


Paulina Grabowska 	Metody Numeryczne
Rozwiązywanie układów równań liniowych - metoda eliminacji Gaussa	

1. Wstęp

Równania liniowe to podstawowe równania matematyczne, których wykresy tworzą linie, płaszczyzny lub hiperpłaszczyzny, a zbiór rozwiązań tworzy przestrzeń liniową. Każde równanie liniowe składa się z liniowej kombinacji zmiennych, zwykle oznaczonych jako x_1, x_2, \dots, x_n gdzie każdy wyraz ma wykładnik 1.

Rozwiązanie układów równań liniowych polega na znalezieniu wartości zmiennych, które spełniają wszystkie równania w układzie. Metody rozwiązywania takich układów mogą być różnorodne, w zależności od wielkości układu, jego właściwości oraz wymagań dotyczących dokładności rozwiązania. Metoda eliminacji Gaussa, oparta na operacjach elementarnych na macierzach, jest jedną z najczęściej stosowanych metod rozwiązywania układów równań liniowych, ze względu na swoją efektywność i wszechstronność.

2. Opis metody eliminacji Gaussa

Metoda eliminacji Gaussa jest kluczowym algorytmem używanym do rozwiązywania układów równań liniowych poprzez przekształcenie macierzy współczynników równań do postaci górnej trójkątnej, a następnie znalezienie rozwiązań poprzez postępowanie odwrotne. Rozważmy układ n równań liniowych z n zmiennymi:

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

Pierwszym krokiem jest przekształcenie macierzy współczynników a oraz wektora wyrazów wolnych b tak, aby wyeliminować elementy pod przekątną macierzy A . Dokonuje się tego poprzez wykonanie odpowiednich operacji na każdym wierszu układu równań. Polega to na

iteracyjnym zerowaniu elementów pod przekątną przez odejmowanie wielokrotności odpowiednich wierszy od wierszy poniżej.

$$\begin{bmatrix} a_{11} + a_{12} + \dots + a_{1n} \\ a_{21} + a_{22} + \dots + a_{2n} \\ \dots \\ a_{n1} + a_{n2} + \dots + a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \quad (2)$$

Po wykonaniu eliminacji, macierz współczynników a zostaje przekształcona do postaci górnej trójkątnej, w której wszystkie elementy pod przekątną są zerami.

Po uzyskaniu macierzy górnej trójkątnej, rozwiązujemy układ równań korzystając z metody podstawiania wstecz. Rozpoczynamy od ostatniego równania i wyznaczamy wartości kolejnych zmiennych x_n, x_{n-1}, \dots, x_1 poprzez podstawienie obliczonych już wartości w kolejnych równaniach.

Metoda podstawiania wstecz jest etapem finalnym procesu eliminacji Gaussa, który służy do znalezienia dokładnych rozwiązań układu równań liniowych po przekształceniu macierzy współczynników do postaci górnej trójkątnej.

Następnie przechodzimy do poprzednich równań, iterując wstecz, i wyznaczamy kolejne zmienne, stosując tę samą metodę. Proces ten kontynuujemy aż do rozwiązania wszystkich zmiennych w układzie równań.

Metoda eliminacji Gaussa jest podstawowym narzędziem w matematyce stosowanym do rozwiązywania układów równań liniowych. Jest szeroko stosowana w różnych dziedzinach nauki i techniki ze względu na swoją efektywność i prostotę implementacji.

3. Implementacja numeryczna

3.1. Funkcja *displayMatrix*

Funkcja *displayMatrix* przyjmuje dwuwymiarową tablicę *matrix* typu *double* o stałej liczbie kolumn (4, ale można zmieniać tę liczbę w zależności od potrzeb) oraz liczbę całkowitą n , która określa liczbę wierszy tej tablicy (*Fragment kodu 1*). Funkcja ta jest odpowiedzialna za wyświetlanie zawartości tablicy *matrix*. Pętla zewnętrzna iteruje przez wiersze tablicy *matrix*, zaczynając od 0 i kończąc na $n-1$.

Pętla wewnętrzna iteruje przez kolumny tablicy *matrix*, zaczynając od 0 i kończąc na n . Jest to zrobione w ten sposób, ponieważ tablica *matrix* ma n kolumn oraz dodatkową kolumnę, co jest wskazane przez $n+1$. Wewnętrzna pętla wyświetla wartości elementów tablicy *matrix*. Elementy są wyświetlane zgodnie z ich indeksami $[i][j]$. Po wyświetleniu każdego elementu jest dodawany znak tabulacji ($\backslash t$), aby oddzielić wartości.

Po wyświetleniu wszystkich elementów w danym wierszu jest wypisywana dodatkowa pusta linia celem zapewnienia większej i rozdzielni zawartość wierszy tablicy *matrix*.

```

1. void displayMatrix(double matrix[][4], int n) {
2.     for (int i = 0; i < n; ++i) {
3.         for (int j = 0; j < n + 1; ++j) {
4.             cout << matrix[i][j] << "\t";
5.         }
6.         cout << endl;
7.     }
8. }
9.

```

Fragment kodu 1: Funkcja *displayMatrix*

3.2. Funkcja *gaussianElimination*

Funkcja ta realizuje algorytm eliminacji Gaussa, który jest często stosowany do rozwiązywania układów równań liniowych (*Fragment kodu 2*). Na początku, dla każdej kolumny i , znajduje się wiersz z największym elementem bezwzględny w tej kolumnie i zamienia go miejscami z wierszem i . To zapewnia, że elementy na przekątnej macierzy są największe w swojej kolumnie, co ułatwia dalsze operacje.

Następnie, iteruje przez pozostałe wiersze (od $i + 1$ do n) i aktualizuje elementy macierzy w celu uzyskania macierzy górnej trójkątnej. W ten sposób, macierz jest przekształcana w formę, w której elementy pod przekątną są zerami.

Po uzyskaniu macierzy górnej trójkątnej, algorytm przechodzi do postępowania odwrotnego. Rozpoczyna od ostatniego równania i oblicza wartości zmiennych, zaczynając od tej ostatniej. Iteruje wstecz przez pozostałe równania, aktualizując wartości zmiennych, aby ostatecznie znaleźć rozwiązania dla wszystkich zmiennych.

Rozwiązania te są przechowywane w tablicy *solution*. Na koniec, funkcja wyświetla te rozwiązania na ekranie w formie " $x_i = \text{wartość}$ ". Dzięki temu łatwo jest odczytać wartości poszczególnych zmiennych i zrozumieć, jakie są rozwiązania dla danego układu równań. Cały

proces odnajdywania rozwiązań układu równań jest kompleksowy, ale algorytm eliminacji Gaussa jest skutecznym sposobem na znalezienie tych rozwiązań.

```

1. void gaussianElimination(double matrix[][4], int n) {
2.     for (int i = 0; i < n; ++i) {
3.         double maxEl = abs(matrix[i][i]);
4.         int maxRow = i;
5.         for (int k = i + 1; k < n; ++k) {
6.             if (abs(matrix[k][i]) > maxEl) {
7.                 maxEl = abs(matrix[k][i]);
8.                 maxRow = k;
9.             }
10.        }
11.
12.        for (int k = i; k < n + 1; ++k) {
13.            double tmp = matrix[maxRow][k];
14.            matrix[maxRow][k] = matrix[i][k];
15.            matrix[i][k] = tmp;
16.        }
17.
18.        for (int k = i + 1; k < n; ++k) {
19.            double c = -matrix[k][i] / matrix[i][i];
20.            for (int j = i; j < n + 1; ++j) {
21.                if (i == j) {
22.                    matrix[k][j] = 0;
23.                } else {
24.                    matrix[k][j] += c * matrix[i][j];
25.                }
26.            }
27.        }
28.    }
29.
30.    // postepowanie odwrotne
31.    double solution[n];
32.    for (int i = n - 1; i >= 0; --i) {
33.        solution[i] = matrix[i][n] / matrix[i][i];
34.        for (int k = i - 1; k >= 0; --k) {
35.            matrix[k][n] -= matrix[k][i] * solution[i];
36.        }
37.    }
38.
39.    cout << "\n output:" << endl;
40.    for (int i = 0; i < n; ++i) {
41.        cout << "x" << i + 1 << " = " << solution[i] << endl;
42.    }
43. }
44.

```

Fragment kodu 2: funkcja *gaussianElimination*

3.3. Funkcja *main*

Funkcja *main* jest centralnym punktem programu, gdzie wywoływane są funkcje *displayMatrix* i *gaussianElimination*, a także zdefiniowane są dane wejściowe w postaci macierzy (Fragment kodu 3). Po pierwsze, tworzona jest dwuwymiarowa tablica *matrix* o rozmiarze 3x4, zawierająca wartości współczynników równań oraz wyrazów wolnych. Następnie zmienna *n* jest ustawiana na wartość 3, co odpowiada liczbie równań w układzie.

Po zainicjowaniu danych wejściowych, program wyświetla macierz wejściową na ekranie za pomocą funkcji *displayMatrix*, która została wcześniej zdefiniowana. Następnie, wywoływana jest funkcja *gaussianElimination*, która przetwarza macierz wejściową, stosując algorytm eliminacji Gaussa w celu znalezienia rozwiązania układu równań.

```

1. int main() {
2.     double matrix[3][4] = {{10, -7, 0, 6},
3.                             {-3, 2, 6, 4},
4.                             {5, -1, 5, 3}};
5.     int n = 3;                                // rozmiar macierzy
6.
7.     cout << "input matrix:" << endl;
8.     displayMatrix(matrix, n);
9.
10.    gaussianElimination(matrix, n);
11.
12.    return 0;
13. }
14.

```

Fragment kodu 3: Funkcja *main*

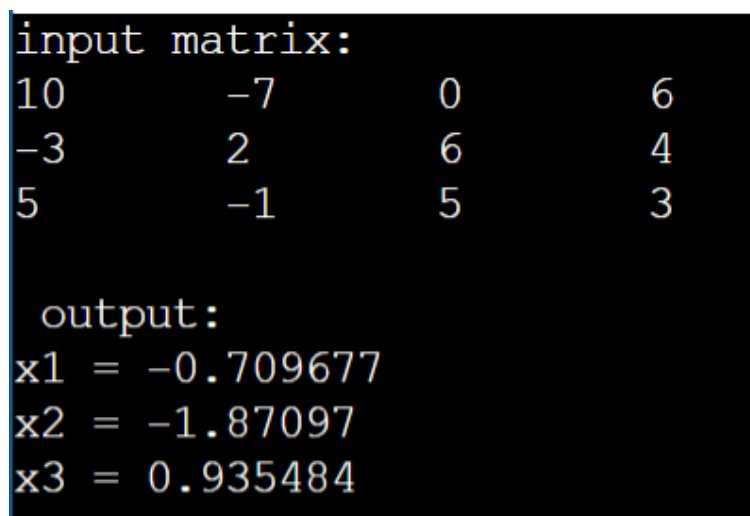
Sprawdzono poprawność działania funkcji dla macierzy:

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \\ 3 \end{bmatrix}$$

Spodziewanymi wartościami są:

$$\begin{aligned} x_1 &= -0,709677 \\ x_2 &= -1,87097 \\ x_3 &= 0,935484 \end{aligned}$$

Taki też wynik otrzymano (Rys. 1), co pozwala przypuszczać, że funkcja została zaimplementowana poprawnie.



```

input matrix:
10      -7      0      6
-3       2      6      4
5       -1      5      3

output:
x1 = -0.709677
x2 = -1.87097
x3 = 0.935484

```

Rys.1 Ekran konsoli dla danych testowych (zestaw 1)

Sprawdzono również poprawność działania funkcji dla macierzy:

$$\begin{bmatrix} 4 & -2 & 4 & -2 \\ 3 & 1 & 4 & 2 \\ 2 & 4 & 2 & 1 \\ 2 & -2 & 4 & 2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 10 \\ 2 \end{bmatrix}$$

W przypadku tej macierzy wymagana była zmiana rozmiarów tablic w argumentach przyjmowanych przez obie funkcje. Spodziewanymi wartościami są:

$$\begin{aligned} x_1 &= -1 \\ x_2 &= 2 \\ x_3 &= 3 \\ x_4 &= -2 \end{aligned}$$

Taki też wynik otrzymano (Rys. 2), co pozwala przypuszczać, że funkcja została zaimplementowana poprawnie.

```
input matrix:
4      -2      4      -2      8
3       1      4       2      7
2       4      2       1     10
2      -2      4       2      2

output:
x1 = -1
x2 = 2
x3 = 3
x4 = -2
```

Rys.2 Ekran konsoli dla danych testowych (zestaw 2)

4. Testy jednostkowe

4.1. Test dla dużej liczby współczynników

Badany układ równań:

$$\begin{cases} 2x_1 + 2x_2 - x_3 + 4x_4 + 5x_5 + 2x_6 = 20 \\ 2x_1 - x_2 + 3x_3 + x_4 - 2x_5 + 3x_6 = 12 \\ 5x_1 + x_2 - 2x_3 - x_4 + 2x_5 + x_6 = -1 \\ -x_1 + 3x_2 + x_3 - 2x_4 - x_5 + 2x_6 = 7 \\ 4x_1 - 2x_2 - x_3 + 3x_4 + x_5 + 3x_6 = 10 \\ -2x_1 + x_2 + 3x_3 - x_4 + 2x_5 + 4x_6 = -8 \end{cases}$$

Aby móc wprowadzić powyższy układ równań w postaci macierzy do programu należało zmienić rozmiar tablic przyjmowanych jako argumenty w funkcjach *displayMatrix* oraz *gaussianElimination* z czterech na siedem. Wydruk z konsoli załączono na Rys. 3a, natomiast na Rys. 3b znajduje się wydruk kalkulatora MatrixCalc (oraz opcji rozwiązania metodą Gaussa), który sprawdzał poprawność otrzymanych wyników.

```
input matrix:
2      2      -1      4      5      2      20
2      -1     3      1     -2     3      12
5      1     -2     -1     2      1      -1
-1     3      1     -2     -1     2       7
4     -2     -1     3      1      3      10
-2     1      3     -1     2      4      -8

output:
x1 = 1.32072
x2 = 5.22535
x3 = 0.218558
x4 = 5.89201
x5 = -3.44197
x6 = 0.384093
```

Rys. 3a Wydruk z konsoli dla zestawu danych dla testu dla dużej liczby współczynników

Wynik:

$$\begin{aligned}
 x_1 &= 1,32072 \\
 x_2 &= 5,22535 \\
 x_3 &= 0,21856 \\
 x_4 &= 5,89201 \\
 x_5 &= -3,44197 \\
 x_6 &= 0,38409
 \end{aligned}$$

Ogólne rozwiązanie: $X = \begin{pmatrix} 1,32072 \\ 5,22535 \\ 0,21856 \\ 5,89201 \\ -3,44197 \\ 0,38409 \end{pmatrix}$

Rys. 3b Wyniki otrzymane w programie MatrixCalc dla testu dla dużej liczby niewiadomych

4.2. Test dla bardzo dużej ilości współczynników

Badano układ równań o dwudziestu niewiadomymi. Aby móc wprowadzić ten układ równań w postaci macierzy do programu zmieniono rozmiar tablic przyjmowanych jako argumenty w funkcjach *displayMatrix* oraz *gaussianElimination* na dwadzieścia jeden. Wydruk z konsoli załączono na Rys. 4. Spodziewane wyniki zawarte są w pliku testowym z zajęć.

```

output:      x11 = 11.4703
x1 = 12.1925  x12 = -15.6365
x2 = 7.24097  x13 = 15.9193
x3 = -23.52   x14 = 23.9103
x4 = 9.5621   x15 = -4.76982
x5 = 16.5062  x16 = -5.91822
x6 = -8.72811 x17 = 15.8692
x7 = -13.287  x18 = 10.3476
x8 = -21.118  x19 = -15.9591
x9 = 0.946473 x20 = -2.33728
x10 = -6.07601

```

Rys. 4 Wydruk z konsoli dla zestawu danych dla testu dla bardzo dużej liczby współczynników

4.3. Test dla układu równań sprzecznych

Badany układ równań:

$$\begin{cases} 2x_1 - 4x_2 + x_3 = 3 \\ 8x_1 - 2x_2 + 4x_3 = 7 \\ -4x_1 + x_2 - 2x_3 = -14 \end{cases}$$

Aby móc wprowadzić powyższy układ równań w postaci macierzy do programu zmieniono rozmiar tablic przyjmowanych jako argumenty w funkcjach *displayMatrix* oraz *gaussianElimination* na cztery. Wydruk z konsoli załączono na Rys. 5a, natomiast na Rys. 5b znajduje się wydruk kalkulatora MatrixCalc (oraz opcji rozwiązania metodą Gaussa), który sprawdzał poprawność otrzymanych wyników.

Zaimplementowana funkcja nie zawiera zabezpieczeń dla możliwości układu równań sprzecznych. Test ma na celu sprawdzenie zachowania się funkcji bez takich zabezpieczeń.

```
input matrix:
2      -4      1      3
8      -2      4      7
-4      1     -2     -14

output:
x1 = -nan
x2 = -nan
x3 = -inf
```

Rys. 5a Wydruk z konsoli dla zestawu danych dla testu dla równań sprzecznych

```

      ≡
{ x1 + 2·x2 - x3 + 3·x4 - 2·x5 = 10
  x2 - x3 + 1,4·x4 - 1,6·x5 = 3
    x3 - 5,4·x4 + 0,6·x5 = -15
      x4 + 0,4827586207·x5 = 3,9655172414
      0 = 5
      ≡

Brak rozwiązania.
```

Rys. 5b Wyniki otrzymane w programie MatrixCalc dla testu dla równań sprzecznych

4.4. Test dla układu równań nieoznaczonych

Badany układ równań:

$$\begin{cases} 5x_1 - 2x_2 - x_3 = -1 \\ 3x_1 - x_2 + 2x_3 = 4 \\ -2x_1 + x_2 + 3x_3 = 5 \end{cases}$$

Aby móc wprowadzić powyższy układ równań w postaci macierzy do programu pozostawiono rozmiar tablic przyjmowanych jako argumenty w funkcjach *displayMatrix* oraz *gaussianElimination* jako cztery. Wydruk z konsoli załączono na Rys. 6a, natomiast na Rys. 6b znajduje się wydruk kalkulatora MatrixCalc (oraz opcji rozwiązania metodą Gaussa), który sprawdzał poprawność otrzymanych wyników.

Zaimplementowana funkcja nie zawiera zabezpieczeń dla możliwości układu równań nieoznaczonych. Test ma na celu sprawdzenie zachowania się funkcji bez takich zabezpieczeń.

```
input matrix:
5      -2      -1      -1
3      -1       2       4
-2      1       3       5

output:
x1 = -nan
x2 = -nan
x3 = -nan
```

Rys. 6a Wydruk z konsoli dla zestawu danych dla testu dla równań nieoznaczonych

Wynik:

$$\begin{aligned} x_1 &= 9 - 5x_3 \\ x_2 &= 23 - 13x_3 \\ x_3 &= x_3 \end{aligned}$$

Ogólne rozwiązanie: $X = \begin{pmatrix} 9 - 5x_3 \\ 23 - 13x_3 \\ x_3 \\ \equiv \end{pmatrix}$

Rys. 6b Wyniki otrzymane w programie MatrixCalc dla testu dla równań nieoznaczonych

5. Opracowanie wyników

5.1. Test dla dużej liczby współczynników

Wartości otrzymane zgadzają się z wartościami oczekiwanymi z dokładnością do kilku miejsc po przecinku, co potwierdza poprawność działania programu. Dodatkowo, wyniki zostały zweryfikowane za pomocą kalkulatora MatrixCalc, który potwierdził poprawność otrzymanych rozwiązań. Wyniki dla testu dla dużej liczby współczynników zestawiono w *Tabeli 1*.

Tabela 1: Zestawienie wyników otrzymanych w teście dla dużej liczby współczynników

Niewiadoma	Wartość oczekiwana	Wartość otrzymana
x_1	1.32072	1.32072
x_2	5.22535	5.22535
x_3	0.21856	0.218558
x_4	5.89201	5.89201
x_5	-3.44197	-3.44197
x_6	0.38409	0.384093

5.2. Test dla bardzo dużej ilości współczynników

Wyniki porównano z tymi znajdującymi się w pliku z danymi – obliczone przez program wartości niewiadomych zgadzają się, chociaż względem odpowiedzi są mniej dokładne, o większym przybliżeniu.

5.3. Test dla układu równań sprzecznych

Układ równań zawiera sprzeczność, gdyż równania są względem siebie. Oczekuje się, że funkcja zwróci nieprawidłowy wynik lub wywoła błąd związany z próbą dzielenia przez zero. Funkcja nie zwróciła poprawnego rozwiązania i nie wygenerowała wyjątku – zgodnie z oczekiwaniami, gdyż algorytm nie posiada zabezpieczeń dla układów sprzecznych.

5.4. Test dla układu równań nieoznaczonych

Układ równań zawiera sprzeczność, gdyż drugie równanie jest wielokrotnością pierwszego równania. Oczekuje się, że funkcja zwróci nieprawidłowy wynik lub wygeneruje wyjątek związany z niemożnością przekształcenia dwóch zmiennych w równania względem trzeciej zmiennej. Implementowana funkcja nie zawiera zabezpieczeń dla możliwości układu równań nieoznaczonych, więc wyniki pokrywają się z oczekiwaniami – wyniki zestawiono w *Tabeli 2*.

Tabela 2: Zestawienie wyników otrzymanych w teście dla układu równań nieoznaczonych

Niewiadoma	Wartość oczekiwana	Wartość otrzymana
x_1	$9 - 5x_3$	-nan
x_2	$23 - 13x_3$	-nan
x_3	x_3	-nan

6. Wnioski

Metoda eliminacji Gaussa jest potężnym narzędziem do rozwiązywania układów równań liniowych. Jej zalety obejmują szybkość działania oraz prostotę implementacji. Jednakże, metoda ta wymaga pewnych zabezpieczeń, aby być użyteczną w praktyce.

Implementacja metody powinna zawierać zabezpieczenia dla przypadków układów równań sprzecznych bądź nieoznaczonych. Układy te mogą wystąpić w rzeczywistych problemach matematycznych i brak obsługi dla nich może prowadzić do nieprawidłowych wyników lub błędów wykonania.

W przypadku dużych układów równań, zastosowanie optymalizacji może być konieczne dla zapewnienia efektywnego działania metody eliminacji Gaussa. Istnieją różne techniki optymalizacyjne, takie jak eliminacja Gaussa z częściowym wyborem elementu głównego, które mogą poprawić wydajność metody w przypadku dużych układów równań.

Podsumowując, mimo swojej prostoty i skuteczności, metoda eliminacji Gaussa wymaga odpowiedniej implementacji i zabezpieczeń, aby być użyteczną w praktyce, szczególnie w przypadku rzeczywistych problemów matematycznych.

7. Źródła

Wykłady z Metod Numerycznych autorstwa dr hab. Danuty Szeligi

Prezentacja Rozwiązywanie układów równań liniowych - metoda eliminacji Gaussa autorstwa dr. Hab. Inż. Marcina Hojnego.

Wikipedia – artykuły o równaniach nieliniowych, nieliniowości, metodzie eliminacji Gaussa

Program MatrixCalc: <https://matrixcalc.org/pl/slu.html>