Julia Plewa
MOwNiT
Laboratorium 5

1. **Proszę zaimplementować metodę Jacobiego oraz przetestować jej działanie na kilku znalezionych przez siebie układach równań (nie mniej niż 5 układów, nie więcej niż 10, w miarę możliwości różnorodne). Układy podać w sprawozdaniu.**

Kod:

```
#define K_MAX 1000
#define EPSILON 1e-10

template<typename T>
AGHMatrix<T> AGHMatrix<T>::jacobiMethod(int equation_id, std::ostream
&convergence_file)
{
 // result vector - the initial estimation is 0
 AGHMatrix<T> x(get_rows(), 1, 0);
 // the next estimation of the result
 std::vector<double> y(get_rows(), 0);
 // K_MAX denotes the maximum number of iterations
 for (int k = 0; k < K_MAX; k++)
 {
   // copy current result estimations into a temporary vector
   for (int i = 0; i < get_rows(); i++)
   {
     y[i] = x.matrix[i][0];
   }
   // calculate a new estimation of the variables based on the previous one
   for (int i = 0; i < get_rows(); i++)
   {
     T sum = matrix[i][get_cols() - 1];
     T diag = matrix[i][i];
     for (int j = 0; j < get_rows(); j++)
     {
       if (j != i)
       {
         sum -= matrix[i][j] * y[j];
       }
     }
```

```
      x.matrix[i][0] = sum / diag;
    }
    T norm = 0;
    T max_y = 0;
    for (int i = 0; i < get_rows(); i++)
    {
      T diff = fabs(y[i] - x.matrix[i][0]);
      if (diff > norm)
      {
        norm = diff;
      }
      if (y[i] > max_y)
      {
        max_y = y[i];
      }
    }
    if (equation_id != -1)
    {
      convergence_file << equation_id << ", " << k << ", " << norm << std::endl;
    }
    // return if convergence was reached
    if (norm < EPSILON * max_y)
    {
        std::cout << "CONVERGENCE REACHED IN ITERATION: " << k << std::endl;
        return x;
    }
  }
  std::cout << "ITERATION LIMIT REACHED WITH NO CONVERGENCE" << std::endl;
  return x;
}
```

Uwaga: W kodzie uwzględnione jest również zapisywanie informacji o zbieżności rozwiązania przygotowane na potrzeby dalszych zadań.

Opis:

Metoda Jacobiego to metoda przybliżania rozwiązania układu równań liniowych dla macierzy przekątniowo dominujących, czyli takich, dla których

$$\forall\ 1 \le i \le N\ :\ \left|a_{ii}\right| \ge \sum_{j=1,\, j\neq i}^{N} \left|a_{ij}\right| \ .$$

Metoda Jacobiego opiera się na rozbiciu wejściowej macierzy A na macierz diagonalną D i macierz reszty R:

$$A = D + R\,.$$

Zapis macierzowy kroku iteracji metody wygląda następująco:

$$x^{(k+1)} = D^{-1}\ (b - Rx^{(k)})\,,$$

gdzie x^k to macierz w k-tej iteracji.

Po rozbiciu wzoru na kolejne współrzędne wzór przyjmuje następującą postać:

$$x_i^{(k+1)} = \tfrac{1}{a_{ii}} \left(b_i - \sum_{j=1,\, j\neq i}^{n} a_{ij} x_j^{(k)} \right) \ .$$

Metoda Jacobiego wymaga podania początkowego oszacowania wartości niewiadomych. W powyższej implementacji wykorzystywany jest do tego wektor wypełniony zerami. Na podstawie oszacowanych wyników w każdej kolejnej iteracji obliczane są nowe szacunki - algorytm kończy działanie w przypadku, gdy przekroczony zostanie limit iteracji lub gdy osiąga zbieżność, czyli kiedy pewna metryka tego wektora osiągnie zadaną wartość. W literaturze można się spotkać między innymi z normą maksimową lub euklidesową wektora $x^{(k+1)} - x^{(k)}$ oraz normą maksimową wektora reszta. W powyższej implementacji wybrany został warunek:

$$max_i \left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon\ max_i \left|x_i^{(k)}\right| .$$

Ponadto przyjęte zostało założenie, że macierz na wejściu jest przekątniowo dominująca.

Przykłady działania:

	układ równań	**iteracja**	**rozwiązanie**
1	$\left[\begin{array}{cc\|c} 2 & 1 & 11 \\ 5 & 7 & 13 \end{array}\right]$	45	$\begin{bmatrix} 7.11111 \\ -3.22222 \end{bmatrix}$
2	$\left[\begin{array}{ccc\|c} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{array}\right]$	29	$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$
3	$\left[\begin{array}{ccc\|c} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{array}\right]$	36	$\begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$
4	$\left[\begin{array}{ccc\|c} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$	8	$\begin{bmatrix} 3 \\ -2.5 \\ 7 \end{bmatrix}$
5	$\left[\begin{array}{cccc\|c} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{array}\right]$	27	$\begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$
6	$\left[\begin{array}{ccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 \\ 2 & -1 & 15 & -1 & -11 & 10 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 \end{array}\right]$	471	$\begin{bmatrix} -0.723201 \\ -0.864777 \\ 1.3109 \\ 0.381906 \\ 0.790906 \end{bmatrix}$
7	$\left[\begin{array}{cccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 \end{array}\right]$	90	$\begin{bmatrix} -0.291411 \\ 0.00223246 \\ 0.49996 \\ 0.18339 \\ 0.463313 \\ 0.136549 \end{bmatrix}$

8	$\left[\begin{array}{ccccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 & 1 & 6 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 & 5 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 & 4 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 & 3 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 & 2 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 10 & 0 \end{array}\right]$	340	$\begin{bmatrix} 0.0746492 \\ 0.282422 \\ 1.00119 \\ 0.310072 \\ 0.727873 \\ -0.00956921 \\ -0.824115 \end{bmatrix}$

2. Proszę zaimplementować metodę Gaussa-Seidela oraz przetestować jej działanie na układach równań z poprzedniego zadania.

Kod:

```
template<typename T>
AGHMatrix<T> AGHMatrix<T>::gaussSeidelMethod(int equation_id, std::ostream
&convergence_file){
 return sorMethod(1.0, equation_id, convergence_file);
}
```

Uwaga: Do implementacji metody Gaussa-Seidela wykorzystano fakt, że jest ona równoważna z metodą SOR dla $\omega = 1$.

Opis:

Metoda Gaussa-Seidela opiera się na zmodyfikowaniu metody Jacobiego tak, aby w każdym momencie iteracji korzystać z najbardziej „aktualnych" przybliżeń rozwiązania. Macierz wejściową rozbija się na dolną macierz trójkątną L_* oraz ściśle górną macierz trójkątną U.

W postaci macierzowej rozwiązanie w $k+1$ iteracji ma postać:

$$x^{(k+1)} = L_*^{-1}\left(b - Ux^{(k)}\right).$$

To samo rozwiązanie po rozbiciu na poszczególne współrzędne ma postać:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right).$$

Przykłady działania:

	układ równań	**iteracja**	**rozwiązanie**
1	$\left[\begin{array}{cc\|c} 2 & 1 & 11 \\ 5 & 7 & 13 \end{array}\right]$	22	$\begin{bmatrix} 7.11111 \\ -3.22222 \end{bmatrix}$
2	$\left[\begin{array}{ccc\|c} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{array}\right]$	13	$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$
3	$\left[\begin{array}{ccc\|c} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{array}\right]$	16	$\begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$
4	$\left[\begin{array}{ccc\|c} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$	6	$\begin{bmatrix} 3 \\ -2.5 \\ 7 \end{bmatrix}$
5	$\left[\begin{array}{cccc\|c} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{array}\right]$	10	$\begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$
6	$\left[\begin{array}{ccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 \\ 2 & -1 & 15 & -1 & -11 & 10 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 \end{array}\right]$	23	$\begin{bmatrix} -0.723201 \\ -0.864777 \\ 1.3109 \\ 0.381906 \\ 0.790906 \end{bmatrix}$
7	$\left[\begin{array}{cccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 \end{array}\right]$	28	$\begin{bmatrix} -0.291411 \\ 0.00223246 \\ 0.49996 \\ 0.18339 \\ 0.463313 \\ 0.136549 \end{bmatrix}$

8	$\left[\begin{array}{ccccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 & 1 & 6 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 & 5 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 & 4 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 & 3 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 & 2 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 10 & 0 \end{array}\right]$	40	$\begin{bmatrix} 0.0746492 \\ 0.282422 \\ 1.00119 \\ 0.310072 \\ 0.727873 \\ -0.00956921 \\ -0.824115 \end{bmatrix}$

3. Proszę zaimplementować metodę SOR oraz przetestować jej działanie na układach równań z poprzedniego zadania.

Kod:

```
template<typename T>
AGHMatrix<T> AGHMatrix<T>::sorMethod(T omega, int equation_id, std::ostream
&convergence_file)
{
 // result vector - the initial estimation is 0
 AGHMatrix<T> x(get_rows(), 1, 0);
 // the next estimation of the result
 std::vector<double> y(get_rows(), 0);
 // K_MAX denotes the maximum number of iterations
 for (int k = 0; k < K_MAX; k++)
 {
   // copy current result estimations into a temporary vector
   for (int i = 0; i < get_rows(); i++)
   {
     y[i] = x.matrix[i][0];
   }
   // calculate a new estimation of the variables based on the previous one
   for (int i = 0; i < get_rows(); i++)
   {
     T sum = matrix[i][get_cols() - 1];
     T diag = matrix[i][i];
     for (int j = 0; j < i; j++)
     {
       sum -= matrix[i][j] * x.matrix[j][0];
```

```
    }
    for (int j = i + 1; j < get_rows(); j++)
    {
      sum -= matrix[i][j] * x.matrix[j][0];
    }
    x.matrix[i][0] = sum / diag;
    x.matrix[i][0] = omega * x.matrix[i][0] + (1 - omega) * y[i];
  }
  T norm = 0;
  T max_y = 0;
  for (int i = 0; i < get_rows(); i++)
  {
    T diff = fabs(y[i] - x.matrix[i][0]);
    if (diff > norm)
    {
      norm = diff;
    }
    if (y[i] > max_y)
    {
      max_y = y[i];
    }
  }
  if (equation_id != -1)
  {
    convergence_file << equation_id << ", " << k << ", " << norm << std::endl;
  }
  // return if convergence was reached
  if (norm < EPSILON * max_y)
  {
      std::cout << "CONVERGENCE REACHED IN ITERATION: " << k << std::endl;
      return x;
  }
 }
 std::cout << "ITERATION LIMIT REACHED WITH NO CONVERGENCE" << std::endl;
 return x;
}
```

Opis:

Metoda SOR (successive over-relaxation) jest modyfikacją metody Gaussa–Seidela rozszerzoną o dodatkowy parametr relaksacji ω - kolejne współrzędne nowego przybliżenia wyznaczane są poprzez kombinację poprzedniego przybliżenia $x_i^{(k)}$ oraz współrzędną nowego przybliżenia $\widehat{x}_i^{(k+1)}$.

Macierz wejściowa rozbijana jest na trzy części: macierz przekątną D, macierz dolną ściśle trójkątną L oraz macierz górną ściśle przekątną U. W formie macierzowej metoda SOR jest opisana wzorem:

$$x^{(k+1)} = (D + \omega L)^{-1} (\omega b - [\omega U + (\omega - 1) D] x^k),$$

a po rozbiciu na współrzędne:

$$x_i^{(k+1)} = (1 - \omega) x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right).$$

Przykłady działania:

	układ równań	**iteracja**	**rozwiązanie**
1	$\left[\begin{array}{cc\|c} 2 & 1 & 11 \\ 5 & 7 & 13 \end{array}\right]$	13	$\begin{bmatrix} 7.11111 \\ -3.22222 \end{bmatrix}$
2	$\left[\begin{array}{ccc\|c} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{array}\right]$	13	$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$
3	$\left[\begin{array}{ccc\|c} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{array}\right]$	16	$\begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$
4	$\left[\begin{array}{ccc\|c} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$	11	$\begin{bmatrix} 3 \\ -2.5 \\ 7 \end{bmatrix}$

5	$\left[\begin{array}{cccc\|c} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{array}\right]$	12	$\begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$
6	$\left[\begin{array}{ccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 \\ 2 & -1 & 15 & -1 & -11 & 10 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 \end{array}\right]$	25	$\begin{bmatrix} -0.723201 \\ -0.864777 \\ 1.3109 \\ 0.381906 \\ 0.790906 \end{bmatrix}$
7	$\left[\begin{array}{cccccc\|c} 10 & -1 & 2 & 0 & 6 & 1 & 1 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 \end{array}\right]$	24	$\begin{bmatrix} -0.291411 \\ 0.00223246 \\ 0.49996 \\ 0.18339 \\ 0.463313 \\ 0.136549 \end{bmatrix}$
8	$\left[\begin{array}{cccccccc} 10 & -1 & 2 & 0 & 6 & 1 & 1 & 6 \\ -1 & 11 & -1 & 3 & 5 & -5 & 2 & 5 \\ 2 & -1 & 15 & -1 & -11 & 10 & 3 & 4 \\ 0 & 3 & -1 & 8 & 5.5 & 3.5 & 4 & 3 \\ -4 & 7.5 & -9 & 3 & 17 & -0.8 & 5 & 2 \\ 0 & 1 & 2 & 3 & 4 & 19 & 6 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 10 & 0 \end{array}\right]$	47	$\begin{bmatrix} 0.0746492 \\ 0.282422 \\ 1.00119 \\ 0.310072 \\ 0.727873 \\ -0.00956921 \\ -0.824115 \end{bmatrix}$

4. **Proszę o TEORETYCZNE porównanie powyższych metod (zasadniczo wystarczy przeczytać ze zrozumieniem wstęp teoretyczny i własnymi słowami je porównać). Proszę wziąć pod uwagę aspekt zbieżności oraz rozkładu na składowe macierze.**

   Opis:

   W przypadku metody Jacobiego obliczenie $(k+1)$-szego przybliżenia wymaga przechowywania w pamięci również przybliżenia k. W metodzie Gaussa-Seidela oraz metodzie SOR można zatem zmniejszyć o połowę zapotrzebowanie na pamięć operacyjną oraz dzięki zniwelowaniu konieczności kopiowania wektora przyspieszyć działanie programu (jednak ze względu na wybrany warunek zakończenia iteracji w powyższych implementacjach przechowywany jest również wektor k-ty).

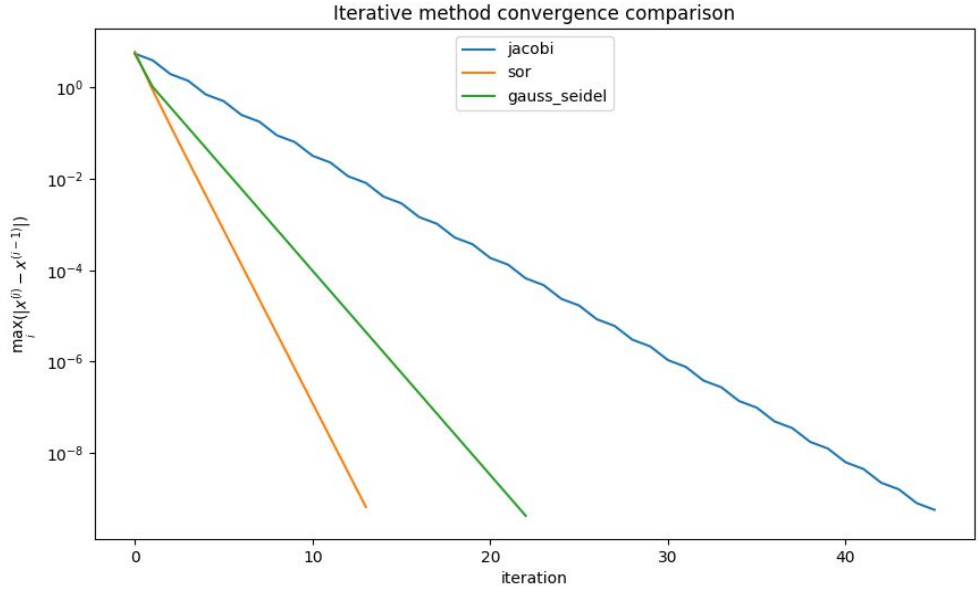
W odróżnieniu od metody Jacobiego, metody Gaussa-Seidela i SOR nie mogą być zaimplementowane z wykorzystaniem mechanizmów zrównoleglenia. Ponadto wynik $x^{(k)}$ jest zależny od kolejności, w której zostały rozważone równania.

Metoda Gaussa-Seidela osiąga zbieżność zauważalnie szybciej od metody Jacobiego. Metoda SOR osiąga zbieżność jeszcze szybciej, jednak konieczne jest dobranie optymalnego współczynnika relaksacji ω.

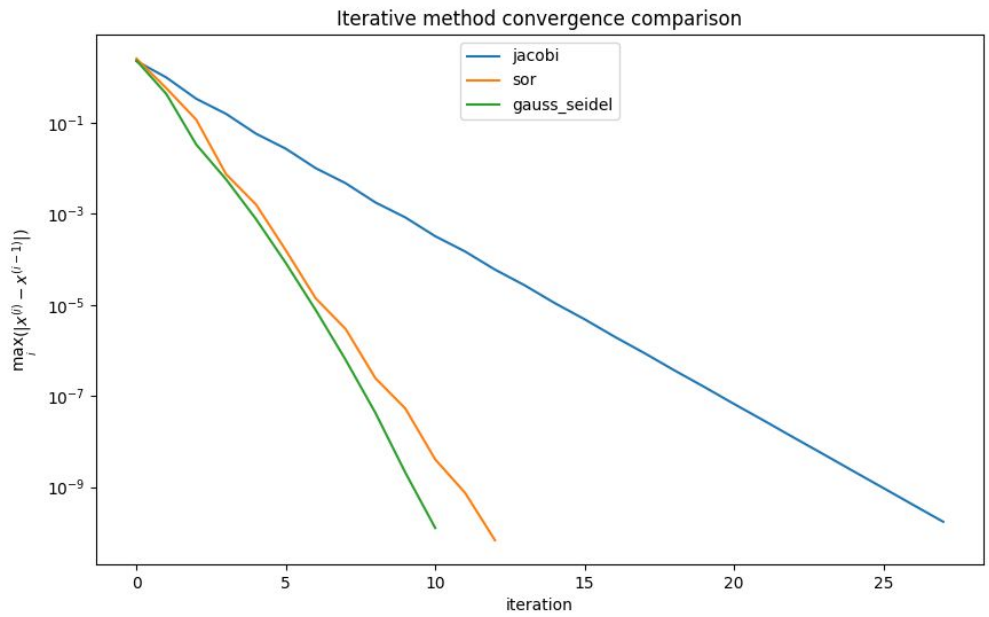
Metoda Jacobiego i metoda Gaussa-Seidela opierają się na rozbiciu macierzy na wejściu na dwie części, jednak w przypadku metody Jacobiego jedna z tych macierzy to sama diagonala, której odwrócenie jest bardzo proste. Metoda SOR opiera się na rozbiciu macierzy na trzy odrębne części.

5. Proszę dla powyższych metod porównać tempo zbiegania do rozwiązania (na wykresie). Co można zaobserwować i o czym to może świadczyć?

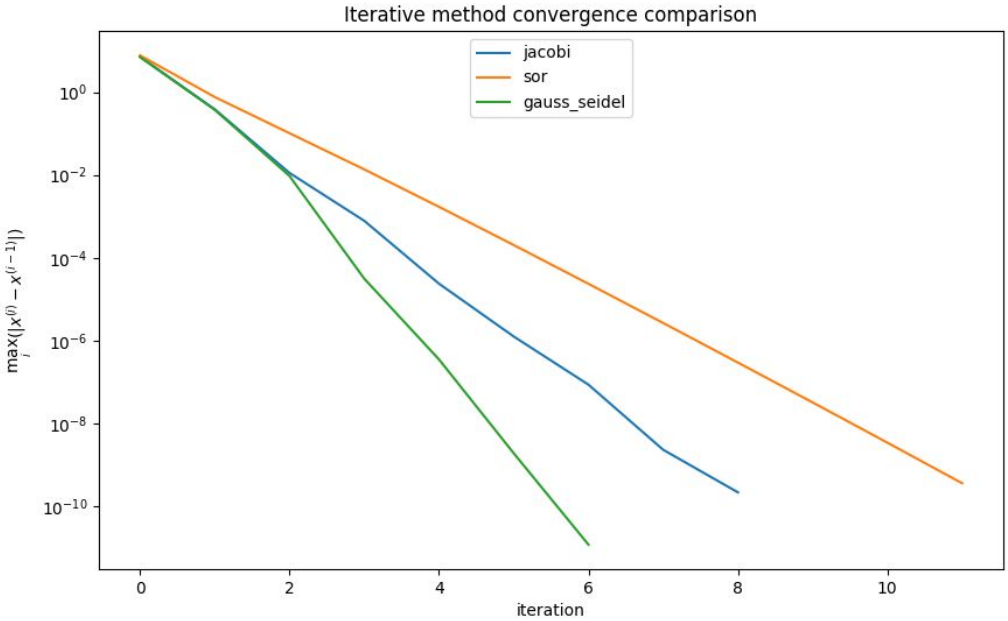
Wykresy:



Wykres 1. *Macierz 1*



Wykres 2. *Macierz 5*



Wykres 3. *Macierz 2*

Obserwacje:

Na powyższych trzech wykresach zaprezentowane zostały pomiary zbieżności dla wybranych trzech macierzy z poprzednich zadań dla parametru relaksacji metody SOR $\omega = 1,1$. Łatwo zauważyć, że parametr ten ma optymalne cechy w przypadku macierzy pierwszej. W przypadku macierzy piątej dla tego parametru metoda SOR wypada mniej optymalnie niż metoda Gaussa-Seidela, ale nadal bardziej optymalnie niż metoda Jacobiego. Dla macierzy drugiej parametr ten sprawia, że metoda SOR wypada najgorzej ze wszystkich trzech. Metoda Gaussa-Seidela zawsze wypada bardziej optymalnie od metody Jacobiego.