MOwNiT2 Lab5

Użyto implementacji macierzy z lab3 (AGHMatrix: dodano domyślny konstruktor).
Implementacje wszystkich metod są zawarte w poniższej klasie:
(metoda decompose() dokonuje podziału macierzy głównej na trzy i zapisuje je do pól klasy, można równie dobrze trzymać macierz w całości, ale skoro już je podzieliłem...)

```
namespace LinearEquations{

 class EquationSolver{
   public:
     EquationSolver(AGHMatrix<double> matrix, AGHMatrix<double> rightSide) :
            b(rightSide){
       if(matrix.get_rows() != matrix.get_cols() ||
            matrix.get_cols() != rightSide.get_rows()){
         std::string txt = "Invalid sizes.";
         throw std::invalid_argument(txt);
       }
       decompose(matrix);
     }

     // solving methods ...

   private:
     AGHMatrix<double> L;
     AGHMatrix<double> D;
     AGHMatrix<double> U;
     AGHMatrix<double> b;

     void decompose(AGHMatrix<double> A){

       AGHMatrix<double> lower(A.get_cols(), A.get_cols(), 0.0);
       AGHMatrix<double> diagonal(A.get_cols(), A.get_cols(), 0.0);
       AGHMatrix<double> upper(A.get_cols(), A.get_cols(), 0.0);

       for(int i=0; i<A.get_rows(); i++){
         for(int j=0; j<A.get_cols(); j++){
           if(i>j) lower(i,j) = A(i,j);
           else if(i<j) upper(i,j) = A(i,j);
           else diagonal(i,j) = A(i,j);
         }
       }
       L = lower;
       D = diagonal;
       U = upper;
     }
 };
} // namespace LinearEquations
```

Zadanie 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.

```
AGHMatrix<double> Jacobi(int iter, AGHMatrix<double> x){

  AGHMatrix<double> newX = x;
  // for each x
  for(int i=0; i<x.get_rows(); i++){
    double sum = 0.0;
    // summing
    for(int j=0; j<L.get_cols(); j++){
      if(i>j) sum += L(i,j)*x(j,0);
      else if(i<j) sum += U(i,j)*x(j,0);
    }
    newX(i,0) = (b(i,0) - sum)/D(i,i);
  }

  std::cout << std::setprecision(10) << "Iteration: " << iter <<
std::endl;
  for(int i=0; i<x.get_rows(); i++){
    std::cout << newX(i,0) << ", ";
  }
  std::cout << std::endl;
  bool check = true;
  for(int i=0; i<newX.get_rows(); i++){
    if(abs(newX(i,0) - x(i,0)) > 1e-10){
      check = false;
    }
  }
  if(check == true) return newX;
  else Jacobi(iter+1,newX);
}
```

We wszystkich metodach za warunek końca uznaję moment, gdy największa różnica między wartościami w kolejnych iteracjach będzie mniejsza od jakiejś stałej (tutaj 10^{-10}).

Macierz	Rozwiązanie	Iteracje
$\begin{matrix} 16 & 3 & 11 \\ 7 & -11 & 13 \end{matrix}$	$\begin{matrix} 0.8121827411 \\ -0.6649746193 \end{matrix}$	24
$\begin{matrix} 2 & 1 & 11 \\ 5 & 7 & 13 \end{matrix}$	$\begin{matrix} 7.111111111 \\ -3.222222222 \end{matrix}$	49
$\begin{matrix} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{matrix}$	$\begin{matrix} 1 \\ 2 \\ -1 \\ 1 \end{matrix}$	29
$\begin{matrix} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{matrix}$	$\begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$	39
$\begin{matrix} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{matrix}$	$\begin{matrix} 3 \\ 2 \\ 1 \end{matrix}$	32

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

```
AGHMatrix<double> GaussSeidel(int iter, AGHMatrix<double> x){

  AGHMatrix<double> newX = x;
  // for each x
  for(int i=0; i<x.get_rows(); i++){
    double sum = 0.0;
    // summing
    for(int j=0; j<L.get_cols(); j++){
      // newX in i-th iteration has new elements in 0:i-1 and old ones
           in i:N-1
      if(i>j) sum += L(i,j)*newX(j,0);
      else if(i<j) sum += U(i,j)*newX(j,0);
    }
    newX(i,0) = (b(i,0) - sum)/D(i,i);
  }

  std::cout << std::setprecision(10) << "Iteration: " << iter <<
std::endl;
  for(int i=0; i<x.get_rows(); i++){
    std::cout << newX(i,0) << ", ";
  }
  std::cout << std::endl;
  bool check = true;
  for(int i=0; i<newX.get_rows(); i++){
    if(abs(newX(i,0) - x(i,0)) > 1e-10){
      check = false;
      break;
    }
  }
  if(check == true) return newX;
  else GaussSeidel(iter+1,newX);
}
```

Macierz	Rozwiązanie	Iteracje
$\begin{matrix} 16 & 3 & 11 \\ 7 & -11 & 13 \end{matrix}$	$\begin{matrix} 0.8121827411 \\ -0.6649746193 \end{matrix}$	13
$\begin{matrix} 2 & 1 & 11 \\ 5 & 7 & 13 \end{matrix}$	$\begin{matrix} 7.111111111 \\ -3.222222222 \end{matrix}$	25
$\begin{matrix} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{matrix}$	$\begin{matrix} 1 \\ 2 \\ -1 \\ 1 \end{matrix}$	12
$\begin{matrix} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{matrix}$	$\begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$	17
$\begin{matrix} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{matrix}$	$\begin{matrix} 3 \\ 2 \\ 1 \end{matrix}$	15

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

```
AGHMatrix<double> SOR(int iter, AGHMatrix<double> x, double relaxation){

  AGHMatrix<double> newX = x;
  // for each x
  for(int i=0; i<x.get_rows(); i++){
    double sum = 0.0;
    // summing
    for(int j=0; j<L.get_cols(); j++){
      // newX in i-th iteration has new elements in 0:i-1 and old ones
           in i:N-1
      if(i>j) sum += L(i,j)*newX(j,0);
      else if(i<j) sum += U(i,j)*newX(j,0);
    }
    newX(i,0) = (1-relaxation)*newX(i,0) +
                 (b(i,0) - sum) / D(i,i) * relaxation;
  }

  std::cout << std::setprecision(10) << "Iteration: " << iter <<
std::endl;
  for(int i=0; i<x.get_rows(); i++){
    std::cout << newX(i,0) << ", ";
  }
  std::cout << std::endl;
  bool check = true;
  for(int i=0; i<newX.get_rows(); i++){
    if(abs(newX(i,0) - x(i,0)) > 1e-10){
      check = false;
      break;
    }
  }
  if(check == true) return newX;
  else SOR(iter+1,newX, relaxation);
}
```

Macierz	Rozwiązanie	Iteracje	Relaksacja
$\begin{matrix} 16 & 3 & 11 \\ 7 & -11 & 13 \end{matrix}$	$\begin{matrix} 0.8121827411 \\ -0.6649746193 \end{matrix}$	10	0.95
$\begin{matrix} 2 & 1 & 11 \\ 5 & 7 & 13 \end{matrix}$	$\begin{matrix} 7.111111111 \\ -3.222222222 \end{matrix}$	16	1.1
$\begin{matrix} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 3 & -1 & 8 & 15 \end{matrix}$	$\begin{matrix} 1 \\ 2 \\ -1 \\ 1 \end{matrix}$	12	1.05
$\begin{matrix} 12 & 3 & -5 & 1 \\ 1 & 5 & 3 & 28 \\ 3 & 7 & 13 & 76 \end{matrix}$	$\begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$	17	1.05
$\begin{matrix} 4 & 1 & 3 & 17 \\ 1 & 5 & 1 & 14 \\ 2 & -1 & 8 & 12 \end{matrix}$	$\begin{matrix} 3 \\ 2 \\ 1 \end{matrix}$	13	1.06

Ze względu na wybraną dokładność (przekłada się na ilość iteracji) nie zawsze byłem w stanie znaleźć współczynnik relaksacji, który zmniejszyłby ilości iteracji względem metody Gaussa-Seidela.

Zadanie 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.

Jacobi:

- Przelicza wartości dla każdej niewiadomej korzystając ze wzoru:
  $x_i^{(k+1)} = (b_i - \sum_{i \neq j} a_{ij} x_j^{(k)}) \; / \; a_{ii}$
- Macierz jest rozkładana na dwie składowe: diagonalną oraz zawierającą pozostałe elementy (suma ściśle dolnotrójkątnej i ściśle górnotrójkątnej)

Gauss-Seidel:

- Tutaj w celu przyspieszenia zbiegania do rozwiązania wartość dla danej niewiadomej jest wyliczana korzystając zawsze z najbardziej aktualnych obliczeń (jakbyśmy korzystali zawsze z jednej tablicy wyników, w której na bieżąco aktualizujemy wartości, metoda Jacobiego “podmieniała” tablice na końcu iteracji)
- Macierz jest rozkładana na dwie składowe: dolnotrójkątną i ściśle górnotrójkątną

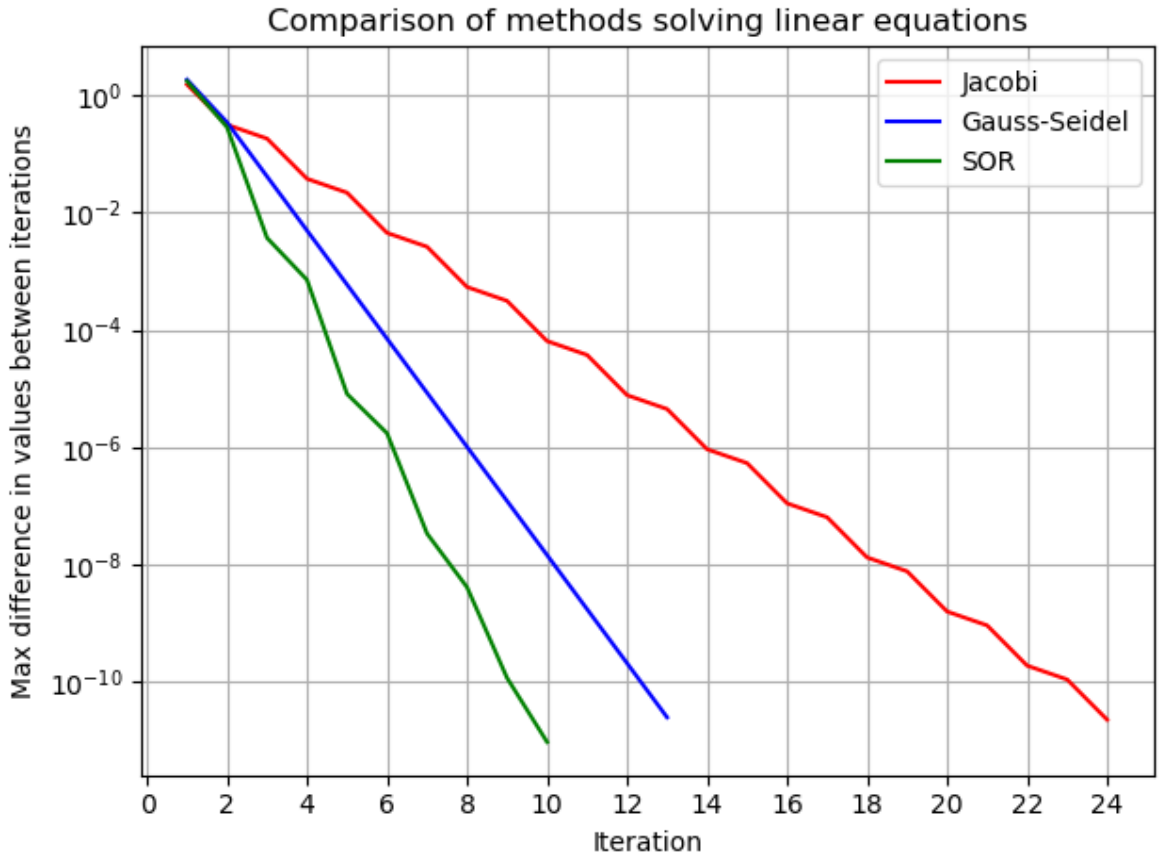
SOR:

- Kolejny dodatek do wzoru, współczynnik relaksacji, który, odpowiednio dobrany, może znacznie przyspieszyć tempo zbiegania do rozwiązania, konkretna wartość jest kombinacją poprzedniego i obecnego przyblbiżenia.
- Macierz jest rozkładana na trzy składowe: diagonalną, ściśle dolnotrójkątną i ściśle górnotrójkątną.

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

Wykres wykonano dla:

$$\begin{pmatrix} 16 & 3 & 11 \\ 7 & -11 & 13 \end{pmatrix}$$



Na podstawie wykresu można zauważyć, że dodawane do algorytmu “ulepszenia” poprawiają tempo zbiegania do rozwiązania.