WPROWADZENIE

Zajmiemy się teraz metodami numerycznego rozwiązywania układów równań liniowych $Ax = b$, gdzie A jest nieosobliwą macierzą $n \times n$ (macierz kwadratowa o takiej samej liczbie wierszy i kolumn równej n). Na kolejnych ćwiczeniach rozważymy także zagadnienia pokrewne, tj. odwracanie macierzy, obliczanie wyznaczników, znajdowanie wektorów własnych i liniowe zadanie najmniejszych kwadratów.

Podstawowe narzędzia matematyczne to algebra macierzy, normy wektorów i macierzy, wektory i wartości własne.

Celem ćwiczeń nr 4 i 5 jest zapoznanie się z *bezpośrednimi metodami* rozwiązywania algebraicznych układów równań liniowych.

Układ *n* równań liniowych z *n* niewiadomymi ma postać

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{cases} \tag{1}$$

w którym współczynniki (a_{ij}) oraz (b_i) są dane, a $x_1, \ldots, x_n$ to szukane niewiadome. Opis takich układów jest wygodny w języku macierzy. W oparciu o definicję mnożenia macierzy możemy układ (1) zapisać następująco:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \tag{2}$$

czyli

$$Ax = b, \tag{3}$$

gdzie

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}. \tag{4}$$

Z algebry liniowej wiadomo, że układ (1) ma jednoznaczne rozwiązanie dla dowolnej prawej strony wtedy i tylko wtedy, gdy $\det A \neq 0$. Warunek ten oznacza również, że macierz układu jest odwracalna, tzn. istnieje macierz A^{-1} taka że:

$$AA^{-1} = A^{-1}A = I, \tag{5}$$

gdzie symbol I oznacza macierz jednostkową

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & & 1 \end{bmatrix}. \tag{6}$$

Posługując się macierzą odwrotną A^{-1} możemy w teorii rozwiązać układ równań następująco:

$$Ax = b \;\Rightarrow\; A^{-1}(Ax) = A^{-1}b \;\Rightarrow\; (A^{-1}A)x = A^{-1}b \;\Rightarrow\; I\,x = A^{-1}b,$$

zatem

$$x = A^{-1}b, \tag{7}$$

gdyż $Ix = x$. Oczywiście głównym problemem jest znalezienie macierzy odwrotnej A^{-1}. Dla macierzy stopnia 2 lub 3 można to w miarę szybko zrobić posługując się wzorami (n = 2):

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \dfrac{a_{22}}{\det A} & -\dfrac{a_{12}}{\det A} \\ -\dfrac{a_{21}}{\det A} & \dfrac{a_{11}}{\det A} \end{bmatrix}, \quad \text{gdzie } \det A = a_{11}a_{22} - a_{12}a_{21}, \tag{8}$$

czyli

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{9}$$

Dla $n = 3$ obliczając macierz dopełnień algebraicznych otrzymamy

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \frac{1}{\det A}\begin{pmatrix} -a_{23}a_{32} + a_{22}a_{33} & a_{13}a_{32} - a_{12}a_{33} & -a_{13}a_{22} + a_{12}a_{23} \\ a_{23}a_{31} - a_{21}a_{33} & -a_{13}a_{31} + a_{11}a_{33} & a_{13}a_{21} - a_{11}a_{23} \\ -a_{22}a_{31} + a_{21}a_{32} & a_{12}a_{31} - a_{11}a_{32} & -a_{12}a_{21} + a_{11}a_{22} \end{pmatrix} \tag{10}$$

$$\text{gdzie } \det A = -a_{13}a_{22}a_{31} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{11}a_{22}a_{33}$$

Dla wyższych stopni znajdowanie macierzy A^{-1} poprzez dopełnienia algebraiczne jest niepraktyczne. Aby móc efektywnie rozwiązywać układy równań postaci (1) należy stosować odpowiednie procedury. Dzielimy je na dwie kategorie: metody bezpośrednie i metody iteracyjne. Zanim przejdziemy do metody eliminacji Gaussa dla pełnej macierzy, zajmiemy się dwoma szczególnymi przypadkami: (i) macierz A jest trójprzekątniowa, (ii) macierz A jest trójkątna (górna lub dolna).

ROZWIĄZYWANIE UKŁADÓW Z MACIERZĄ TRÓJPRZEKĄTNIOWĄ

Rozważmy układ $Ax = b$ z macierzą, w której poza główną przekątną oraz dwiema sąsiednimi występują zera:

$$A = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ a_1 & d_2 & c_2 & & 0 \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & a_{n-2} & d_{n-1} & c_{n-1} \\ 0 & & 0 & a_{n-1} & d_n \end{bmatrix}. \tag{11}$$

Oznacza to, że układ równań ma postać

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1 \\ a_1x_1 + d_2x_2 + c_2x_3 &= b_2 \\ a_2x_2 + d_3x_3 + c_3x_4 &= b_3 \\ \cdots\cdots & \\ a_{n-1}x_{n-1} + d_nx_n &= b_n \end{aligned} \qquad (12)$$

Do rozwiązania układu z taką macierzą możemy posłużyć się następującym algorytmem:

for i=2 **to** n **do** $d_i = d_i - c_{i-1}\cdot(a_{i-1}/d_{i-1}),\ b_i = b_i - b_{i-1}\cdot(a_{i-1}/d_{i-1})$

$x_n = b_n / d_n$

for i=n-1 **to** 1 **do** $x_i = (b_i - c_i x_{i+1}) / d_i$

Łącząc oba fragmenty procedur uzyskamy następujący kod funkcji dla układów trójprzekątniowych. Zaprezentowana funkcja `triDiagSolve1` jest najprostszą realizacją, które nie zmienia wejściowych tablic `a` i `b`.

```
// Rozwiązywanie układu Ax=b z macierzą trójdiagonalną A=diag(a,d,c).
// Tablice jednowymiarowe a, d, c reprezentują elementy poniżej diagonalnej, na diagonalnej
// oraz powyżej. Tablica b reprezentuje prawą stronę układu.
void triDiagSolve1(double a[n-1], double d[n], double c[n-1], double b[n], double x[n]) {
    double dTmp[n], bTmp[n];
    int i;
    // Zapamiętywanie tablic d i b w pomocniczych dTmp i bTmp. Robimy to, aby po wyjściu
    // z funkcji tablice d i b nie były zmienione. Tablic a i c nie trzeba zapamiętywać,
    // gdyż procedura wogóle ich nie modyfikuje.
    for (i=1; i < n; i++) { dTmp[i] = d[i]; bTmp[i] = b[i]; }
    // Właściwy algorytm rozwiązywania układu trójdiagonalnego: eliminacja.
    for (i=1; i < n; i++) {
        d[i] = d[i] - c[i-1]*(a[i-1]/d[i-1]);
        b[i] = b[i] - b[i-1]*(a[i-1]/d[i-1]);
    }
    // Właściwy algorytm rozwiązywania układu trójdiagonalnego: wylicznie rozwiązań od końca.
    x[n-1] = b[n-1]/d[n-1];
    for (i=n-2; i >= 0; i--) x[i] = (b[i] - c[i]*x[i+1])/d[i];
    // Przywracanie wartości w tablicach d i b.
    for (i=1; i < n; i++) { d[i] = dTmp[i]; b[i] = bTmp[i]; }
}
```

Zadanie 1. Zaimplementować i przetestować w języku C/C++ metodę rozwiązywania układu trójprzekątniowego.

Dane sprawdzające: dla układu (n=5) opisanego macierzami

$$A = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 5 & -2 & 0 \\ 0 & 0 & 2 & 7 & 1 \\ 0 & 0 & 0 & 3 & 8 \end{bmatrix},\quad b = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 3 \\ -1 \end{bmatrix},$$

rozwiązanie (z dokładnością do czterech miejsc po przecinku) wynosi:

$$x_1 = 0.3264,\ x_2 = 0.3472,\ x_3 = 0.2848,\ x_4 = 0.3857,\ x_5 = -0.2696.$$

UKŁAD Z MACIERZĄ TRÓJKĄTNĄ GÓRNĄ

Rozważmy następujący układ równań liniowych

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ \quad\quad a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2, \\ \quad\quad\quad\quad a_{33}x_3 + \ldots + a_{3n}x_n = b_3, \\ \quad\quad\quad\quad\quad\quad\quad\quad \vdots \quad\quad \vdots \\ \quad\quad\quad\quad\quad\quad\quad\quad a_{nn}x_n = b_n. \end{cases} \tag{13}$$

Rozwiązywanie tego układu jest bardzo proste. Zaczynamy od „końca”: z ostatniego równania wyliczamy x_n, następnie z przedostatniego x_{n-1} itd. aż do pierwszego, z którego wyliczamy x_1 :

$$\begin{aligned} & x_n = b_n / a_{nn}, \\ & x_{n-1} = (b_{n-1} - a_{n-1,n}x_n) / a_{n-1,n-1}, \\ & x_{n-2} = (b_{n-2} - a_{n-2,n-1}x_{n-1} - a_{n-2,n}x_n) / a_{n-2,n-2}, \\ & \ldots \\ & x_i = (b_i - \textstyle\sum_{j=i+1}^{n} a_{ij}x_j) / a_{ii}. \end{aligned} \tag{14}$$

Zauważmy, że w każdym kroku uzyskujemy kolejną niewiadomą w sekwencji $x_n, x_{n-1},\ldots,x_1$, przy czym korzystamy z już otrzymanych wcześniej rozwiązań.

Poniżej jest przedstawiony algorytm zapisany w języku C rozwiązywania układu trójkątnego. W stosunku do zapisu, który używaliśmy wyżej jest jedna mała zmiana polegająca na tym, że indeksy w tablicy (macierzy) oraz w kolumnach zaczynają się od 0 (a nie od 1) i kończą na *n-1* (a nie na *n*).

```
double a[n][n], b[n], x[n], sum;
int i, j;
x[n-1] = b[n-1]/a[n-1][n-1];
for (i=n-2; i >= 0; i--) {
      sum = 0.0;
      for (j=i+1; j < n; j++) sum = sum + a[i][j]*x[j];
      x[i] = (b[i] - sum)/a[i][i];
}
```

Gdy się przyglądniemy się dokładniej powyższemu algorytmowi, to zauważmy że nie jest potrzebne odrębne przypisanie przed wejściem do pętli **`for`**, gdyż dla pierwszej wyliczanej składowej rozwiązania (tj. dla `x[n-1]` w notacji języka C/C++) pętla wewnętrzne i tak się nie wykona, więc zmienna `sum` pozostanie równa zero. Tak więc bardziej zwarta postać algorytmu jest następująca:

```
double a[n][n], b[n], x[n], sum;
int i, j;
for (i=n-1; i >= 0; i--) {
      sum = 0.0;
      for (j=i+1; j < n; j++) sum = sum + a[i][j]*x[j];
      x[i] = (b[i] - sum)/a[i][i];
}
```

Zadanie 2. Zaimplementować i przetestować metodę rozwiązywanie układu z macierzą trójkątna górną. Rozwiązać przykładowy układ dla $n = 7$ i sprawdzić rozwiązanie przez podstawienie i porównanie z prawą stroną.