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:

- То, что было в прошлом отчете и здесь не приводится:
  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. `vector.h + vector.cpp` — модуль для работы с векторами.
- Новый код:
  1. `diagonal.h + diagonal.cpp` — модуль для работы с матрицами в диагональном формате.
  2. `table_generator.cpp` — программа, которая генерирует таблицы.
  3. `diagonal_test.cpp` — юнит-тестирование модуля для работы с диагональными матрицами.

FILE **diagonal.h**

```cpp
#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);

    //-----------------------------------------------------------------------
    /*
        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

    void save(std::ostream& out) const;
    void load(std::istream& in);

    //-----------------------------------------------------------------------
    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
    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();

    void save(std::ostream& out) const;
```

```cpp
    void load(std::istream& in);

    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;
};
```

FILE **diagonal.cpp**

```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;
    }
}

//----------------------------------------------------------------------------
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));
}

//----------------------------------------------------------------------------
void MatrixDiagonal::save(std::ostream& out) const {
    out << n << ” ” << fi.size() << std::endl;
    for (const auto& i : fi)
        out << i << ” ”;
    out << std::endl;

    for (int i = 0; i < getDiagonalsCount(); ++i) {
        for (auto j = begin(i); j != end(i); ++j) {
            out << (*j) << ” ”;
        }
        out << std::endl;
    }
    out << std::endl;
}

//----------------------------------------------------------------------------
void MatrixDiagonal::load(std::istream& in) {
    int n, m;
    in >> n >> m;
    std::vector<int> format(m, 0);
    for (int i = 0; i < m; ++i)
        in >> format[i];
    resize(n, format);
    for (int i = 0; i < getDiagonalsCount(); ++i) {
        for (auto j = begin(i); j != end(i); ++j) {
            in >> (*j);
        }
    }
}

//----------------------------------------------------------------------------
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();
}

//----------------------------------------------------------------------------
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;

```

```cpp
    // 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) {
}

//----------------------------------------------------------------------------
void SolverSLAE_Iterative::save(std::ostream& out) const {
    out << w << std::endl;
    out << isLog << std::endl;
    start.save(out);
    out << std::scientific;
    out << epsilon << std::endl;
    out << std::defaultfloat;
    out << maxIterations << std::endl;
}

//----------------------------------------------------------------------------
void SolverSLAE_Iterative::load(std::istream& in) {
    in >> w >> isLog;
    start.load(in);
    in >> epsilon >> maxIterations;
}

//----------------------------------------------------------------------------
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));
    }
}
```

FILE table_generator.cpp

```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);
```

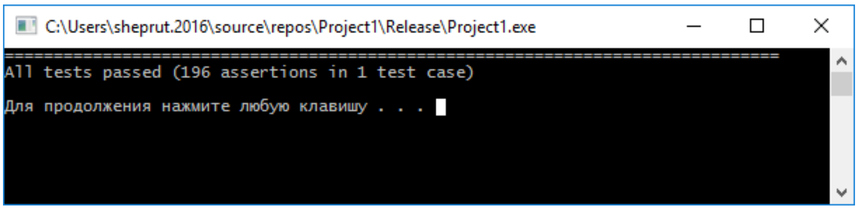
```cpp
    for (int i = 0; i < a_dense.height(); i++) {
        for (int j = 0; j < a_dense.width(); j++) {
            int d = Diagonal(a_dense.height()).calcDiag_byLR(i, j);
            bool isOnFormat = std::find(format.begin(), format.end(), d) != format.end();
            if (isOnFormat)
                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;
        }
    };

    // Заполняем массивы расчитанными данными
    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);

    //-----------------------------------------------------------------------
    // Выводим преамбулу таблицы
    // 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”
        << ”\\hhline{*{6}{-}~*{6}{-}}\n”
        << ”$w$ & $x$ & $x-x^*$ & \\tcell{\\tiny Относительная\\\\\\tiny невязка} & {\\tiny $\\mathop{cond}(A) >$} & {\\tiny Итераций} && $w$ & $x$ & $x-x^*$ & \\tcell{\\tiny Относительная\\\\\\tiny невязка} & {\\tiny $\\mathop{cond}(A) >$} & {\\tiny Итераций} \\\\\n”
        << ”\\hhline{*{6}{-}~*{6}{-}}\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, bool isWrite1, bool isWrite2, std::string color1, std::string color2) {
        int doublePrec = 16;
        if (isWrite1) {
            std::string before_cell = ”\\cellcolor{” + color1 + ”}{”;
            std::string after_cell = ”}”;
            if (color1 == ”white”) {
                before_cell = ””;
                after_cell = ””;
            }

            fout << std::fixed << std::setprecision(2)
                 << before_cell << w1[i] << after_cell <<” & ”;

            fout << std::fixed << std::setprecision(doublePrec)
                 << ”\\tiny{” << before_cell;
                 write_vector(x1[i]);
            fout << after_cell << ”} & ”;

            fout << std::scientific << std::setprecision(1)
                  << ”\\tiny{” << before_cell;
                 write_vector(xsub1[i]);
            fout << after_cell << ”} & ”;

            fout << std::scientific << std::setprecision(1)
                 << before_cell << rr1[i] << after_cell << ” & ”;

            fout << std::fixed << std::setprecision(2)
                 << before_cell << va1[i] << after_cell << ” & ”;

            fout << before_cell << it1[i] << after_cell << ” & ”;
        } else {
            fout << ”& & & & & &”;
        }

        fout << ” & ”;

        if (isWrite2) {
            std::string before_cell = ”\\cellcolor{” + color2 + ”}{”;
            std::string after_cell = ”}”;
            if (color2 == ”white”) {
                before_cell = ””;
                after_cell = ””;
            }

            fout << std::fixed << std::setprecision(2)
                 << before_cell << w2[i] << after_cell <<” & ”;

            fout << std::fixed << std::setprecision(doublePrec)
                 << ”\\tiny{” << before_cell;
                 write_vector(x2[i]);
            fout << after_cell << ”} & ”;

            fout << std::scientific << std::setprecision(1)
                  << ”\\tiny{” << before_cell;
                 write_vector(xsub2[i]);
            fout << after_cell << ”} & ”;

            fout << std::scientific << std::setprecision(1)
                 << before_cell << rr2[i] << after_cell << ” & ”;

            fout << std::fixed << std::setprecision(2)
                 << before_cell << va2[i] << after_cell << ” & ”;

            fout << before_cell << it2[i] << after_cell << ” \\\\”;
        } else {
            fout << ”& & & & & \\\\”;
        }

        fout << ”\n”;
        fout << ”\\hhline{*{6}{-}~*{6}{-}}\n”;
    };

    //-----------------------------------------------------------------------
    // Формируем массив тех значений, которые надо вывести
    int tableSize = std::max(count1, count2);
    std::vector<int> to_write;
    for (int i = 10; i < tableSize; i+=10)
        to_write.push_back(i);
    to_write.push_back(min1-1);
    to_write.push_back(min1);
    to_write.push_back(min1+1);
    to_write.push_back(min2-1);
    to_write.push_back(min2);
    to_write.push_back(min2+1);

    std::sort(to_write.begin(), to_write.end(), std::less<int>());

    // Удаляем дубликаты
    to_write.erase(std::unique(to_write.begin(), to_write.end()), to_write.end());

    // Удаляем отрицательные значения
    while (to_write.front() <= 0)
        to_write.erase(to_write.begin());

    // Удаляем значения за допустимыми пределами
    while (to_write.back() >= tableSize)
        to_write.pop_back();

    // Выводим каждую строку
    std::string orange = ”orange!30”;
    std::string green = ”green!30”;
    for (auto& i : to_write) {
        std::string color1 = ”white”, color2 = ”white”;
        if (i == min1)
            color1 = green;
        if (abs(i-min1) == 1)
            color1 = orange;
        if (i == min2)
            color2 = green;
        if (abs(i-min2) == 1)
            color2 = orange;
        write_line(i, i < count1, i < count2, color1, color2);
    }

    //-----------------------------------------------------------------------
    // Выводим данные для построения графика
    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”;

    //-----------------------------------------------------------------------
    // Выводим график
    fout
        << ”\\subsubsection{График зависимости числа итераций от параметра релаксации}\n\n”
        << ”\\noindent\\begin{tikzpicture}\n”
        << ”\\begin{semilogyaxis}[xlabel=w,ylabel=Итераций,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{Метод Якоби,Метод Зейделя}\n”
        << ”\\end{semilogyaxis}\n”
        << ”\\end{tikzpicture}”;

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

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

int main() {
    MatrixDiagonal a, b;
    Vector x;
    Vector y_a, y_b;
    SolverSLAE_Iterative solver;

    std::ifstream fin(”num.txt”);
    a.load(fin);
    b.load(fin);
    x.load(fin);
    solver.load(fin);
    fin.close();

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

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

    Matrix a_dense, b_dense;
    a.toDenseMatrix(a_dense);
    b.toDenseMatrix(b_dense);

    std::ofstream fout(”matrixes.txt”);
    fout << std::defaultfloat;
    a_dense.save(fout);
    b_dense.save(fout);
    fout.close();

    /*// Сохраняем сгенерированные данные в файл
    std::ofstream fout(”num.txt”);

    generateDiagonallyDominantMatrix(10, makeSevenDiagonalFormat(10, 3, 2), true, a);
    b = a;
    for (int i = 0; i < b.getDiagonalsCount(); i++)
        for (auto j = b.begin(i); j != b.end(i); j++)
            (*j) = std::fabs(*j);
    x.generate(10);
    solver.w = 0;
    solver.isLog = false;
    solver.start = Vector(10, 0);
    solver.epsilon = 1e-14;
    solver.maxIterations = 1e5;

    a.save(fout);
    b.save(fout);
    x.save(fout);
    solver.save(fout);

    fout.close();*/
}
```

diagonal_test.cpp

```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+=5) {
        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;
}
```

3 Тестирование

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



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

w	x	$x - x^*$	Относительная невязка	$cond(A) >$	Итераций
0.10	0.9999999999997312 1.9999999999994800 2.9999999999994125 3.9999999999994142 4.9999999999995302 5.9999999999994751 6.9999999999994902 7.9999999999994200 8.9999999999994404 9.9999999999994262	2.7e-13 5.2e-13 5.9e-13 5.9e-13 4.7e-13 5.2e-13 5.1e-13 5.8e-13 5.6e-13 5.7e-13	9.9e-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	2.6e-13 5.0e-13 5.7e-13 5.7e-13 4.5e-13 5.1e-13 4.9e-13 5.6e-13 5.4e-13 5.6e-13	1.0e-14	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	2.6e-13 5.0e-13 5.6e-13 5.6e-13 4.5e-13 5.0e-13 4.8e-13 5.5e-13 5.4e-13 5.5e-13	9.9e-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	2.6e-13 5.0e-13 5.6e-13 5.6e-13 4.5e-13 5.0e-13 4.9e-13 5.6e-13 5.4e-13 5.5e-13	9.9e-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	2.6e-13 5.0e-13 5.7e-13 5.6e-13 4.5e-13 5.0e-13 4.9e-13 5.6e-13 5.4e-13 5.6e-13	9.9e-15	8.21	1124
0.60	0.9999999999997411 1.9999999999994975 2.9999999999994320 3.9999999999994347 4.9999999999995453 5.9999999999994946 6.9999999999995097 7.9999999999994404 8.9999999999994564 9.9999999999994422	2.6e-13 5.0e-13 5.7e-13 5.7e-13 4.5e-13 5.1e-13 4.9e-13 5.6e-13 5.4e-13 5.6e-13	9.9e-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	2.5e-13 4.9e-13 5.6e-13 5.5e-13 4.4e-13 4.9e-13 4.8e-13 5.5e-13 5.3e-13 5.5e-13	9.6e-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	2.6e-13 5.0e-13 5.6e-13 5.6e-13 4.5e-13 5.0e-13 4.8e-13 5.5e-13 5.4e-13 5.5e-13	9.8e-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	2.5e-13 4.9e-13 5.6e-13 5.6e-13 4.5e-13 5.0e-13 4.8e-13 5.5e-13 5.3e-13 5.5e-13	9.8e-15	8.17	618

Метод Зейделя

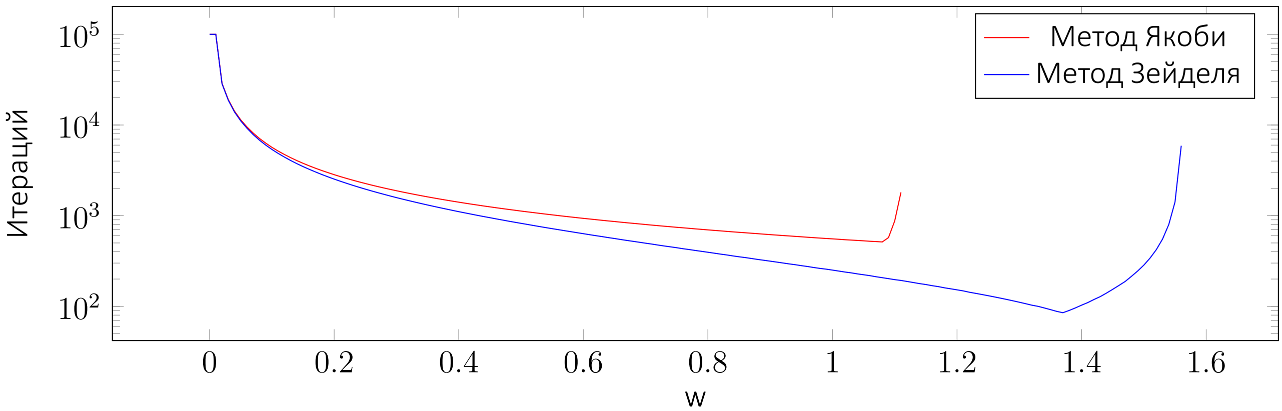
w	x	$x - x^*$	Относительная невязка	$cond(A) >$	Итераций
0.10	0.9999999999997365 1.9999999999994895 2.9999999999994222 3.9999999999994245 4.9999999999995381 5.9999999999994866 6.9999999999995008 7.9999999999994333 8.9999999999994529 9.9999999999994351	2.6e-13 5.1e-13 5.8e-13 5.8e-13 4.6e-13 5.1e-13 5.0e-13 5.7e-13 5.5e-13 5.6e-13	9.9e-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	9.9e-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	1.0e-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	1.0e-14	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	9.8e-15	7.92	823
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	9.7e-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	9.5e-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	9.4e-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	9.5e-15	6.43	315

1.00	0.9999999999997509 1.9999999999995151 2.9999999999994529 3.9999999999994547 4.9999999999995612 5.9999999999995133 6.9999999999995266 7.9999999999994609 8.9999999999994760 9.9999999999994618	2.5e-13 4.8e-13 5.5e-13 5.5e-13 4.4e-13 4.9e-13 4.7e-13 5.4e-13 5.2e-13 5.4e-13	9.7e-15	8.14	555
1.07	0.9999999999997543 1.9999999999995248 2.9999999999994631 3.9999999999994653 4.9999999999995692 5.9999999999995222 6.9999999999995355 7.9999999999994698 8.9999999999994849 9.9999999999994724	2.5e-13 4.8e-13 5.4e-13 5.3e-13 4.3e-13 4.8e-13 4.6e-13 5.3e-13 5.2e-13 5.3e-13	9.4e-15	8.26	518
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
1.09	1.0000000000000042 1.9999999999999687 2.9999999999999876 4.0000000000000027 4.9999999999999689 6.0000000000000195 6.9999999999999796 7.9999999999999911 8.9999999999999947 9.9999999999999662	-4.2e-15 3.1e-14 1.2e-14 -2.7e-15 3.1e-14 -2.0e-14 2.0e-14 8.9e-15 5.3e-15 3.4e-14	1.0e-14	0.33	573
1.10	1.0000000000000113 1.9999999999999800 3.0000000000000013 4.0000000000000124 4.9999999999999813 6.0000000000000284 6.9999999999999893 8.0000000000000053 9.0000000000000053 9.9999999999999822	-1.1e-14 2.0e-14 -1.3e-15 -1.2e-14 1.9e-14 -2.8e-14 1.1e-14 -5.3e-15 -5.3e-15 1.8e-14	9.1e-15	0.27	875

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	9.0e-15	5.84	251
1.07	0.9999999999998176 1.9999999999996478 2.9999999999996079 3.9999999999996327 4.9999999999997087 5.9999999999996820 6.9999999999996811 7.9999999999996554 8.9999999999996732 9.9999999999996749	1.8e-13 3.5e-13 3.9e-13 3.7e-13 2.9e-13 3.2e-13 3.2e-13 3.4e-13 3.3e-13 3.3e-13	9.9e-15	5.32	212
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	9.6e-15	5.30	207
1.09	0.9999999999998253 1.9999999999996623 2.9999999999996239 3.9999999999996474 4.9999999999997220 5.9999999999996962 6.9999999999996945 7.9999999999996705 8.9999999999996891 9.9999999999996909	1.7e-13 3.4e-13 3.8e-13 3.5e-13 2.8e-13 3.0e-13 3.1e-13 3.3e-13 3.1e-13 3.1e-13	9.6e-15	5.22	202
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	9.6e-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	8.7e-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	9.3e-15	3.31	111
1.36	0.9999999999999248 1.9999999999998550 2.9999999999998419 3.9999999999998672 4.9999999999998979 5.9999999999998881 6.9999999999998828 7.9999999999998854 8.9999999999998952 9.9999999999999005	7.5e-14 1.4e-13 1.6e-13 1.3e-13 1.0e-13 1.1e-13 1.2e-13 1.1e-13 1.0e-13 9.9e-14	7.4e-15	2.59	88
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	9.1e-15	1.49	85
1.38	1.0000000000000264 2.0000000000000280 2.9999999999999640 4.0000000000000036 5.0000000000000364 6.0000000000000373 6.9999999999999574 7.9999999999999813 9.0000000000000107 10.0000000000000195	-2.6e-14 -2.8e-14 3.6e-14 -3.6e-15 -3.6e-14 -3.7e-14 4.3e-14 1.9e-14 -1.1e-14 -2.0e-14	6.6e-15	0.70	90

						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	7.3e-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	8.5e-15	0.79	285

4.1.1 График зависимости числа итераций от параметра релаксации



4.1.2 Расчет числа обусловленности через MathCad

$$conde(A) = 133.86$$
$$condi(A) = 111.728$$
$$cond1(A) = 200$$
$$cond2(A) = 74.622$$
$$\frac{\lambda^A_{max}}{\lambda^A_{min}} = \frac{13.709}{0.278} = 49.313$$

4.2 Матрица с обратным знаком внедиагональных элементов

$$B = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 13 & 4 & 0 & 0 & 4 & 0 & 2 & 0 & 0 \\ 0 & 0 & 7 & 3 & 0 & 0 & 2 & 0 & 2 & 0 \\ 0 & 0 & 3 & 8 & 2 & 0 & 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 2 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 3 & 7 & 1 & 0 \\ 0 & 0 & 2 & 0 & 4 & 0 & 0 & 3 & 9 & 0 \\ 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 4 & 9 \end{pmatrix}, X = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}, F = \begin{pmatrix} 9 \\ 81 \\ 65 \\ 81 \\ 41 \\ 19 \\ 56 \\ 98 \\ 131 \\ 154 \end{pmatrix}$$

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

Метод Якоби					
w	x	$x - x^*$	Относительная невязка	$cond(A) >$	Итераций
0.10	0.9999999999999448 1.9999999999999298 2.9999999999995093 4.0000000000006404 4.9999999999995248 6.0000000000004841 7.0000000000002656 7.9999999999995683 9.0000000000005063 9.9999999999993410	5.5e-14 7.0e-14 4.9e-13 -6.4e-13 4.8e-13 -4.8e-13 -2.7e-13 4.3e-13 -5.1e-13 6.6e-13	1.0e-14	7.35	1249
0.20	0.9999999999997924 2.0000000000001532 2.9999999999992975 4.0000000000007567 4.9999999999995675 6.0000000000003331 7.0000000000005098 7.9999999999993401 9.0000000000006466 9.9999999999993125	2.1e-13 -1.5e-13 7.0e-13 -7.6e-13 4.3e-13 -3.3e-13 -5.1e-13 6.6e-13 -6.5e-13 6.9e-13	9.6e-15	9.22	618
0.30	0.9999999999996179 2.0000000000004308 2.9999999999991189 4.0000000000007976 4.9999999999996909 6.0000000000000782 7.0000000000007621 7.9999999999991331 9.0000000000007425 9.9999999999993783	3.8e-13 -4.3e-13 8.8e-13 -8.0e-13 3.1e-13 -7.8e-14 -7.6e-13 8.7e-13 -7.4e-13 6.2e-13	9.7e-15	10.63	409
0.40	0.9999999999995401 2.0000000000006173 2.9999999999992113 4.0000000000005675 4.9999999999999485 5.9999999999997415 7.0000000000008020 7.9999999999991864 9.0000000000005933 9.9999999999996767	4.6e-13 -6.2e-13 7.9e-13 -5.7e-13 5.2e-14 2.6e-13 -8.0e-13 8.1e-13 -5.9e-13 3.2e-13	9.7e-15	9.62	308
0.50	0.9999999999995763 2.0000000000006102 2.9999999999993903 4.0000000000003553 5.0000000000000906 5.9999999999996110 7.0000000000006883 7.9999999999993481 9.0000000000004192 9.9999999999998810	4.2e-13 -6.1e-13 6.1e-13 -3.6e-13 -9.1e-14 3.9e-13 -6.9e-13 6.5e-13 -4.2e-13 1.2e-13	9.5e-15	8.14	247
0.60	0.9999999999996063 2.0000000000005911 2.9999999999995000 4.0000000000002345 5.0000000000001625 5.9999999999995524 7.0000000000006111 7.9999999999994520 9.0000000000003180 9.9999999999999947	3.9e-13 -5.9e-13 5.0e-13 -2.3e-13 -1.6e-13 4.5e-13 -6.1e-13 5.5e-13 -3.2e-13 5.3e-15	9.5e-15	7.21	205
0.70	0.9999999999996703 2.0000000000005156 2.9999999999996398 4.0000000000001084 5.0000000000002069 5.9999999999995488 7.0000000000004858 7.9999999999995888 9.0000000000002025 10.0000000000000853	3.3e-13 -5.2e-13 3.6e-13 -1.1e-13 -2.1e-13 4.5e-13 -4.9e-13 4.1e-13 -2.0e-13 -8.5e-14	8.8e-15	6.41	175
0.80	0.9999999999997253 2.0000000000004499 2.9999999999997562 4.0000000000000115 5.0000000000002363 5.9999999999995532 7.0000000000003819 7.9999999999997051 9.0000000000001066 10.0000000000001581	2.7e-13 -4.5e-13 2.4e-13 -1.2e-14 -2.4e-13 4.5e-13 -3.8e-13 2.9e-13 -1.1e-13 -1.6e-13	8.3e-15	5.71	152
0.90	0.9999999999997036 2.0000000000005018 2.9999999999997837 3.9999999999999458 5.0000000000003126 5.9999999999994564 7.0000000000003917 7.9999999999997238 9.0000000000000622 10.0000000000002469	3.0e-13 -5.0e-13 2.2e-13 5.4e-14 -3.1e-13 5.4e-13 -3.9e-13 2.8e-13 -6.2e-14 -2.5e-13	9.7e-15	5.45	133

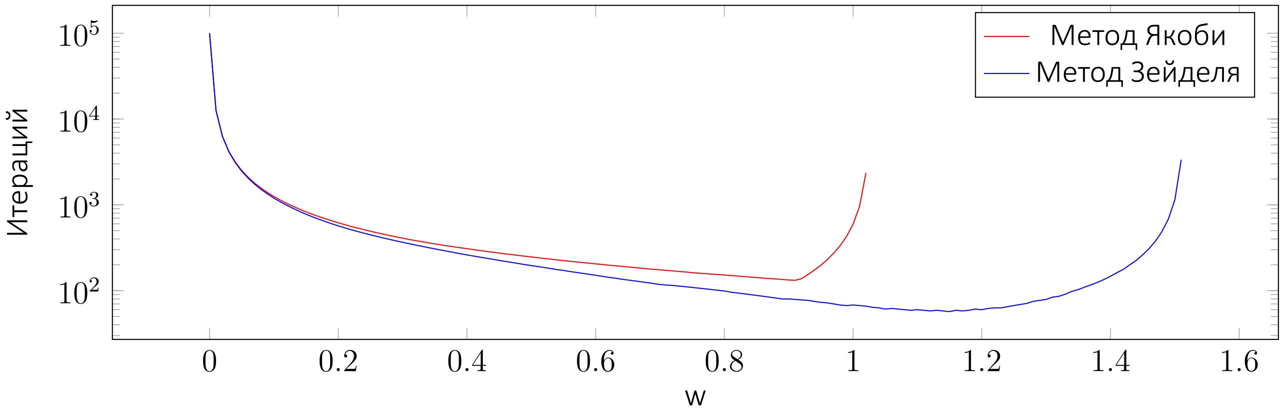
Метод Зейделя					
w	x	$x - x^*$	Относительная невязка	$cond(A) >$	Итераций
0.10	0.9999999999998730 2.0000000000000338 2.9999999999993920 4.0000000000007159 4.9999999999995328 6.0000000000004166 7.0000000000003917 7.9999999999994449 9.0000000000005844 9.9999999999993232	1.3e-13 -3.4e-14 6.1e-13 -7.2e-13 4.7e-13 -4.2e-13 -3.9e-13 5.6e-13 -5.8e-13 6.8e-13	9.9e-15	8.21	1198
0.20	0.9999999999995949 2.0000000000004694 2.9999999999990785 4.0000000000008127 4.9999999999997042 6.0000000000000391 7.0000000000007976 7.9999999999991083 9.0000000000007478 9.9999999999994067	4.1e-13 -4.7e-13 9.2e-13 -8.1e-13 3.0e-13 -3.9e-14 -8.0e-13 8.9e-13 -7.5e-13 5.9e-13	9.7e-15	10.90	569
0.30	0.9999999999995429 2.0000000000006515 2.9999999999993157 4.0000000000004032 5.0000000000000773 5.9999999999996154 7.0000000000007301 7.9999999999993188 9.0000000000004423 9.9999999999998668	4.6e-13 -6.5e-13 6.8e-13 -4.0e-13 -7.7e-14 3.8e-13 -7.3e-13 6.8e-13 -4.4e-13 1.3e-13	9.8e-15	8.42	366
0.40	0.9999999999996304 2.0000000000005715 2.9999999999995834 4.0000000000001297 5.0000000000002141 5.9999999999995408 7.0000000000005151 7.9999999999995852 9.0000000000002007 10.0000000000000870	3.7e-13 -5.7e-13 4.2e-13 -1.3e-13 -2.1e-13 4.6e-13 -5.2e-13 4.1e-13 -2.0e-13 -8.7e-14	9.7e-15	6.20	261
0.50	0.9999999999997562 2.0000000000004174 2.9999999999998570 3.9999999999998987 5.0000000000002824 5.9999999999995666 7.0000000000002665 7.9999999999998570 8.9999999999999840 10.0000000000002292	2.4e-13 -4.2e-13 1.4e-13 1.0e-13 -2.8e-13 4.3e-13 -2.7e-13 1.4e-13 1.6e-14 -2.3e-13	9.0e-15	4.64	196
0.60	0.9999999999999404 2.0000000000001767 3.0000000000002229 3.9999999999996207 5.0000000000003348 5.9999999999996518 6.9999999999999281 8.0000000000002061 8.9999999999997264 10.0000000000003677	6.0e-14 -1.8e-13 -2.2e-13 3.8e-13 -3.3e-13 3.5e-13 7.2e-14 -2.1e-13 2.7e-13 -3.7e-13	8.4e-15	5.14	151
0.70	1.0000000000004183 1.9999999999994906 3.0000000000009850 3.9999999999991811 5.0000000000002647 6.0000000000000453 6.9999999999991829 8.0000000000008722 8.9999999999993072 10.0000000000004601	-4.2e-13 5.1e-13 -9.8e-13 8.2e-13 -2.6e-13 -4.5e-14 8.2e-13 -8.7e-13 6.9e-13 -4.6e-13	9.5e-15	11.08	118
0.80	1.0000000000001437 1.9999999999997418 3.0000000000000266 4.0000000000001412 4.9999999999997859 6.0000000000002691 6.9999999999999014 7.9999999999999956 9.0000000000000817 9.9999999999998188	-1.4e-13 2.6e-13 -2.7e-14 -1.4e-13 2.1e-13 -2.7e-13 9.9e-14 4.4e-15 -8.2e-14 1.8e-13	7.1e-15	3.78	99
0.90	0.9999999999996279 2.0000000000005511 2.9999999999994373 4.0000000000002709 5.0000000000000995 5.9999999999996732 7.0000000000005222 7.9999999999995781 9.0000000000002416 9.9999999999999964	3.7e-13 -5.5e-13 5.6e-13 -2.7e-13 -9.9e-14 3.3e-13 -5.2e-13 4.2e-13 -2.4e-13 3.6e-15	9.0e-15	6.84	80

0.91	0.9999999999997643 2.0000000000003961 2.9999999999998503 3.9999999999999067 5.0000000000002727 5.9999999999995275 7.0000000000002860 7.9999999999997993 9.0000000000000071 10.0000000000002274	2.4e-13 -4.0e-13 1.5e-13 9.3e-14 -2.7e-13 4.7e-13 -2.9e-13 2.0e-13 -7.1e-15 -2.3e-13	7.7e-15	5.62	132
0.92	0.9999999999999978 1.9999999999999263 3.0000000000000084 3.9999999999998774 4.9999999999999876 5.9999999999999289 6.9999999999998970 8.0000000000000071 8.9999999999998863 10.0000000000000018	2.2e-15 7.4e-14 -8.4e-15 1.2e-13 1.2e-14 7.1e-14 1.0e-13 -7.1e-15 1.1e-13 -1.8e-15	8.9e-15	1.27	138
1.00	1.0000000000000275 2.0000000000000546 3.0000000000000591 4.0000000000000622 5.0000000000000488 6.0000000000000560 7.0000000000000542 8.0000000000000586 9.0000000000000604 10.0000000000000586	-2.8e-14 -5.5e-14 -5.9e-14 -6.2e-14 -4.9e-14 -5.6e-14 -5.4e-14 -5.9e-14 -6.0e-14 -5.9e-14	9.7e-15	0.91	595

0.91	0.9999999999996774 2.0000000000005023 2.9999999999996070 4.0000000000001092 5.0000000000001705 5.9999999999996536 7.0000000000003917 7.9999999999997273 9.0000000000001137 10.0000000000000870	3.2e-13 -5.0e-13 3.9e-13 -1.1e-13 -1.7e-13 3.5e-13 -3.9e-13 2.7e-13 -1.1e-13 -8.7e-14	9.5e-15	5.17	79
0.92	0.9999999999997662 2.0000000000003832 2.9999999999997917 3.9999999999999769 5.0000000000001945 5.9999999999996989 7.0000000000002345 7.9999999999998801 9.0000000000000071 10.0000000000001350	2.3e-13 -3.8e-13 2.1e-13 2.3e-14 -1.9e-13 3.0e-13 -2.3e-13 1.2e-13 -7.1e-15 -1.4e-13	8.5e-15	4.09	78
1.00	1.0000000000001998 1.9999999999996729 3.0000000000001941 4.0000000000000098 4.9999999999998446 6.0000000000002407 6.9999999999998046 8.0000000000000924 8.9999999999999964 9.9999999999998916	-2.0e-13 3.3e-13 -1.9e-13 -9.8e-15 1.6e-13 -2.4e-13 2.0e-13 -9.2e-14 3.6e-15 1.1e-13	7.5e-15	3.90	68
1.10	1.0000000000001097 1.9999999999997926 3.0000000000000244 4.0000000000001226 4.9999999999998463 6.0000000000001634 6.9999999999999671 7.9999999999999440 9.0000000000000853 9.9999999999998810	-1.1e-13 2.1e-13 -2.4e-14 -1.2e-13 1.5e-13 -1.6e-13 3.3e-14 5.6e-14 -8.5e-14 1.2e-13	7.1e-15	2.76	60
1.14	1.0000000000000333 2.0000000000000209 3.0000000000002847 3.9999999999996687 5.0000000000001688 5.9999999999999494 6.9999999999997700 8.0000000000002878 8.9999999999997744 10.0000000000001350	-3.3e-14 -2.1e-14 -2.8e-13 3.3e-13 -1.7e-13 5.1e-14 2.3e-13 -2.9e-13 2.3e-13 -1.4e-13	8.2e-15	4.09	58
1.15	1.0000000000003704 1.9999999999995046 3.0000000000007843 3.9999999999994986 5.0000000000000195 6.0000000000002940 6.9999999999993570 8.0000000000005471 8.9999999999996785 10.0000000000000195	-3.7e-13 5.0e-13 -7.8e-13 5.0e-13 -2.0e-14 -2.9e-13 6.4e-13 -5.5e-13 3.2e-13 -2.0e-14	9.9e-15	7.58	57
1.16	0.9999999999999343 2.0000000000000568 2.9999999999997615 4.0000000000002212 4.9999999999999210 5.9999999999999796 7.0000000000001972 7.9999999999997922 9.0000000000001474 9.9999999999999361	6.6e-14 -5.7e-14 2.4e-13 -2.2e-13 7.9e-14 2.0e-14 -2.0e-13 2.1e-13 -1.5e-13 6.4e-14	4.5e-15	5.38	59
1.20	0.9999999999998656 2.0000000000001279 2.9999999999995715 4.0000000000003775 4.9999999999998908 5.9999999999999183 7.0000000000003659 7.9999999999996323 9.0000000000002398 9.9999999999999325	1.3e-13 -1.3e-13 4.3e-13 -3.8e-13 1.1e-13 8.2e-14 -3.7e-13 3.7e-13 -2.4e-13 6.8e-14	8.0e-15	5.35	60
1.30	0.9999999999998268 2.0000000000001878 2.9999999999995537 4.0000000000003508 4.9999999999999529 5.9999999999998233 7.0000000000004077 7.9999999999996350 9.0000000000001972 10.0000000000000284	1.7e-13 -1.9e-13 4.5e-13 -3.5e-13 4.7e-14 1.8e-13 -4.1e-13 3.7e-13 -2.0e-13 -2.8e-14	8.6e-15	5.18	79
1.40	0.9999999999998201 2.0000000000002327 2.9999999999996545 4.0000000000001883 5.0000000000000604 5.9999999999997646 7.0000000000003144 7.9999999999997886 9.0000000000000568 10.0000000000001386	1.8e-13 -2.3e-13 3.5e-13 -1.9e-13 -6.0e-14 2.4e-13 -3.1e-13 2.1e-13 -5.7e-14 -1.4e-13	6.7e-15	5.22	147

						1.50	1.0000000000002334 1.9999999999996918 3.0000000000004157 3.9999999999997984 4.9999999999998836 6.0000000000003437 6.9999999999996145 8.0000000000002274 8.9999999999999911 9.9999999999997407	-2.3e-13 3.1e-13 -4.2e-13 2.0e-13 1.2e-13 -3.4e-13 3.9e-13 -2.3e-13 8.9e-15 2.6e-13	9.6e-15	4.62	1137

4.2.1 График зависимости числа итераций от параметра релаксации



4.2.2 Расчет числа обусловленности через MathCad

$conde(B) = 44.931$

$condi(B) = 34.897$

$cond1(B) = 53.051$

$cond2(B) = 20.536$

$$\frac{\lambda^B_{max}}{\lambda^B_{min}} = \frac{14.284}{0.93} = 15.359$$

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 один ярко выраженный минимум, что позволяет подобрать его благодаря методам поиска минимума, либо различным эвристикам.

Так же было оценено число обусловленности по относительной невязке и погрешности: $cond(A) > 8.54$, $cond(B) > 10.90$. Смотря на расчет числа обусловленности через специальные программы, можем заметить, что в реальности оно в несколько раз больше, чем было оценено.