


| | |
|--|-------------------|
| Paulina Grabowska  | Metody Numeryczne |
| Rozwiązanie układów równań liniowych - metoda eliminacji Gaussa (metoda wyboru elementu głównego) | |

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 elementu głównego)

Metoda Gaussa z wyborem elementu głównego jest rozszerzeniem podstawowej metody eliminacji Gaussa, która ma na celu poprawę jej efektywności i stabilności numerycznej poprzez wybór odpowiedniego elementu głównego podczas eliminacji w przód.

Kluczowym krokiem metody eliminacji Gaussa jest wybór elementu głównego w danej kolumnie. Element główny to ten, który ma największą wartość bezwzględną w kolumnie, i jest wybierany w celu minimalizacji błędów zaokrągleń i utraty cyfr znaczących podczas obliczeń. Metoda elementu głównego polega na tym, że przed eliminacją w danym kroku iteracyjnym wybierany jest jako element główny ten, który pozwoli na najmniejsze przekształcenie wartości pozostałych elementów w kolumnie. Dzięki temu minimalizowane są błędy numeryczne i poprawiana jest stabilność algorytmu.

Rozpoczyna się od wyboru elementu głównego - dla każdego kroku eliminacji w przód wybieramy jako element główny ten, który ma największą wartość bezwzględną w danej kolumnie (bądź wierszu). Dla każdego wybranego elementu głównego wykonuje się operacje elementarne, aby wyeliminować zmienne w danej kolumnie poza elementem głównym.

Po zakończeniu eliminacji w przód, otrzymana macierz jest macierzą trójkątną górną. Następnie rozwiązujemy układ równań trójkątnych metodą wstecznej substytucji, zaczynając od ostatniego równania i przechodząc w górę, aby wyznaczyć wartości zmiennych nieznanych.

Metoda Gaussa z wyborem elementu głównego jest bardziej wydajna i stabilna numerycznie niż podstawowa metoda eliminacji Gaussa, co sprawia, że jest szeroko stosowana w praktyce do rozwiązywania układów równań liniowych. Jednakże wymaga ona dodatkowych obliczeń związanych z wyborem elementu głównego, co może zwiększyć złożoność obliczeniową algorytmu.

3. Implementacja numeryczna

3.1. funkcja *printMatrix* i stała *EPSILON* (Fragment kodu 1)

Kod rozpoczyna się od definicji stałej *EPSILON*, która jest bardzo małą liczbą. Jest to wartość używana do porównań, szczególnie w kontekście porównywania liczb zmiennoprzecinkowych bliskich zeru. Wartość ta pomaga uniknąć problemów związanych z niedokładnościami arytmetyki zmiennoprzecinkowej.

Następnie definiowana jest funkcja *printMatrix*, która przyjmuje jako argumenty dwuwymiarową tablicę wskaźników do typu *double* (reprezentującą macierz) oraz liczbę całkowitą n , określającą rozmiar macierzy. Pierwsza pętla iteruje od 0 do $n-1$, reprezentując numer wiersza macierzy.

Wewnętrzna pętla *for* iteruje od 0 do n , z dodatkowym krokiem, aby uwzględnić dodatkową kolumnę w macierzy (często jest to kolumna wyników, stąd $n + 1$). Reprezentuje ona numer kolumny w danym wierszu macierzy.

Następnie wypisywana jest wartość elementu macierzy znajdującego się w aktualnie iterowanym wierszu i i kolumnie j na standardowym wyjściu ekranie.

```

1. const double EPSILON = 1e-10;
2.
3. void printMatrix(double **matrix, int n) {
4.     for (int i = 0; i < n; ++i) {
5.         for (int j = 0; j < n + 1; ++j) {
6.             cout << matrix[i][j] << " ";
7.         }
8.         cout << endl;
9.     }
10. }

```

Fragment kodu 1: Funkcja *printMatrix*

3.2. funkcja *findPivot* (Fragment kodu 2)

Funkcja ta służy do znajdowania indeksu elementu głównego w danej kolumnie macierzy w kontekście metody eliminacji Gaussa z wyborem elementu głównego. Funkcja przyjmuje trzy argumenty: dwuwymiarową tablicę wskaźników do typu *double* (reprezentującą macierz), *k* jako numer kolumny, w której chcemy znaleźć element główny, oraz *n* jako rozmiar macierzy.

Tworzona jest zmienna *pivot_row*, która początkowo przyjmuje wartość *k*, czyli numer wiersza, od którego zaczynamy przeszukiwanie w kolumnie. Następnie zmienna *max_val* jest wartością bezwzględną elementu na przekątnej macierzy w kolumnie *k*. Jest to wartość, którą będzie porównywana z innymi elementami w kolumnie, aby znaleźć element główny.

Pętla *for* rozpoczyna się od $k + 1$, ponieważ przeszukiwane są tylko wiersze pod przekątną, ponieważ elementy powyżej przekątnej już zostały sprawdzone i nie mogą być elementem głównym. Sprawdzany jest warunek, czy wartość bezwzględna aktualnie rozpatrywanego elementu w kolumnie jest większa od dotychczasowego *max_val*. Jeśli wartość bezwzględna aktualnego elementu jest większa od *max_val*, następuje aktualizacja *max_val* oraz przypisanie *pivot_row* wartość *i*, czyli numer wiersza, w którym znajduje się ten nowy maksymalny element.

```

1. int findPivot(double **matrix, int k, int n) {
2.     int pivot_row = k;
3.     double max_val = abs(matrix[k][k]);
4.
5.     for (int i = k + 1; i < n; ++i) {
6.         if (abs(matrix[i][k]) > max_val) {
7.             max_val = abs(matrix[i][k]);
8.             pivot_row = i;
9.         }
10.    }
11.
12.    return pivot_row;
13. }

```

Fragment kodu 2: Funkcja *findPivot*

3.3. funkcja *swapRows* (Fragment kodu 3)

Ta funkcja służy do zamiany miejscami dwóch wierszy w macierzy. Funkcja przyjmuje cztery argumenty: *matrix* (dwuwymiarową tablicę wskaźników do typu *double*, reprezentującą macierz), *row1* i *row2* (numery wierszy, które zostaną zamienione) oraz *n* (rozmiar macierzy - liczba kolumn).

Pętla *for* iteruje przez wszystkie kolumny macierzy, włącznie z dodatkową kolumną, która często przechowuje wyniki. Zmienna *temp* przechowuje wartość elementu macierzy w *row1* i *j*. Jest to tymczasowe miejsce przechowywania wartości, które zostaną zamienione. Wartość elementu w *row2* i *j* jest przypisywana do elementu w *row1* i *j*, co skutkuje zamianą wartości między wierszami *row1* i *row2* dla kolumny *j*. Wartość tymczasowa *temp* jest przypisywana do elementu w *row2* i *j*, co kończy operację zamiany miejscami.

```
1. void swapRows(double **matrix, int row1, int row2, int n) {
2.     for (int j = 0; j < n + 1; ++j) {
3.         double temp = matrix[row1][j];
4.         matrix[row1][j] = matrix[row2][j];
5.         matrix[row2][j] = temp;
6.     }
7. }
```

Fragment kodu 3: Funkcja *printMatrix*

3.4. funkcja *solveEquations* (Fragment kodu 4)

Pętla *for* iteruje przez wszystkie kolumny macierzy. Każda iteracja odpowiada kolejnej kolumnie, w której wykonywana będzie eliminacja Gaussa. Dla każdej iteracji znajdowany jest indeks elementu głównego w kolumnie za pomocą funkcji *findPivot*, która zwraca indeks wiersza, w którym znajduje się element główny.

Sprawdzany jest warunek, czy wartość bezwzględna elementu głównego jest mniejsza niż wartość *EPSILON*. Jeśli tak, oznacza to, że macierz jest osobliwa, co prowadzi do wyświetlenia komunikatu o błędzie i zakończenia działania programu.

Następuje zamiana miejscami wierszy tak, aby element główny znalazł się na przekątnej. W dwóch zagnieżdżonych pętlach *for*, iterowane są po wierszach pod przekątną (wiersze od $k + 1$ do $n - 1$) i wykonujemy operacje elementarne, aby wyzerować elementy pod przekątną w danej kolumnie. Po zakończeniu eliminacji w przód, zostaje otrzymana macierz trójkątna górna, co umożliwia rozwiązanie układu równań metodą wstecznej substytucji.

Tworzona jest dynamiczna tablica *solution*, która będzie przechowywać rozwiązania układu równań. W drugiej pętli *for* rozwiązywany jest układ równań trójkątnych, zaczynając od ostatniego równania i przechodząc w górę, aby wyznaczyć wartości zmiennych nieznanych. Każde rozwiązanie jest przechowywane w tablicy *solution*. Zwracany zostaje wskaźnik do tablicy zawierającej rozwiązania układu równań.

```

1. double* solveEquations(double **matrix, int n) {
2.     // eliminacja w przód
3.     for (int k = 0; k < n; ++k) {
4.         int pivot_row = findPivot(matrix, k, n);
5.         if (abs(matrix[pivot_row][k]) < EPSILON) {
6.             cerr << "singular matrix. " << endl;
7.             exit(1);
8.         }
9.         swapRows(matrix, k, pivot_row, n);
10.
11.         for (int i = k + 1; i < n; ++i) {
12.             double factor = matrix[i][k] / matrix[k][k];
13.             for (int j = k; j < n + 1; ++j) {
14.                 matrix[i][j] -= factor * matrix[k][j];
15.             }
16.         }
17.     }
18.
19.     // macierz trójkątna
20.     double* solution = new double[n];
21.     for (int i = n - 1; i >= 0; --i) {
22.         solution[i] = matrix[i][n];
23.         for (int j = i + 1; j < n; ++j) {
24.             solution[i] -= matrix[i][j] * solution[j];
25.         }
26.         solution[i] /= matrix[i][i];
27.     }
28.
29.     return solution;
30. }

```

Fragment kodu 4: Funkcja *solveEquations*

3.5. funkcja *main* (Fragment kodu 5)

Definiowana jest macierz *matrix* typu *double*, która reprezentuje układ równań liniowych. Tworzona jest dynamiczna tablica *dynamic_matrix*, która jest dwuwymiarową tablicą wskaźników do typu *double*. Jest to potrzebne, aby przekazać macierz do funkcji *solveEquations*.

W pętli *for* konwertowana jest macierz *matrix* na dynamiczną tablicę *dynamic_matrix*, aby móc przekazać ją jako argument do funkcji *solveEquations*. Wywoływana jest funkcja *solveEquations*. Wyświetlany zostaje wynik na standardowy wyjściu. Na koniec zwalniana jest zaalokowana pamięć dla tablicy *dynamic_matrix* oraz dla tablicy *solution*, aby uniknąć wycieków pamięci.

```

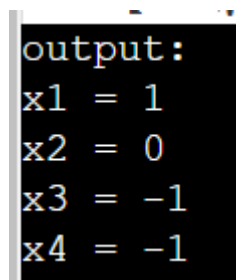
1. int main() {
2.     int n = 4;
3.     double matrix[4][5] = {
4.         {-1, -5, 0.5, 5.5, 9.5 },
5.         {-2, 0, -1, 3, -3},
6.         {-1.5, -1.25, 0.5, -0.75, -1.5},
7.         {0, -1.25, 0.5, 5.5, 9.5}
8.     };
9.
10.    // konwersja macierzy do tablicy
11.    double **dynamic_matrix = new double*[n];
12.    for (int i = 0; i < n; ++i) {
13.        dynamic_matrix[i] = new double[n + 1];
14.        for (int j = 0; j < n + 1; ++j) {
15.            dynamic_matrix[i][j] = matrix[i][j];
16.        }
17.    }
18.
19.    double* solution = solveEquations(dynamic_matrix, n);
20.
21.    cout << "output: " << endl;
22.    for (int i = 0; i < n; ++i) {
23.        cout << "x" << i + 1 << " = " << solution[i] << endl;
24.    }
25.
26.    for (int i = 0; i < n; ++i) {
27.        delete[] dynamic_matrix[i];
28.    }
29.    delete[] dynamic_matrix;
30.    delete[] solution;
31.
32.    return 0;
33. }

```

Fragment kodu 5: Funkcja *main*

Podczas zajęć testowano kod na dwóch zestawach danych celem sprawdzenia poprawności działania algorytmu. Pierwsza testowana macierz oraz spodziewany wynik zawiera wzór (1), a na Rys.1 załączono zrzut ekranu konsoli.

$$\begin{bmatrix} -2 & -4 & 5 & -2 & -5 \\ 6,5 & 2 & 1,25 & 3,5 & 1,75 \\ 1,75 & 7,25 & -8,75 & 5,5 & 5 \\ 3,25 & -2,75 & -3,75 & 1 & 6 \end{bmatrix} \Rightarrow x = \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \end{bmatrix} \quad (1)$$



```

output:
x1 = 1
x2 = 0
x3 = -1
x4 = -1

```

Rys.1 Zrzut ekranu konsoli dla pierwszego zestawu danych testowych

Druga testowana macierz oraz spodziewany wynik zawiera wzór (2), a Rys.2. przedstawia wydruk z konsoli.

$$\begin{bmatrix} -1 & -5 & 0,5 & 5,5 & 9,5 \\ -2 & 0 & -1 & 3 & -3 \\ 1,5 & -1,25 & 0,5 & -0,75 & -1,5 \\ 0 & -1,25 & 0,5 & 5,5 & 9,5 \end{bmatrix} \Rightarrow x = \begin{bmatrix} 1,84 \\ -0,49 \\ 3,27 \\ 1,32 \end{bmatrix} \quad (2)$$

```
output:
x1 = 1.84091
x2 = -0.490909
x3 = 3.27273
x4 = 1.31818
```

Rys.2 Zrzut ekranu konsoli dla drugiego zestawu danych testowych

Oba wyniki pokrywają się ze spodziewanymi, więc można stwierdzić, że algorytm został zaimplementowany prawidłowo.

4. Testy jednostkowe

4.1. Test dla bardzo dużej ilości współczynników

Badano układ równań o dwudziestu niewiadomych. Wydruk z konsoli załączono na Rys. 3. Spodziewane wyniki zawarte są w pliku testowym z zajęć.

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

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

4.2. Test dla układu równań sprzecznych

Badany układ równań:

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

Wydruk z konsoli załączono na Rys. 4. Test ma na celu sprawdzenie, czy algorytm zwraca informację o niemożności rozwiązania dla macierzy układu równań sprzecznej oraz upewnienie się, że algorytm zwraca oczekiwany komunikat w przypadku sprzecznej macierzy (tj. wiadomość *singular matrix*).



Rys. 4 Wydruk z konsoli dla zestawu danych dla testu dla równań sprzecznych

4.3. Test dla układu równań nieoznaczonych

Badany układ równań:

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

Wydruk z konsoli załączono na Rys. 5. Test ma na celu sprawdzenie, czy algorytm zwraca informację o niemożności rozwiązania dla macierzy układu równań sprzecznej oraz upewnienie się, że algorytm zwraca oczekiwany komunikat w przypadku nieoznaczonej macierzy (tj. wiadomość *singular matrix*).



Rys. 5 Wydruk z konsoli dla zestawu danych dla testu dla równań nieoznaczonych

5. Opracowanie wyników

5.1. 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.

5.2. Test dla układu równań sprzecznych

Układ równań zawiera sprzeczność, gdyż równania są względem siebie sprzeczne. Oczekuje się, że funkcja zwróci informację o występowaniu macierzy nieosobliwej. Taki też komunikat został wyświetlony na konsoli, co spełnia oczekiwania wobec wykonanego testu.

5.3. Test dla równań nieoznaczonych

Układ równań wprowadzony do programu to układ równa nieoznaczonych – drugie równanie jest wielokrotnością pierwszego. Również w tym przypadku oczekuje się od funkcji zwrócenia informacji o nieosobliwości macierzy, ponieważ nieosobliwa macierz, mająca wyznacznik zerowy, pozwala na stwierdzenie, że układ równań albo jest nieoznaczony albo sprzeczny. Taki też komunikat został wyświetlony na konsoli, co spełnia oczekiwania wobec wykonanego testu.

6. Wnioski

Równania liniowe są fundamentalnym elementem matematyki, których rozwiązania definiują przestrzenie liniowe. Metoda eliminacji Gaussa, oparta na operacjach elementarnych na macierzach, jest jedną z najefektywniejszych metod rozwiązywania układów równań liniowych. Metoda elementu głównego w metodzie eliminacji Gaussa pomaga w poprawie jej efektywności i stabilności numerycznej poprzez wybór odpowiedniego elementu głównego w danej kolumnie.

Wyniki testów potwierdziły, że algorytm poprawnie radzi sobie z różnymi przypadkami układów równań liniowych. Pomimo nieco mniejszej dokładności w przypadku bardzo dużych ilości współczynników, algorytm nadal działał poprawnie. Testy dla układów równań

sprzecznych i nieoznaczonych potwierdziły, że algorytm zwraca odpowiednie komunikaty w przypadku wystąpienia macierzy nieosobliwej.

Metoda eliminacji Gaussa z wyborem elementu głównego jest skutecznym narzędziem do rozwiązywania układów równań liniowych, a przeprowadzone testy potwierdziły jej poprawność i działanie. W przypadku dużych układów równań lub szczególnych przypadków, algorytm może wymagać optymalizacji lub dostosowania do konkretnych potrzeb aplikacji.

7. Źródła

Wykłady z Metod Numerycznych autorstwa dr hab. Danuty Szeligi

Prezentacja Rozwiązywanie układów równań liniowych - metoda eliminacji Gaussa (metody wyboru elementu głównego) autorstwa dr. Hab. Inż. Marcina Hojnego.

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