, std::ostream& out)

{

double\* tmp\_x = new double[n]; double error = INFINITY;

nt k = 0;

wh le (error > eps && k < max\_k) { k++;

f (n < 20 && k < 50) {

std::cout << "Итерация = " << k << std::endl; out << "Итерация = " << k << std::endl;

}

for ( nt = 0; < n; ++) {

tmp\_x[ ] = (a[ ][n] + add\_form(a, n, x, )) / a[ ][ ];

std::cout << "x" << + 1 << " = " << std::setw(10) << tmp\_x[ ] << " |

";

f (n < 20 && k < 50) {

out << "x" << + 1 << " = " << std::setw(10) << tmp\_x[ ] << " | ";

}

}

error = get\_error(x, tmp\_x, n); f (n < 20 && k < 50) {

std::cout << "Ошибка = " << error; out << "Ошибка = " << error; std::cout << std::endl;

out << std::endl;

}

for ( nt = 0; < n; ++) x[ ] = tmp\_x[ ]; f (error > 1000000)

break;

}

std::cout << std::endl; out << std::endl;

f (k == max\_k) {

std::cout << "Решение расходится. Количество итераций превысило максимальное значение: " << max\_k << std::endl;

out << "Решение расходится. Количество итераций превысило максимальное значение: " << max\_k << std::endl;

delete[] tmp\_x; return false;

}

else f (error > 1000000) {

std::cout << "Решение расходится. Ошибка превысила значение 1000000" << std::endl;

out << "Решение расходится. Ошибка превысила значение 1000000" << std::endl; delete[] tmp\_x;

return false;

}

else {

std::cout << "Решение сходится." << std::endl; out << "Решение сходится." << std::endl; std::cout << "Вектор X:" << std::endl;

out << "Вектор X:" << std::endl; for ( nt = 0; < n; ++) {

std::cout << "x" << + 1 << " = " << x[ ] << std::endl; out << "x" << + 1 << " = " << x[ ] << std::endl;

}

}

std::cout << "Количество итераций = " << k << std::endl; std::cout << "Точность = " << eps << std::endl;

out << "Количество итераций = " << k << std::endl; out << "Точность = " << eps << std::endl;

delete[] tmp\_x; return true;

}

// Метод Зейделя с выводом промежуточных результатов и записью в файл

bool ze del(double\*\* a, nt n, double\* x, nt max\_k, double eps, std::ofstream& out)

{

double\* tmp\_x = new double[n]; double\* x\_copy = new double[n]; double error = INFINITY;

nt k = 0;

for ( nt = 0; < n; ++) x\_copy[ ] = x[ ]; wh le (error > eps && k < max\_k) {

k++;

f (n < 20 && k < 50) {

std::cout << "Итерация = " << k << std::endl; out << "Итерация = " << k << std::endl;

| ";

}

for ( nt = 0; < n; ++) {

tmp\_x[ ] = (a[ ][n] + add\_form(a, n, x\_copy, )) / a[ ][ ]; x\_copy[ ] = tmp\_x[ ];

f (n < 20 && k < 50) {

std::cout << "x" << + 1 << " = " << std::setw(10) << tmp\_x[ ] << "

out << "x" << + 1 << " = " << std::setw(10) << tmp\_x[ ] << " | ";

}

}

error = get\_error(x, tmp\_x, n); f (n < 20 && k < 50) {

std::cout << "Ошибка = " << error; out << "Ошибка = " << error; std::cout << std::endl;

out << std::endl;

}

for ( nt = 0; < n; ++) x[ ] = tmp\_x[ ]; f (error > 1000000)

break;

}

std::cout << std::endl; out << std::endl;

f (k == max\_k) {

std::cout << "Решение расходится. Количество итераций превысило максимальное значение: " << max\_k << std::endl;

out << "Решение расходится. Количество итераций превысило максимальное значение: " << max\_k << std::endl;

delete[] tmp\_x; delete[] x\_copy; return false;

}

else f (error > 1000000) {

std::cout << "Решение расходится. Ошибка превысила значение 1000000" << std::endl;

out << "Решение расходится. Ошибка превысила значение 1000000" << std::endl; delete[] tmp\_x;

delete[] x\_copy; return false;

}

else {

std::cout << "Решение сходится." << std::endl; out << "Решение сходится." << std::endl; std::cout << "Вектор X:" << std::endl;

out << "Вектор X:" << std::endl; for ( nt = 0; < n; ++) {

std::cout << "x" << + 1 << " = " << x[ ] << std::endl; out << "x" << + 1 << " = " << x[ ] << std::endl;

}

}

std::cout << "Количество итераций = " << k << std::endl;

std::cout << "Точность = " << eps << std::endl; out << "Количество итераций = " << k << std::endl; out << "Точность = " << eps << std::endl;

delete[] tmp\_x; delete[] x\_copy; return true;

}

// Освобождение памяти

vo d free\_mem(double\*\* matr x, nt n) { for ( nt = 0; < n; ++)

delete[] matr x[ ]; delete[] matr x;

}

// Рандомное заполнение матрицы

vo d f ll\_matr x(double\*\* matr x, nt n, nt m)

{

for ( nt = 0; < n; ++) {

for ( nt j = 0; j < m; j++) {

matr x[ ][j] = rand() % 200 - 100;

}

}

}

// Проверка, есть ли нули на главной диагонали bool check\_zeros(double\*\* a, nt n)

{

for ( nt = 0; < n; ++) {

f (a[ ][ ] == 0) return true;

}

return false;

}

// Ввод входных значений n, eps, max\_k

vo d nput\_values( nt& n, double& eps, nt& max\_k)

{

do {

std::cout << std::endl << "Введите размерность матрицы: "; std::c n >> n;

} wh le (n < 2); do {

std::cout << "Введите точность: "; std::c n >> eps;

} wh le (eps < 0); do {

std::cout << "Введите максимальное число итераций: "; std::c n >> max\_k;

} wh le (max\_k < 1);

}

// Проверка диагонального преобладания

bool check\_d agonal\_dom nance(double\*\* a, nt n)

{

nt sum = 0;

bool more = false;

for ( nt = 0; < n; ++) { sum = 0;

for ( nt j = 0; j < n; j++) { f ( != j)

sum += abs(a[ ][j]);

}

f (abs(a[ ][ ]) > sum) more = true;

f (sum > abs(a[ ][ ])) return false;

}

return more;

}

// Перестановки строк (Попытка добиться диагонального преобладания)

vo d str ng\_permutat on(double\*\* a, nt n)

{

nt max\_row; double max; double\* tmp\_row;

for ( nt = 0; < n; ++) { max\_row = ;

max = a[ ][ ];

// Перестановки строк

for ( nt p = ; p < n; p++) { f (fabs(a[p][ ]) > max) {

max\_row = p;

max = fabs(a[p][ ]);

}

}

f (!(max\_row == )) { tmp\_row = a[max\_row]; a[max\_row] = a[ ];

a[ ] = tmp\_row;

}

}

}

// Вывод меню

vo d pr nt\_menu()

{

std::cout << "Меню:" << std::endl;

std::cout << "1. Решение СЛАУ из файла методом Якоби" << std::endl; std::cout << "2. Решение СЛАУ из файла методом Зейделя" << std::endl;

std::cout << "3. Решение автоматически сгенерированной СЛАУ методом Якоби" << std::endl;

std::cout << "4. Решение автоматически сгенерированной СЛАУ методом Зейделя" << std::endl;

std::cout << "5. Метод Якоби. Тест сходимости на 50000 случайных СЛАУ" << std::endl;

std::cout << "6. Метод Зейделя. Тест сходимости на 50000 случайных СЛАУ" << std::endl;

std::cout << "7. Выход из программы" << std::endl; std::cout << std::endl;

std::cout << "Выберите пункт меню: ";

}

// Вывод подменю и ввод пункта подменю bool submenu()

{

nt cho ce = 0;

std::cout << std::endl << "Использовать перестановку строк, чтобы попытаться добиться диагонального преобладания?" << std::endl;

std::cout << "1. Да" << std::endl;

std::cout << "2. Нет" << std::endl << std::endl; do {

std::cout << "Ввод: "; std::c n >> cho ce;

} wh le (cho ce < 1 || cho ce > 2); f (cho ce == 1)

return true; return false;

}

// Нахождение нормы невязки

double norma\_nevyazk (double\*\* a, double\* x, nt n)

{

double tmp, nevyazka\_tmp, norma = 0; for ( nt = 0; < n; ++) {

tmp = 0;

for ( nt j = 0; j < n; j++) { tmp += a[ ][j] \* x[j];

}

nevyazka\_tmp = a[ ][n] - tmp; f (nevyazka\_tmp > norma)

norma = nevyazka\_tmp;

}

return norma;

}

nt ma n()

{

setlocale(LC\_ALL, "Russ an"); nt n, max\_k;

double eps; nt menu\_ tem;

wh le (true) { srand(t me(0));

system("cls"); pr nt\_menu();

std::c n >> menu\_ tem; sw tch (menu\_ tem)

{

case 1: // Метод Якоби с чтением матрицы из файла и записью результатов в

файл

{

std:: fstream np("Input.txt"); std::ofstream out("Result.txt");

f ( np. s\_open()) { np >> n;

np >> eps; np >> max\_k;

double\*\* a = new double\* [n]; double\* x = new double[n]; for ( nt = 0; < n; ++) {

a[ ] = new double[n + 1]; x[ ] = 0;

std::endl; std::endl;

}

read\_matr x(a, n, n + 1, np);

std::cout << "Исходная матрица:" << std::endl; out << "Исходная матрица:" << std::endl;

wr te\_matr x(a, n, n + 1, out); pr nt\_matr x(a, n, n + 1);

f (check\_d agonal\_dom nance(a, n)) {

std::cout << "Матрица обладает диагональным преобладанием" << out << "Матрица обладает диагональным преобладанием" <<

}

std::cout << std::endl; out << std::endl;

f (check\_zeros(a, n)) { free\_mem(a, n); delete[] x;

out << "Деление на ноль! Поменяйте местами уравнения."; out.close();

np.close();

std::cout << "Деление на ноль! Поменяйте местами уравнения."; break;

std::endl; std::endl;

}

f (jacob (a, n, x, max\_k, eps, out)) {

std::cout << "Норма невязки = " << norma\_nevyazk (a, x, n) << out << "Норма невязки = " << norma\_nevyazk (a, x, n) <<

}

free\_mem(a, n); delete[] x;

}

else std::cout << "Не удается открыть файл nput.txt"; out.close();

np.close();

system("pause"); break;

}

файл

case 2: // Метод Зейделя с чтением матрицы из файла и записью результатов в

{

std:: fstream np("Input.txt"); std::ofstream out("Result.txt");

f ( np. s\_open()) { np >> n;

np >> eps; np >> max\_k;

double\*\* a = new double\* [n]; double\* x = new double[n]; for ( nt = 0; < n; ++) {

a[ ] = new double[n + 1]; x[ ] = 0;

std::endl; std::endl;

}

read\_matr x(a, n, n + 1, np);

std::cout << "Исходная матрица:" << std::endl; out << "Исходная матрица:" << std::endl;

wr te\_matr x(a, n, n + 1, out); pr nt\_matr x(a, n, n + 1);

f (check\_d agonal\_dom nance(a, n)) {

std::cout << "Матрица обладает диагональным преобладанием" << out << "Матрица обладает диагональным преобладанием" <<

}

std::cout << std::endl; out << std::endl;

f (check\_zeros(a, n)) { free\_mem(a, n); delete[] x;

out << "Деление на ноль! Поменяйте местами уравнения."; out.close();

np.close();

std::cout << "Деление на ноль! Поменяйте местами уравнения."; break;

std::endl; std::endl;

}

f (ze del(a, n, x, max\_k, eps, out)) {

std::cout << "Норма невязки = " << norma\_nevyazk (a, x, n) << out << "Норма невязки = " << norma\_nevyazk (a, x, n) <<

}

free\_mem(a, n); delete[] x;

}

else std::cout << "Не удается открыть файл nput.txt"; np.close();

out.close();

system("pause"); break;

}

case 3: // Метод Якоби для случайной матрицы с записью результатов в файл

{

std::ofstream out("Result.txt"); nput\_values(n, eps, max\_k);

double\*\* a = new double\* [n]; double\* x = new double[n]; for ( nt = 0; < n; ++) {

a[ ] = new double[n + 1]; x[ ] = 0;

}

std::endl;

f ll\_matr x(a, n, n + 1);

std::cout << "Исходная матрица:" << std::endl; out << "Исходная матрица:" << std::endl;

wr te\_matr x(a, n, n + 1, out); pr nt\_matr x(a, n, n + 1);

f (check\_d agonal\_dom nance(a, n)) {

std::cout << "Матрица обладает диагональным преобладанием" <<

out << "Матрица обладает диагональным преобладанием" << std::endl;

}

std::cout << std::endl; out << std::endl;

f (check\_zeros(a, n)) { free\_mem(a, n); delete[] x;

out << "Деление на ноль! Поменяйте местами уравнения."; out.close();

std::cout << "Деление на ноль! Поменяйте местами уравнения."; break;

std::endl;

}

f (jacob (a, n, x, max\_k, eps, out)) {

std::cout << "Норма невязки = " << norma\_nevyazk (a, x, n) <<

out << "Норма невязки = " << norma\_nevyazk (a, x, n) << std::endl;

}

free\_mem(a, n); delete[] x; out.close();

system("pause"); break;

}

case 4: // Метод Зейделя для случайной матрицы с записью результатов в файл

{

std::ofstream out("Result.txt"); nput\_values(n, eps, max\_k);

double\*\* a = new double\* [n]; double\* x = new double[n]; for ( nt = 0; < n; ++) {

a[ ] = new double[n + 1]; x[ ] = 0;

std::endl;

}

f ll\_matr x(a, n, n + 1);

std::cout << "Исходная матрица:" << std::endl; out << "Исходная матрица:" << std::endl;

wr te\_matr x(a, n, n + 1, out); pr nt\_matr x(a, n, n + 1);

f (check\_d agonal\_dom nance(a, n)) {

std::cout << "Матрица обладает диагональным преобладанием" <<

out << "Матрица обладает диагональным преобладанием" << std::endl;

}

std::cout << std::endl; out << std::endl;

f (check\_zeros(a, n)) { free\_mem(a, n); delete[] x;

out << "Деление на ноль! Поменяйте местами уравнения."; out.close();

std::cout << "Деление на ноль! Поменяйте местами уравнения."; break;

}

pr nt\_matr x(a, n, n + 1);

std::endl;

}

f (ze del(a, n, x, max\_k, eps, out)) {

std::cout << "Норма невязки = " << norma\_nevyazk (a, x, n) <<

out << "Норма невязки = " << norma\_nevyazk (a, x, n) << std::endl;

}

free\_mem(a, n); delete[] x; out.close();

system("pause"); break;

case 5: // Решение 50000 СЛАУ методом Якоби

{

bool permutat ons; permutat ons = submenu();

nput\_values(n, eps, max\_k); double\*\* a = new double\* [n]; double\* x = new double[n];

nt k, k\_overall = 0;

for ( nt = 0; < n; ++) { a[ ] = new double[n + 1]; x[ ] = 0;

}

nt count = 0, no\_d v de\_by\_zero = 50000; for ( nt = 0; < 50000; ++) {

f ll\_matr x(a, n, n + 1); f (permutat ons)

str ng\_permutat on(a, n);

for ( nt = 0; < n; ++) x[ ] = 0; f (check\_zeros(a, n)) {

no\_d v de\_by\_zero--; cont nue;

}

f (jacob (a, n, x, max\_k, eps, k)) { count++;

k\_overall += k;

}

}

std::cout << "Процент сходимости = " << stat c\_cast<float>(count) / no\_d v de\_by\_zero \* 100 << "%" << std::endl;

std::cout << "Среднее число итераций = " << k\_overall / count <<

std::endl;

}

free\_mem(a, n); delete[] x; system("pause"); break;

case 6: // Решение 50000 СЛАУ методом Зейделя

{

bool permutat ons; permutat ons = submenu();

nput\_values(n, eps, max\_k); double\*\* a = new double\* [n]; double\* x = new double[n];

nt k, k\_overall = 0;

for ( nt = 0; < n; ++) { a[ ] = new double[n + 1]; x[ ] = 0;

}

nt count = 0, no\_d v de\_by\_zero = 50000; for ( nt = 0; < 50000; ++) {

f ll\_matr x(a, n, n + 1); f (permutat ons)

str ng\_permutat on(a, n);

for ( nt = 0; < n; ++) x[ ] = 0;

f (check\_zeros(a, n)) { no\_d v de\_by\_zero--; cont nue;

}

f (ze del(a, n, x, max\_k, eps, k)) { count++;

k\_overall += k;

}

}

std::cout << "Процент сходимости = " << stat c\_cast<float>(count) / no\_d v de\_by\_zero \* 100 << "%" << std::endl;

std::cout << "Среднее число итераций = " << k\_overall / count <<

std::endl;

}

free\_mem(a, n); delete[] x; system("pause"); break;

case 7:

return 0; default:

break;

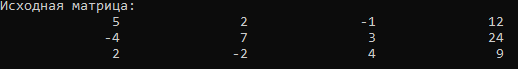
}

}

}

# Исследование зависимости числа итераций от начального приближения, точности, выбора метода решения.

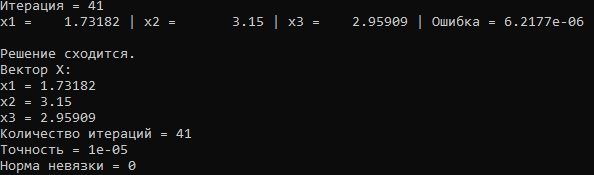
* 1. Зависимость числа итераций от начального приближения. Исходная расширенная матрица:



Начальное приближение 1:

X1 = 0; X2 = 0; X3 = 0

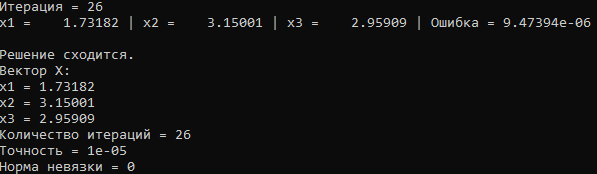
Решение:



Начальное приближение 2:

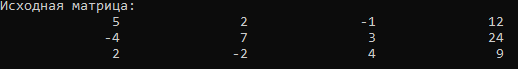
X1 = 1.7; X2 = 3.1; X3 = 2.9

Решение:

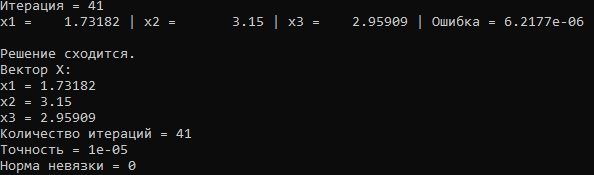


Вывод: Чем ближе значения начального приближения к корням СЛАУ, тем меньше итераций потребуется для решения этой СЛАУ.

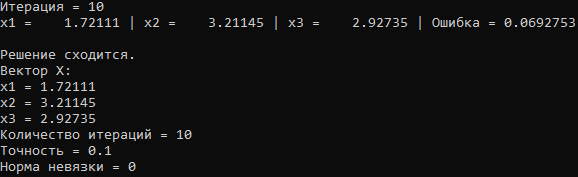
* 1. Зависимость числа итераций от точности. Исходная расширенная матрица:



Точность = 0.00001 Решение:

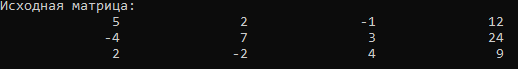


Точность = 0.1 Решение:

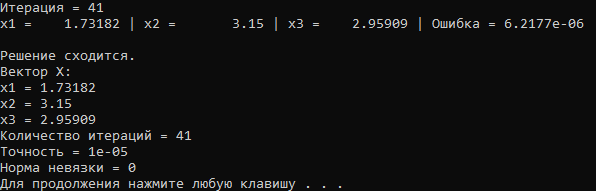


Вывод: Чем больше значение точности, тем меньше итераций потребуется для решения СЛАУ.

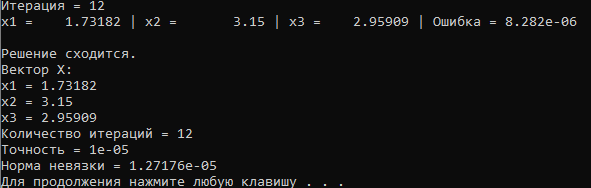
* 1. Зависимость числа итераций от выбора метода решения. Исходная расширенная матрица:



Метод Якоби:



Метод Зейделя:

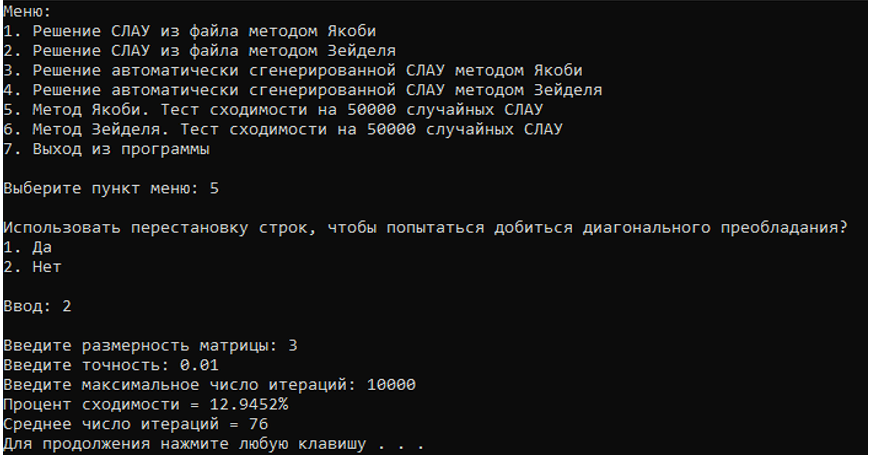


Вывод: Метод Зейделя сходится быстрее метода Якоби.

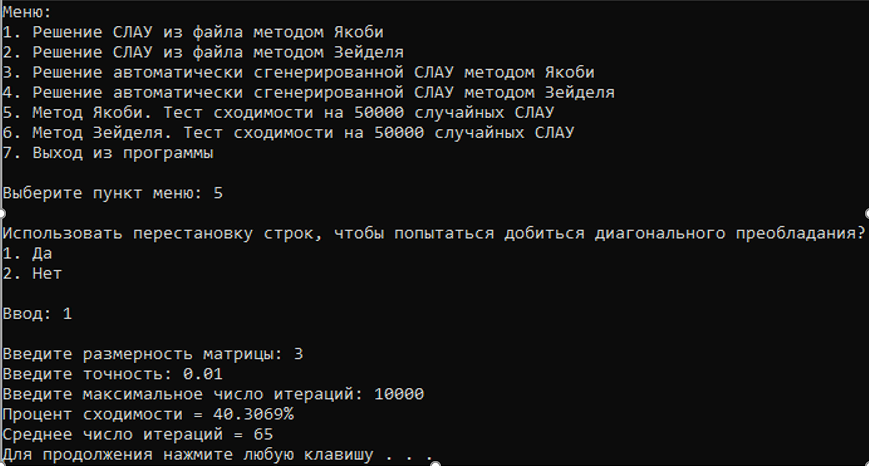
# Изучить влияние на сходимость величины диагонального преобладания матрицы, то есть величины отношения суммы модулей недиагональных элементов строки к модулю диагонального элемента.

Для изучения влияния диагонального преобладания на сходимость я проводил тестирование на 50000 различных случайных матрицах двумя способами. В первом способе я никак не изменял случайно заполненную матрицу. Во втором способе я пытался привести исходную матрицу к матрице с диагональным преобладанием. И сравнивал процент сходимости решений методами Якоби и Зейделя:

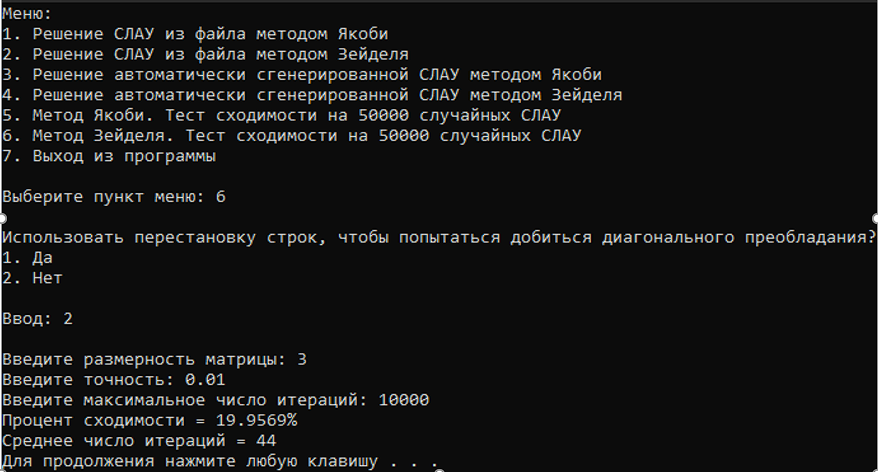
Метод Якоби без изменения матрицы:



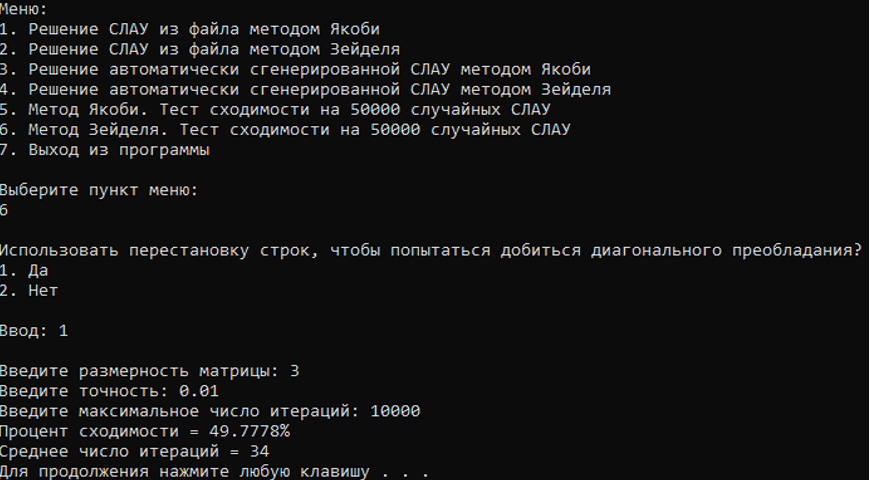
Метод Якоби с попыткой добиться диагонального преобладания:



Метод Зейделя без изменения матрицы:



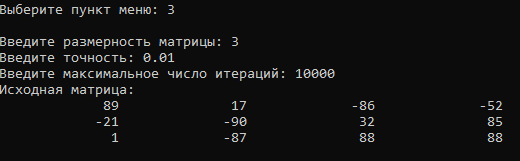
Метод Зейделя с попыткой добиться диагонального преобладания:

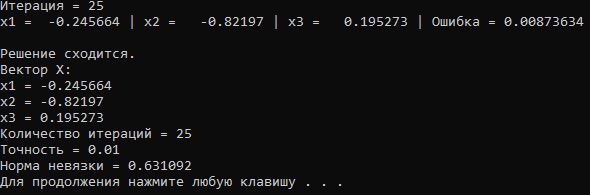


Выводы: Перестановки строк с попыткой вывести наибольшие элементы матрицы на главную диагональ действительно влияют на сходимость данных методов, но так как не существует универсального алгоритма получения диагонального преобладания в матрице, 100% сходимости данных методов добиться невозможно.

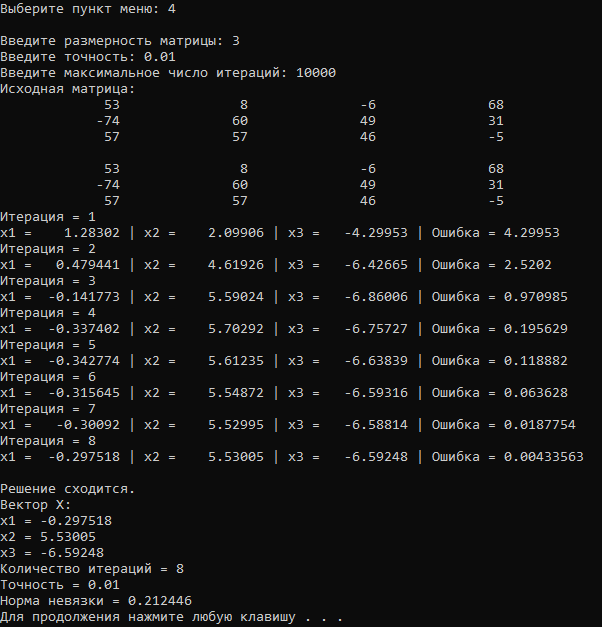
# Подобрать примеры, показывающие, что диагональное преобладание не является необходимым условием сходимости.

* 1. Пример 1 (Метод Якоби)





* 1. Пример 2 (Метод Зейделя)



Вывод: Диагональное преобладание не является необходимым условием сходимости.