Laboratorium 7

Artem Buhera
GĆ01 135678

03.05.2021

Dany laboratorium polega na rozwiązywaniu poniższego układu równań w postaci $Ax=b$ przy użyciu metod iteracyjnych Jacobiego, Gaussa-Seidela oraz SOR z parametrem ω=0.5:

$$\begin{pmatrix} 100 & -1 & 2 & -3 \\ 1 & 200 & -4 & 5 \\ -2 & 4 & 300 & -6 \\ 3 & -5 & 6 & 400 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 116 \\ -226 \\ 912 \\ -1174 \end{pmatrix} \quad \text{dla} \quad \vec{x}_0 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Zaczynając od początkowego wektora $\vec{x}_0$ obliczamy kolejne przybliżenia wektora rozwiązań $\vec{x}_n$

We wszystkich trzech metodach najpierw rozkładamy macierz A na sumę L + D + U, gdzie L, D oraz U - odpowiednio macierz dolnotrójkątna, diagonalna oraz górnotrójkatna.

Obliczanie kończymy przy przekroczeniu maksymalnej ilości iteracji bądź przy spełnieniu dwóch kryteriów:

- Kryterium dokładności wyznaczenia x_n: $\|\vec{x}_n - \vec{x}_{n-1}\|_\infty \leq TOLX$
- Kryterium wiarygodności x_n jako przybliżenia wektora rozwiązań: $\|A\vec{x}_n - \vec{b}\|_\infty \leq TOLF$

Metoda Jacobiego

Wzór operacyjny: $D\vec{x}_n = -(L+U)\vec{x}_{n-1} + \vec{b}$

$\vec{x}_n$ wyznaczamy poprzez rozwiązanie układu równań z macierzą diagonalną D

Metoda Gaussa-Seidela

Wzór operacyjny: $(L+D)\vec{x}_n = -U\vec{x}_{n-1} + \vec{b}$

$\vec{x}_n$ wyznaczamy poprzez rozwiązanie układu z macierzą dolnątrójkątną $L+D$

Metoda SOR

Wzór operacyjny: $\left(L+\frac{1}{\omega}D\right)\vec{x}_n = -\left(\left(1-\frac{1}{\omega}\right)D + U\right)\vec{x}_{n-1} + \vec{b}$

$\vec{x}_n$ analogicznie wyznaczamy poprzez rozwiązanie układu z macierzą dolnątrójkątną $L+\frac{1}{\omega}D$

Klasy Matrix oraz Vector napisane do reprezentacji macierzy i wektorów użyte w programie:

matrix.h

```
#include "vector.h"

#ifndef MATRIX_H
#define MATRIX_H
using namespace std;

template<int N, int M>
class Matrix : array<Vector<N>, M> {
public:
    Matrix<N, M>() = default;
    Matrix<N, M>(array<Vector<N>, M> a) : array<Vector<N>, M> (a) {}

    using array<Vector<N>, M>::operator[];

    void swapRows(int a, int b) {
        swap((*this)[a], (*this)[b]);
    };

    void print(int width=8) const {
        for (Vector row : *this)
            row.print(width);
        std::cout << std::endl;
    }

    void print(string text, int width=8) const {
        std::cout << text << std::endl;
        this->print(width);
    }

    Matrix<N, M> operator*(double scalar) {
        Matrix<N, M> result;
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                result[i][j] = (*this)[i][j] * scalar;
        return result;
    }

    template <int, int>
    friend Vector<N> operator*(const Matrix<N, M> m, const Vector<M> &x);

    Matrix<N, M> operator+(const Matrix<N, M> &other) {
        Matrix<N, M> result;
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                result[i][j] = (*this)[i][j] + other[i][j];
        return result;
    }
```

```cpp
    Matrix<N, M> operator*(const Matrix<N, M> &other) {
        Matrix<N, M> result;

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                double sum = 0.0;
                for (int k = 0; k < M; k++) {
                    sum += (*this)[i][k] * other[k][j];
                }
                result[i][j] = sum;
            }
        }

        return result;
    }
};

template <int N, int M>
Vector<N> operator*(const Matrix<N, M> m, const Vector<M> &x) {
    Vector<N> result;
    for (int i = 0; i < N; i++) {
        double sum = 0.0;
        for (int j = 0; j < M; j++)
            sum += m[i][j] * x[j];
        result[i] = sum;
    }
    return result;
}

#endif
```

vector.h

```
#include <array>
#include <algorithm>
#include <ostream>
#include <iostream>
#include <iomanip>

#ifndef VECTOR_H
#define VECTOR_H
using namespace std;

template <int N>
class Vector: public array<double,N> {
public:
    template <int>
    friend Vector<N> operator-(const Vector<N> &left, const Vector<N> &right);

    template <int>
    friend Vector<N> &operator+(Vector<N> &left, const Vector<N> &right);

    Vector<N> operator*(double scalar) {
        Vector<N> result;
        for(int i = 0; i < N; ++i)
            result[i] = (*this)[i] + scalar;
        return result;
    }

    void swapElements(int a, int b) {
        std::swap((*this)[a], (*this)[b]);
    }

    void print(int width=8) {
        for (auto value : (*this))
            std::cout << std::setw(width) << value << " ";
        std::cout << std::endl;
    }

    void print(const std::string &text, int width=10) {
        std::cout << text << std::endl;
        print(width);
    }

    template <int>
    friend ostream& operator<<(ostream &os, const Vector<N> &vect);

    double normMax() {
        return *std::max_element((*this).begin(), (*this).end());
    }
};

template <int N>
ostream &operator<<(ostream &os, const Vector<N> &vect) {
    os << '[';
    for (double value : vect)
        os << setw(11) << setprecision(8) << value << ", ";
    os << "\b\b]";

    return os;
}
```

```cpp
template <int N>
Vector<N> operator-(const Vector<N> &left, const Vector<N> &right) {
    Vector<N> result;
    for(int i = 0; i < N; i++)
        result[i] = left[i] - right[i];
    return result;
}

template <int N>
Vector<N> &operator+(Vector<N> &left, const Vector<N> &right) {
    Vector<N> result;
    for(int i = 0; i < N; i++)
        result[i] = left[i] + right[i];
    return result;
}

#endif
```

Kod głównego programu:

```cpp
#include <iostream>
#include <vector.h>
#include <matrix.h>

using namespace std;


#define NMAX 30
#define TOLF 1e-8
#define TOLX 1e-8

template <int N>
void LDU_decomposition(const Matrix<N, N> &A, Matrix<N, N> &L, Matrix<N, N> &D,
                       Matrix<N, N> &U) {
    L = {{0}};
    D = {{0}};
    U = {{0}};

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double value = A[i][j];
            if (i > j)
                L[i][j] = value;
            else if (i == j)
                D[i][j] = value;
            else
                U[i][j] = value;
        }
    }
}

template <int N>
Vector<N> solveDiagonal(const Matrix<N, N> &M, const Vector<N> &y) {
    Vector<N> x;
    for (int i = 0; i < N; i++)
        x[i] = y[i] / M[i][i];
    return x;
}

template <int N, int M>
Vector<N> solveLowerTriangular(const Matrix<N, M> &L, const Vector<N> &y) {
    Vector<N> x;
    for (int n = 0; n < N; n++) {
        double sum = 0.0;
        for (int j = 0; j < n; j++)
            sum += L[n][j] * x[j];
        x[n] = (y[n] - sum) / L[n][n];
    }
    return x;
}

void printHeader(int N) {
    cout << " n   " << setw(14*N) << "xn    " << "    residuum      estimator" << endl;
}

template <int N>
void printIteration(int n, const Vector<N> &xn, double residuum, double estimator) {
    cout << setw(2) << n << "    " << xn << " " << setw(14)
         << residuum << " " << setw(14) << estimator << endl;
}
```

```cpp
template <int N>
Vector<N> solveJacobi(Matrix<N, N> &A, Vector<N> &b, Vector<N> &x0) {
    Vector<N> x = x0, xnext, y;
    Matrix<N, N> L, D, U;
    double residuum, estimator;

    LDU_decomposition(A, L, D, U);

    Matrix LU = L + U;

    printHeader(N);
    cout << " 0   " << x << endl;

    for (int n = 1; n <= NMAX; n++) {
        // D * xnext = -(L + U) * x + b
        Vector<N> D_xnext = b - (LU * x);

        xnext = solveDiagonal(D, D_xnext);

        residuum  = (A * xnext - b).normMax();
        estimator = (xnext - x).normMax();

        x = xnext;

        printIteration(n, x, residuum, estimator);

        if (residuum < TOLF && estimator < TOLX) {
            cout << "TOLF i TOLX po " << n << " iteracjach" << endl;
            return x;
        }
    }

    cout << "przekroczono NMAX" << endl;
    return x;
}

template <int N>
Vector<N> solveGaussSeidel(const Matrix<N, N> &A, const Vector<N> &b, const Vector<N>
&x0) {
    Vector<N> x = x0, xnext, y;
    Matrix<N, N> L, D, U;
    double residuum, estimator;

    LDU_decomposition(A, L, D, U);

    Matrix LD = L + D;

    printHeader(N);
    cout << " 0   " << x << endl;

    for (int n = 1; n <= NMAX; n++) {
        // (L + D) * xnext = -U * x + b
        Vector<N> LD_xnext = b - U * x;

        xnext = solveLowerTriangular(LD, LD_xnext);

        residuum  = (A * xnext - b).normMax();
        estimator = (xnext - x).normMax();

        x = xnext;

        printIteration(n, x, residuum, estimator);
```

```cpp
        if (residuum < TOLF && estimator < TOLX) {
            cout << "TOLF i TOLX po " << n << " iteracjach" << endl;
            return x;
        }
    }

    cout << "przekroczono NMAX" << endl;
    return x;
}

template <int N>
Vector<N> solveSOR(const Matrix<N, N> &A, const Vector<N> &b, const Vector<N> &x0, double
omega) {
    Vector<N> x = x0, xnext, y;
    Matrix<N, N> L, D, U;
    double residuum, estimator;

    LDU_decomposition(A, L, D, U);

    // L + 1/omega*D
    Matrix L_omega_D =  L + (D * (1/omega));
    // (1 - 1/omega)*D + U
    Matrix omega_DU = (D * (1 - 1/omega)) + U;

    printHeader(N);
    cout << " 0    " << x << endl;

    for (int n = 1; n <= NMAX; n++) {
        // (L + 1/omega*D) * xnext = -[(1 - 1/omega)*D + U] * x + b
        Vector<N> L_omega_D_xnext  = b - (omega_DU * x);

        xnext = solveLowerTriangular(L_omega_D, L_omega_D_xnext);

        residuum  = (A * xnext - b).normMax();
        estimator = (xnext - x).normMax();

        x = xnext;

        printIteration(n, x, residuum, estimator);

        if (residuum < TOLF && estimator < TOLX) {
            cout << "TOLF i TOLX po " << n << " iteracjach" << endl;
            return x;
        }
    }

    cout << "przekroczono NMAX" << endl;
    return x;
}

int main() {
    Matrix<4, 4> A = {{
        100.0, -1.0, 2.0, -3.0,
        1.0, 200.0, -4.0, 5.0,
        -2.0, 4.0, 300.0, -6.0,
        3.0, -5.0, 6.0, 400.0
    }};

    Vector<4> b = {{116.0, -226.0, 912.0, -1174.0}};
    Vector<4> x0 = {{2.0, 2.0, 2.0, 2.0}};
```

```
    cout << "Metoda Jacobi:" << endl;
    solveJacobi(A,b, x0);

    cout << endl << "Metoda Gaussa-Seidela:" << endl;
    solveGaussSeidel(A, b, x0);

    cout << endl << "Metoda SOR z parametrem 0.75:" << endl;
    solveSOR(A, b, x0, 0.75);

    cout << endl << "Metoda SOR z parametrem 0.5:" << endl;
    solveSOR(A, b, x0, 0.5);
}
```

Wynik działania programu:

```
Metoda Jacobi:
 n                                                         xn        residuum       estimator
 0   [          2,           2,           2,            2]
 1   [        1.2,       -1.15,   3.0666667,       -2.955]       20.148333       1.0666667
 2   [ 0.99851667, -1.0007917,   3.0042333,    -3.004375]         1.29605      0.14920833
 3   [ 0.99977617, -0.99979854,   2.9999132,   -3.0000623]     0.040103813    0.0043127292
 4   [  1.0000019, -0.99999906,   2.9999946,   -2.9999945]    0.0021682022   0.00022571646
 5   [  1.0000003,  -1.0000003,   3.0000001,   -2.9999999]   3.4375044e-05   5.5337701e-06
 6   [          1,          -1,           3,           -3]   2.1028574e-06    2.541894e-07
 7   [          1,          -1,           3,           -3]   6.4384352e-08   6.8638948e-09
 8   [          1,          -1,           3,           -3]   3.4769982e-09   3.6122372e-10
TOLF i TOLX po 8 iteracjach

Metoda Gaussa-Seidela:
 n                                                         xn        residuum       estimator
 0   [          2,           2,           2,            2]
 1   [        1.2,      -1.146,     3.10328,   -3.0048742]       30.029245         1.10328
 2   [ 0.99632817, -0.99779419,   2.9998486,   -2.9999426]       0.4383834      0.14820581
 3   [  1.0000268,  -1.0000046,   3.0000014,   -3.0000003]    0.0026889167    0.0036986331
 4   [ 0.99999992, -0.99999996,           3,           -3]   6.9793075e-06   4.6312048e-06
 5   [          1,          -1,           3,           -3]   5.2052798e-08   8.2610947e-08
 6   [          1,          -1,           3,           -3]   1.4742341e-10   1.6423263e-10
TOLF i TOLX po 6 iteracjach

Metoda SOR z parametrem 0.75:
 n                                                         xn        residuum       estimator
 0   [          2,           2,           2,            2]
 1   [        1.4,    -0.36025,   2.8206025,   -1.7442341]       499.23122       0.8206025
 2   [  1.1357438, -0.86680811,   2.9733339,   -2.6852734]       125.47191      0.15273141
 3   [  1.0424162,  -0.9731622,   2.9979981,   -2.9212828]        31.46792     0.024664167
 4   [  1.0126065,  -0.9948438,   3.0006917,   -2.9803511]       7.8757652    0.0026936681
 5   [   1.003622, -0.99908257,   3.0004766,   -2.9951049]       1.9671787 -0.00021514125
 6   [  1.0010154, -0.99985909,   3.0001962,   -2.9987828]      0.49038978 -0.00028036104
 7   [  1.0002793,  -0.9999857,   3.0000686,   -2.9996979]      0.12201218 -0.00012661197
 8   [  1.0000757,  -1.0000013,   3.0000221,   -2.9999252]     0.030300118  -1.5646506e-05
 9   [  1.0000203,  -1.0000015,   3.0000068,   -2.9999815]    0.0075106576  -1.4036406e-07
10   [  1.0000054,  -1.0000006,    3.000002,   -2.9999954]    0.0018582974   8.4729936e-07
11   [  1.0000014,  -1.0000002,   3.0000006,   -2.9999989]    0.0004589501   4.1666481e-07
12   [  1.0000004,  -1.0000001,   3.0000002,   -2.9999997]   0.00011314527   1.5128858e-07
13   [  1.0000001,          -1,           3,   -2.9999999]   2.7844169e-05   4.8547488e-08
14   [          1,          -1,           3,           -3]   6.8401221e-06    1.454699e-08
15   [          1,          -1,           3,           -3]   1.6773695e-06   4.1698562e-09
16   [          1,          -1,           3,           -3]   4.1061162e-07   1.1580408e-09
17   [          1,          -1,           3,           -3]   1.0033978e-07   3.1390757e-10
18   [          1,          -1,           3,           -3]   2.4476549e-08   8.3437923e-11
19   [          1,          -1,           3,           -3]   5.9601462e-09    2.181233e-11
TOLF i TOLX po 19 iteracjach
```

```
Metoda SOR z parametrem 0.5:
 n                                                      xn      residuum     estimator
 0   [         2,          2,          2,          2]
 1   [       1.6,      0.426,  2.5424933, -0.4899062]     995.96248     0.54249333
 2   [ 1.3493565, -0.32382463,  2.7930043, -1.7404846]    500.23137     0.25051096
 3   [ 1.1990218, -0.68022377,  2.9076289, -2.3682973]    251.12505     0.11462457
 4   [  1.111509, -0.84921065,  2.9594979, -2.6833206]    126.00933     0.05186903
 5   [ 1.0616637,  -0.929123,  2.9826488, -2.8413184]     63.199132     0.02315088
 6   [   1.03374, -0.96680288,  2.9928024, -2.9205243]    31.682341    0.010153582
 7   [ 1.0183001, -0.98451261,  2.9971537, -2.9602126]    15.875339   0.0043513309
 8   [ 1.0098527, -0.99280674,  2.9989596, -2.9800905]    7.9511515   0.0018059184
 9   [ 1.0052714, -0.99667582,  2.9996743, -2.9900418]    3.9805205  0.00071470308
10   [ 1.0028049, -0.99847266,  2.9999359, -2.9950214]    1.9918375  0.00026159595
11   [ 1.0014854, -0.99930292,   3.000018,  -2.997512]   0.99626152  8.2138707e-05
12   [ 1.0007833, -0.99968434,  3.0000344, -2.9987572]   0.49808027  1.6365309e-05
13   [ 1.0004115, -0.99985839,  3.0000301, -2.9993795]   0.24890453 -4.3483223e-06
14   [ 1.0002155, -0.99993719,  3.0000215, -2.9996903]    0.1243296 -8.5249612e-06
15   [ 1.0001125, -0.99997253,  3.0000141, -2.9998455]  0.062076224 -7.4784348e-06
16   [ 1.0000586,  -0.9999882,  3.0000087,  -2.999923]  0.030980297 -5.3664122e-06
17   [ 1.0000304, -0.99999505,  3.0000052, -2.9999616]  0.015454531 -3.5058786e-06
18   [ 1.0000158, -0.99999799,   3.000003, -2.9999809] 0.0077061306 -2.1685445e-06
19   [ 1.0000081, -0.99999923,  3.0000017, -2.9999905]   0.00384085 -1.2323056e-06
20   [ 1.0000042, -0.99999973,   3.000001, -2.9999953] 0.0019135017 -4.9921235e-07
21   [ 1.0000022, -0.99999992,  3.0000005, -2.9999976] 0.00095288562 -1.9224382e-07
22   [ 1.0000011, -0.99999999,  3.0000003, -2.9999988] 0.00047431072 -6.8014009e-08
23   [ 1.0000006,         -1,  3.0000002, -2.9999994] 0.00023599077 -2.0249137e-08
24   [ 1.0000003,         -1,  3.0000001, -2.9999997] 0.00011736445  -3.398283e-09
25   [ 1.0000001,         -1,          3, -2.9999999]  5.834277e-05  1.5854331e-09
26   [ 1.0000001,         -1,          3, -2.9999999] 2.8989833e-05  2.3946278e-09
27   [         1,         -1,          3,         -3] 1.4398318e-05   1.977551e-09
28   [         1,         -1,          3,         -3] 7.1479947e-06  1.3682773e-09
29   [         1,         -1,          3,         -3] 3.5470066e-06   8.684562e-10
30   [         1,         -1,          3,         -3] 1.7593161e-06  5.2361004e-10
przekroczono NMAX
```

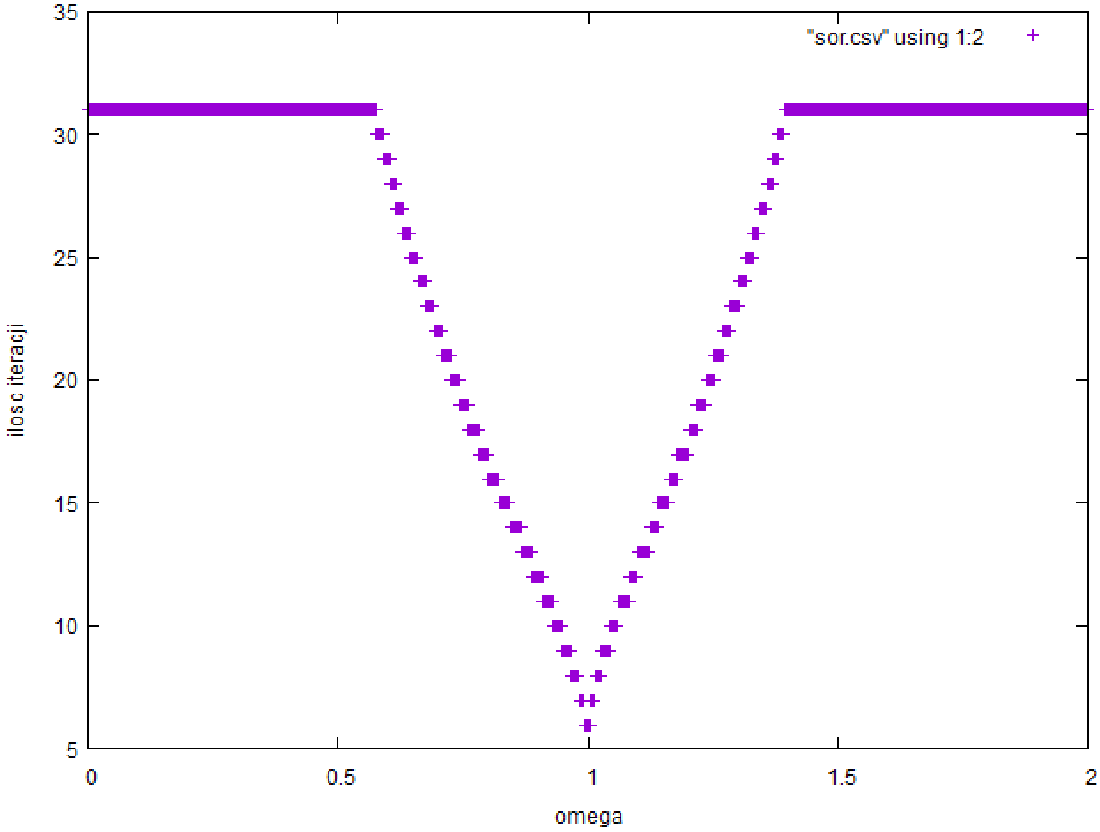
Używając metody Gaussa-Seidela oraz Jacobiego znaleźliśmy rozwiązania dosyć szybko - po 6 i 8 iteracjach.

W przypadku metody SOR możemy zauważyć, że im bardziej parametr omega się różni od 1 - tym więcej iteracji potrzebuje program. Możemy sprawdzić tą zależność po zmodyfikowaniu programu tak, żeby funkcja `solveSOR` zwracała ilość wykonanych iteracji i sporządzimy wykres zależności ilości iteracji od parametru ω.

Zmodyfikowane fragmenty kodu:

```cpp
template <int N>
Vector<N> solveSOR(const Matrix<N, N> &A, const Vector<N> &b, const Vector<N> &x0, double
omega, int &iterations) {
    Vector<N> x = x0, xnext, y;
    Matrix<N, N> L, D, U;
    double residuum, estimator;

    LDU_decomposition(A, L, D, U);

    // L + 1/omega*D
    Matrix L_omega_D =  L + (D * (1/omega));
    // (1 - 1/omega)*D + U
    Matrix omega_DU = (D * (1 - 1/omega)) + U;


    int n = 1;
    for (; n <= NMAX; n++) {
        // (L + 1/omega*D) * xnext = -[(1 - 1/omega)*D + U] * x + b
        Vector<N> L_omega_D_xnext  = b - (omega_DU * x);

        xnext = solveLowerTriangular(L_omega_D, L_omega_D_xnext);

        residuum  = (A * xnext - b).normMax();
        estimator = (xnext - x).normMax();

        x = xnext;

        if (residuum < TOLF && estimator < TOLX) {
            iterations = n;
            return x;
        }
    }

    iterations = n;
    return x;
}

int main() {
    Matrix<4, 4> A = {{
        100.0, -1.0, 2.0, -3.0,
        1.0, 200.0, -4.0, 5.0,
        -2.0, 4.0, 300.0, -6.0,
        3.0, -5.0, 6.0, 400.0
    }};

    Vector<4> b = {{116.0, -226.0, 912.0, -1174.0}};
    Vector<4> x0 = {{2.0, 2.0, 2.0, 2.0}};

    ofstream out;
    out.open("../lab7/sor.csv");

    for (double omega = 0.001; omega < 2.0; omega += 0.001) {
        int iterations;
        solveSOR(A, b, x0, omega, iterations);
        out << omega << "," << iterations << endl;
    }

    out.close();
}
```

Otrzymany wykres:



Jak widać, najlepsze wyniki dla danego układu równań są dla ω ≈ 1 - 6 iteracji. Tyle samo uzyskaliśmy przy użyciu metody Gaussa-Seidela, ponieważ przy ω = 1 metoda SOR działa w ten sam sposób. Niewielkie zmiany ω powodują szybki wzrost liczby iteracji i przy ω < 0.6 oraz ω > 1.4 program już przekracza ustalony limit.