

# Eliminacja Gaussa

*Metody numeryczne*

## 1. Wprowadzenie teoretyczne

Metoda eliminacji Gaussa jest jednym z podstawowych algorytmów stosowanych do rozwiązywania układów równań liniowych. Jest to metoda bezpośrednia, co oznacza, że po skończonej liczbie kroków dostajemy dokładne rozwiązanie (w granicach błędu numerycznego).

Głównym celem metody eliminacji Gaussa jest przekształcenie pierwotnego układu równań do postaci, w której łatwo można znaleźć rozwiązanie. Dokonuje się tego poprzez eliminację niewiadomych - stąd nazwa metody.

$$Ax = b \quad (1)$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

macierz n x n

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

szukane rozwiązanie

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

wektor danych

W metodzie eliminacji Gaussa wyróżniamy dwa etapy:

- postępowanie proste (eliminacja w przód),
- postępowanie odwrotne (wsteczne).

### 1.1 postępowanie proste

**Krok I:** odejmowany od i-tego wiersza układu jest wiersz pierwszy pomnożony przez:

$$m_{i1} = \frac{a_{i1}^1}{a_{11}^1}, \quad (2)$$

gdzie:

a – poszczególne elementy macierzy,

i – 2, 3, ..., n

**Krok II:** odejmowany od i-tego wiersza układu jest drugi wiersz pomnożony przez  $m_{i2}$ , a kolejne mnożniki wyznaczone ze wzoru:

$$m_{ij} = \frac{a_{ij}^k}{a_{jj}^k}, \quad (3)$$

gdzie:

$$k = 1, \dots, n$$

$$i = 2, \dots, n$$

$$j = 1, \dots, n - 1$$

Po wykonaniu  $n-1$  eliminacji uzyskany zostaje trójkątny układ równań.

## 1.2 Postępowanie odwrotne

Rozpoczynamy od ostatniego równania w układzie, które zawiera tylko jedną niewiadomą. Rozwiązujemy to równanie, co daje nam wartość tej niewiadomej. Następnie przechodzimy do przedostatniego równania. Znając już wartość niewiadomej z ostatniego równania, możemy ją podstawić do przedostatniego równania i rozwiązać je, co daje nam wartość kolejnej niewiadomej. Powtarzamy ten proces, przechodząc do góry przez układ równań, aż znajdziemy wartości wszystkich niewiadomych.

- Wyznaczenie ostatniej niewiadomej wartości z ostatniego równania:

$$x_n = \frac{b_n^n}{a_{nn}^n}, \quad (4)$$

- Wyznaczenie pozostałych wartości  $x_i$ :

$$x_i = \frac{b_i^n - \sum_{k=i+1}^n a_{i,k}^n x_k}{a_{ii}^n}, \quad (5)$$

gdzie:

$$i = n - 1, \dots, 1$$

## 2. Opis implementacji numerycznej

Zmienna **tab** to dwuwymiarowa tablica, która reprezentuje układ równań. Każdy wiersz tablicy reprezentuje jedno równanie, a każda kolumna reprezentuje jedną zmienną. Ostatnia kolumna reprezentuje wektor prawych stron równań.

Zmienna **n** to liczba wierszy jak i kolumn badanej macierzy.

Pierwsza pętla **for** to główna pętla eliminacji Gaussa. Dla każdego kroku **k**, pętla eliminuje zmienną **k** z wszystkich równań poniżej **k**. Wewnętrzna pętla przechodzi przez wszystkie równania poniżej równania **k**. Najbardziej wewnętrzna pętla aktualizuje wszystkie elementy w równaniu **i** poprzez odejmowanie od nich odpowiednio przeskalowanego równania **k**. Skalowanie jest realizowane przez wyrażenie  $\text{tab}[i][k] / \text{tab}[k][k]$ , które jest współczynnikiem, przez który mnożone jest równanie **k** przed odjęciem go od równania **i**.

```
for (int k = 0; k < n; k++) {  
    for (int i = k + 1; i < n; i++) {  
        for (int j = n; j >= 0; j--) {  
            tab[i][j] -= tab[k][j] * (tab[i][k] / tab[k][k]);  
        }  
    }  
}
```

Fragment kodu 1: sprowadzanie macierzy do postaci trójkątnej górnej

Po wykonaniu tego fragmentu kodu, układ równań jest w postaci trójkątnej górnej, co oznacza, że można go łatwo rozwiązać za pomocą podstawienia wstecznego.

```
for (int i = n - 1; i >= 0; i--) {  
    x[i] = tab[i][n];  
    for (int j = i + 1; j < n; j++) {  
        x[i] -= tab[i][j] * x[j];  
    }  
    x[i] /= tab[i][i];  
    cout << "x" << i + 1 << " = " << x[i] << endl;  
}
```

Fragment kodu 2: postępowanie wsteczne

Podstawienie wsteczne zaczyna się od ostatniego równania w układzie i przechodzi w górę, rozwiązując każde równanie po kolei.

Pętla **for (int i = n - 1; i >= 0; i--)** przechodzi przez każde równanie w układzie, zaczynając od ostatniego i idąc w górę. Po wykonaniu tego fragmentu kodu, wszystkie **x[i]** są obliczone, co daje rozwiązanie układu równań.

### 3. Testy kodu numerycznego

#### Test I

Macierz  $3 \times 3 = \{( 10, -7, 0 ); ( -3, 2, 6 ); ( 5, -1, 5 )\}$

wektor danych =  $\{( 6 ); ( 4 ); ( 3 )\}$

Wynik programu Matrix calculator:

$$x_1 = -0.7096774$$

$$x_2 = -1.8709677$$

$$x_3 = 0.9354839$$

Wynik zaimplementowanego programu:

```
x1 = -0.709677
x2 = -1.87097
x3 = 0.935484
```

*Fragment konsoli: wynik programu dla testu I*

#### Test II

Macierz  $4 \times 4 = \{( 4, -2, 4, -2 ); ( 3, 1, 4, 2 ); ( 2, 4, 2, 1 ); ( 2, -2, 4, 2 )\}$

wektor danych =  $\{( 8 ); ( 7 ); ( 10 ); ( 2 )\}$

Wynik programu Matrix calculator:

$$x_1 = -1$$

$$x_2 = 2$$

$$x_3 = 3$$

$$x_4 = -2$$

Wynik zaimplementowanego programu:

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

*Fragment konsoli: wynik programu dla testu II*

### Test III

Macierz 20x20 = {

[	5.10433490e+00,	1.75969137e+00,	3.34269222e+00,	1.69270851e+00,
	2.10571531e+00,	2.88859060e+00,	7.58084997e+00,	7.89813209e+00,
	3.22961138e+00,	2.74727035e+00,	1.64047227e+00,	8.50491285e-01,
	9.92490806e+00,	8.17924897e+00,	5.55289459e+00,	1.42494093e+00,
	2.76959304e+00,	1.56532834e-01,	6.66525688e+00,	1.67808636e+00
	];			
[	2.51787444e+00,	8.85065540e+00,	6.96539888e+00,	7.74759713e+00,
	1.69969840e-01,	9.25868652e+00,	8.47749420e+00,	2.61761845e-01,
	3.94321306e+00,	5.89072001e+00,	8.29337823e+00,	2.20212145e+00,
	7.22467370e-03,	4.30461845e+00,	3.28503270e+00,	8.94091309e+00,
	4.78059184e+00,	8.36653930e+00,	1.44848560e+00,	3.45006255e+00
	];			
[	3.74780917e-02,	5.07160375e+00,	3.63896851e+00,	4.77241115e+00,
	4.97797037e+00,	5.43678447e+00,	6.30858043e+00,	5.55483663e+00,
	3.29617157e+00,	2.44656662e+00,	8.70682154e+00,	7.59035931e+00,
	3.75598973e+00,	6.20465356e+00,	9.60523352e-01,	4.15748008e-01,
	5.67764815e-01,	2.29292958e+00,	4.43041021e-01,	8.20206925e+00
	];			
[	1.00599731e+00,	6.22345945e+00,	7.93202559e+00,	7.97800763e+00,
	3.94323078e-01,	5.71393977e+00,	5.80092904e+00,	1.73848000e+00,
	7.03076629e+00,	2.15698396e+00,	6.28935358e+00,	6.76508355e+00,
	5.28644884e+00,	7.80555007e+00,	5.22660948e+00,	5.42241227e+00,
	6.22280648e+00,	2.29594435e+00,	3.66192245e+00,	8.94837190e+00
	];			
[	5.04506405e+00,	5.80367597e+00,	8.92849022e+00,	7.47350443e+00,
	3.81950675e+00,	9.55397235e+00,	7.86375898e+00,	8.97189930e+00,
	2.04109623e+00,	1.04662443e+00,	7.18326849e+00,	4.74199460e+00,
	7.47098892e+00,	6.52205664e+00,	5.90895425e+00,	6.80910265e+00,
	6.89623538e+00,	7.82277807e+00,	3.03142408e+00,	3.93553100e-01
	];			
[	4.45523365e+00,	6.01620888e+00,	2.88395450e+00,	6.96428198e+00,
	7.73912909e+00,	2.85965153e+00,	3.60405756e+00,	6.10323893e+00,
	6.55659667e+00,	6.79311561e+00,	2.92094880e-01,	5.40647851e+00,
	6.55074767e-01,	3.23988216e+00,	3.70867922e+00,	8.76834262e+00,
	6.61527717e+00,	5.05942327e+00,	3.81894900e+00,	6.78664542e+00
	];			
[	9.27976892e+00,	3.91287173e-01,	9.40598010e+00,	6.93501657e+00,
	1.25907761e+00,	6.07041081e+00,	3.73698707e+00,	3.32356886e+00,
	8.02306992e+00,	9.09421238e+00,	9.05273293e+00,	1.01063740e+00,
	3.34950663e+00,	7.67360842e+00,	1.21020805e+00,	6.41917805e+00,
	5.82974058e-01,	8.64370265e+00,	7.49893468e+00,	7.34494266e+00
	];			

[ 2.36519035e+00, 2.69722816e+00, 4.01132931e+00, 6.04524995e-01,  
2.78986742e+00, 5.88113190e-01, 1.70759426e+00, 1.88147281e+00,  
4.97805120e+00, 7.15255989e+00, 6.75748205e+00, 8.66745105e+00,  
9.63209183e+00, 8.22806126e+00, 9.77046017e+00, 9.87722962e+00,  
2.27708536e+00, 3.42119156e+00, 8.35105691e+00, 8.55444398e+00 ];

[ 6.32598011e+00, 5.56797387e+00, 3.27672707e+00, 4.49173411e+00,  
3.88967740e+00, 3.79663182e-01, 5.55830142e+00, 6.05870363e+00,  
4.26713016e+00, 6.96345639e+00, 7.35682525e+00, 1.57858773e+00,  
1.64768488e+00, 5.11799385e+00, 4.69080581e+00, 9.54082419e+00,  
1.31438264e+00, 7.26865764e+00, 7.52952520e+00, 3.16112249e+00 ];

[ 5.96547032e+00, 5.78959748e+00, 3.53345854e+00, 8.37157841e-01,  
4.77173930e+00, 9.19910453e+00, 1.06503884e+00, 2.26380068e+00,  
6.26659347e+00, 6.87270384e+00, 4.48992683e+00, 4.45212059e+00,  
5.91484727e+00, 2.87956094e+00, 7.56822762e+00, 5.35478736e+00,  
5.91397316e+00, 5.94583961e-01, 6.59043999e+00, 4.43922465e+00 ];

[ 3.63303691e+00, 9.11389066e+00, 4.23603836e+00, 8.73014942e-01,  
3.65089960e+00, 4.18420322e+00, 3.35587359e+00, 8.28711266e+00,  
7.58771886e+00, 9.94696512e+00, 8.40620542e+00, 1.25716918e+00,  
4.40411940e+00, 5.83188472e+00, 3.93880154e-01, 4.96834824e+00,  
4.52895273e+00, 5.21410603e+00, 8.60665147e+00, 4.54284683e+00 ];

[ 2.91323767e+00, 1.44008056e+00, 4.23208327e+00, 3.00957861e-01,  
1.08987750e+00, 5.48323873e+00, 1.96486696e+00, 1.01730182e+00,  
6.35255648e+00, 2.96065467e+00, 9.72233337e+00, 9.70441754e+00,  
3.07306254e+00, 9.53766596e+00, 5.87427411e+00, 5.46489467e+00,  
1.04610751e+00, 1.82749805e+00, 2.88475357e+00, 7.11506000e+00 ];

[ 9.96272469e+00, 6.94720317e+00, 5.76208837e+00, 5.14148927e+00,  
2.42928342e-01, 4.78850571e-01, 7.60312581e+00, 4.07857444e+00,  
5.42740508e+00, 9.59580480e+00, 3.29723088e+00, 3.32570968e+00,  
2.07962047e+00, 4.36821045e+00, 1.38199237e+00, 1.21833966e+00,  
3.84247551e+00, 6.13067429e+00, 4.12853462e+00, 2.31645282e+00 ];

[ 1.81356132e+00, 6.85345417e-01, 9.34653337e+00, 8.10179528e+00,  
8.14915767e+00, 7.30401328e+00, 4.24759787e+00, 6.43647092e-02,  
2.89913710e+00, 4.44501423e+00, 4.07712828e+00, 3.01926374e+00,  
3.11920370e+00, 5.86888955e+00, 2.47016437e+00, 9.53229639e+00,  
3.44072622e+00, 2.41160594e+00, 4.47801568e+00, 1.02601112e+00 ];

[ 2.36171217e+00, 3.80714231e+00, 9.04112234e+00, 9.93247987e-01,  
8.01372338e+00, 5.36806789e+00, 6.64952989e+00, 3.21883742e+00,  
5.76103675e+00, 2.28765379e+00, 8.76735948e-01, 2.63635605e+00,  
2.21634567e+00, 8.30723171e+00, 2.47507677e+00, 3.78674492e+00,

```

      8.32553680e+00, 6.59268067e+00, 7.52608251e+00, 8.38006564e+00 ];
[
  4.46482512e+00, 1.13144056e+00, 5.85889396e+00, 8.64117482e+00,
  6.10507694e+00, 4.26822267e+00, 6.75249314e+00, 9.39769297e+00,
  8.62397307e+00, 7.48413287e-02, 9.05092405e+00, 1.34185519e+00,
  6.93917786e+00, 3.36307954e+00, 1.46575447e+00, 8.38722476e+00,
  3.96298772e+00, 7.21608290e-01, 3.99208685e+00, 5.62599782e+00 ];
[
  8.81342133e+00, 4.45953067e+00, 2.31022845e+00, 4.58171052e+00,
  2.41465605e+00, 4.49131090e+00, 8.04915367e+00, 8.59212755e-01,
  9.99860407e+00, 3.93193991e-02, 3.36702510e+00, 6.13047314e+00,
  2.92667166e+00, 2.16410499e+00, 8.77746429e+00, 2.77522324e+00,
  8.81347467e+00, 4.08634046e-01, 7.88836051e+00, 3.50529396e+00 ];
[
  4.54377058e+00, 8.57130416e+00, 9.57173070e-01, 1.81131982e-01,
  2.61306214e+00, 1.08561613e+00, 9.88020918e+00, 9.61983697e+00,
  7.98149672e+00, 1.02130046e+00, 8.11118393e+00, 8.99452642e+00,
  3.42575761e+00, 8.93780489e+00, 7.82694356e+00, 6.70572668e+00,
  3.81870093e+00, 9.97967854e+00, 5.14824629e+00, 6.68100978e+00 ];
[
  4.09244084e+00, 6.47744012e+00, 4.31626204e-01, 2.01357125e+00,
  9.07116695e+00, 5.00847293e+00, 9.34224113e+00, 7.91980139e+00,
  2.45861412e+00, 8.37514505e+00, 8.19399706e+00, 1.19978850e+00,
  3.10794710e+00, 4.02921482e+00, 5.37454753e+00, 2.27273053e+00,
  1.46724630e+00, 6.26394725e+00, 7.51526370e+00, 6.64156544e+00 ];
[
  9.76335803e+00, 5.02048297e+00, 8.16110497e+00, 3.17586938e+00,
  6.24951419e+00, 1.97398801e+00, 5.05391652e+00, 4.50493279e+00,
  9.73979315e+00, 8.58993726e+00, 6.24727732e+00, 6.89333240e+00,
  9.52123357e+00, 2.02189392e-01, 8.13574765e+00, 9.03349868e-01,
  9.87113674e+00, 6.77803126e+00, 9.99926084e+00, 6.43465967e+00 ]}

```

wektor danych = {

```

    [0.37420695];    [3.21757695];    [7.84957695];    [6.72922658];
    [2.00267396];    [5.23242394];    [4.27969739];    [4.81143087];
    [2.13077584];    [1.79740411];    [2.78116646];    [9.07648293];
    [7.60278802];    [0.44804225];    [3.03032564];    [1.08351171];
    [6.82213138];    [6.96424131];    [6.5750299];     [2.76621875]
}

```

Wynik programu Matrix calculator:

$x_1 = 12.19254145$

$x_2 = 7.24097684$



$x_3 = -23.52001299$   
 $x_4 = 9.56210165$   
 $x_5 = 16.50620674$   
 $x_6 = -8.72811736$   
 $x_7 = -13.28704939$   
 $x_8 = -21.11798518$   
 $x_9 = 0.9464735$   
 $x_{10} = -6.07601471$   
 $x_{11} = 11.47032627$   
 $x_{12} = -15.63654823$   
 $x_{13} = 15.9192917$   
 $x_{14} = 23.91029996$   
 $x_{15} = -4.76981873$   
 $x_{16} = -5.91822258$   
 $x_{17} = 15.86916816$   
 $x_{18} = 10.3475958$   
 $x_{19} = -15.95905739$   
 $x_{20} = -2.3372826486$

Wynik zaimplementowanego programu:

```
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
```

*Fragment konsoli: wynik programu dla testu III*

## Test IV

Macierz  $6 \times 6 = \{(-4, 2, -1, 3, 0, -4); (1, 0, 5, -1, 0, 3); (-3, -2, -3, -2, 4, 5); (3, 3, 3, 1, -2, 0); (-2, 1, 0, -4, -4, -2); (4, -3, 4, -4, 4, -2)\}$

wektor danych =  $\{(3); (0); (4); (5); (4); (-5)\}$

Wynik programu Matrix calculator:

$$x_1 = -0,2583113$$

$$x_2 = 3,5002639$$

$$x_3 = -0,4575198$$

$$x_4 = -1,2580475$$

$$x_5 = 1,0476253$$

$$x_6 = 0,4292876$$

Wynik zaimplementowanego programu:

```
x1 = -0.258311
x2 = 3.50026
x3 = -0.45752
x4 = -1.25805
x5 = 1.04763
x6 = 0.429288
```

*Fragment konsoli: wynik programu dla testu IV*

## 4. Analiza wyników

W celu głębszej analizy powstała tabela, w której porównane zostały wyniki zaimplementowanych metod z wynikami programu „Matrix calculator” na podstawie powyższych testów.

*Tabela 1: wyniki Testu I*

wartości	Matrix Calculator	Program
$x_1$	-0.7096774	-0.709677
$x_2$	-1.8709677	-1.87097
$x_3$	0.9354839	0.935484

Tabela 2: wyniki Testu II

wartości	Matrix Calculator	Program
$x_1$	-1	-1
$x_2$	2	2
$x_3$	3	3
$x_4$	-2	-2

Tabela 3: wyniki Testu III

wartości	Matrix Calculator	Program
$x_1$	12.19254145	12.1925
$x_2$	7.24097684	7.24097
$x_3$	-23.52001299	-23.52
$x_4$	9.56210165	9.5621
$x_5$	16.50620674	16.5062
$x_6$	-8.72811736	-8.72811
$x_7$	-13.28704939	-13.287
$x_8$	-21.11798518	-21.118
$x_9$	0.9464735	0.946473
$x_{10}$	-6.07601471	-6.07601
$x_{11}$	11.47032627	11.4703
$x_{12}$	-15.63654823	-15.6365
$x_{13}$	15.9192917	15.9193
$x_{14}$	23.91029996	23.9103
$x_{15}$	-4.76981873	-4.76982
$x_{16}$	-5.91822258	-5.91822
$x_{17}$	15.86916816	15.8692
$x_{18}$	10.3475958	10.3476

$x_{19}$	-15.95905739	-15.9591
$x_{20}$	-2.3372826486	-2.33728

Tabela 4: wyniki Testu IV

wartości	Matrix Calculator	Program
$x_1$	-0,2583113	-0,258311
$x_2$	3,5002639	3,50026
$x_3$	-0,4575198	-0,45752
$x_4$	-1,2580475	-1,25805
$x_5$	1,0476253	1,04763
$x_6$	0,4292876	0,429288

Dzięki analizie wyników można stwierdzić, że przedstawiona metoda została zaimplementowana poprawnie. Wyniki uzyskane za pomocą programu są bardzo bliskie z wynikami uzyskanymi za pomocą kalkulatora macierzy. Jednakże, w przypadku testu III zauważono istotne rozbieżności, które mogą wynikać z dużego rozmiaru macierzy oraz precyzji jej elementów - liczby zmiennoprzecinkowe o dużej liczbie miejsc po przecinku. W trakcie obliczeń numerycznych, takich jak eliminacja Gaussa, mogą wystąpić błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych. Te błędy mogą się kumulować w trakcie obliczeń, prowadząc do zaokrąglenia wyników.

## 5. Wnioski

W tym sprawozdaniu przedstawiono oraz omówiono metodę eliminacji Gaussa służącą do rozwiązywania układów równań liniowych. Pokazano oraz wytłumaczono jej prostą implementację w języku c++. Z analizy przeprowadzonych testów wynika że metoda eliminacji Gaussa jest skutecznym narzędziem do rozwiązywania układów równań liniowych. Testy wykazały, że zastosowana implementacja jest efektywna pod względem czasu obliczeń, szczególnie dla dużych układów równań, natomiast mogą wystąpić błędy zaokrąglenia wynikające z ograniczonej precyzji liczb zmiennoprzecinkowych.

## 6. Źródła

- prezentacja z zajęć „lab 8.pdf”
- matrixcalc.org