Metody numeryczne - Laboratorium 3
Układy równań liniowych

Uwaga: Instrukcje dotyczące przebiegu zajęć laboratoryjnych zawarte są w części 2 tego dokumentu.

1 Część teoretyczna

- metody ścisłe
  - wzory Cramera
  - metody eliminacyjne:
    - metoda eliminacji Gaussa
    - metoda Gaussa-Jordana
  - metody dekompozycyjne:
    - rozkład LU przy pomocy eliminacji Gaussa
    - metoda Gaussa-Doolittle'a
    - metoda Gaussa-Crouta
    - metoda Choleskiego (Choleskiego-Crouta/Banachiewicza/pierwiastków kwadratowych) (dla macierzy symetrycznych)
- metody iteracyjne (przybliżone)
  - metoda Jacobiego (iteracji prostej)
  - metoda Gaussa-Seidla
  - in.

Na zajęciach będziemy rozważać problem znalezienia rozwiązania poniższego układu równań liniowych (URL)

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & \vdots & & & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{array} \tag{1}$$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2}$$

$$\mathbf{A}\ \mathbf{x} = \mathbf{b} \tag{3}$$

1.1 Metody bezpośrednie

1.1.1 Metoda eliminacji Gaussa

W celu przedstawienia metody eliminacji Gaussa zapiszmy układ (2) w postaci

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \dots & a_{2n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \dots & a_{nn}^{(0)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(0)} \\ \vdots \\ a_{n,n+1}^{(0)} \end{bmatrix} \tag{4}$$

gdzie wartości $a_{i,n+1}^{(0)}$ to wyrazy wolne, które odpowiadają wartościom b_i z równania (2), natomiast . Jeśli $a_{11} \neq 0$ można odjąć pierwsze równanie pomnożone przez a_{i1}/a_{11} od pozostałych równań ($i = 2, 3, \dots, n$). Otrzyma się wtedy układ zredukowany, w którym wszystkie elementy pierwszej kolumny poniżej pierwszego wiersza będą zerami:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{n,n+1}^{(1)} \end{bmatrix} \tag{5}$$

$a_{ij}^{(1)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} a_{1j}^{(0)}$, $i = 2, 3, \dots, n$, $j = 2, 3, \dots, n$.

W kolejnym kroku (o ile $a_{22}^{(1)} \neq 0$), odejmujemy drugie równanie układu (5) pomnożone przez $a_{i2}^{(1)}/a_{22}^{(1)}$ od równań $3, 4, \dots, n$. Uzyskuje się wtedy drugi układ zredukowany:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \dots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ a_{3,n+1}^{(2)} \\ \vdots \\ a_{n,n+1}^{(2)} \end{bmatrix} \tag{6}$$

gdzie $a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)}$, $i = 3, \dots, n$, $j = 3, \dots, n$.

Wykonując analogiczne przekształcenia dla niżej położonych wierszy, po $n-1$ krokach otrzyma się układ z macierzą trójkątna górną:

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1,n-1}^{(0)} & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \dots & a_{2,n-1}^{(1)} & a_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n-1,n-1}^{(n-2)} & a_{2n}^{(n-2)} \\ 0 & 0 & \dots & 0 & a_{nn}^{(n-1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_3 \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,n+1}^{(0)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{3,n+1}^{(n-2)} \\ a_{n,n+1}^{(n-1)} \end{bmatrix} \tag{7}$$

Uogólniając, układ (7) otrzymuje się tworząc tzw. macierz rozszerzoną $\mathbf{A_r}$ układu (2) powstałą poprzez połączenie macierzy $\mathbf{A}$ i wektora wyrazów wolnych $\mathbf{b}$ ($\mathbf{A_r} = [\mathbf{Ab}]_{n x n+1}$). Następnie na macierzy tej dokonuje się serii przekształceń (zastępowanie wierszy poprzez ich kombinacje liniowe), w taki sposób, aby zerować kolejne elementy macierzy poniżej przekątnej głównej, tzn. zgodnie ze wzorem:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}$$
$$k = 1, 2, \dots, n-1$$
$$j = k+1, \dots, n+1$$
$$i = k+1, \dots, n$$

Po otrzymaniu układu (7) rozwiązanie można uzyskać dokonując redukcji wstecznej. Wartość x_n wyznaczamy bezpośrednio z ostatniego równania układu (7): $x_n = c_n/t_{nn}$. Wtedy w pozostałych równaniach n-tą kolumnę możemy przenieść na prawą stronę równania, zmniejszyć rozmiar URL do $n-1$ x $n-1$ i ponownie w łatwy sposób obliczyć niewiadomą x_{n-1}. Schemat ten powtarzamy, aż do uzyskania wszystkich szukanych wartości.

Dane: A, b, N
Wyniki: Ur
  $Ur \leftarrow [Ab]$;
  for $k \leftarrow 1$ to $N-1$ **do**
    if $A(k,k) = 0$ **then**
      return error;
    end if
    for $j \leftarrow k+1$ to $N+1$ **do**
      for $i \leftarrow k+1$ to N **do**
        $Ur(i,j) = Ur(i,j) - \frac{Ur(i,k)}{Ur(k,k)} \cdot Ur(k,j)$;
      end for
    end for
  end for

Algorytm 1: Metoda eliminacji Gaussa

Dane: $\mathbf{U}r$, N
Wyniki: $\mathbf{x}$
  $\mathbf{x} \leftarrow \mathbf{0}$;
  for $i \leftarrow N$ to 1 **do**
    $S \leftarrow 0$
    for $j \leftarrow i+1$ to N **do**
      $S \leftarrow S + \mathbf{U}r(i,j) \cdot x(j)$;
    end for
    $x(i) \leftarrow \frac{\mathbf{U}r(i,n+1) - S}{\mathbf{U}r(i,i)}$;
  end for

Algorytm 2: Redukcja wsteczna

Powyższe działania można zapisać jako:

$$
\begin{aligned}
x_n &= a_{n,n+1}^{(n-1)} / a_{nn}^{(n-1)} \\
x_i &= \frac{1}{a_{ii}^{(n-1)}} \left(a_{i,n+1}^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j \right) \quad dla \quad i = n-1, n-2, ..., 1
\end{aligned}
\tag{8}
$$

1.1.2 Metoda Gaussa-Jordana

W eliminacji Gaussa zerujemy tylko elementy poniżej (lub powyżej) przekątnej głównej macierzy. Modyfikacja powyższego algorytmu polegająca na zerowaniu zarówno elementów poniżej jak i powyżej przekątnej prowadzi do metody Gaussa-Jordana. Metoda ta ma tę przewagę nad zwykłą eliminacja Gaussa, że nie jest potrzebna redukcja wsteczna w celu uzyskania rozwiązania. Macierz układu jest przekształcana do macierzy jednostkowej, a rozwiązanie otrzymuje się niejako 'automatycznie' w ostatniej kolumnie macierzy $\mathbf{A_r}$ (wektor wyrazów wolnych).

W każdym k-tym kroku wykonywania algorytmu Gaussa-Jordana, k-ty wiersz macierzy $\mathbf{A_r}$ jest dzielony przez element $a_{kk}^{(k-1)}$. Następnie wiersz ten pomnożony przez $a_{i1}^{(0)}$ jest odejmowany od każdego i-tego wiersza ($i \neq k$), tak aby wyzerowały się wszystkie elementy k-tej kolumny poza elementem leżącym na przekątnej głównej.

Tak więc, przekształcenie układu równań (4) w pierwszym kroku tej metody daje układ postaci:

$$
\begin{bmatrix}
1 & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\
0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\
\vdots & \vdots & \ddots & \vdots \\
0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)}
\end{bmatrix}
\cdot
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}
=
\begin{bmatrix} a_{1,n+1}^{(1)} \\ a_{2,n+1}^{(1)} \\ \vdots \\ a_{n,n+1}^{(1)} \end{bmatrix}
\tag{9}
$$

W kolejnym kroku drugie równanie dzielimy obustronnie przez $a_{22}^{(1)}$ i od i-tego wiersza ($i = 1,\ 3, \dots, n$) odejmujemy wiersz drugi pomnożony przez $a_{i2}^{(1)}$. W ten sposób otrzyma się układ:

$$
\begin{bmatrix}
1 & 0 & \dots & a_{1n}^{(2)} \\
0 & 1 & \dots & a_{2n}^{(2)} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & a_{nn}^{(2)}
\end{bmatrix}
\cdot
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}
=
\begin{bmatrix} a_{1,n+1}^{(2)} \\ a_{2,n+1}^{(2)} \\ \vdots \\ a_{n,n+1}^{(2)} \end{bmatrix}
\tag{10}
$$

W końcu, po $n-1$ krokach eliminacji układ jest przekształcany do postaci

$$
\begin{bmatrix}
1 & 0 & \dots & 0 \\
0 & 1 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & 1
\end{bmatrix}
\cdot
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}
=
\begin{bmatrix} a_{1,n+1}^{(n)} \\ a_{2,n+1}^{(n)} \\ \vdots \\ a_{n,n+1}^{(n)} \end{bmatrix}
\tag{11}
$$

gdzie prawa strona układu jest rozwiązaniem.

Algorytm metody Gaussa-Jordana można opisać następująco:

- Utwórz macierz rozszerzoną $\mathbf{A_r}$ dla układu $\mathbf{Ax} = \mathbf{b}$
- Dla każdego k-tego wiersza ($k = 1, 2, \ldots, n$) macierzy $\mathbf{A_r}$:
  - Dla każdej j-tego elementu k-tego wiersza, podziel j-ty element przez element $\mathbf{A_r}(\mathbf{k}, \mathbf{k})$

$$\begin{aligned} a_{kj}^{(k)} = &\quad a_{kj}^{(k)} / a_{kk}^{(k)} \\ j = &\quad 1, 2, \ldots, n+1 \end{aligned} \tag{12}$$

  - Dla każdego i-tego ($i = 1, 2, \ldots, n,\ i \neq k$) wiersza
    Dla $j = 1, 2, \ldots, n+1$

$$\begin{aligned} \lambda_i &= \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - \lambda a_{kj}^{(k-1)} \end{aligned} \tag{13}$$

1.1.3 Rozkład LU, metoda Gaussa-Doolittle'a

Innym sposobem na rozwiązanie URL są metody dekompozycyjne. Opierają się one na przekształceniu macierzy $\mathbf{A}$ na iloczyn dwóch macierzy trójkątnych: dolnej $\mathbf{L}$ (ang. lower) oraz górnej $\mathbf{U}$ (ang. upper):

$$A = LU \tag{14}$$

Aby rozkład był jednoznaczny zakłada się, że wszystkie elementy przekątnej macierzy L lub U są równe 1 (metoda Gaussa-Doolittle'a lub Gaussa-Crouta).

Uwzględniając (14) w (3) otrzymuje się

$$LUx = b \tag{15}$$

W ten sposób rozwiązanie URL można podzielić na 2 etapy. W pierwszej kolejności rozwiązuje się

$$Ly = b \Rightarrow y = L^{-1}b \tag{16}$$

a następnie

$$Ux = y \Rightarrow x = U^{-1}y \tag{17}$$

Rozkład LU przy pomocy metody Gaussa-Doolittle'a opisują poniższe wzory:

$$\begin{aligned} &u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \qquad dla \quad j = i, i+1, \ldots, n \\ &l_{ji} = \frac{1}{u_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right) \quad dla \quad j = i+1, i+2, \ldots, n \\ &i = 1, 2, \ldots, n \end{aligned}$$

Dane: n, (a_{ij})
Wyniki: l_{ij}, u_{ij}
  for $k \leftarrow 1$ to n **do**
    $l_{kk} \leftarrow 1$;
    for $j \leftarrow k$ to n **do**
      $u_{kj} \rightarrow a_{kj} - \sum_{s=1}^{k-1} l_{ks} U_{sj}$
    end for
    for $i \rightarrow k+1$ to N **do**
      $l_{ik} \rightarrow (a_{ik} - \sum_{s=1}^{k-1} l_{is} U_{sk})/u_{kk}$
    end for
  end for

Algorytm 3: Rozkład LU przy pomocy metody Gaussa-Doolittle'a

1.1.4 Wybór elementów podstawowych

We wszystkich powyższych metodach istnieje możliwość wystąpienia zerowych, lub bardzo małych elementów na przekątnej macierzy **A**, co jest powodem niestabilności numerycznej algorytmu. Dlatego też w praktyce stosuje się ich zmodyfikowane wersje. Mianowicie zakłada się, że w każdej iteracji do eliminacji elementów wybiera się element o największej co do modułu wartości (ang. pivoting). Wyróżnić można dwa sposoby wyboru

- wybór częściowy elementu podstawowego – polega na wyszukiwaniu przed każdym k-tym krokiem eliminacji zmiennych największego co do modułu elementu spośród elementów znajdujących się poniżej k-tej kolumny. Po ustaleniu numeru r równania, w którym ten element występuje, wiersz o numerze r zamieniany jest miejscem z wierszem o numerze k. Czynności przestawiania wierszy musi towarzyszyć zmiana znaków w dowolnym spośród przestawianych równań, jeśli wyznacznik macierzy współczynników układu powstałego w wyniku przestawiania wierszy ma być równy wyznacznikowi macierzy współczynników układu wyjściowego.

- wybór pełny elementu podstawowego – w k-tym kroku wyszukiwany jest największy co do modułu element w całej macierzy [...]. Po ustaleniu numeru r wiersza i numeru s kolumny następuje przestawienie wiersza o numerze r z wierszem o numerze k, a następnie kolumny o numerze s z kolumną o numerze k. Liczba przestawień wierszy i kolumn jest zapamiętywana i uwzględniana przy obliczaniu wyznacznika macierzy współczynników.

1.2 Metody iteracyjne

1.2.1 Metoda Jacobiego (metoda iteracji prostej)

Przekształćmy układ (1) do postaci:

$$\begin{array}{lllllllll} x_1 & = & c_1 & + & & d_{12}x_2 & + & \ldots & + & d_{1n}x_n \\ x_2 & = & c_2 & + & d_{21}x_1 & + & & \ldots & + & d_{2n}x_n \\ & & & & & \vdots & & & & \\ x_n & = & c_n & + & d_{n1}x_1 & + & d_{n2}x_2 & + & \ldots & \end{array} \tag{18}$$

gdzie

$$\begin{aligned} c_i = b_i/a_{ii} \quad dla \quad i = 1, 2, ..., n \\ d_{ij} = -a_{ij}/a_{ii} \quad dla \quad i = 1, 2, ..., n; j = 1, 2, ..., n; i \neq j \end{aligned} \tag{19}$$

Przyjmując:

$$C = \begin{bmatrix} c_1 \\ c_2 \\ \ldots \\ c_3 \end{bmatrix} \text{ oraz } D = \begin{bmatrix} 0 & d_{12} & \ldots & d_{1n} \\ d_{21} & 0 & \ldots & d_{2n} \\ \ldots & & & \\ d_{n1} & d_{n2} & \ldots & 0 \end{bmatrix} \tag{20}$$

układ (18) można zapisać macierzowo

$$x = C + Dx \tag{21}$$

Na podstawie ostatniego równania konstruowany jest ciąg przybliżeń

$$x^{(k+1)} = C + Dx^{(k)} \tag{22}$$

gdzie za początkowy wektor x^0 jest dowolny, najczęściej przyjmuje się $x^0 = C$.

Mówiąc inaczej, $(k+1)$-sze przybliżenie i-tej niewiadomej układu równań można obliczyć przy pomocy

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{ij} * x_j^{(k)}}{a_{ii}} \tag{23}$$

Jako warunku kończącego wykonywanie iteracji można użyć np:

$$\sum_{i=1}^{n} |x_i^{(k+1)} - x_i^{(k)}| < tol \tag{24}$$

lub

$$\sum_{i=1}^{n} |\tilde{b}_i^{(k)} - b_i| < tol, \text{ gdzie } \tilde{b}^{(k)} = Ax^{(k)} \tag{25}$$

gdzie tol – zadana dokładność.

Warunkiem koniecznym zbieżności ciągu kolejnych przybliżeń jest:

$$\wedge_{i\in\{1,2,\ldots,n\}}|a_{ii}| > \sum_{j=1,j\neq i}^{n}|a_{ij}| \tag{26}$$

lub

$$\wedge_{j\in\{1,2,\ldots,n\}}|a_{jj}| > \sum_{i=1,i\neq j}^{n}|a_{ij}| \tag{27}$$

Macierz współczynników układu (3) można wyrazić w postaci sumy:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \tag{28}$$

gdzie macierze $\mathbf{L}$, $\mathbf{D}$ i $\mathbf{U}$ są poniższej postaci

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ * & * & * & * & 0 \end{bmatrix} + \begin{bmatrix} * & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & * \end{bmatrix} + \begin{bmatrix} 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{29}$$

Posługując się poniższą notacją można zapisać metodę Jacobiego następująco:

$$\mathbf{x^{(k+1)}} = -\mathbf{D^{-1}}(\mathbf{L}+\mathbf{U})\mathbf{x^{(k)}} + \mathbf{D^{-1}}\mathbf{b} \tag{30}$$

1.3 Metoda Gaussa-Seidla

Przyspieszenie zbieżności tego prostego procesu iteracyjnego można uzyskać poprzez wykorzystanie w każdej $(k+1)$-ej iteracji nowoobliczonych wartości wektora $\mathbf{x}$, zamiast wartości z poprzedniej iteracji jak to ma miejsce w metodzie Jacobiego. Uzyskuje się w ten sposób metodę Gaussa-Seidla, którą można wyrazić macierzowo:

$$\mathbf{x}^{(k+1)} = (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} - (\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(k)} \tag{31}$$

lub równoważnie w postaci wzoru dla pojedynczego elementu poszukiwanego wektora:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{1}{a_{ii}}\left(\sum_{j=1,i>1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i}^{n} a_{ij}x_j^{(k)} - b_i\right) = x_i^{(k)} + r_i^{(k)} \tag{32}$$

w którym część odpowiadającą elementowi $r_i^{(k)}$ można interpretować jako poprawkę wprowadzaną w celu ulepszenia rozwiązania otrzymanego w poprzednim kroku iteracyjnym.

Tabela 1: Funkcje implementujące operacje na macierzach

Funkcja	Opis
det(A)	wyznacznik macierzy
cond(A)	wskaźnik uwarunkowania macierzy
norm(A,)	norma macierzy
inv(A) A^-1	macierz odwrotna
pinv(A)	pseudoinwersja macierzy
lu(A)	rozkład LU macierzy
chol(A)	rozkład Choleskiego
eig(A)	wartości własne
qr(A)	rozkład QR macierzy

1.4 Metoda SOR

Okazuje się, że jeśli bieżącą poprawkę we wzorze (32) przemnoży się dodatkowo przez pewien parametr można uzyskać dodatkowe przyspieszenie procesu iteracyjnego

$$x_i^{(k+1)} = x_i^{(k)} + \omega r_i^{(k)} = x_i^{(k)} - \frac{\omega}{a_{ii}} \left(\sum_{j=1, i>1}^{i-1} a_{ij} x_j^{(k+1)} + \sum_{j=i}^{n} a_{ij} x_j^{(k)} - b_i \right) \quad (33)$$

co można zapisać macierzowo:

$$\mathbf{x}^{(k+1)} = (\mathbf{D} + \omega \mathbf{L})^{-1} (\omega \mathbf{b} - (\omega \mathbf{U} + (\omega - 1)\mathbf{D}) \mathbf{x}^{(k)}) \quad (34)$$

Parametr ω jest nazywany parametrem relaksacji, a sama metoda nosi nazwę metody kolejnych nadrelaksacji (ang. succesive overrelaxation - SOR). Udowodniono, że zbieżność procesu iteracyjnego jest zapewniona pod warunkiem, że $\omega \in (0, 2)$. W przypadku gdy $\omega = 1$ metoda SOR oczywiście upraszcza się do metody Gaussa-Seidla.

1.5 Octave/Matlab i URL

W programach typu Matlab istnieje pokaźna biblioteka funkcji umożliwiających szereg zaawansowanych operacji na macierzach i wektorach, dzięki czemu, w wielu wypadkach zbędne jest pisanie własnych procedur. Ich część wraz z krótkim opisem zawiera Tabela (1).

Zgodnie z teorią, do rozwiązania URL zapisanego w postaci $Ax = b$ w programie Matlab/Octave można posłużyć się poleceniem

```
>> x = inv(A) * b
```

lub

```
>> x = A^-1 * b
```

W praktyce zalecane jest używanie lewostronnego operatora dzielenia
$>>$ x = A \ b

ponieważ powyższe działanie jest o wiele szybsze, niż rozwiązywanie URL poprzez jawne odwracanie macierzy.

1.6 Zadanie

W programie Octave zapoznaj się z działaniem i sposobami wywoływania poniżej podanych funkcji:

- size
- sum
- triu
- diag

2 Ćwiczenia

W czasie zajęć będziemy szukać rozwiązania układu równań liniowych postaci:

$$\begin{array}{rcrcrcr} 3x_1 & + & x_2 & - & x_3 & = & 6 \\ -x_1 & + & 5x_2 & - & x_3 & = & 10 \\ 2x_1 & + & 4x_2 & + & 8x_3 & = & 2 \end{array} \tag{35}$$

1. Utwórz skrypt `main.m`, a w nim zdefiniuj zmienne `A` i `b` będące odpowiednio macierzą współczynników i wektorem wyrazów wolnych układu (35).

2. Rozwiąż powyższy układ równań przy pomocy operatora lewostronnego dzielenia (rozwiązaniem są liczby całkowite).

3. Na podstawie algorytmów 1 i 2 napisz (w oddzielnym pliku) funkcję rozwiązującą URL przy pomocy metody eliminacji Gaussa bez wyboru elementu podstawowego. Postać jej wywołania to:

   ```
   [x, Ur] = f_gauss(A,b)
   ```

   gdzie `A` – kwadratowa macierz współczynników, `b` – wektor wyrazów wolnych, `x` – wektor szukanych (rozwiązanie URL), `Ur` – macierz rozszerzona układu przekształcona do macierzy trójkątnej górnej.

   Uwaga: Macierz `Ur` utworzona zgodnie z algorytmem 1 nie będzie macierzą z zerami poniżej przekątnej głównej (elementów tych nie wykorzystuje się na dalszym etapie obliczeń).

   Uwaga 2: Przed rozpoczęciem głównej pętli algorytmu redukcji wstecznej wektor **x należy** odpowiednio zainicjalizować (**x** to nie skalar, tylko wektor zerowy o 1 kolumnie i n wierszach, gdzie n to stopień macierzy **A**).

   Uwaga 3: W Octave operator `==` to nie to samo co `=`.

   Uwaga 4 (Dla tych, którzy użyli funkcji `size` to określenia stopnia macierzy): Proszę zwracić uwagę na to co zwraca ta funkcja w zależności od podanych parametrów wejściowych i wyjściowych (→ `help size`).

4. Wewnątrz skrypu `main.m` wywołaj napisaną przez siebie funkcję `f_gauss` na rzecz zmiennych `A`, `b`. Poprawność funkcji można sprawdzić porównując jej wynik z rozwiązaniem otrzymanym w zad. 2.

5. Dokonaj rozkładu macierzy `A` na macierze trójkątną dolną i górną przy pomocy funkcji programu Matlab/Octave `lu`. Postaci wywołania tej funkcji można sprawdzić wpisując `help lu` w linii poleceń.

6. Na podstawie algorytmu 3 zaimplementuj rozkład LU przy pomocy metody Gaussa-Doolittle'a bez wyboru elementu podstawowego. Wywołanie funkcji powinno wyglądać następująco:

   ```
   [L, U] = f_gauss_doolittle(A)
   ```

7. Sprawdź poprawność zaimplementowanej przez siebie funkcji porównując zwrócony przez nią wynik z rezultatem wywołania funkcji `[L,U]=lu(A)`.

8. Napisz funkcję `f_gauss_jordan` na podstawie algorytmu opisanego w rozdziale 1.1.2 (wzory (12),(13)). Postać jej wywołania to:

   ```
   [x] = f_gauss_jordan(A,b)
   ```

   gdzie A - kwadratowa macierz współczynników, b - wektor wyrazów wolnych, x - wektor szukanych (rozwiązanie URL).

9. (Dla zaawansowanych) Przepisz funkcje `f_gauss`, `f_gauss_jordan` oraz `f_jacobi`, tak aby obliczenia wykonywać na całych wierszach (operator :), a nie na każdym elemencie wiersza z osobna.

10. (Dla zaawansowanych) Spróbuj rozwiązać za pomocą napisanej przez siebie funkcji `f_gauss` układ równań liniowych $Ax = b$, gdzie

    $$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 1 \\ -2 & 1 & 3 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$$

    Jego rozwiązaniem jest wektor $x = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$. Co jest powodem błędnego działania funkcji? Uzupełnij algorytm o część implementującą wybór częściowy elementu podstawowego, tak by funkcja dawała poprawny rezultat.

---

11. Napisz funkcję rozwiązującą URL przy pomocy metody iteracji prostej (rozdział 1.2.1 → wzory (22) albo (23) albo (30)). Jej deklaracja powinna wyglądać:

    ```
    [xx, kk] = f_jacobi(A,b, k_max, tol)
    ```

    gdzie `A` – kwadratowa macierz współczynników, `b` – wektor wyrazów wolnych, `k_max` - maksymalna ilość iteracji, `tol` – żądana dokładność rozwiązania. `xx` – wektor szukanych (rozwiązanie URL), `kk` – numer iteracji na której zakończono obliczenia.

    Uwaga: Ze względu na prostotę implementacji, zalecanym sposobem jest wykorzystanie wzoru (30) (dzięki czemu łatwo będzie można zmodyfikować funkcję celem implementacji kolejnych metod, por. wzory (31) i (34)).

W takim przypadku przydatne będą funkcje `diag`, `triu`, `tril`, w celu rozkładu macierzy $\mathbf{A}$ wg wzoru (28).

12. Rozwiąż układ równań z zadania 1 przy pomocy funkcji `f_jacobi` . Czy ten układ równań spełnia warunki zbieżności metody iteracji prostej? Podaj przykład takiego URL, dla którego metoda Jacobiego będzie rozbieżna. Sprawdź działanie swojej metody dla tego przykładu.

13. Korzystając z podanych wzorów zaimplementuj metody iteracyjne Gaussa-Seidla i SOR.

14. Dobierz eksperymentalnie optymalną wartość parametru relaksacji w metodzie SOR dla układu równań (35).

    Podpowiedź: Potraktuj ω jako parametr i wywołuj funkcję SOR dla różnych parametrów. Zapisanie liczby iteracji potrzebnych na uzyskanie zbieżności dla każdej wartości ω pozwoli odczytać (przybliżoną) optymalną wartość z wykresu (`plot(omega, liczba_iteracji)`).

15. Pobierz pliki zgodnie z instrukcjami prowadzącego. Otwórz funkcję `generujMacierz.m`. Funkcja dla zadanego parametru `n` generuje macierz o rozmiarach $n^2 \times n^2$ o poniższej strukturze:

$$\begin{bmatrix} 4 & -1 & 0 & \cdots & -1 & \cdots & 0 \\ -1 & 4 & -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & -1 & 4 & \ddots & \ddots & \ddots & -1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ -1 & \ddots & \ddots & \ddots & 4 & -1 & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 & 4 & -1 \\ 0 & \cdots & -1 & \cdots & 0 & -1 & 4 \end{bmatrix} \tag{36}$$

    Wywołaj te funkcję w linii poleceń dla niewielkich wartości `n`. Tego typu macierze powstają w sposób naturalny w przypadku dyskretyzacji dwuwymiarowych zagadnień opisywanych równaniem różniczkowym Laplace'a lub Poissona metodą różnic skończonych.

    W skrypcie dodaj linie z kodem:

```
n = 20 ;
M = generujMacierz(n) ; % nie zapomnij o średniku!
x = ones(length(M),1) ; b = M*x ;
S = sparse(M)  % zwróć uwagę na sposób przedstawiania S
size(M)
size(S)
sizeof(M) % Uwaga: rozmiar zmiennej podawany w bajtach
```

`sizeof(S)`

`nnz(S)`

`spy(S)`

Macierze M i S będą macierzami o tych samych wartość ale będą zapamiętywane jako pełna i rzadka. Sprawdź rozmiar tych macierzy, ilość pamięci potrzebnej na przechowywanie jednej i drugiej zmiennej, oraz liczbę elementów niezerowych. Jaki procent macierzy to elementy niezerowe?

16. Porównaj zbieżność metod iteracyjnych napisanych przez Ciebie (iteracji prostych, Gaussa-Seidla, SOR) oraz dwóch najpopularniejszych obecnie metod – metod przestrzeni Kryłowa: CG, GMRES, wykorzystanych do rozwiązania układu

$$\mathbf{Mx} = \mathbf{b}$$

gdzie, **M** będzie macierzą wygenerowaną przy pomocy funkcji z zadania 15, a wektor **b** będzie wynikał z przyjętego wektora rozwiązania.

17. Rzuć okiem na kody funkcji `f_cg` oraz `f_gmres`.

18. Porównaj czas rozwiązywania zagadnienia w przypadku użycia macierzy M i S:

    `tic; xx = M\b ; toc`

    `tic; xx = S\b ; toc`

    Jakie wnioski się nasuwają?

19. Porównaj czas rozwiązywania zagadnienia funkcjami ww. metodami, a rozwiązaniem poprzez użycie lewostronnego operatora dzielenia.

    Jakie wnioski się nasuwają?

20. Wartości własne/ wektory własne ...