Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Лабораторная работа №2
по дисциплине «Численные методы»

Итерационные методы решения СЛАУ

Факультет: ПМИ

Группа: ПМ-63

Студент: Шепрут И.И.

Вариант: 11

Преподаватель: Задорожный А.Г.

Новосибирск
2018

1 Цель работы

Разработать программы решения СЛАУ методами Якоби, Гаусса-Зейделя с хранением матрицы в диагональном формате. Исследовать сходимость методов для различных тестовых матриц и её зависимость от параметра релаксации. Изучить возможность оценки порядка числа обусловленности матрицы путем вычислительного эксперимента.

Вариант 11: 7-ми диагональная матрица с параметрами m, k — количество нулевых диагоналей, n — размерность матрицы.

2 Код программы

Программа состоит из нескольких частей:

1. `common.h + common.cpp` — пара общих функций и объявление вещественных типов.
2. `matrix.h + matrix.cpp` — модуль для работы с матрицами в плотном формате.
3. `diagonal.h + diagonal.cpp` — модуль для работы с матрицами в диагональном формате.
4. `table_generator.cpp` — программа, которая генерирует таблицы.

common.h

```
#pragma once

#ifdef ALL_FLOAT
typedef float real;
typedef float sumreal;
#endif

#ifdef ALL_DOUBLE
typedef double real;
typedef double sumreal;
#endif

#ifdef ALL_FLOAT_WITH_DOUBLE
typedef float real;
typedef double sumreal;
#endif

#ifndef ALL_FLOAT
#ifndef ALL_DOUBLE
#ifndef ALL_FLOAT_WITH_DOUBLE
#error "Type isn't defined"
typedef double real;
typedef double sumreal;
#endif
#endif
#endif


bool isNear(double a, double b);
double random(void);
int intRandom(int min, int max);
```

matrix.h

```
#pragma once

#include <string>
#include <vector>
#include "common.h"
```

```cpp

//---------------------------------------------------------------------------
class Matrix
{
public:
    Matrix(int n = 0, int m = 0, real fill = 0); // n - количество столбцов, m -
    ↪ количество строк
    void loadFromFile(std::string fileName);
    void saveToFile(std::string fileName) const;

    void getFromVector(int n, int m, const std::vector<real>& data);

    void resize(int n, int m, real fill = 0);
    void negate(void);

    bool isSymmetric(void) const;
    bool isLowerTriangular(void) const;
    bool isUpperTriangular(void) const;
    bool isDiagonal(void) const;
    bool isDiagonalIdentity(void) const;
    bool isDegenerate(void) const;

    real& operator()(int i, int j);
    const real& operator()(int i, int j) const;

    int width(void) const;
    int height(void) const;

private:
    std::vector<std::vector<real>> m_matrix;
    int m_n, m_m;
};

//---------------------------------------------------------------------------
void generateSparseSymmetricMatrix(
    int n,
    int min, int max,
    real percent,
    Matrix& result
);

void generateLMatrix(
    int n,
    int min, int max,
    real percent,
    Matrix& result
);

void generateDiagonalMatrix(
    int n,
    int min, int max,
    Matrix& result
);

void generateVector(
    int n,
    int min, int max,
    Matrix& result
);
```

```
void generateVector(
    int n,
    Matrix& result
);

void generateGilbertMatrix(int n, Matrix& result);

void generateTestMatrix(int n, int profileSize, Matrix& result);

//-----------------------------------------------------------------------------
bool mul(const Matrix& a, const Matrix& b, Matrix& result);
bool sum(const Matrix& a, const Matrix& b, Matrix& result);

sumreal sumAllElementsAbs(const Matrix& a);

bool transpose(Matrix& a);

bool calcLDL(const Matrix& a, Matrix& l, Matrix& d);
bool calcGaussianReverseOrder(const Matrix& l, const Matrix& y, Matrix& x);
bool calcGaussianFrontOrder(const Matrix& l, const Matrix& y, Matrix& x);
bool calcGaussianCentralOrder(const Matrix& d, const Matrix& y, Matrix& x);
bool solveSLAE_by_LDL(const Matrix& a, const Matrix& y, Matrix& x);

bool solveSLAE_byGaussMethod(const Matrix& a, const Matrix& y, Matrix& x);
```

FILE **diagonal.h**

```
#pragma once

#include <string>
#include <vector>
#include <iostream>
#include <map>
#include <functional>
#include "../1/common.h"
#include "../1/vector.h"
#include "../1/matrix.h"

class MatrixDiagonal;
class matrix_diagonal_iterator;
class SolverSLAE_Iterative;

//-----------------------------------------------------------------------------
/** Класс для вычислений различных параметров с диагональными матрицами. */
class Diagonal
{
public:
    int n;

    //-------------------------------------------------------------------------
    Diagonal(int n);

    int calcDiagonalsCount(void);
    int calcMinDiagonal(void);
    int calcMaxDiagonal(void);
    int calcDiagonalSize(int d);

    bool isLineIntersectDiagonal(int line, int d);
    bool isRowIntersectDiagonal(int row, int d);
```

```cpp

    //-----------------------------------------------------------------------
    /*
        R - Row - столбец
        L - Line - строка
        P - Pos - номер элемента в диагонали
        D - Diag - формат диагонали
    */

    int calcLine_byDP(int d, int pos);
    int calcRow_byDP(int d, int pos);

    int calcDiag_byLR(int line, int row);
    int calcPos_byLR(int line, int row);

    int calcPos_byDL(int d, int line);
    int calcPos_byDR(int d, int row);

    int calcRow_byDL(int d, int line);
    int calcLine_byDR(int d, int row);
};

//-----------------------------------------------------------------------
/** Матрица в диагональном формате. */
/** 0-я диагональ всегда главная диагональ. */
class MatrixDiagonal
{
public:
    typedef std::vector<real>::iterator iterator;
    typedef std::vector<real>::const_iterator const_iterator;

    //-----------------------------------------------------------------------
    MatrixDiagonal();
    MatrixDiagonal(int n, std::vector<int> format); // format[0] must be 0,
    ↪  because it's main diagonal
    MatrixDiagonal(const Matrix& a);

    void toDenseMatrix(Matrix& dense) const;
    void resize(int n, std::vector<int> format); // format[0] must be 0, because
    ↪  it's main diagonal

    //-----------------------------------------------------------------------
    int dimension(void) const;
    int getDiagonalsCount(void) const;
    int getDiagonalSize(int diagNo) const;
    int getDiagonalPos(int diagNo) const;

    std::vector<int> getFormat(void) const;

    //-----------------------------------------------------------------------
    matrix_diagonal_iterator posBegin(int diagNo) const;
    matrix_diagonal_iterator posEnd(int diagNo) const;

    iterator begin(int diagNo);
    const_iterator begin(int diagNo) const;

    iterator end(int diagNo);
    const_iterator end(int diagNo) const;

```

```
private:
    std::vector<std::vector<real>> di;
    std::vector<int> fi;
    int n;
    Diagonal dc;
};

std::vector<int> makeSevenDiagonalFormat(int n, int m, int k);
std::vector<int> generateRandomFormat(int n, int diagonalsCount);

void generateDiagonalMatrix(
    int n,
    int min, int max,
    std::vector<int> format,
    MatrixDiagonal& result
);

void generateDiagonallyDominantMatrix(
    int n,
    std::vector<int> format,
    bool isNegative,
    MatrixDiagonal& result
);

bool mul(const MatrixDiagonal& a, const Vector& x, Vector& y);

//-------------------------------------------------------------------------------
/** Матричный ”итератор” для движения по диагонали. */
class matrix_diagonal_iterator
{
public:
    matrix_diagonal_iterator(int n, int d, bool isEnd);

    matrix_diagonal_iterator& operator++();
    matrix_diagonal_iterator  operator++(int);

    bool operator==(const matrix_diagonal_iterator& b) const;
    bool operator!=(const matrix_diagonal_iterator& b) const;

    matrix_diagonal_iterator& operator+=(const ptrdiff_t& movement);

    int i, j;
};

/** Матричный ”итератор” для движения по строке между различными диагоналями.
 ↪ Может обрабатывать как всю строку, так и только нижний треугольник. */
class matrix_diagonal_line_iterator
{
public:
    matrix_diagonal_line_iterator(int n, std::vector<int> format, bool
     ↪ isOnlyLowTriangle);

    matrix_diagonal_line_iterator& operator++();
    matrix_diagonal_line_iterator  operator++(int);

    bool isLineEnd(void) const;
    bool isEnd(void) const;

    int i, j; // i - текущая строка,  j - текущий столбец
```

```cpp
    int d, dn, di; // d - формат текущей диагонали, d - номер текущей диагонали, di - номер текущего элемента в диагонали
private:
    std::map<int, int> m_map;
    std::vector<int> m_sorted_format;

    int line, start, end, pos;
    Diagonal dc;

    bool m_isLineEnd;
    bool m_isEnd;

    void calcPos(void);
};

//-----------------------------------------------------------------------------
/** Класс итеративного решателя СЛАУ для диагональной матрицы. */
struct IterationsResult
{
    int iterations;
    double relativeResidual;
};

class SolverSLAE_Iterative
{
public:
    SolverSLAE_Iterative();

    IterationsResult jacobi(const MatrixDiagonal& a, const Vector& y, Vector& x) const;
    IterationsResult seidel(const MatrixDiagonal& a, const Vector& y, Vector& x) const;

    double              w;
    bool               isLog;
    std::ostream&      log;
    Vector              start;
    double              epsilon;
    int               maxIterations;

private:
    mutable Vector x1;

    // До итерации: x - текущее решение. После итерации x - следующее решение.
    void iteration_jacobi(const MatrixDiagonal& a, const Vector& y, Vector& x) const;
    void iteration_seidel(const MatrixDiagonal& a, const Vector& y, Vector& x) const;

    typedef std::function<void(const SolverSLAE_Iterative*, const MatrixDiagonal&, const Vector&, Vector&)> step_function;

    IterationsResult iteration_process(
        const MatrixDiagonal& a,
        const Vector& y,
        Vector& x,
        step_function step
    ) const;
};
```

```cpp

//----------------------------------------------------------------------------
/** Класс итеративного решателя СЛАУ для плотной матрицы. */
class SolverSLAE_Iterative_matrix
{
public:
    SolverSLAE_Iterative_matrix();

    IterationsResult jacobi(const Matrix& a, const Vector& y, Vector& x) const;
    IterationsResult seidel(const Matrix& a, const Vector& y, Vector& x) const;

    double             w;
    bool              isLog;
    std::ostream&      log;
    Vector             start;
    double             epsilon;
    int              maxIterations;
private:
    mutable Vector x1;

    // До итерации: x - текущее решение. После итерации x - следующее решение.
    void iteration_jacobi(const Matrix& a, const Vector& y, Vector& x) const;
    void iteration_seidel(const Matrix& a, const Vector& y, Vector& x) const;

    typedef std::function<void(const SolverSLAE_Iterative_matrix*, const
    ↪ Matrix&, const Vector&, Vector&)> step_function;

    IterationsResult iteration_process(
        const Matrix& a,
        const Vector& y,
        Vector& x,
        step_function step
    ) const;
};
```

FILE common.cpp

```cpp
#include <cmath>
#include "common.h"

//----------------------------------------------------------------------------
bool isNear(double a, double b) {
    if (a != 0) {
        if (fabs(a - b)/a > 0.0001)
            return false;
    } else {
        if (fabs(b) > 0.0001)
            return false;
    }

    return true;
}

//----------------------------------------------------------------------------
double random(void) {
    return std::rand() / double(RAND_MAX);
}

//----------------------------------------------------------------------------
int intRandom(int min, int max) {
```

```
    return min + random() * (max - min);
}
```

matrix.cpp

```
#include <fstream>
#include <iomanip>
#include ”matrix.h”

//---------------------------------------------------------------------------
Matrix::Matrix(int n, int m, real fill) : m_matrix(m, std::vector<real>(n, fill)), m_n(n), m_m(m) {
}

//---------------------------------------------------------------------------
void Matrix::loadFromFile(std::string fileName) {
    std::ifstream fin(fileName);

    m_matrix.clear();
    int n, m;
    fin >> n >> m;
    resize(n, m);
    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j < width(); ++j) {
            fin >> operator()(i, j);
        }
    }

    fin.close();
}

//---------------------------------------------------------------------------
void Matrix::saveToFile(std::string fileName) const {
    std::ofstream fout(fileName);

    fout << m_n << ”\t” << m_m << std::endl;

    fout.precision(std::numeric_limits<real>::digits10);
    int w = std::numeric_limits<real>::digits10 + 4;

    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j < width(); ++j)
            fout << ”\t” << operator()(i, j);
        fout << std::endl;
    }

    fout.close();
}

//---------------------------------------------------------------------------
void Matrix::getFromVector(int n, int m, const std::vector<real>& data) {
    resize(n, m);
    for (int i = 0; i < data.size(); ++i)
        operator()(i / n, i % n) = data[i];
}

//---------------------------------------------------------------------------
void Matrix::resize(int n, int m, real fill) {
    if (m_n != n || m_m != m) {
        m_n = n;
```

```
        m_m = m;
        m_matrix.clear();
        m_matrix.resize(m_m, std::vector<real>(m_n, fill));
    }
}

//------------------------------------------------------------------------------
void Matrix::negate(void) {
    for (auto& i : m_matrix) {
        for (auto& j : i) {
            j = -j;
        }
    }
}

//------------------------------------------------------------------------------
bool Matrix::isSymmetric(void) const {
    if (height() != width())
        return false;

    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j <= i; ++j) {
            const real& a = operator()(i, j);
            const real& b = operator()(j, i);
            if (!isNear(a, b))
                return false;
        }
    }

    return true;
}

//------------------------------------------------------------------------------
bool Matrix::isLowerTriangular(void) const {
    if (height() != width())
        return false;

    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j < i; ++j) {
            if (fabs(operator()(j, i)) > 0.000001)
                return false;
        }
    }

    return true;
}

//------------------------------------------------------------------------------
bool Matrix::isUpperTriangular(void) const {
    if (height() != width())
        return false;

    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j < i; ++j) {
            if (operator()(i, j) != 0)
                return false;
        }
    }

```

```
    return true;
}

//---------------------------------------------------------------------------
bool Matrix::isDiagonal(void) const {
    if (height() != width())
        return false;

    for (int i = 0; i < height(); ++i) {
        for (int j = 0; j < i; ++j) {
            if (operator()(j, i) != 0 && operator()(i, j) != 0)
                return false;
        }
    }

    return true;
}

//---------------------------------------------------------------------------
bool Matrix::isDiagonalIdentity(void) const {
    if (height() != width())
        return false;

    for (int i = 0; i < height(); ++i) {
        if (operator()(i, i) != 1)
            return false;
    }

    return true;
}

//---------------------------------------------------------------------------
bool Matrix::isDegenerate(void) const {
    // TODO
    return false;
}

//---------------------------------------------------------------------------
real& Matrix::operator()(int i, int j) {
    return m_matrix[i][j];
}

//---------------------------------------------------------------------------
const real& Matrix::operator()(int i, int j) const {
    return m_matrix[i][j];
}

//---------------------------------------------------------------------------
int Matrix::width(void) const {
    return m_n;
}

//---------------------------------------------------------------------------
int Matrix::height(void) const {
    return m_m;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

```
//--------------------------------------------------------------------

//--------------------------------------------------------------------
void generateSparseSymmetricMatrix(int n, int min, int max, real percent,
↪   Matrix& result) {
    result.resize(n, n, 0);

    int count = percent * n * n;

    for (int k = 0; k < count; ++k) {
        int i = intRandom(0, n-1);
        int j = intRandom(0, i-1);
        result(i, j) = intRandom(min, max);
        result(j, i) = result(i, j);
    }

    for (int i = 0; i < n; i++)
        result(i, i) = intRandom(1, max - min);
}

//--------------------------------------------------------------------
void generateLMatrix(int n, int min, int max, real percent, Matrix& result) {
    result.resize(n, n, 0);

    int count = percent * n * n / 2;

    for (int k = 0; k < count; ++k) {
        int i = intRandom(0, n-1);
        int j = intRandom(0, i-1);
        result(i, j) = intRandom(min, max);
    }

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

//--------------------------------------------------------------------
void generateDiagonalMatrix(int n, int min, int max, Matrix& result) {
    result.resize(n, n, 0);

    for (int i = 0; i < n; ++i)
        result(i, i) = intRandom(min, max);
}

//--------------------------------------------------------------------
void generateVector(int n, int min, int max, Matrix& result) {
    result.resize(1, n, 0);

    for (int i = 0; i < n; ++i)
        result(i, 0) = intRandom(min, max);
}

//--------------------------------------------------------------------
void generateVector(int n, Matrix& result) {
    result.resize(1, n, 0);

    for (int i = 0; i < n; ++i)
        result(i, 0) = i+1;
```

```
}

//-------------------------------------------------------------------------------
void generateGilbertMatrix(int n, Matrix& result) {
    result.resize(n, n);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            result(i, j) = double(1.0)/double((i+1)+(j+1)-1);
        }
    }
}

//-------------------------------------------------------------------------------
void generateTestMatrix(int n, int profileSize, Matrix& result) {
    result.resize(n, n);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < profileSize; ++j) if (i-j-1 >= 0) {
            result(i, i-j-1) = -intRandom(0, 5);
            result(i-j-1, i) = result(i, i-j-1);
        }
    }

    for (int i = 0; i < n; ++i) {
        sumreal sum = 0;
        for (int j = 0; j < n; ++j) if (i != j) {
            sum += result(i, j);
        }
        result(i, i) = -sum;
    }
}

//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
bool mul(const Matrix& a, const Matrix& b, Matrix& result) {
    // result = rus_a * b
    if (a.width() != b.height())
        return false;

    result.resize(b.width(), a.height());

    for (int i = 0; i < b.width(); ++i) {
        for (int j = 0; j < a.height(); ++j) {
            real sum = 0;
            for (int k = 0; k < a.width(); ++k) {
                sum += a(j, k) * b(k, i);
            }
            result(j, i) = sum;
        }
    }

    return true;
}

//-------------------------------------------------------------------------------
```

```
bool sum(const Matrix& a, const Matrix& b, Matrix& result) {
    // result = rus_a + b
    if (a.width() != b.width() || a.height() != b.height())
        return false;

    result.resize(a.width(), a.height());

    for (int i = 0; i < a.width(); ++i) {
        for (int j = 0; j < a.height(); ++j) {
            result(j, i) = a(j, i) + b(j, i);
        }
    }

    return true;
}

//------------------------------------------------------------------------------
bool transpose(Matrix& a) {
    // rus_a = rus_a^T
    if (a.height() != a.width())
        return false;

    for (int i = 0; i < a.height(); ++i) {
        for (int j = 0; j < i; ++j) {
            std::swap(a(j, i), a(i, j));
        }
    }

    return true;
}

//------------------------------------------------------------------------------
sumreal sumAllElementsAbs(const Matrix& a) {
    sumreal sum = 0;
    for (int i = 0; i < a.height(); ++i) {
        for (int j = 0; j < a.width(); ++j) {
            sum += fabs(a(i, j));
        }
    }

    return sum;
}

//------------------------------------------------------------------------------
bool calcLDL(const Matrix& a, Matrix& l, Matrix& d) {
    // l * d * l^T = rus_a
    if (!a.isSymmetric())
        return false;

    l.resize(a.width(), a.height(), 0);
    d.resize(a.width(), a.height(), 0);

    for (int i = 0; i < a.height(); ++i) {
        // Считаем элементы матрицы L
        for (int j = 0; j < i; ++j) {
            real sum = 0;
            for (int k = 0; k < j; ++k)
                sum += d(k, k) * l(j, k) * l(i, k);

```

```
            if (fabs(d(j, j)) < 0.0001)
                l(i, j) = 0;
            else
                l(i, j) = (a(i, j) - sum) / d(j, j);
        }

        // Считаем диагональный элемент
        {
            real sum = 0;
            for (int j = 0; j < i; ++j)
                sum += d(j, j) * l(i, j) * l(i, j);
            d(i, i) = a(i, i) - sum;
        }
    }

    for (int i = 0; i < l.height(); i++)
        l(i, i) = 1;

    return true;
}

//----------------------------------------------------------------------------
bool calcGaussianReverseOrder(const Matrix& l, const Matrix& y, Matrix& x) {
    // l * x = y, l - нижнетреугольная матрица
    if (!l.isLowerTriangular() || !l.isDiagonalIdentity())
        return false;

    x.resize(1, y.height());

    for (int i = x.height() - 1; i >= 0; --i) {
        real sum = 0;
        for (int j = i; j < x.height(); ++j)
            sum += l(j, i) * x(j, 0);
        x(i, 0) = y(i, 0) - sum;
    }

    return true;
}

//----------------------------------------------------------------------------
bool calcGaussianFrontOrder(const Matrix& l, const Matrix& y, Matrix& x) {
    // l * x = y, l - верхнетреугольная матрица
    if (!l.isLowerTriangular() || !l.isDiagonalIdentity())
        return false;

    x.resize(1, y.height());

    for (int i = 0; i < x.height(); ++i) {
        real sum = 0;
        for (int j = 0; j < i; ++j)
            sum += l(i, j) * x(j, 0);
        x(i, 0) = y(i, 0) - sum;
    }

    return true;
}

//----------------------------------------------------------------------------
bool calcGaussianCentralOrder(const Matrix& d, const Matrix& y, Matrix& x) {
```

```cpp
    // d * x = y, d - диагональная матрица
    if (!d.isDiagonal())
        return false;

    x.resize(1, y.height());

    for (int i = 0; i < x.height(); ++i)
        x(i, 0) = y(i, 0) / d(i, i);

    return true;
}

//--------------------------------------------------------------------------------
bool solveSLAE_by_LDL(const Matrix& a, const Matrix& y, Matrix& x) {
    // rus_a * x = y, rus_a - симметричная матрица
    if (!(a.width() == a.height() && a.width() == y.height() &&
    ↪  !a.isDegenerate()))
        return false;

    Matrix l, d, z, w;

    if (!calcLDL(a, l, d))
        return false;

    if (!calcGaussianFrontOrder(l, y, z))
        return false;

    if (!calcGaussianCentralOrder(d, z, w))
        return false;

    if (!calcGaussianReverseOrder(l, w, x))
        return false;

    return true;
}

//--------------------------------------------------------------------------------
bool solveSLAE_byGaussMethod(const Matrix& a1, const Matrix& y1, Matrix& x1) {
    if (!(a1.width() == a1.height() && y1.height() == a1.width() &&
    ↪  !a1.isDegenerate()))
        return false;

    Matrix a(a1);
    Matrix y(y1);

    for (int i = 0; i < a.height(); ++i) {
        // Находим максимальный элемент
        int maxI = i;
        for (int j = i+1; j < a.height(); ++j)
            if (fabs(a(j, i)) > fabs(a(maxI, i)))
                maxI = j;

        // Переставляем эту строчку с текущей
        for (int j = i; j < a.width(); ++j)
            std::swap(a(i, j), a(maxI, j));
        std::swap(y(i, 0), y(maxI, 0));

        // Перебираем все строчки ниже и отнимаем текущую строчку от них
        for (int j = i+1; j < a.height(); ++j) {
```

```
            real m = a(j, i) / a(i, i);
            for (int k = i; k < a.width(); ++k)
                a(j, k) -= m * a(i, k);
            y(j, 0) -= m * y(i, 0);
        }

        // Делим текущую строку на ее ведущий элемент, чтобы на диагонали были
        ↪  единицы
        double m = a(i, i);
        for (int j = i; j < a.width(); ++j)
            a(i, j) /= m;
        y(i, 0) /= m;
    }

    // Считаем обратный ход Гаусса
    transpose(a);
    calcGaussianReverseOrder(a, y, x1);

    return true;
}
```

FILE diagonal.cpp

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "diagonal.h"

//------------------------------------------------------------------------------
Diagonal::Diagonal(int n) : n(n) {
}

//------------------------------------------------------------------------------
int Diagonal::calcDiagonalsCount(void) {
    return 2 * n - 1;
}

//------------------------------------------------------------------------------
int Diagonal::calcMinDiagonal(void) {
    return -(n-1);
}

//------------------------------------------------------------------------------
int Diagonal::calcMaxDiagonal(void) {
    return n - 1;
}

//------------------------------------------------------------------------------
int Diagonal::calcDiagonalSize(int d) {
    return n - std::abs(d);
}

//------------------------------------------------------------------------------
bool Diagonal::isLineIntersectDiagonal(int line, int d) {
    if (d <= 0)
        return (line+d >= 0);

    if (d > 0)
        return (line < calcDiagonalSize(d));
```

```
}

//----------------------------------------------------------------------------
bool Diagonal::isRowIntersectDiagonal(int row, int d) {
    return isLineIntersectDiagonal(row, -d);
}

//----------------------------------------------------------------------------
int Diagonal::calcLine_byDP(int d, int pos) {
    if (d <= 0)
        return -d + pos;

    if (d > 0)
        return pos;
}

//----------------------------------------------------------------------------
int Diagonal::calcRow_byDP(int d, int pos) {
    if (d <= 0)
        return pos;

    if (d > 0)
        return pos + d;
}

//----------------------------------------------------------------------------
int Diagonal::calcDiag_byLR(int line, int row) {
    return row - line;
}

//----------------------------------------------------------------------------
int Diagonal::calcPos_byLR(int line, int row) {
    return calcPos_byDL(calcDiag_byLR(line, row), line);
}

//----------------------------------------------------------------------------
int Diagonal::calcPos_byDL(int d, int line) {
    if (d <= 0)
        return line+d;

    if (d > 0)
        return line;
}

//----------------------------------------------------------------------------
int Diagonal::calcPos_byDR(int d, int row) {
    return calcPos_byDL(d, calcLine_byDR(d, row));
}

//----------------------------------------------------------------------------
int Diagonal::calcRow_byDL(int d, int line) {
    return line+d;
}

//----------------------------------------------------------------------------
int Diagonal::calcLine_byDR(int d, int row) {
    return calcRow_byDL(-d, row);
}

```

```
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
MatrixDiagonal::MatrixDiagonal() : n(0), dc(n) {
}

//------------------------------------------------------------------------------
MatrixDiagonal::MatrixDiagonal(int n, std::vector<int> format) : dc(n) {
    resize(n, format);
}

//------------------------------------------------------------------------------
MatrixDiagonal::MatrixDiagonal(const Matrix& a) : dc(n) {
    if (a.width() != a.height())
        throw std::exception();

    n = a.width();
    dc.n = n;

    // Определяем формат
    std::vector<int> format;
    format.clear();
    format.push_back(0);
    for (int i = dc.calcMinDiagonal(); i <= dc.calcMaxDiagonal(); ++i)
        if (i != 0) {
            auto mit = matrix_diagonal_iterator(n, i, false);
            auto mite = matrix_diagonal_iterator(n, i, true);
            for (; mit != mite; ++mit) {
                if (a(mit.i, mit.j) != 0) {
                    format.push_back(i);
                    break;
                }
            }
        }

    // Создаем формат
    resize(n, format);

    // Обходим массив и записываем элементы
    for (int i = 0; i < getDiagonalsCount(); ++i) {
        auto mit = posBegin(i);
        for (auto it = begin(i); it != end(i); ++it, ++mit)
            *it = a(mit.i, mit.j);
    }
}

//------------------------------------------------------------------------------
void MatrixDiagonal::toDenseMatrix(Matrix& dense) const {
    dense.resize(n, n, 0);

    // Обходим массив и записываем элементы
    for (int i = 0; i < getDiagonalsCount(); ++i) {
        auto mit = posBegin(i);
        for (auto it = begin(i); it != end(i); ++it, ++mit)
            dense(mit.i, mit.j) = *it;
    }
}
```

```cpp

//---------------------------------------------------------------------------
void MatrixDiagonal::resize(int n1, std::vector<int> format) {
    if (format[0] != 0)
        throw std::exception();

    dc.n = n1;
    n = n1;
    fi = format;

    di.clear();
    for (const auto& i : format)
        di.push_back(std::vector<real>(dc.calcDiagonalSize(i), 0));
}

//---------------------------------------------------------------------------
int MatrixDiagonal::dimension(void) const {
    return n;
}

//---------------------------------------------------------------------------
int MatrixDiagonal::getDiagonalsCount(void) const {
    return di.size();
}

//---------------------------------------------------------------------------
int MatrixDiagonal::getDiagonalSize(int diagNo) const {
    return di[diagNo].size();
}

//---------------------------------------------------------------------------
int MatrixDiagonal::getDiagonalPos(int diagNo) const {
    return fi[diagNo];
}

//---------------------------------------------------------------------------
std::vector<int> MatrixDiagonal::getFormat(void) const {
    return fi;
}

//---------------------------------------------------------------------------
matrix_diagonal_iterator MatrixDiagonal::posBegin(int diagNo) const {
    return matrix_diagonal_iterator(n, fi[diagNo], false);
}

//---------------------------------------------------------------------------
matrix_diagonal_iterator MatrixDiagonal::posEnd(int diagNo) const {
    return matrix_diagonal_iterator(n, fi[diagNo], true);
}

//---------------------------------------------------------------------------
MatrixDiagonal::iterator MatrixDiagonal::begin(int diagNo) {
    return di[diagNo].begin();
}

//---------------------------------------------------------------------------
MatrixDiagonal::const_iterator MatrixDiagonal::begin(int diagNo) const {
    return di[diagNo].begin();
}
```

```cpp

//---------------------------------------------------------------------------
MatrixDiagonal::iterator MatrixDiagonal::end(int diagNo) {
    return di[diagNo].end();
}

//---------------------------------------------------------------------------
MatrixDiagonal::const_iterator MatrixDiagonal::end(int diagNo) const {
    return di[diagNo].end();
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
matrix_diagonal_iterator::matrix_diagonal_iterator(int n, int d, bool isEnd) {
    Diagonal dc(n);
    if (isEnd) {
        i = dc.calcLine_byDP(d, dc.calcDiagonalSize(d));
        j = dc.calcRow_byDP(d, dc.calcDiagonalSize(d));
    } else {
        i = dc.calcLine_byDP(d, 0);
        j = dc.calcRow_byDP(d, 0);
    }
}

//---------------------------------------------------------------------------
matrix_diagonal_iterator& matrix_diagonal_iterator::operator++() {
    i++;
    j++;
    return *this;
}

//---------------------------------------------------------------------------
matrix_diagonal_iterator matrix_diagonal_iterator::operator++(int) {
    i++;
    j++;
    return *this;
}

//---------------------------------------------------------------------------
bool matrix_diagonal_iterator::operator==(const matrix_diagonal_iterator& b) const {
    return b.i == i && b.j == j;
}

//---------------------------------------------------------------------------
bool matrix_diagonal_iterator::operator!=(const matrix_diagonal_iterator& b) const {
    return b.i != i || b.j != j;
}

//---------------------------------------------------------------------------
matrix_diagonal_iterator& matrix_diagonal_iterator::operator+=(const ptrdiff_t& movement) {
    i += movement;
    j += movement;
    return *this;
```

```
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
matrix_diagonal_line_iterator::matrix_diagonal_line_iterator(int n, std::vector<int> format, bool isOnlyLowTriangle) : dc(n), m_isEnd(false), m_isLineEnd(false) {
    // Создаем обратное преобразование из формата диагонали в ее номер в формате
    for (int i = 0; i < format.size(); ++i)
        if ((isOnlyLowTriangle && format[i] < 0) || !isOnlyLowTriangle)
            m_map[format[i]] = i;

    // Создаем сортированный формат, чтобы по нему двигаться
    if (isOnlyLowTriangle) {
        for (int i = 0; i < format.size(); ++i)
            if (format[i] < 0)
                m_sorted_format.push_back(format[i]);
    } else
        m_sorted_format = format;
    std::sort(m_sorted_format.begin(), m_sorted_format.end());

    line = 0;
    pos = 0;
    start = m_sorted_format.size() - 1;
    end = start;

    // Находим, с какой диагонали начинается текущая строка
    for (int i = 0; i < m_sorted_format.size(); ++i) {
        if (dc.isLineIntersectDiagonal(line, m_sorted_format[i])) {
            start = i;
            break;
        }
    }

    // Находим на какой диагонали кончается текущая строка
    for (int i = 0; i < m_sorted_format.size(); ++i) {
        int j = m_sorted_format.size() - i - 1;
        if (dc.isLineIntersectDiagonal(line, m_sorted_format[j])) {
            end = j;
            break;
        }
    }

    calcPos();
}

//---------------------------------------------------------------------------
matrix_diagonal_line_iterator& matrix_diagonal_line_iterator::operator++() {
    if (!m_isEnd) {
        if (m_isLineEnd) {
            // Сдвигаемся на одну строку
            line++;

            // Определяем какие диагонали пересекают эту строку
            if (start != 0)
                if (dc.isLineIntersectDiagonal(line, m_sorted_format[start-1]))
```

```
                    start = start-1;

            if (end != 0)
                if (!dc.isLineIntersectDiagonal(line, m_sorted_format[end]))
                    if (start != end)
                        end = end-1;

            m_isLineEnd = false;
            if (line == dc.n)
                m_isEnd = true;

            pos = 0;
            calcPos();
        } else {
            // Сдвигаемся на один столбец
            pos++;
            calcPos();
        }
    }

    return *this;
}

//-------------------------------------------------------------------------------
matrix_diagonal_line_iterator matrix_diagonal_line_iterator::operator++(int) {
    return operator++();
}

//-------------------------------------------------------------------------------
bool matrix_diagonal_line_iterator::isLineEnd(void) const {
    return m_isLineEnd;
}

//-------------------------------------------------------------------------------
bool matrix_diagonal_line_iterator::isEnd(void) const {
    return m_isEnd;
}

//-------------------------------------------------------------------------------
void matrix_diagonal_line_iterator::calcPos(void) {
    // Вычисляет все текущие положения согласно переменным start, pos и формату
    if (!dc.isLineIntersectDiagonal(line, m_sorted_format[end]) || (start + pos
    > end)) {
        m_isLineEnd = true;
        i = line;
        j = 0;
        d = 0;
        di = 0;
        dn = 0;
    } else {
        i = line;
        d = m_sorted_format[start + pos];
        dn = m_map[d];
        di = dc.calcPos_byDL(d, i);
        j = dc.calcRow_byDL(d, i);
    }
}

//-------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
std::vector<int> makeSevenDiagonalFormat(int n, int m, int k) {
    std::vector<int> result;

    if (1+m+k >= n)
        throw std::exception();

    result.push_back(0);

    result.push_back(1);
    result.push_back(1+m);
    result.push_back(1+m+k);

    result.push_back(-1);
    result.push_back(-1-m);
    result.push_back(-1-m-k);

    return result;
}

//-----------------------------------------------------------------------------
std::vector<int> generateRandomFormat(int n, int diagonalsCount) {
    Diagonal d(n);

    std::vector<int> result;
    result.push_back(0);

    // Создаем массив всех возможных диагоналей
    std::vector<int> diagonals;
    for (int i = d.calcMinDiagonal(); i <= d.calcMaxDiagonal(); ++i)
        if (i != 0)
            diagonals.push_back(i);

    diagonalsCount = std::min<int>(diagonals.size(), diagonalsCount);

    // Заполняем результат случайными диагоналями из этого массива
    for (int i = 0; i < diagonalsCount; ++i) {
        int pos = intRandom(0, diagonals.size());
        result.push_back(diagonals[pos]);
        diagonals.erase(diagonals.begin() + pos);
    }

    return result;
}

//-----------------------------------------------------------------------------
void generateDiagonalMatrix(int n, int min, int max, std::vector<int> format,
↪   MatrixDiagonal& result) {
    result.resize(n, format);
    for (int i = 0; i < result.getDiagonalsCount(); ++i) {
        auto mit = result.posBegin(i);
        for (auto it = result.begin(i); it != result.end(i); ++it, ++mit)
            (*it) = intRandom(min, max);
    }
}

```

```
//---------------------------------------------------------------------------
void generateDiagonallyDominantMatrix(int n, std::vector<int> format, bool
↪ isNegative,  MatrixDiagonal& result) {
    result.resize(n, format);

    for (int i = 0; i < result.getDiagonalsCount(); ++i) {
        auto mit = result.posBegin(i);
        for (auto it = result.begin(i); it != result.end(i); ++it, ++mit) {
            if (isNegative)
                *it = -intRandom(0, 5);
            else
                *it = intRandom(0, 5);
        }
    }

    matrix_diagonal_line_iterator mit(n, format, false);
    for (; !mit.isEnd(); ++mit) {
        sumreal& sum = result.begin(0)[mit.i];
        sum = 0;
        for (; !mit.isLineEnd(); ++mit)
            if (mit.i != mit.j)
                sum += result.begin(mit.dn)[mit.di];
        sum = std::fabs(sum);
    }

    result.begin(0)[0] += 1;
}


//---------------------------------------------------------------------------
bool mul(const MatrixDiagonal& a, const Vector& x, Vector& y) {
    if (x.size() != a.dimension())
        return false;

    y.resize(x.size());

    // Зануление результата
    y.zero();
    for (int i = 0; i < a.getDiagonalsCount(); ++i) {
        auto mit = a.posBegin(i);
        for (auto it = a.begin(i); it != a.end(i); ++it, ++mit)
            y(mit.i) += (*it) * x(mit.j);
    }

    return true;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
SolverSLAE_Iterative::SolverSLAE_Iterative() :
    w(1),
    isLog(false),
    log(std::cout),
    start(),
    epsilon(0.00001),
    maxIterations(100) {
```

```
}

//--------------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative::jacobi(const MatrixDiagonal& a, const
↪ Vector& y, Vector& x) const {
    return iteration_process(a, y, x, &SolverSLAE_Iterative::iteration_jacobi);
}

//--------------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative::seidel(const MatrixDiagonal& a, const
↪ Vector& y, Vector& x) const {
    return iteration_process(a, y, x, &SolverSLAE_Iterative::iteration_seidel);
}

//--------------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative::iteration_process(const MatrixDiagonal&
↪ a, const Vector& y, Vector& x, step_function step) const {
    if (a.dimension() != y.size() || start.size() != y.size())
        throw std::exception();

    // Считаем норму матрицы: ее максимальный элемент по модулю
    real yNorm = calcNorm(y);
    x1.resize(y.size());
    x = start;

    // Цикл по итерациям
    int i = 0;
    real relativeResidual = epsilon + 1;
    for (; i < maxIterations && relativeResidual > epsilon; ++i) {
        // Итерационный шаг
        step(this, a, y, x);

        // Считаем невязку
        mul(a, x, x1);
        x1.negate();
        sum(x1, y, x1);
        relativeResidual = fabs(calcNorm(x1)) / yNorm;

        // Выводим данные
        if (isLog)
            log << i << "\t" << std::scientific << std::setprecision(3) <<
            ↪ relativeResidual << std::endl;
    }

    return {i, relativeResidual};
}

//--------------------------------------------------------------------------------
void SolverSLAE_Iterative::iteration_jacobi(const MatrixDiagonal& a, const
↪ Vector& y, Vector& x) const {
    // Умножаем матрицу на решение
    mul(a, x, x1);

    // x^(k+1) = x^k + w/a(i, i) * x^(k+1)
    auto it = a.begin(0);
    for (int i = 0; i < x1.size(); ++i, ++it)
        x(i) += w / (*it) * (y(i)-x1(i));
}

```

```
//-----------------------------------------------------------------------------
void SolverSLAE_Iterative::iteration_seidel(const MatrixDiagonal& a, const Vector& y, Vector& x) const {
    // Умножем верхний треугольник на решение
    x1.zero();
    for (int i = 0; i < a.getDiagonalsCount(); ++i)
        if (a.getDiagonalPos(i) >= 0) {
            auto mit = a.posBegin(i);
            for (auto it = a.begin(i); it != a.end(i); ++it, ++mit)
                x1(mit.i) += (*it) * x(mit.j);
        }

    // Проходим по нижнему треугольнику и считаем все параметры
    matrix_diagonal_line_iterator mit(a.dimension(), a.getFormat(), true);
    for (; !mit.isEnd(); ++mit) {
        for (; !mit.isLineEnd(); ++mit)
            x1(mit.i) += a.begin(mit.dn)[mit.di] * x(mit.j);
        x(mit.i) = x(mit.i) + w/a.begin(0)[mit.i] * (y(mit.i) - x1(mit.i));
    }
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
SolverSLAE_Iterative_matrix::SolverSLAE_Iterative_matrix() :
    w(1),
    isLog(false),
    log(std::cout),
    start(),
    epsilon(0.00001),
    maxIterations(100) {
}

//-----------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative_matrix::jacobi(const Matrix& a, const Vector& y, Vector& x) const {
    return iteration_process(a, y, x, &SolverSLAE_Iterative_matrix::iteration_jacobi);
}

//-----------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative_matrix::seidel(const Matrix& a, const Vector& y, Vector& x) const {
    return iteration_process(a, y, x, &SolverSLAE_Iterative_matrix::iteration_seidel);
}

//-----------------------------------------------------------------------------
void SolverSLAE_Iterative_matrix::iteration_jacobi(const Matrix& a, const Vector& y, Vector& x) const {
    for (int i = 0; i < a.height(); ++i) {
        sumreal sum = 0;
        for (int j = 0; j < a.height(); ++j)
            sum += a(i, j) * x(j);
        x1(i) = x(i) + w / a(i, i) * (y(i) - sum);
    }

```

```
    x = x1;
}

//---------------------------------------------------------------------------
void SolverSLAE_Iterative_matrix::iteration_seidel(const Matrix& a, const
↪ Vector& y, Vector& x) const {
    for (int i = 0; i < a.height(); ++i) {
        sumreal sum = 0;
        for (int j = 0; j < a.height(); ++j)
            sum += a(i, j) * x(j);
        x(i) = x(i) + w / a(i, i) * (y(i) - sum);
    }
}

//---------------------------------------------------------------------------
IterationsResult SolverSLAE_Iterative_matrix::iteration_process(const Matrix& a,
↪ const Vector& y, Vector& x, step_function step) const {
    if (a.width() != a.height() || a.width() != y.size() || start.size() !=
    ↪ y.size())
        throw std::exception();

    // Считаем норму матрицы: ее максимальный элемент по модулю
    real yNorm = calcNorm(y);
    x1.resize(y.size());
    x = start;

    // Цикл по итерациям
    int i = 0;
    real relativeResidual = epsilon + 1;
    for (; i < maxIterations && relativeResidual > epsilon; ++i) {
        // Итерационный шаг
        step(this, a, y, x);

        // Считаем невязку
        mul(a, x, x1);
        x1.negate();
        sum(x1, y, x1);
        relativeResidual = calcNorm(x1) / yNorm;

        // Выводим данные
        if (isLog)
            log << i << "\t" << std::scientific << std::setprecision(3) <<
            ↪ relativeResidual << std::endl;
    }

    return {i, relativeResidual};
}
```

FILE table_generator.cpp

```
#include <fstream>
#include <cmath>
#include <iomanip>
#include <algorithm>
#include "diagonal.h"

typedef std::function<IterationsResult(const SolverSLAE_Iterative*, const
↪ MatrixDiagonal& a, const Vector& y, Vector& x)> method_function;

//---------------------------------------------------------------------------
```

```
void makeTable(
    const MatrixDiagonal& a,
    const Vector& x_precise,
    const Vector& y,
    SolverSLAE_Iterative& solver,
    std::string fileName
) {
    std::ofstream fout(fileName + ".tex");
    std::ofstream fout1(fileName + ".dat");

    // Выводим матрицу и остальное в виде формулы
    auto format = a.getFormat();
    fout << "$$ " << fileName << "=\\left(\\quad\\begin{matrix}\n";
    Matrix a_dense;
    a.toDenseMatrix(a_dense);
    for (int i = 0; i < a_dense.height(); i++) {
        for (int j = 0; j < a_dense.width(); j++) {
            int d = Diagonal(a_dense.height()).calcDiag_byLR(i, j);
            bool isOnFormat = std::find(format.begin(), format.end(), d) !=
            ↪  format.end();
            if (isOnFormat)
                fout << "\\cellcolor{green!30}";
            fout << int(a_dense(i, j));
            if (j + 1 != a_dense.width())
                 fout << " & ";
        }
        if (i + 1 != a_dense.height())
            fout << " \\\\\n";
        else
            fout << " \n";
    }
    fout << "\\end{matrix}\\quad\\right), X=\\begin{pmatrix}";
    for (int i = 0; i < x_precise.size(); i++) {
        if (i + 1 != x_precise.size())
            fout << int(x_precise(i)) << " \\\\\n";
        else
            fout << int(x_precise(i)) << " \n";
    }
    fout << "\\end{pmatrix}, F=\\begin{pmatrix}";
    for (int i = 0; i < y.size(); i++) {
        if (i + 1 != y.size())
            fout << int(y(i)) << " \\\\\n";
        else
            fout << int(y(i)) << " \n";
    }
    fout << "\\end{pmatrix} $$\n\n";

    // Выводим параметры решателя
    int exponent = floor(log10(solver.epsilon));
    double number = solver.epsilon / pow(10.0, exponent);
    fout
        << "$$ \\varepsilon = ";
    if (fabs(number - 1) >= 0.01)
        fout
            << std::setprecision(2) << std::fixed << number
            << " \\cdot ";
    fout
        << "10^{" << exponent << "}, \\quad iterations_{max} = "
        << solver.maxIterations << ", \\quad start = \\begin{pmatrix} ";
```

```
    fout << std::defaultfloat;
    for (int i = 0; i < solver.start.size(); i++) {
        fout << solver.start(i);
        if (i + 1 != solver.start.size())
            fout << ” & ”;
        else
            fout << ” ”;
    }
    fout << ”\\end{pmatrix}^T $$\n\n”;

    std::vector<double> w1(200), w2(200);
    std::vector<Vector> x1(200), x2(200);
    std::vector<Vector> xsub1(200), xsub2(200);
    std::vector<double> rr1(200), rr2(200); // relativeResidual
    std::vector<double> va1(200), va2(200);
    std::vector<int> it1(200), it2(200);

    int min1, min2;
    int count1, count2;

    auto one_method = [&a, &x_precise, &y, &solver] (
        std::vector<double>& w,
        std::vector<Vector>& x,
        std::vector<Vector>& xsub,
        std::vector<double>& rr,
        std::vector<double>& va,
        std::vector<int>& it,

        int& min,
        int& count,

        method_function method
    ) {
        min = 0;
        count = 200;
        Vector x_solve(x_precise.size());
        Vector x_sub(x_precise.size());
        real xNorm = calcNorm(x_precise);

        for (int i = 0; i < 200; ++i) {
            solver.w = i / 100.0;
            auto result = method(&solver, a, y, x_solve);

            // Если начинается ошибки после 100 итерации, то и потом ничего
            ↪ кроме них не будет, поэтому заканчиваем цикл
            if ((result.relativeResidual > solver.epsilon && i >= 100) ||
                (result.relativeResidual != result.relativeResidual)) {
                count = i;
                break;
            }

            // Вычисления разности точного и приближенного решений
            x_sub = x_solve;
            x_sub.negate();
            sum(x_sub, x_precise, x_sub);
            real x_subNorm = calcNorm(x_sub);

            w[i] = solver.w;
            x[i] = x_solve;
```

```
            xsub[i] = x_sub;
            rr[i] = result.relativeResidual;
            va[i] = x_subNorm / xNorm / result.relativeResidual;
            it[i] = result.iterations;

            // Находим минимум
            if (result.iterations < it[min])
                min = i;
        }
    };

    one_method(w1, x1, xsub1, rr1, va1, it1, min1, count1, &SolverSLAE_Iterative::jacobi);
    one_method(w2, x2, xsub2, rr2, va2, it2, min2, count2, &SolverSLAE_Iterative::seidel);

    // w, x, x-x*, относительная невязка, vA, итераций
    fout
        << ”\\setlength{\\tabcolsep}{2pt}\n”
        << ”\\tabulinesep=0.3mm\n”
        << ”\\noindent{\\scriptsize\\texttt{”
        << ”\\begin{longtabu}{\n”
        << ”|X[-1,c]||X[-1,c]|X[-1,c]|X[-1,c]|X[-1,c]|X[-1,c]|\n”
        << ”p{0.05cm}\n|X[-1,c]||X[-1,c]|X[-1,c]|X[-1,c]|X[-1,c]|X[-1,c]|}\n”
        << ”\\cline{1-6}\\cline{8-13}\n”
        << ”\\multicolumn{6}{|c|}{Метод Якоби} && \\multicolumn{6}{c|}{Метод Зейделя} \\\\\n”
        << ”\\cline{1-6}\\cline{8-13}\n”;

    auto write_vector = [&fout] (const Vector& a) {
        fout << ”\\tcell{”;
        for (int i = 0; i < a.size(); ++i)
            if (i + 1 != a.size())
                fout << a(i) << ” \\\\ ”;
            else
                fout << a(i) << ”}”;
    };

    auto write_line = [&] (int i, int colorNo) {
        int doublePrec = 16;
        if (i < count1) {
            if (colorNo == 1) {
                fout << std::fixed << std::setprecision(2) << ”\\cellcolor{green!30}{” << w1[i] << ”} & ”;
                fout << std::fixed << std::setprecision(doublePrec) << ”\\tiny{\\cellcolor{green!30}{”;
                write_vector(x1[i]);
                fout << ”}} & ” << ”\\tiny{\\cellcolor{green!30}{”;
                fout << std::scientific << std::setprecision(1);
                write_vector(xsub1[i]);
                fout << ”}} & ”;
                fout << std::scientific << std::setprecision(1) << ”\\cellcolor{green!30}{” << rr1[i] << ”} & ”;
                fout << std::fixed << std::setprecision(2) << ”\\cellcolor{green!30}{” << va1[i] << ”} & ”;
                fout << ”\\cellcolor{green!30}{” << it1[i] << ”} & ”;
            } else {
                fout << std::fixed << std::setprecision(2) << w1[i] << ” & ”;
                fout << std::fixed << std::setprecision(doublePrec) << ”\\tiny{”;
```

```
                write_vector(x1[i]);
                fout << "} & " << "\\tiny{";
                fout << std::scientific << std::setprecision(2);
                write_vector(xsub1[i]);
                fout << "} & ";
                fout << std::scientific << std::setprecision(2) << rr1[i] << " & ";
                fout << std::fixed << std::setprecision(2) << va1[i] << " & ";
                fout << it1[i] << " & ";
            }
        } else {
            fout << "& & & & & &";
        }

        fout << " & ";

        if (i < count2) {
            if (colorNo == 2) {
                fout << std::fixed << std::setprecision(2) << "\\cellcolor{green!30}{" << w2[i] << "} & ";
                fout << std::fixed << std::setprecision(doublePrec) << "\\tiny{\\cellcolor{green!30}{";
                write_vector(x2[i]);
                fout << "}} & " << "\\tiny{\\cellcolor{green!30}{";
                fout << std::scientific << std::setprecision(1);
                write_vector(xsub2[i]);
                fout << "}} & ";
                fout << std::scientific << std::setprecision(2) << "\\cellcolor{green!30}{" << rr2[i] << "} & ";
                fout << std::fixed << std::setprecision(2) << "\\cellcolor{green!30}{" << va2[i] << "} & ";
                fout << "\\cellcolor{green!30}{" << it2[i] << "} \\\\";
            } else {
                fout << std::fixed << std::setprecision(2) << w2[i] << " & ";
                fout << std::fixed << std::setprecision(doublePrec) << "\\tiny{";
                write_vector(x2[i]);
                fout << "} & " << "\\tiny{";
                fout << std::scientific << std::setprecision(1);
                write_vector(xsub2[i]);
                fout << "} & ";
                fout << std::scientific << std::setprecision(2) << rr2[i] << " & ";
                fout << std::fixed << std::setprecision(2) << va2[i] << " & ";
                fout << it2[i] << " \\\\";
            }
        } else {
            fout << "& & & & & \\\\";
        }

        fout << "\n";
        fout << "\\cline{1-6}\\cline{8-13}\n";
    };

    for (int i = 0; i < std::max(count1, count2); i+=10) {
        if (min1 == i) {
            write_line(i, 1);
        } else {
            if (min2 == i) {
                write_line(i, 2);
```

```
            } else {
                write_line(i, 0);
            }
        }
        if (i + 10 > min1 && min1 > i) {
            write_line(min1, 1);
        } else {
            if (i + 10 > min2 && min2 > i) {
                write_line(min2, 2);
            }
        }
    }

    fout1 << ”w1\tit1\tw2\tit2” << std::endl;
    fout1 << std::fixed << std::setprecision(2);
    for (int i = 0; i < std::max(count1, count2); ++i) {
        if (i >= count1)
            fout1 << w1[count1-1] << ”\t” << it1[count1-1] << ”\t”;
        else
            fout1 << w1[i] << ”\t” << it1[i] << ”\t”;
        if (i >= count2)
            fout1 << w2[count2-1] << ”\t” << it2[count2-1] << std::endl;
        else
            fout1 << w2[i] << ”\t” << it2[i] << std::endl;
    }

    fout << ”\\end{longtabu}}\n\n”;

    fout
        << ”\\noindent\\begin{tikzpicture}\n”
        << ”\\begin{semilogyaxis}[xlabel=w,ylabel=Iterations,width=\\textwidth,height=6cm]\n”
        << ”\\addplot[red, no markers] table [y=it1, x=w1]{” << fileName << ”.dat};\n”
        << ”\\addplot[blue, no markers] table [y=it2, x=w2]{” << fileName << ”.dat};\n”
        << ”\\legend{Jacobi,Seidel}\n”
        << ”\\end{semilogyaxis}\n”
        << ”\\end{tikzpicture}”;

    fout.close();
    fout1.close();
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

int main() {
    // Получаем необходимые матрицы
    MatrixDiagonal a, b;
    generateDiagonallyDominantMatrix(10, makeSevenDiagonalFormat(10, 3, 2), true, a);
    generateDiagonallyDominantMatrix(10, makeSevenDiagonalFormat(10, 2, 4), false, b);

    Vector x;
    x.generate(10);

```

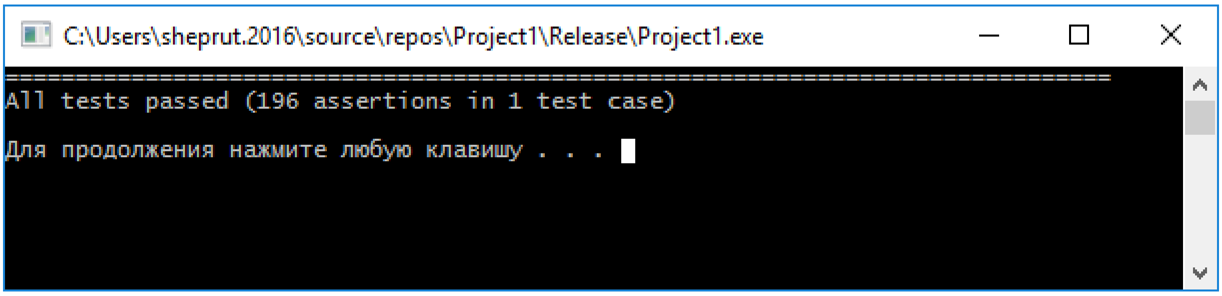
```
    Vector y_a, y_b;
    mul(a, x, y_a);
    mul(b, x, y_b);

    // Начальные присвоения
    SolverSLAE_Iterative solver;
    solver.w = 0;
    solver.isLog = false;
    solver.start = Vector(10, 0);
    solver.epsilon = 1e-14;
    solver.maxIterations = 1e5;

    // Создаем таблицы
    makeTable(a, x, y_a, solver, ”A”);
    makeTable(b, x, y_b, solver, ”B”);
}
```

3 Тестирование

Для тестирования использовалось юнит-тестирование и библиотека Catch. Было протестировано получение необходимой относительной невязки на матрицах с диагональным преобладанием.



diagonal_test.cpp

```
#define CATCH_CONFIG_RUNNER

#include ”../1/catch.hpp”
#include ”../1/matrix.h”
#include ”../1/vector.h”
#include ”diagonal.h”

typedef std::function<IterationsResult(const SolverSLAE_Iterative*, const MatrixDiagonal& a, const Vector& y, Vector& x)> method_function;

//-----------------------------------------------------------------------------
void testResidual(const MatrixDiagonal& a, const Vector& x, method_function method, real epsilon) {
    SolverSLAE_Iterative solver;
    solver.w = 1;
    solver.start = Vector(a.dimension(), 5);
    solver.epsilon = epsilon;
    solver.maxIterations = 1e5;

    Vector y;
    mul(a, x, y);

```

```
    Vector x1;
    auto result = method(&solver, a, y, x1);

    Vector y1;
    mul(a, x1, y1);

    y1.negate();
    sum(y, y1, y1);
    real relativeResidual = calcNorm(y1) / calcNorm(y);

    if (relativeResidual == relativeResidual) {
        CHECK(fabs(relativeResidual - result.relativeResidual) / relativeResidual < 0.01);

        if (result.iterations < solver.maxIterations) {
            CHECK(relativeResidual <= epsilon);
        }
    }
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

TEST_CASE("Test residual of methods") {
    for (int i = 10; i < 100; ++i) {
        for (int j = 0; j < 3; ++j) {
            MatrixDiagonal a;
            auto format = generateRandomFormat(i, intRandom(10, Diagonal(i).calcDiagonalsCount()));
            generateDiagonallyDominantMatrix(i, format, intRandom(0, 10) % 2, a);

            Vector x;
            x.generate(i);

            testResidual(a, x, &SolverSLAE_Iterative::jacobi, 1e-10);
            testResidual(a, x, &SolverSLAE_Iterative::seidel, 1e-10);
        }
    }
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

int main(int argc, char* const argv[]) {
    int result = Catch::Session().run(argc, argv);

    system("pause");
    return result;
}
```

4 Исследования

4.1 Матрица с диагональным преобладанием

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -3 & 13 & -4 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\ 0 & 0 & 7 & -3 & 0 & 0 & -2 & 0 & -2 & 0 \\ 0 & 0 & -3 & 8 & -2 & 0 & 0 & 0 & 0 & -3 \\ -2 & 0 & 0 & -2 & 5 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 2 & 0 & 0 & 0 & 0 \\ -2 & 0 & -4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & 0 & 0 & -3 & 7 & -1 & 0 \\ 0 & 0 & -2 & 0 & -4 & 0 & 0 & -3 & 9 & 0 \\ 0 & 0 & 0 & -1 & 0 & -4 & 0 & 0 & -4 & 9 \end{pmatrix}, X = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}, F = \begin{pmatrix} -5 \\ -29 \\ -23 \\ -17 \\ 9 \\ 5 \\ 28 \\ 14 \\ 31 \\ 26 \end{pmatrix}$$

$$\varepsilon = 10^{-14}, \quad iterations_{max} = 100000, \quad start = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$$

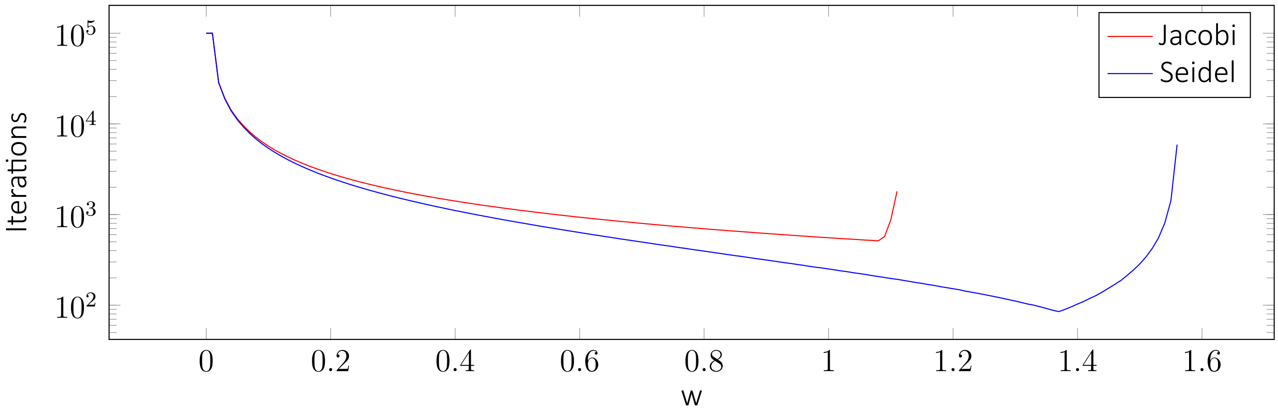
Метод Якоби					
0.00	0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000	1.00e+00 2.00e+00 3.00e+00 4.00e+00 5.00e+00 6.00e+00 7.00e+00 8.00e+00 9.00e+00 1.00e+01	1.00e+00	1.00	100000
0.10	0.9999999999997312 1.9999999999994800 2.9999999999994125 3.9999999999994142 4.9999999999995302 5.9999999999994751 6.9999999999994902 7.9999999999994200 8.9999999999994404 9.9999999999994262	2.69e-13 5.20e-13 5.88e-13 5.86e-13 4.70e-13 5.25e-13 5.10e-13 5.80e-13 5.60e-13 5.74e-13	9.93e-15	8.54	5678
0.20	0.9999999999997413 1.9999999999994982 2.9999999999994329 3.9999999999994347 4.9999999999995453 5.9999999999994946 6.9999999999995097 7.9999999999994413 8.9999999999994564 9.9999999999994404	2.59e-13 5.02e-13 5.67e-13 5.65e-13 4.55e-13 5.05e-13 4.90e-13 5.59e-13 5.44e-13 5.60e-13	9.96e-15	8.22	2833
0.30	0.9999999999997444 1.9999999999995042 2.9999999999994396 3.9999999999994413 4.9999999999995506 5.9999999999995008 6.9999999999995159 7.9999999999994476 8.9999999999994635 9.9999999999994476	2.56e-13 4.96e-13 5.60e-13 5.59e-13 4.49e-13 4.99e-13 4.84e-13 5.52e-13 5.36e-13 5.52e-13	9.86e-15	8.21	1884
0.40	0.9999999999997434 1.9999999999995017 2.9999999999994365 3.9999999999994382 4.9999999999995488 5.9999999999994991 6.9999999999995133 7.9999999999994449 8.9999999999994618 9.9999999999994458	2.57e-13 4.98e-13 5.64e-13 5.62e-13 4.51e-13 5.01e-13 4.87e-13 5.55e-13 5.38e-13 5.54e-13	9.90e-15	8.21	1409
0.50	0.9999999999997422 1.9999999999994995 2.9999999999994347 3.9999999999994369 4.9999999999995470 5.9999999999994973 6.9999999999995115 7.9999999999994422 8.9999999999994582 9.9999999999994440	2.58e-13 5.00e-13 5.65e-13 5.63e-13 4.53e-13 5.03e-13 4.88e-13 5.58e-13 5.42e-13 5.56e-13	9.94e-15	8.21	1124

Метод Зейделя					
0.00	0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000	1.0e+00 2.0e+00 3.0e+00 4.0e+00 5.0e+00 6.0e+00 7.0e+00 8.0e+00 9.0e+00 1.0e+01	1.00e+00	1.00	100000
0.10	0.9999999999997365 1.9999999999994895 2.9999999999994222 3.9999999999994245 4.9999999999995381 5.9999999999994866 6.9999999999995008 7.9999999999994333 8.9999999999994529 9.9999999999994351	2.6e-13 5.1e-13 5.8e-13 5.8e-13 4.6e-13 5.1e-13 5.0e-13 5.7e-13 5.5e-13 5.6e-13	9.95e-15	8.36	5381
0.20	0.9999999999997410 1.9999999999994977 2.9999999999994329 3.9999999999994373 4.9999999999995479 5.9999999999994991 6.9999999999995124 7.9999999999994458 8.9999999999994635 9.9999999999994511	2.6e-13 5.0e-13 5.7e-13 5.6e-13 4.5e-13 5.0e-13 4.9e-13 5.5e-13 5.4e-13 5.5e-13	9.93e-15	8.19	2532
0.30	0.9999999999997394 1.9999999999994948 2.9999999999994307 3.9999999999994369 4.9999999999995488 5.9999999999994991 6.9999999999995115 7.9999999999994467 8.9999999999994653 9.9999999999994529	2.6e-13 5.1e-13 5.7e-13 5.6e-13 4.5e-13 5.0e-13 4.9e-13 5.5e-13 5.3e-13 5.5e-13	1.00e-14	8.14	1582
0.40	0.9999999999997397 1.9999999999994955 2.9999999999994316 3.9999999999994396 4.9999999999995506 5.9999999999995026 6.9999999999995142 7.9999999999994511 8.9999999999994689 9.9999999999994582	2.6e-13 5.0e-13 5.7e-13 5.6e-13 4.5e-13 5.0e-13 4.9e-13 5.5e-13 5.3e-13 5.4e-13	9.99e-15	8.10	1107
0.50	0.9999999999997498 1.9999999999995155 2.9999999999994547 3.9999999999994644 4.9999999999995710 5.9999999999995257 6.9999999999995355 7.9999999999994778 8.9999999999994955 9.9999999999994866	2.5e-13 4.8e-13 5.5e-13 5.4e-13 4.3e-13 4.7e-13 4.6e-13 5.2e-13 5.0e-13 5.1e-13	9.76e-15	7.92	823

0.60	0.9999999999997411 1.9999999999994975 2.9999999999994320 3.9999999999994347 4.9999999999995453 5.9999999999994946 6.9999999999995097 7.9999999999994404 8.9999999999994564 9.9999999999994422	2.59e-13 5.02e-13 5.68e-13 5.65e-13 4.55e-13 5.05e-13 4.90e-13 5.60e-13 5.44e-13 5.58e-13	9.89e-15	8.29	934
0.70	0.9999999999997469 1.9999999999995088 2.9999999999994449 3.9999999999994476 4.9999999999995559 5.9999999999995062 6.9999999999995204 7.9999999999994520 8.9999999999994689 9.9999999999994547	2.53e-13 4.91e-13 5.55e-13 5.52e-13 4.44e-13 4.94e-13 4.80e-13 5.48e-13 5.31e-13 5.45e-13	9.64e-15	8.31	799
0.80	0.9999999999997445 1.9999999999995040 2.9999999999994396 3.9999999999994413 4.9999999999995515 5.9999999999995008 6.9999999999995159 7.9999999999994476 8.9999999999994618 9.9999999999994493	2.55e-13 4.96e-13 5.60e-13 5.59e-13 4.49e-13 4.99e-13 4.84e-13 5.52e-13 5.38e-13 5.51e-13	9.83e-15	8.23	697
0.90	0.9999999999997456 1.9999999999995062 2.9999999999994418 3.9999999999994440 4.9999999999995532 5.9999999999995035 6.9999999999995186 7.9999999999994502 8.9999999999994671 9.9999999999994511	2.54e-13 4.94e-13 5.58e-13 5.56e-13 4.47e-13 4.96e-13 4.81e-13 5.50e-13 5.33e-13 5.49e-13	9.85e-15	8.17	618
1.00	0.9999999999997509 1.9999999999995151 2.9999999999994529 3.9999999999994547 4.9999999999995612 5.9999999999995133 6.9999999999995266 7.9999999999994609 8.9999999999994760 9.9999999999994618	2.49e-13 4.85e-13 5.47e-13 5.45e-13 4.39e-13 4.87e-13 4.73e-13 5.39e-13 5.24e-13 5.38e-13	9.70e-15	8.14	555
1.08	0.9999999999997538 1.9999999999995233 2.9999999999994595 3.9999999999994640 4.9999999999995683 5.9999999999995204 6.9999999999995355 7.9999999999994680 8.9999999999994831 9.9999999999994706	2.5e-13 4.8e-13 5.4e-13 5.4e-13 4.3e-13 4.8e-13 4.6e-13 5.3e-13 5.2e-13 5.3e-13	9.3e-15	8.35	513
1.10	1.0000000000000113 1.9999999999999800 3.0000000000000013 4.0000000000000124 4.9999999999999813 6.0000000000000284 6.9999999999999893 8.0000000000000053 9.0000000000000053 9.9999999999999822	-1.13e-14 2.00e-14 -1.33e-15 -1.24e-14 1.87e-14 -2.84e-14 1.07e-14 -5.33e-15 -5.33e-15 1.78e-14	9.12e-15	0.27	875

0.60	0.9999999999997568 1.9999999999995293 2.9999999999994706 3.9999999999994831 4.9999999999995870 5.9999999999995435 6.9999999999995524 7.9999999999994973 8.9999999999995168 9.9999999999995097	2.4e-13 4.7e-13 5.3e-13 5.2e-13 4.1e-13 4.6e-13 4.5e-13 5.0e-13 4.8e-13 4.9e-13	9.68e-15	7.69	633
0.70	0.9999999999997663 1.9999999999995477 2.9999999999994920 3.9999999999995066 4.9999999999996056 5.9999999999995648 6.9999999999995719 7.9999999999995230 8.9999999999995399 9.9999999999995346	2.3e-13 4.5e-13 5.1e-13 4.9e-13 3.9e-13 4.4e-13 4.3e-13 4.8e-13 4.6e-13 4.7e-13	9.53e-15	7.46	497
0.80	0.9999999999997817 1.9999999999995768 2.9999999999995257 3.9999999999995417 4.9999999999996350 5.9999999999995977 6.9999999999996039 7.9999999999995604 8.9999999999995772 9.9999999999995737	2.2e-13 4.2e-13 4.7e-13 4.6e-13 3.7e-13 4.0e-13 4.0e-13 4.4e-13 4.2e-13 4.3e-13	9.42e-15	6.99	395
0.90	0.9999999999997934 1.9999999999996005 2.9999999999995537 3.9999999999995723 4.9999999999996607 5.9999999999996261 6.9999999999996296 7.9999999999995932 8.9999999999996092 9.9999999999996074	2.1e-13 4.0e-13 4.5e-13 4.3e-13 3.4e-13 3.7e-13 3.7e-13 4.1e-13 3.9e-13 3.9e-13	9.54e-15	6.43	315
1.00	0.9999999999998201 1.9999999999996521 2.9999999999996123 3.9999999999996323 4.9999999999997087 5.9999999999996803 6.9999999999996820 7.9999999999996536 8.9999999999996678 9.9999999999996696	1.8e-13 3.5e-13 3.9e-13 3.7e-13 2.9e-13 3.2e-13 3.2e-13 3.5e-13 3.3e-13 3.3e-13	9.02e-15	5.84	251
1.08	0.9999999999998229 1.9999999999996569 2.9999999999996190 3.9999999999996430 4.9999999999997176 5.9999999999996909 6.9999999999996900 7.9999999999996643 8.9999999999996803 9.9999999999996856	1.8e-13 3.4e-13 3.8e-13 3.6e-13 2.8e-13 3.1e-13 3.1e-13 3.4e-13 3.2e-13 3.1e-13	9.64e-15	5.30	207
1.10	0.9999999999998258 1.9999999999996636 2.9999999999996265 3.9999999999996509 4.9999999999997247 5.9999999999996989 6.9999999999996971 7.9999999999996723 8.9999999999996891 9.9999999999996927	1.7e-13 3.4e-13 3.7e-13 3.5e-13 2.8e-13 3.0e-13 3.0e-13 3.3e-13 3.1e-13 3.1e-13	9.63e-15	5.19	197
1.20	0.9999999999998664 1.9999999999997420 2.9999999999997149 3.9999999999997380 4.9999999999997948 5.9999999999997771 6.9999999999997735 7.9999999999997611 8.9999999999997744 9.9999999999997797	1.3e-13 2.6e-13 2.9e-13 2.6e-13 2.1e-13 2.2e-13 2.3e-13 2.4e-13 2.3e-13 2.2e-13	8.69e-15	4.28	152
1.30	0.9999999999998852 1.9999999999997773 2.9999999999997558 3.9999999999997824 4.9999999999998312 5.9999999999998179 6.9999999999998126 7.9999999999998055 8.9999999999998206 9.9999999999998295	1.1e-13 2.2e-13 2.4e-13 2.2e-13 1.7e-13 1.8e-13 1.9e-13 1.9e-13 1.8e-13 1.7e-13	9.30e-15	3.31	111
1.37	0.9999999999999157 1.9999999999998914 2.9999999999998836 3.9999999999999565 4.9999999999999458 5.9999999999998996 6.9999999999998996 7.9999999999999503 8.9999999999999432 9.9999999999999130	8.4e-14 1.1e-13 1.2e-13 4.4e-14 5.4e-14 1.0e-13 1.0e-13 5.0e-14 5.7e-14 8.7e-14	9.10e-15	1.49	85

						1.40	1.0000000000000024 1.9999999999999885 3.0000000000000351 3.9999999999999760 4.9999999999999698 5.9999999999999920 7.0000000000000480 8.0000000000000018 8.9999999999999876 10.0000000000000000	-2.4e-15 1.2e-14 -3.5e-14 2.4e-14 3.0e-14 8.0e-15 -4.8e-14 -1.8e-15 1.2e-14 0.0e+00	7.35e-15	0.51	103
						1.50	0.9999999999999558 1.9999999999999600 3.0000000000000373 4.0000000000000036 4.9999999999999520 5.9999999999999396 7.0000000000000524 8.0000000000000355 8.9999999999999840 9.9999999999999556	4.4e-14 4.0e-14 -3.7e-14 -3.6e-15 4.8e-14 6.0e-14 -5.2e-14 -3.6e-14 1.6e-14 4.4e-14	8.49e-15	0.79	285



4.2 Матрица с обратным знаком внедиагональных элементов

$$B = \begin{pmatrix} 9 & 4 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 \\ 4 & 8 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 9 & 1 & 0 & 3 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 9 & 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 5 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 & 0 & 3 & 8 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 2 & 0 & 3 & 10 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 2 & 7 & 1 \\ 0 & 0 & 3 & 0 & 0 & 0 & 3 & 0 & 1 & 7 \end{pmatrix}, X = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}, F = \begin{pmatrix} 45 \\ 48 \\ 75 \\ 44 \\ 97 \\ 59 \\ 100 \\ 148 \\ 113 \\ 109 \end{pmatrix}$$

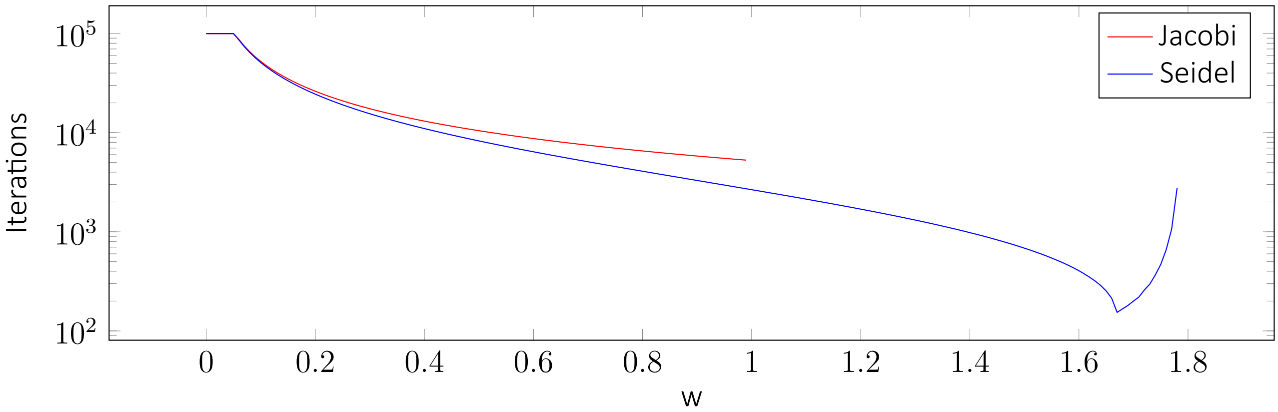
$$\varepsilon = 10^{-14}, \quad iterations_{max} = 100000, \quad start = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$$

Метод Якоби						Метод Зейделя					
0.00	0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000	1.00e+00 2.00e+00 3.00e+00 4.00e+00 5.00e+00 6.00e+00 7.00e+00 8.00e+00 9.00e+00 1.00e+01	1.00e+00	1.00	100000	0.00	0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000 0.0000000000000000	1.0e+00 2.0e+00 3.0e+00 4.0e+00 5.0e+00 6.0e+00 7.0e+00 8.0e+00 9.0e+00 1.0e+01	1.00e+00	1.00	100000
0.10	0.9999999999748445 2.0000000000272502 2.9999999999713296 4.0000000000295977 4.9999999999709344 6.0000000000290692 6.9999999999705302 8.0000000000289546 8.9999999999708287 10.0000000000291998	2.52e-11 -2.73e-11 2.87e-11 -2.96e-11 2.91e-11 -2.91e-11 2.95e-11 -2.90e-11 2.92e-11 -2.92e-11	9.99e-15	461.37	52448	0.10	0.9999999999750441 2.0000000000270322 2.9999999999715627 4.0000000000293552 4.9999999999711697 6.0000000000288320 6.9999999999707736 8.0000000000287201 8.9999999999710614 10.0000000000289617	2.5e-11 -2.7e-11 2.8e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11	9.99e-15	457.30	51082

0.20	0.9999999999751141 2.0000000000269584 2.9999999999716391 4.0000000000292735 4.9999999999712452 6.0000000000287601 6.9999999999708482 8.0000000000286509 8.9999999999711324 10.0000000000288924	2.49e-11 -2.70e-11 2.84e-11 -2.93e-11 2.88e-11 -2.88e-11 2.92e-11 -2.87e-11 2.89e-11 -2.89e-11	9.98e-15	456.98	26225
0.30	0.9999999999749128 2.0000000000271765 2.9999999999714091 4.0000000000295106 4.9999999999710134 6.0000000000289937 6.9999999999706111 8.0000000000288818 8.9999999999708997 10.0000000000291251	2.51e-11 -2.72e-11 2.86e-11 -2.95e-11 2.90e-11 -2.90e-11 2.94e-11 -2.89e-11 2.91e-11 -2.91e-11	9.99e-15	459.80	17472
0.40	0.9999999999749905 2.0000000000270926 2.9999999999714961 4.0000000000294218 4.9999999999711005 6.0000000000289040 6.9999999999706999 8.0000000000287912 8.9999999999709903 10.0000000000290363	2.50e-11 -2.71e-11 2.85e-11 -2.94e-11 2.89e-11 -2.89e-11 2.93e-11 -2.88e-11 2.90e-11 -2.90e-11	1.00e-14	458.23	13103
0.50	0.9999999999750855 2.0000000000269904 2.9999999999716045 4.0000000000293081 4.9999999999712124 6.0000000000287947 6.9999999999708136 8.0000000000286811 8.9999999999710987 10.0000000000289262	2.49e-11 -2.70e-11 2.84e-11 -2.93e-11 2.88e-11 -2.88e-11 2.92e-11 -2.87e-11 2.89e-11 -2.89e-11	9.88e-15	461.69	10482
0.60	0.9999999999751421 2.0000000000269287 2.9999999999716702 4.0000000000292371 4.9999999999712772 6.0000000000287272 6.9999999999708828 8.0000000000286171 8.9999999999711662 10.0000000000288587	2.49e-11 -2.69e-11 2.83e-11 -2.92e-11 2.87e-11 -2.87e-11 2.91e-11 -2.86e-11 2.88e-11 -2.89e-11	9.97e-15	456.64	8734
0.70	0.9999999999750842 2.0000000000269909 2.9999999999716036 4.0000000000293081 4.9999999999712097 6.0000000000287956 6.9999999999708136 8.0000000000286846 8.9999999999711005 10.0000000000289297	2.49e-11 -2.70e-11 2.84e-11 -2.93e-11 2.88e-11 -2.88e-11 2.92e-11 -2.87e-11 2.89e-11 -2.89e-11	9.94e-15	459.33	7484
0.80	0.9999999999750737 2.0000000000270037 2.9999999999715921 4.0000000000293205 4.9999999999711990 6.0000000000288081 6.9999999999707994 8.0000000000286988 8.9999999999710880 10.0000000000289386	2.49e-11 -2.70e-11 2.84e-11 -2.93e-11 2.88e-11 -2.88e-11 2.92e-11 -2.87e-11 2.89e-11 -2.89e-11	9.99e-15	457.18	6547
0.90	0.9999999999751336 2.0000000000269380 2.9999999999716600 4.0000000000292504 4.9999999999712665 6.0000000000287361 6.9999999999708704 8.0000000000286260 8.9999999999711573 10.0000000000288676	2.49e-11 -2.69e-11 2.83e-11 -2.93e-11 2.87e-11 -2.87e-11 2.91e-11 -2.86e-11 2.88e-11 -2.89e-11	9.95e-15	457.88	5819
0.99	0.9999999999751349 2.0000000000269420 2.9999999999716618 4.0000000000292522 4.9999999999712701 6.0000000000287415 6.9999999999708731 8.0000000000286313 8.9999999999711573 10.0000000000288729	2.5e-11 -2.7e-11 2.8e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11	9.9e-15	461.89	5289

0.20	0.9999999999751906 2.0000000000268678 2.9999999999717359 4.0000000000291758 4.9999999999713474 6.0000000000286517 6.9999999999709601 8.0000000000285354 8.9999999999712532 10.0000000000287717	2.5e-11 -2.7e-11 2.8e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11 2.9e-11 -2.9e-11	1.00e-14	454.29	24500
0.30	0.9999999999755919 2.0000000000264286 2.9999999999722000 4.0000000000286953 4.9999999999718163 6.0000000000281757 6.9999999999714468 8.0000000000280593 8.9999999999717382 10.0000000000282867	2.4e-11 -2.6e-11 2.8e-11 -2.9e-11 2.8e-11 -2.8e-11 2.9e-11 -2.8e-11 2.8e-11 -2.8e-11	9.97e-15	447.84	15542
0.40	0.9999999999758989 2.0000000000260898 2.9999999999725584 4.0000000000283276 4.9999999999721778 6.0000000000278071 6.9999999999718225 8.0000000000276934 8.9999999999721130 10.0000000000279083	2.4e-11 -2.6e-11 2.7e-11 -2.8e-11 2.8e-11 -2.8e-11 2.8e-11 -2.8e-11 2.8e-11 -2.8e-11	9.98e-15	441.87	11012
0.50	0.9999999999762602 2.0000000000256919 2.9999999999729794 4.0000000000278932 4.9999999999726050 6.0000000000273754 6.9999999999722657 8.0000000000272617 8.9999999999725553 10.0000000000274643	2.4e-11 -2.6e-11 2.7e-11 -2.8e-11 2.7e-11 -2.7e-11 2.8e-11 -2.7e-11 2.7e-11 -2.7e-11	1.00e-14	434.20	8267
0.60	0.9999999999767676 2.0000000000251359 2.9999999999735665 4.0000000000272884 4.9999999999732019 6.0000000000267715 6.9999999999728812 8.0000000000266560 8.9999999999731681 10.0000000000268496	2.3e-11 -2.5e-11 2.6e-11 -2.7e-11 2.7e-11 -2.7e-11 2.7e-11 -2.7e-11 2.7e-11 -2.7e-11	9.97e-15	425.61	6421
0.70	0.9999999999775033 2.0000000000243299 2.9999999999744165 4.0000000000264126 4.9999999999740625 6.0000000000259011 6.9999999999737659 8.0000000000257909 8.9999999999740510 10.0000000000259668	2.2e-11 -2.4e-11 2.6e-11 -2.6e-11 2.6e-11 -2.6e-11 2.6e-11 -2.6e-11 2.6e-11 -2.6e-11	9.95e-15	412.64	5092
0.80	0.9999999999784299 2.0000000000233187 2.9999999999754836 4.0000000000253113 4.9999999999751452 6.0000000000248104 6.9999999999748779 8.0000000000247002 8.9999999999751576 10.0000000000248601	2.2e-11 -2.3e-11 2.5e-11 -2.5e-11 2.5e-11 -2.5e-11 2.5e-11 -2.5e-11 2.5e-11 -2.5e-11	9.91e-15	396.99	4087
0.90	0.9999999999792958 2.0000000000223714 2.9999999999764833 4.0000000000242828 4.9999999999761586 6.0000000000237881 6.9999999999759179 8.0000000000236771 8.9999999999761933 10.0000000000238192	2.1e-11 -2.2e-11 2.4e-11 -2.4e-11 2.4e-11 -2.4e-11 2.4e-11 -2.4e-11 2.4e-11 -2.4e-11	9.96e-15	378.68	3297
0.99	0.9999999999806718 2.0000000000208731 2.9999999999780660 4.0000000000226503 4.9999999999777600 6.0000000000221787 6.9999999999775557 8.0000000000220712 8.9999999999778186 10.0000000000221902	1.9e-11 -2.1e-11 2.2e-11 -2.3e-11 2.2e-11 -2.2e-11 2.2e-11 -2.2e-11 2.2e-11 -2.2e-11	9.88e-15	356.25	2720
1.00	0.9999999999806523 2.0000000000208935 2.9999999999780425 4.0000000000226716 4.9999999999777405 6.0000000000221974 6.9999999999775380 8.0000000000220890 8.9999999999778026 10.0000000000222062	1.9e-11 -2.1e-11 2.2e-11 -2.3e-11 2.2e-11 -2.2e-11 2.2e-11 -2.2e-11 2.2e-11 -2.2e-11	9.98e-15	352.86	2661

						1.10	0.9999999999823919 2.0000000000189981 2.9999999999800377 4.0000000000206155 4.9999999999797637 6.0000000000201643 6.9999999999796030 8.0000000000200586 8.9999999999798543 10.0000000000201545	1.8e-11 -1.9e-11 2.0e-11 -2.1e-11 2.0e-11 -2.0e-11 2.0e-11 -2.0e-11 2.0e-11 -2.0e-11	9.89e-15	323.59	2136
						1.20	0.9999999999845336 2.0000000000166689 2.9999999999824927 4.0000000000180869 4.9999999999822533 6.0000000000176659 6.9999999999821370 8.0000000000175664 8.9999999999823750 10.0000000000176321	1.5e-11 -1.7e-11 1.8e-11 -1.8e-11 1.8e-11 -1.8e-11 1.8e-11 -1.8e-11 1.8e-11 -1.8e-11	9.77e-15	286.99	1694
						1.30	0.9999999999868533 2.0000000000141469 2.9999999999851510 4.0000000000153442 4.9999999999849436 6.0000000000149640 6.9999999999848859 8.0000000000148717 8.9999999999850981 10.0000000000149054	1.3e-11 -1.4e-11 1.5e-11 -1.5e-11 1.5e-11 -1.5e-11 1.5e-11 -1.5e-11 1.5e-11 -1.5e-11	9.79e-15	242.59	1314
						1.40	0.9999999999892694 2.0000000000115188 2.9999999999879177 4.0000000000124878 4.9999999999877529 6.0000000000121450 6.9999999999877476 8.0000000000120615 8.9999999999879368 10.0000000000120615	1.1e-11 -1.2e-11 1.2e-11 -1.2e-11 1.2e-11 -1.2e-11 1.2e-11 -1.2e-11 1.2e-11 -1.2e-11	9.87e-15	195.50	981
						1.50	0.9999999999922587 2.0000000000082725 2.9999999999913363 4.0000000000089670 4.9999999999912159 6.0000000000086713 6.9999999999912701 8.0000000000085993 8.9999999999914326 10.0000000000085638	7.7e-12 -8.3e-12 8.7e-12 -9.0e-12 8.8e-12 -8.7e-12 8.7e-12 -8.6e-12 8.6e-12 -8.6e-12	9.88e-15	139.64	684
						1.60	0.9999999999954424 2.0000000000048197 2.9999999999949667 4.0000000000052127 4.9999999999948992 6.0000000000049862 6.9999999999950093 8.0000000000049276 8.9999999999951275 10.0000000000048566	4.6e-12 -4.8e-12 5.0e-12 -5.2e-12 5.1e-12 -5.0e-12 5.0e-12 -4.9e-12 4.9e-12 -4.9e-12	9.61e-15	82.83	405
						1.67	1.0000000000007827 1.9999999999992648 3.0000000000007390 3.9999999999992268 5.0000000000007505 5.9999999999993525 7.0000000000006084 7.9999999999993765 9.0000000000005471 9.9999999999994689	-7.8e-13 7.4e-13 -7.4e-13 7.7e-13 -7.5e-13 6.5e-13 -6.1e-13 6.2e-13 -5.5e-13 5.3e-13	7.87e-15	13.93	154
						1.70	1.0000000000032723 1.9999999999963636 3.0000000000038507 3.9999999999960201 5.0000000000039044 5.9999999999960316 7.0000000000040501 7.9999999999960476 9.0000000000040181 9.9999999999959677	-3.3e-12 3.6e-12 -3.9e-12 4.0e-12 -3.9e-12 4.0e-12 -4.1e-12 4.0e-12 -4.0e-12 4.0e-12	4.71e-15	132.54	200



5 Выводы

Исследования показали, что для различных матриц необходим различный параметр релаксации, и что иногда он может лежать за допустимыми пределами (как это было для метода Якоби, где $w = 1.08$). График зависимости числа итераций от параметра релаксации имеет непростой вид, что сильно затрудняет его выбор без перебора всех вариантов.

Так же было оценено число обусловленности: $cond(A) > 1.49$, $cond(B) > 13.93$.