Sprawozdanie - Metody numeryczne i optymailzacja

Jakub Andryszczak 259519,
Jakub Żak 244255,
Maciej Cierpisz 249163

Spis treści

1 Zadanie nr. 1

Rozwiązać ręcznie i komputerowo metodą eliminacji Gaussa poniższy układ równań liniowych. Znaleźć elementy podstawowe (pivots).

$$\begin{cases} 2u - v = 0 \\ -u + 2v - w = 0 \\ -v + 2w - z = 0 \\ -w + 2z = 5 \end{cases} . \tag{1}$$

Rozpocząć proces iteracyjny od zerowej wartości początkowej. Przedtaw krzywe błędów residualnych i aproksymacji rozwiązania (dwa rysunki). Porównaj z rozwiązaniem uzyskanym z eliminacji Gaussa.

Wykorzystano algorytmy metod iteracyjnych:

Landwebera – jako wartość parametru α przyjęto 0.14

Jacobiego – macierz A jest przekątniowo dominująca

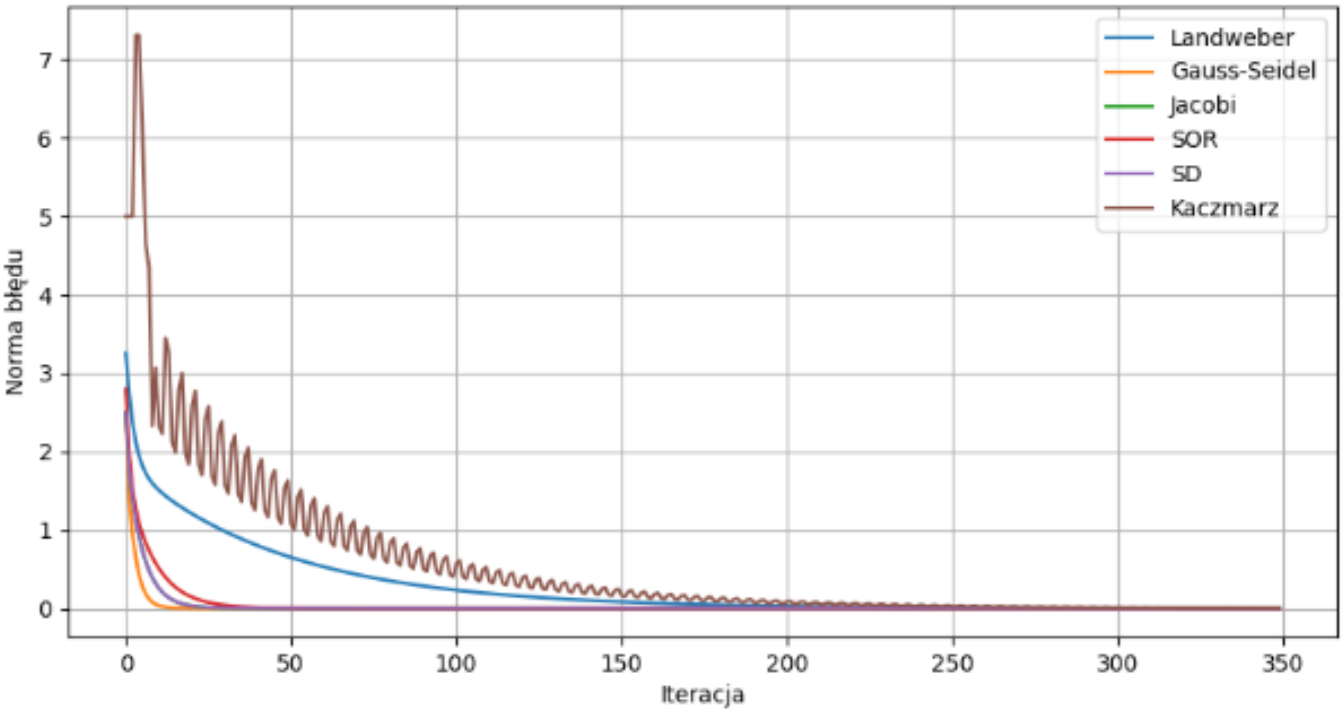
Gaussa-Seidela – podobnie jak metoda Jacobiego, ze względu na przekątinowo dominującą macierz

SOR – jako wartość współczynnika relaksacji przyjęto 0.5

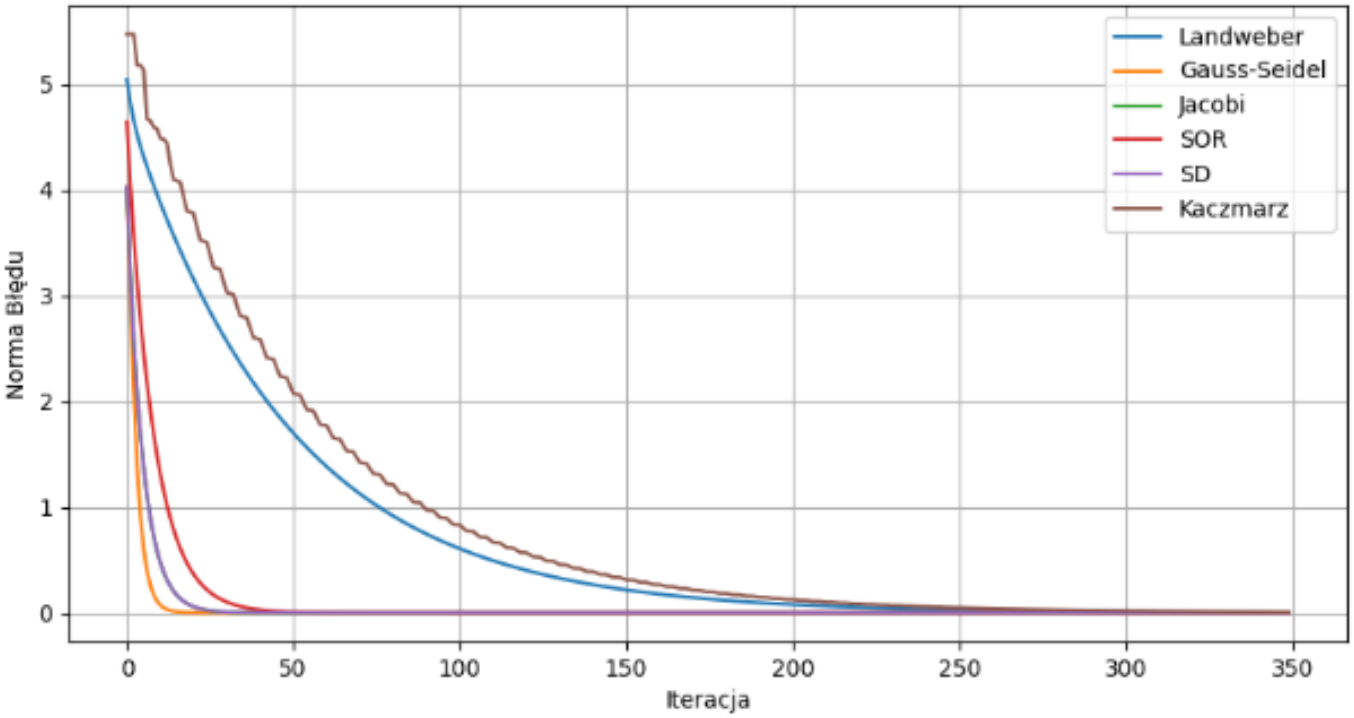
SD – macierz A jest dodatnio określona

Kaczmarza – jako współczynnik zbieżności metody, kontrolujący krok iteracji przyjęto 2.9

Dla każdego z algorytmów wykonano 350 iteracji.



Wykres 3.2. Zależność błędu rezydualnego do iteracji



Wykres 3.2. Zależność błędu rozwiązania do iteracji

2 Zadanie nr. 2

Rozwiązać następujący układ równań przy pomocy wybranych metod iteracyjnych:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ x_1 + x_2 + 2x_3 = 2 \\ x_1 + 2x_2 + 2x_3 = 1 \end{cases} . \tag{2}$$

Rozpocznij iteracje od $x^{(0)} = 0$. Przedstaw krzywe błędów residualnych i aproksymacji rozwiązania (dwa rysunki). Porównaj z rozwiązaniem uzyskanym z eliminacji Gaussa. Uzasadnij matematycznie (na podstawie promienia zbieżności) dlaczego nie udaje się uzyskać zbieżności niektórymi metodami iteracyjnymi.

W przypadku metody Jacobi wyliczono jego wektor własny

$$\begin{bmatrix} 2.1091 \\ -0.6498 \\ -1.4593 \end{bmatrix} \tag{3}$$

Wartością spektralną będzie maksymalny moduł wartości własnej, a w naszym wypadku będzie on wynosić p = 2.1091. Oznacza to, iż podczas wykorzystywania tej metody dla podanej macierzy nie jesteśmy w stanie dojść do rozwiązania.

W przypadku metody Gaussa-seidela sprawa wygląda podobnie ponieważ promień zbieżności również jest większy od wartości 1.

Metoda SOR jest zależna od współczynnika relaksacji ω. Nie udało się znaleźć optymalnej wartości co skutkowało nie znalezieniem żadnego rozwiązania, które spełniałoby warunek zbieżności dla promienia spektralnego.
Metoda Landweber'a zależy od parametru α, który kontroluje szybkość zbieżności. W tym wypadku wyznaczamy największą zmodułowaną wartość własną, a następnie podstawiamy do wzoru:

$$2 * |\lambda_{max}(A^T A)|^{-1} = 2 * 17.4887 = 0.1144 \tag{4}$$

Metodę Kaczmarza można wykorzystać dla tego układu równań ponieważ macierz A ma pełny rząd.
Obie metody dały wyniki przybliżone do rozwiązania uzyskanego za pomocą eliminacji Gaussa.

$$x = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \tag{5}$$

W przypadku metody Landweber'a liczba iteracji wyniosła 746, a dla znalezienia rozwiązania metodą Kaczmarza wystarczyło jedynie 190 iteracji. Poniżej przedstawiono wykres zależności błędu residualnego i rozwiązania dla metody Landweber'a i Kaczmarza.

3 Zadanie nr. 3

Rozwiąż układ równań liniowych: Ax = b, gdzie:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -1 & 1 & 2 & 1 \\ 0 & 2 & 1 & 3 \\ 0 & 0 & 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^T \tag{6}$$

Dla podanego układu równań ustalono, że podane metody iteracyjne nie zadziałają poprawnie:

Landwebera – brak zbieżności układu

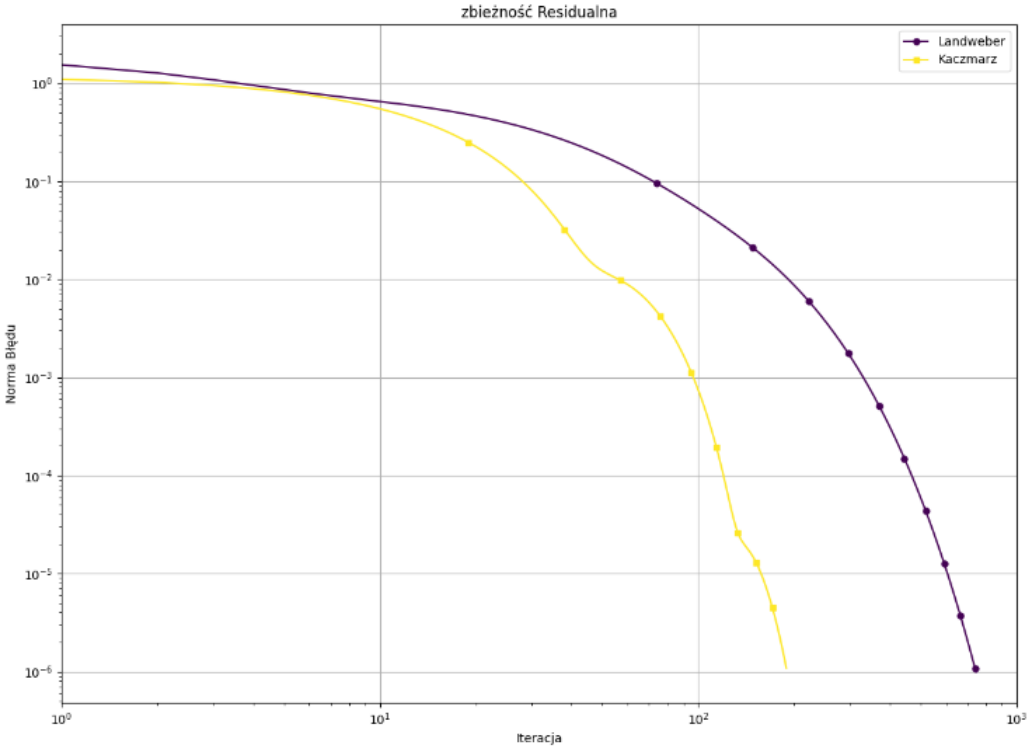
Jacobiego – brak diagonalnej ani przekątniowej dominacji

Gaussa-Seidela – analogicznie do metody Jacobiego

SOR – jako wartość współczynnika relaksacji przyjęto 0.5

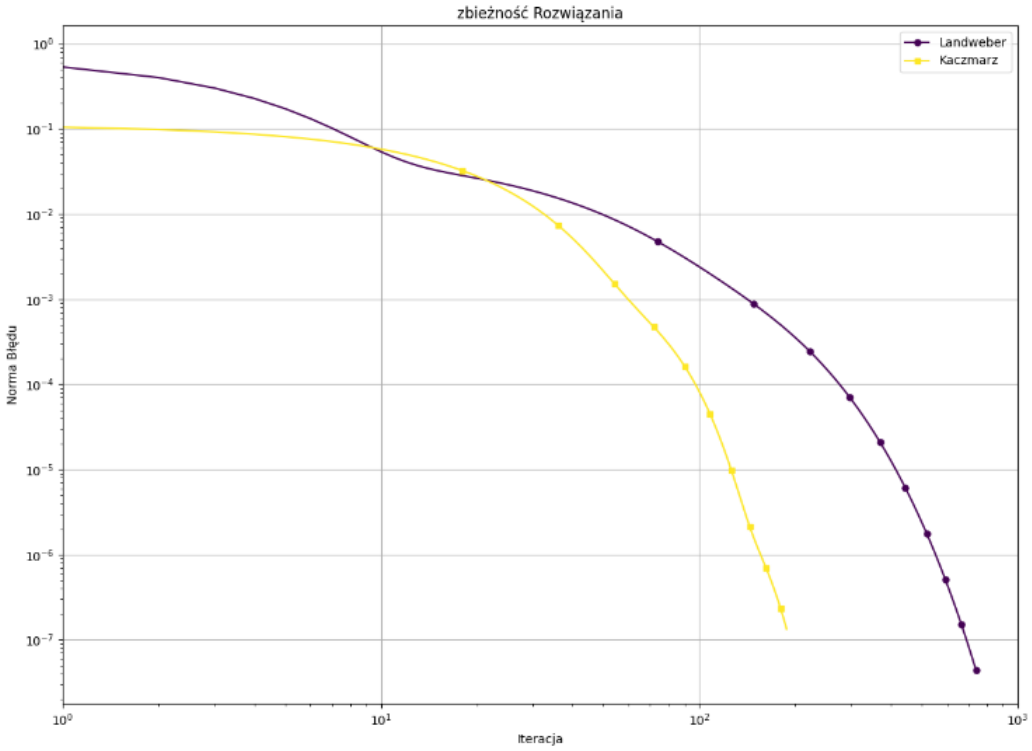
SD – macierz A nie jest dodatnio określona

Kaczmarza – żadna wartość współczynnika nie pozwala na zbieżność

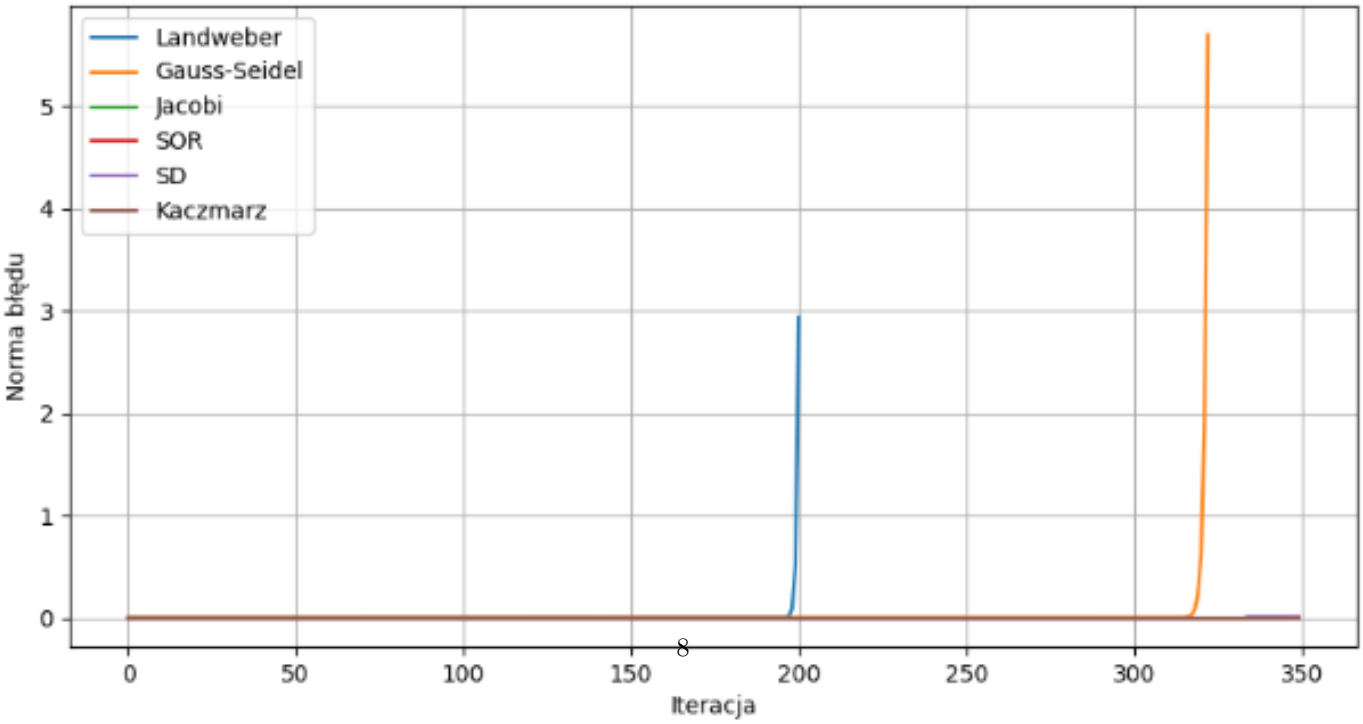


Wykres.2.1. Zależność n-tej iteracji metody do błędu residualnego

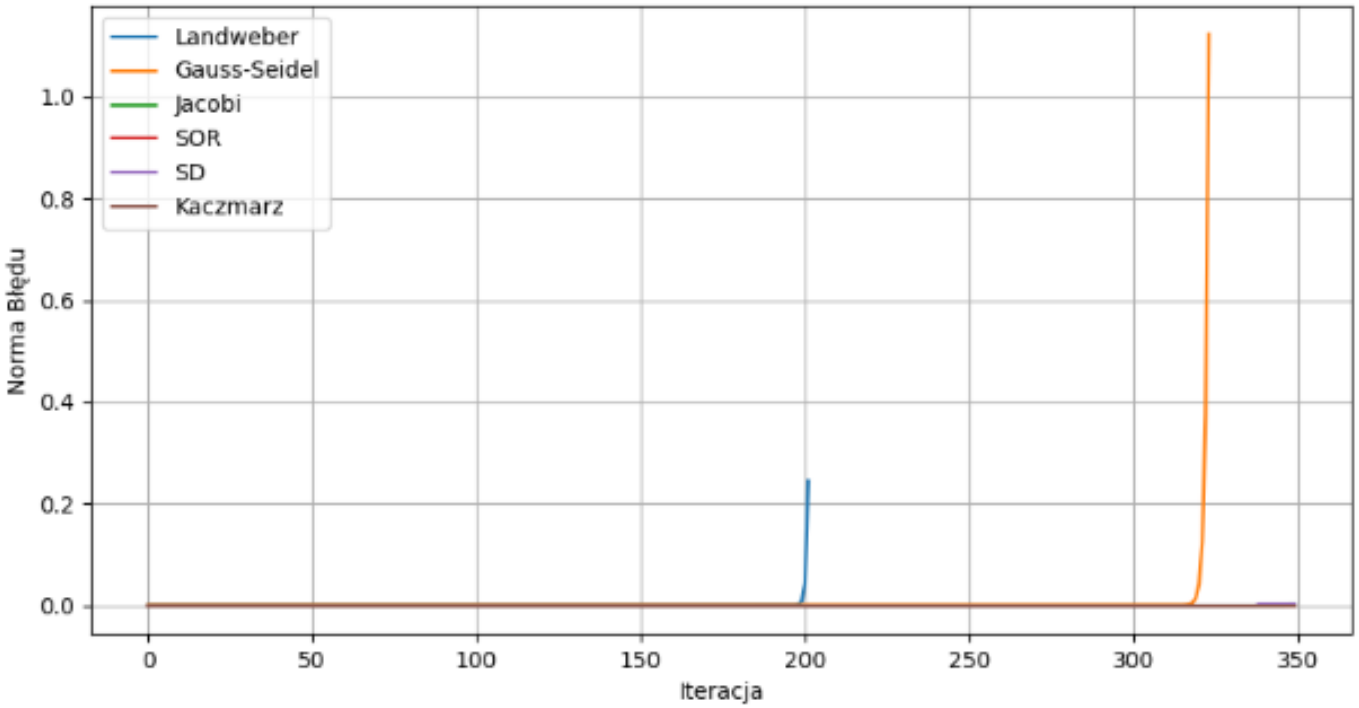
Pomimo ustaleń przedstawiono uzyskane przy pomocy algorytmów wykresy na rysunkach:



Wykres.2.2. Zależność n-tej iteracji metody do błędu rozwiązania



Wykres 3.1. Zależność błędu rezydualnego do iteracji



Wykres 3.2. Zależność błędu rozwiązania do iteracji

4 Zadanie nr. 4

Niech $A = [a_{ij}] \in R^{NxN}$, gdzie $a_{ij} = \frac{1}{i+j-1}$ (macierz Hilberta) oraz $\begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^T \in R^N$. Rozwiąż układ równań Ax=b dla N= 5,10 i 20, stosując wybrane metody iteracyjne. Rozpocząć interacje od $x^{(0)} = 0$. Przedstaw krzywe błędu residualnego: $r^{(k)} = ||b - Ax^{(k)}||_2$.

Do zrealizowania zadania zdecydowano się na metody Gaussa-Siedela, Steepest Descent oraz metodę Kaczmarza. Poniżej matematyczne wyjaśnienie wykorzystanych metod:
Metoda Gaussa-Siedela:

$$x^{(k-1)} = S^{-1}(Tx^{(k)} + b) \tag{7}$$

$$A = S - T \tag{8}$$

S- macierz dolnotrójkątna, T- macierz górnotrójkątna

$$G = s^{-1}T, c = S^{-1}b \tag{9}$$

Metoda Steepest Descent:
Obliczenie residuum:

$$r^{(k)} = b - Ax^{(k)} \tag{10}$$

Obliczenie kierunku:

$$d^{(k)} = -r^{(k)} \tag{11}$$

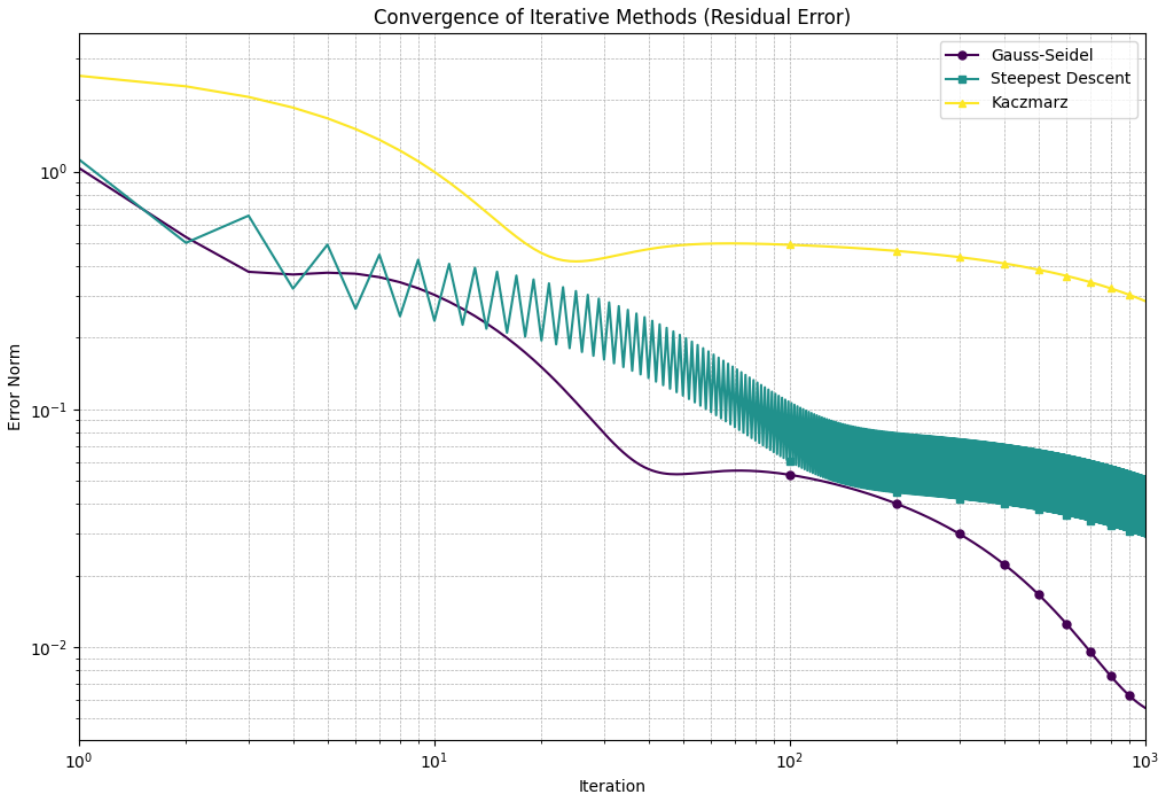
Obliczenie pozycji:

$$x^{(k+1)} \tag{12}$$

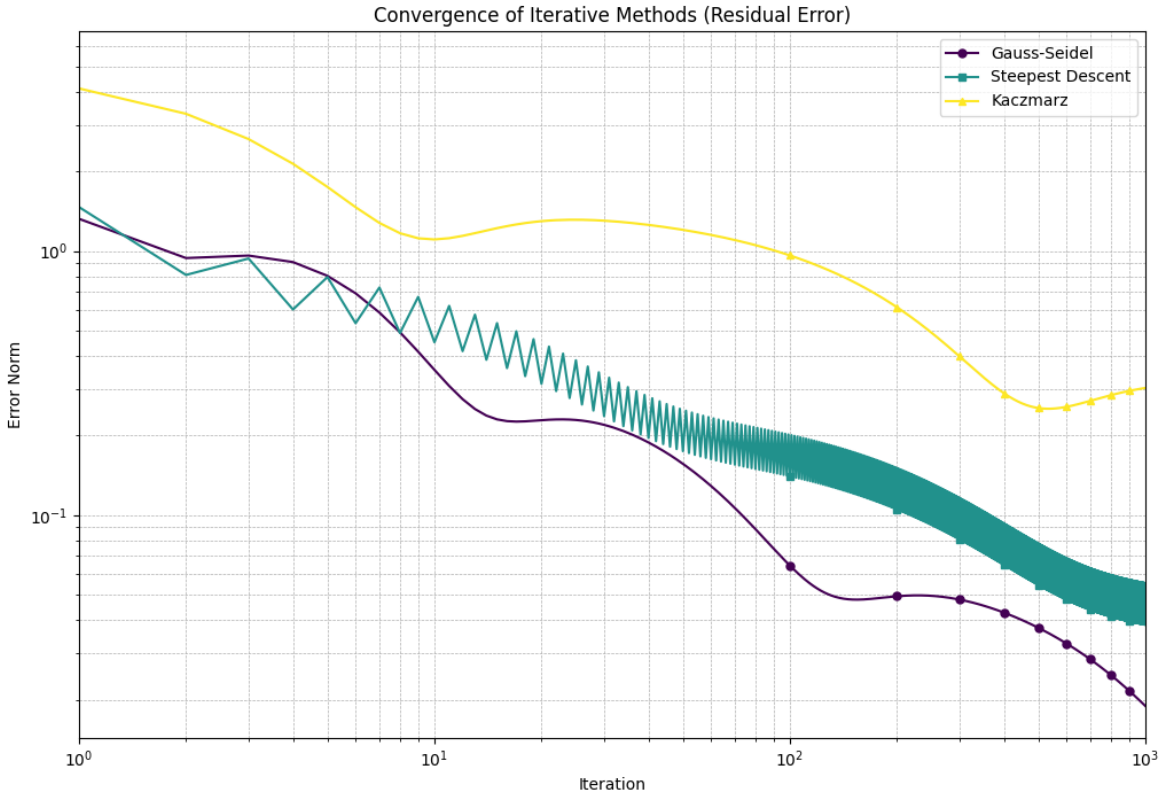
Sprawdzenie warunku zatrzymania pętli:
Jeśli osiagniemy zadowalający poziom dokładności lub maksymalny poziom iteracji, opuszczamy pętlę.
Metoda Kaczmarza:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{(b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)})}{a_{ii}}, \tag{13}$$

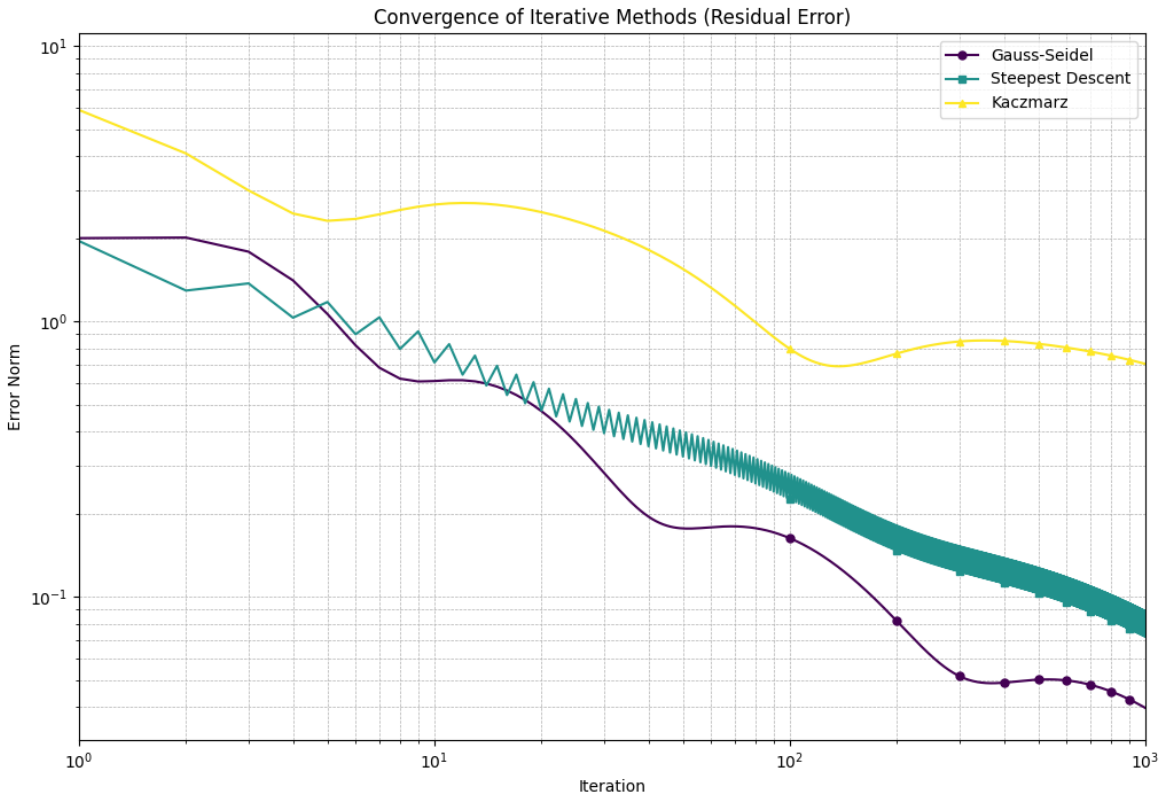
gdzie a_{ij} oznacza i-ty wiersz i j-tą kolumne macierzy A, a b_i to i-ta wartość wektora b.
Poniżej umieszczono wykresy reprezentujące krzywe błędów residualnych wybranych metod iteracyjnych dla zadanych wartości N:



Wykres.4.1. Krzywe błędów residualnych dla N=5



Wykres.4.2. Krzywe błędów residualnych dla N=10



Wykres.4.3. Krzywe błędów residualnych dla N=20

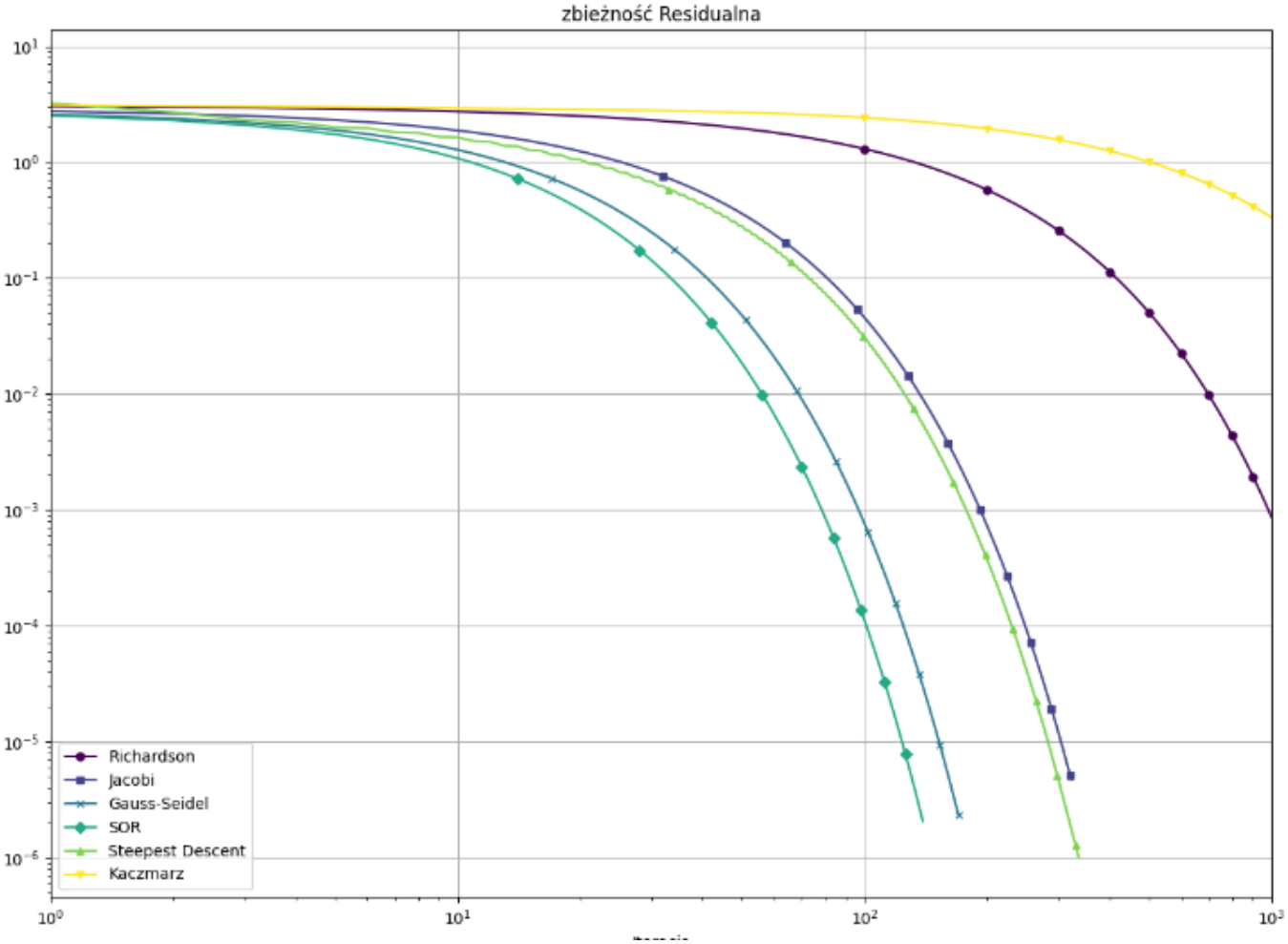
Metoda obarczona najmniejszym błędem rezidualnym dla wszystkich wartości N to metoda Gaussa-Siedela a największym metoda Kaczmarza, Wyjasnieniem takiego wyniku moze byc fakt iż macierz Hilberta może zawierać wiersze zerowe.

5 Zadanie nr. 5

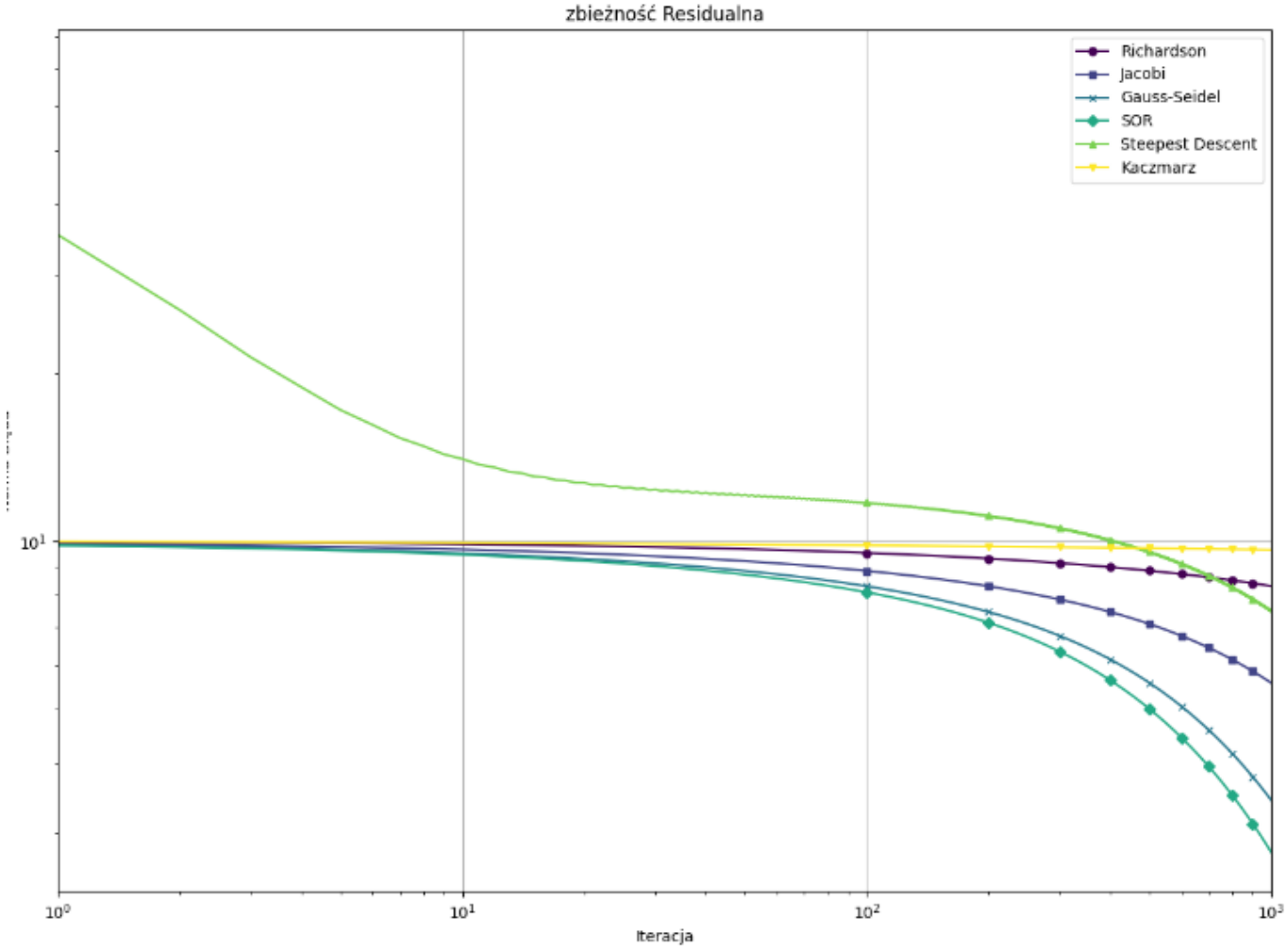
Niech

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots \\ -1 & 2 & \dots & 0 \\ 0 & \dots & 2 & -1 \\ \dots & 0 & -1 & 2 \end{bmatrix} \in R^{NxN}, b = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^T \in R^N \qquad (14)$$

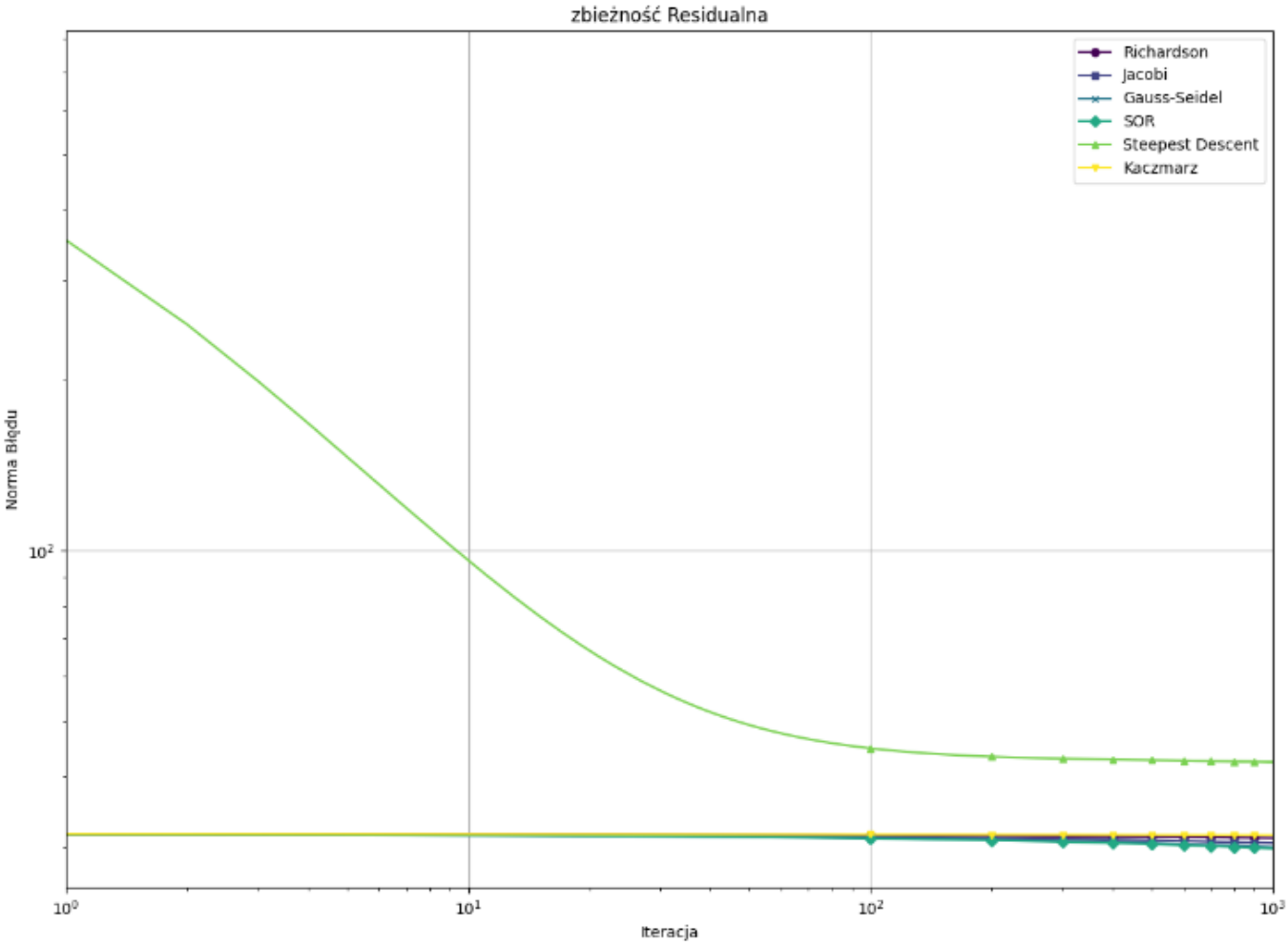
Rozwiązać układ równań Ax=b przy pomocy wybranych iteracyjnych solverów dla N = 10, 100, 1000, 10000, 100000. Wykorzystaj reprezentację liczb rzadkich dla dużych macierzy. Dla każdego N narysować błąd residualny $r^{(k)} = ||b - Ax^{(k)}||_2$ w zależności od k-tego kroku iteracyjnego. Wyjaśnić różnicę w zachowaniu zbieżności.
Poniżej przedstawiono zależność błędów rozwiązania i rezydualnego w zależności od ilości iteracji.



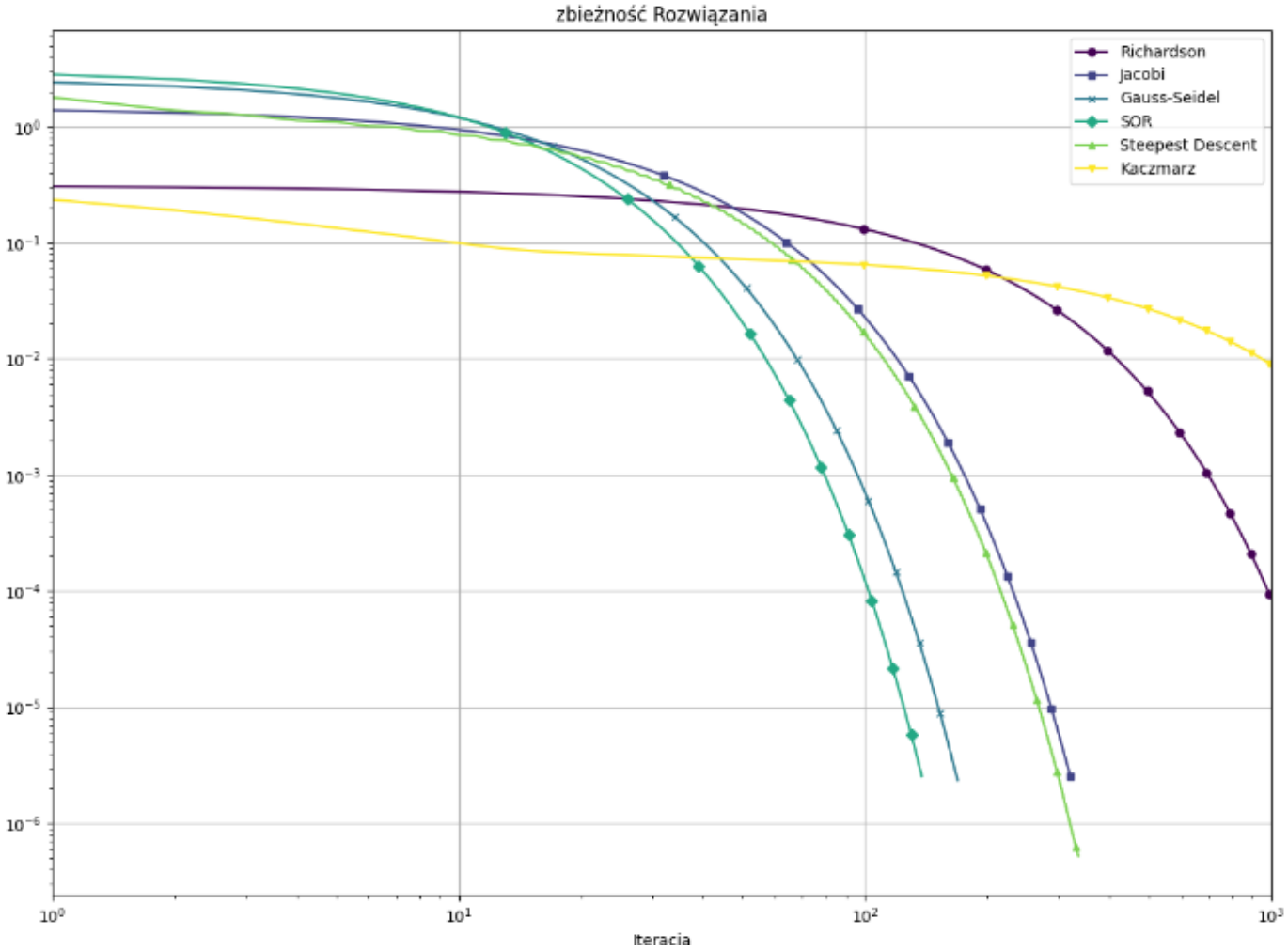
Wykres.5.1. Zależność błędu rezydualnego dla macierzy 10x10 i ilości iteracji= 1000



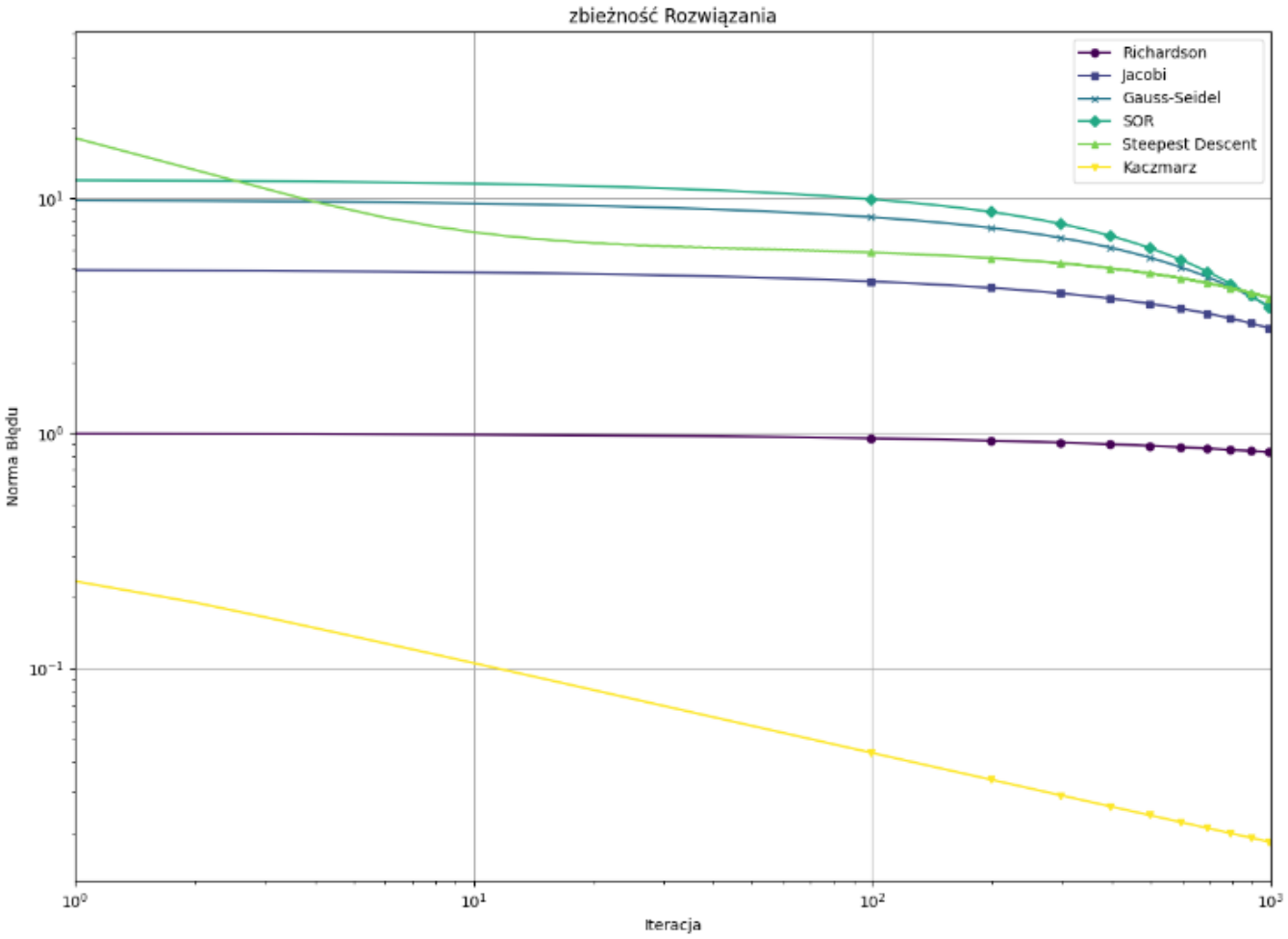
Wykres.5.2. Zależność błędu rezydualnego dla macierzy 100x100 i ilości iteracji= 1000



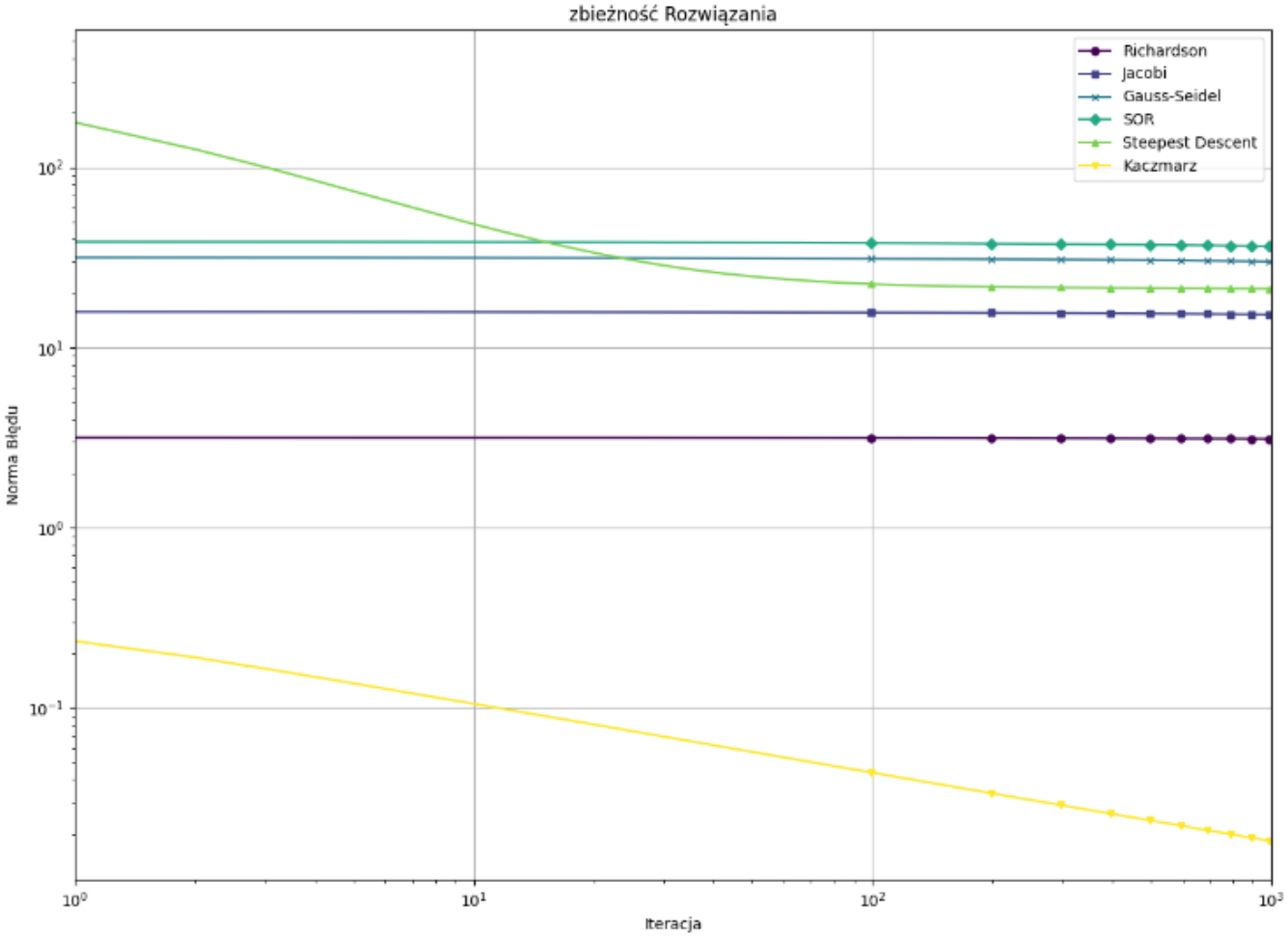
Wykres.5.3. Zależność błędu rezydualnego dla macierzy 1000x1000 i ilości iteracji= 1000



Wykres.5.1. Zależność błędu rozwiązania dla macierzy 10x10 i ilości iteracji= 1000



Wykres.5.1. Zależność błędu rozwiązania dla macierzy 100x100 i ilości iteracji= 1000



Wykres.5.1. Zależność błędu rozwiązania dla macierzy 1000x1000 i ilości iteracji= 1000

Niestety ze względu na coraz to dłuższy czas przeliczeń oraz obciążenia komputera jakie za sobą niosły zaprzestano wykonywania zadania tym samym nie wykonując je dla N=10 000 i N= 100 000. Wraz ze wzrostem wielkości macierzy zwiększała się również liczba iteracji która była potrzebna do wyliczenia prawidłowego rozwiązania o podanej tolerancji. W przypadku macierzy 100x100 tysięczna liczba iteracji nie była w stanie wyliczyć poprawnych wyników. Ze względu na czas i ograniczenia sprzętowe zwiększono iteracje dla dwóch wybranych metod.

Liczba iteracji dla N=10

Richardson: 361
Jacobi: 323
Gauss-Seidel: 171
SD: 337
Kaczmarz: 6775

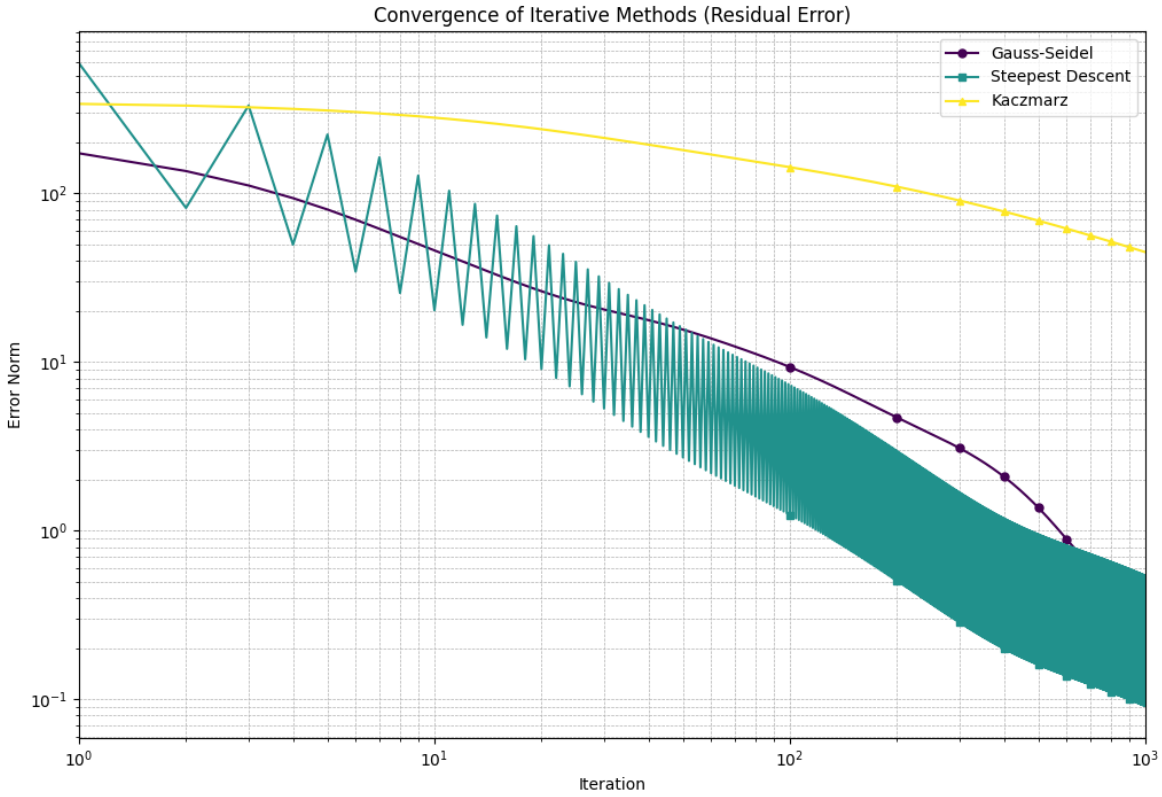
Liczba iteracji dla N=100

Richardson: 33107
SD: 33691

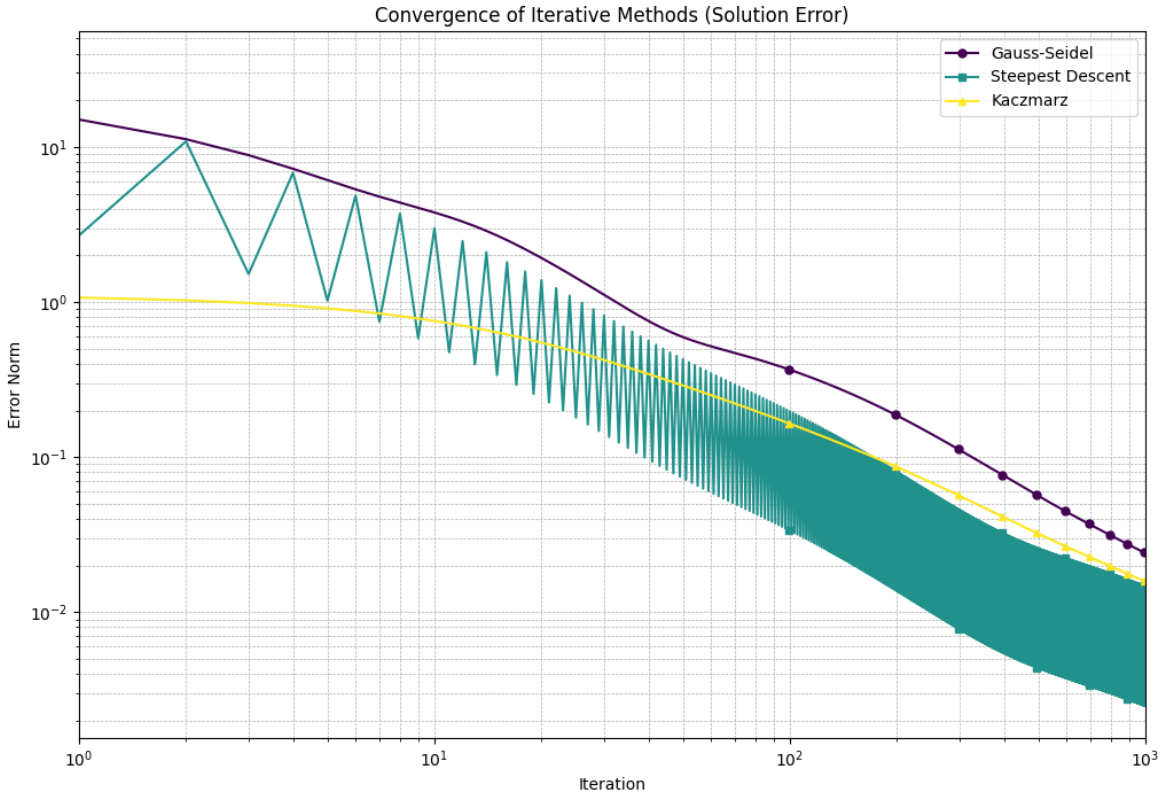
6 Zadanie nr. 6

Niech $Ax = b$, $gdzieA = I_N \bigotimes C^T C$, symbol $\bigotimes$ oznacza iloczyn Kroneckera, $I_N \leftarrow R^{NxN}$ jest macierzą jednostkową, $C \leftarrow R^{MxM}$ jest macierzą losową wygenerowaną z rozkładu normalnego(rand), M=200, N=30, oraz x $N(0, I_{MN})$ (rozkład normalny - randn). Porównaj na jednym rysunku krzywe błędu residualnego uzyskane różnymi metodami iteracyjnymi, a na drugim rysunku krzywe błędu aproksymacji rozwiązania $||x^* - x^{(k)}||_2$. Porównaj czas wykonywania się algorytmów.

Zdecydowano się na ponowne wykorzystanie algorytmów użytych w zadaniu nr. 4. Dla kazdego algorytmu wykonano 1000 iteracjii.
Poniżej wykaz wykresów przedstawiających krzywe błędów residualnych oraz aproksymacji wybranych algorytmów, jak i tabela przedstawiająca porównanie czasów wykonywania się algorytmów:



Wykres.4.2. Krzywe błędów residualnych.



Wykres.4.2. Krzywe błędów aproksymacji

Metoda	Czas [s]
Gauss-Seidel	6522
Steepest Descent	12
Kaczmarz	94

Długi czas wykonywania metody Gaussa-Seidela moze wynikac z wykorzystania języka wysokiego poziomu oraz ograniczonej wartosci pamieci RAM komputera. Niemniej jednak algorytm ten nie sprawdza sie w przypadku macierzy zainicjowanej iloczynem kroneckera poniewaz wartosci bledu aproksymacji jest największa. Wyniki sugerują, że mimo że metoda Steepest Descent daje najmniejszy błąd aproksymacji, to metoda Kaczmarsza osiąga najlepsze dopasowanie do danych, wyrażone przez najmniejszy błąd residualny. Metoda Gaussa-Siedela wydaje się być najmniej skuteczna z punktu widzenia zarówno błędu aproksymacji, jak i błędu residualnego.

7 Algorytmy

```
def richardson(A, b, x0, tau, max_iter=100, tol=1e-6):

  history = []
  x = x0
  for i in range(max_iter):
      r = b - np.dot(A, x) # obliczenie residuum
      if np.linalg.norm(r) < tol:
          print(f"Iteracja {i}: Znaleziono rozwiązanie z odpowiednią
              łśdokadnocią.")
          return x, history
      x = x + tau * r # aktualizacja rozwiązania
      history.append(x.copy())
  print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
  return x, history
```

```
def landweber(A, b, x0, tau = 0.1, max_iter=100, tol=1e-6):
  history = []
  x = x0
  A_transpose = A.T # Transpozycja macierzy A

  for i in range(max_iter):
      r = b - np.dot(A, x) # Obliczenie residuum
      if np.linalg.norm(r) < tol:
          print(f"Iteracja {i}: Znaleziono rozwiązanie z odpowiednią
              łśdokadnocią.")
          return x, history
      x = x + tau * np.dot(A_transpose, r) # Aktualizacja rozwiązania
          z wykorzystaniem gradientu
      history.append(x.copy())
  print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
  return x,history
```

```
def jacobi(A, b, x0, max_iter=100, tol=1e-6):

  history = []
  n = len(b)
  x = x0.copy()
  for it in range(max_iter):
      x_new = np.zeros_like(x)

      for i in range(n):
          s = sum(A[i, j] * x[j] for j in range(n) if i != j)
          x_new[i] = (b[i] - s) / A[i, i]

      if np.linalg.norm(x - x_new, ord=np.inf) < tol:
```

```
            print(f"Iteracja {it}: Znaleziono rozwiązanie z odpowiednią
                łśdokadnocią.")
            return x_new, history
        x = x_new
        history.append(x.copy())

    print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
    return x, history
```

```
def gauss_seidel(A, b, x0, max_iter=100, tol=1e-6):

    history = []
    n = len(b)
    x = x0.copy()
    for it in range(max_iter):
        x_new = np.copy(x)

        for i in range(n):
            s1 = sum(A[i, j] * x_new[j] for j in range(i))
            s2 = sum(A[i, j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i, i]

        if np.linalg.norm(x - x_new, ord=np.inf) < tol:
            print(f"Iteracja {it}: Znaleziono rozwiązanie z odpowiednią
                łśdokadnocią.")
            return x_new, history
        x = x_new
        history.append(x.copy())

    print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
    return x, history
```

```
def sor(A, b, x0, omega, max_iter=100, tol=1e-6):

    history = []
    n = len(b)
    x = x0.copy()
    for it in range(max_iter):
        x_new = np.copy(x)

        for i in range(n):
            s1 = sum(A[i, j] * x_new[j] for j in range(i))
            s2 = sum(A[i, j] * x[j] for j in range(i + 1, n))
            x_new[i] = (1 - omega) * x[i] + omega * (b[i] - s1 - s2) /
                A[i, i]

        if np.linalg.norm(x - x_new, ord=np.inf) < tol:
            print(f"Iteracja {it}: Znaleziono rozwiązanie z odpowiednią
```

```
            łśdokadnocią.")
        return x_new, history
    x = x_new
    history.append(x.copy())
  print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
  return x, history
```

```
def steepest_descent(A, b, x0, max_iter=100, tol=1e-6):

  history = []
  x = x0
  r = b - np.dot(A, x) # początkowe residuum
  for it in range(max_iter):
      Ar = np.dot(A, r)
      alpha = np.dot(r, r) / np.dot(r, Ar) # obliczenie łśdugoci kroku
      x = x + alpha * r # aktualizacja rozwiązania
      r = b - np.dot(A, x) # aktualizacja residuum
      if np.linalg.norm(r) < tol:
          print(f"Iteracja {it}: Znaleziono rozwiązanie z odpowiednią
              łśdokadnocią.")
          return x, history
      history.append(x.copy())
  print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
  return x, history
```

```
def kaczmarz(A, b, x0, max_iter=100, tol=1e-6):

  history = []
  n, m = A.shape
  x = x0.copy()

  for it in range(max_iter):
      for i in range(n):
          ai = A[i, :]
          lambda_i = (b[i] - np.dot(ai, x)) / np.dot(ai, ai)
          x += lambda_i * ai

      r = b - np.dot(A, x)
      if np.linalg.norm(r) < tol:
          print(f"Iteracja {it}: Znaleziono rozwiązanie z odpowiednią
              łśdokadnocią.")
          return x, history
      history.append(x.copy())

  print("Nie znaleziono rozwiązania w maksymalnej liczbie iteracji.")
  return x, history
```