Eliminare gaussiană cu pivotare totală și scalare. Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal.

Colaboratori: Andrei STAN, Florin Pop, Mihaela Andreea-Vasile

February 21, 2025

Cuprins

1 Obiective laborator

În urma parcurgerii acestui laborator, studentul va fi capabil să:

- Transforme o matrice nesingulară într-o matrice superior sau inferior triunghiulară folosind una din metodele prezentate de eliminare gaussiană;
- Implementeze algoritmii de eliminare gaussiană prezentați;
- Implementeze algoritmul Thomas.

2 Noțiuni teoretice

Eliminarea Gaussiană sau *reducerea coloanelor* este un algoritm fundamental în algebra liniară. Acesta poate fi folosit pentru rezolvarea sistemelor liniare, găsirea rang-ului unei matrice, inversarea unei matrice, sau calcularea determinantului.

Algoritmul se bazează pe efectuarea unor operații elementarea pentru a permuta, scala și combina liniile unei matrici. Gândindu-ne la un sistem de ecuații liniare, acesta nu se schimbă dacă facem aceste operații. Dacă mai adăugăm și permutarea coloanelor, atunci soluția sistemului se schimbă doar prin aceste permutări, ușor de inversat. Obiecivul este să aducem matricea la *forma eșalon* sau *forma eșalon redusă*.

Așadar, opeațiile elementare cu liniile sunt:

1. permutarea liniilor;
2. înmulțirea unei linii cu un scalar nenul;
3. adunarea unei linii scalate la altă linie.

2.1 Forma eșalon a unei matrice

Pentru fiecare linie dintr-o matrice, numim prima valoare diferită de zero *coeficient principal* sau *pivot*. Dacă doi pivoți se găsesc pe aceeași coloană, putem mereu să facem pe unul dintre ei 0. Ne folosim apoi de permutari, pentru a aranja liniile în așa fel încât pivotul unei linii să fie la dreapta pivotului liniei anterioare.

De exemplu, matricea de mai jos este în forma eșalon. Observați cum toate liniile pline de 0 sunt jos.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

În plus, putem spune despre o matrice că este în *forma eșalon redusă* dacă toți pivoții sunt 1 (lucru care se poate obține prin scalare) și toate celelalte elemente de pe coloana unui pivot sunt 0 (lucru care se poate obține prin adunarea unei linii scalate la altă linie).

2.2 Matrici elementare

O matrice elementară este o matrice care se obține din matricea identitate prin efectuarea unei operații elementare cu liniile.

Matricea de permutare. Interschimbă liniile i și j.

$$P_{ij} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

Construire:
$P = eye(n)$
$P(i,i) = P(j,j) = 0$
$P(i,j) = P(j,i) = 1$

Proprietăți:

- $P = P^{-1} = P^T \implies P$ ortogonală
- $\det P = -1 \implies \det(PA) = -\det A$

Matricea de scalare. Înmulțește linia i cu un scalar α.

$$S_i(\alpha) = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & \alpha & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

Construire:
$S = eye(n)$
$S(i,i) = \alpha$

Proprietăți:

- $D_i(m)^{-1} = D_i(\frac{1}{m})$
- $\det S = \alpha \implies \det(SA) = \alpha \det A$

Matricea de adunare. Adună linia i la linia j înmulțită cu un scalar α.

$$E_{ij}(\alpha) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & \alpha & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

Construire:
$E = eye(n)$
$E(i,j) = \alpha$

Proprietăți:

- $E_{ij}(\alpha)^{-1} = E_{ij}(-\alpha)$
- $\det E = 1 \implies \det(EA) = \det A$

2.2.1 Descompunerea LU

Prin aplicare unor astfel de matrice și ajungând la forma eșalon, am ajuns practic la o matrice superior triunghiulară.

$$\begin{aligned} T_n T_{n-1} \dots T_1 A &= U \\ A &= T_1^{-1} T_2^{-1} \dots T_n^{-1} U \\ A &= LU, \quad L = T_1^{-1} T_2^{-1} \dots T_n^{-1} \end{aligned}$$

2.2.2 Calcularea determinantului

După cum am spus la primul laborator, determinantul se calculează prin formula lui Leibniz în $O(n!)$ operații. Eliminarea Gaussiană are complexitate $O(n^3)$.

Pentru o matrice triunghiulară U avem că $\det U = \prod_{i=1}^{n} u_{ii}$.

$$\det A = \frac{\det U}{\prod_{i=1}^{n} \det T_i}$$

2.3 Eliminarea gaussiană - G

Exemplu.

$$\begin{bmatrix} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 11 & 8 & 35 \end{bmatrix} \to \begin{bmatrix} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 5 & 8 \end{bmatrix} \to \begin{bmatrix} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 3 & 0 \end{bmatrix}$$

Agoritmul în MATLAB poate fi gândit astfel:

1. Folosesc un p ca să ma plimb pe coloane, până când p este egal cu numărul de linii sau de coloane.
2. Pivotul meu va fi a_{pp}.
3. Iau restul de elemente de pe coloană folosind un index i și calculez $\mu_{ip} = \frac{a_{ip}}{a_{pp}}$. Acești coeficienți îi pun în matricea S, $S(i,p) = -\mu_{ip}$.
4. Calculez produsul SA și continui cu următoarea coloană.

```
A = [1 3 1 9; 1 1 -1 1; 3 11 8 35];

[m, n] = size(A);
maxP = min(m, n);

for p = 1 : maxP
  T = eye(m);
  mu = A(p + 1 : m, p) / A(p, p);
  T(p + 1 : m, p) = -mu;

  A = T * A;
end

disp(A);
```

De notat că nu este nevoie să aduce matricea la forma eșalon redusă pentru a rezolva un sistem, trebuie doar să fie triunghiulară. În codul de mai sus, A este matricea extinsă a sistemului. Ba mai mult, am putea afecta stabilitatea numerică din cauza efectuării mai multor operații.

Alt lucru de observat este că dacă pivotul selectat este 0, atunci programul "crapă" (womp-womp). Pentru a rezolva acest lucru introducem conceptul de *pivotare*.

2.4 Pivotare

Pivotarea constă în folosirea permutărilor și a scalării pentru a alege cel mai bun pivot. Este clar că eliminearea Gauss poate fi aplicată pe orice matrice, deci în continuare discuția se va baza pe analiză numerică.

2.4.1 De ce este pivotarea importantă?

Fie un calculator care folosește aritmetică în virgulă mobilă, cu 3 zecimale, cu trunchiere. Fie următorul sistem:

$$\begin{bmatrix} 0.913 & 0.659 \\ 0.780 & 0.563 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.254 \\ 0.217 \end{bmatrix}$$

$$\mu = \frac{0.780}{0.913} = 0.854 \quad \text{(trunchiere)}$$

Sistemul devine:

$$\begin{bmatrix} 0.913 & 0.659 \\ 0 & 0.001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.254 \\ 0.001 \end{bmatrix}$$

Soluțiile sunt $x_2 = 1$ și $x_1 = \frac{0.254 - 0.659}{0.913} = -0.417$.

Comparăm soluția găsită, $x_* = \begin{bmatrix} -0.417 \\ 1 \end{bmatrix}$, cu soluția reală, $x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$:

$$||x - x_*|| = || \begin{bmatrix} -0.557 \\ 0 \end{bmatrix} || = 0.557$$
$$||x|| = || \begin{bmatrix} -1 \\ 1 \end{bmatrix} || = \sqrt{2}$$
$$\frac{||x - x_*||}{||x||} = \frac{0.557}{\sqrt{2}} = 0.39$$

Eroarea relativă este de aproape 40%. Foarte mare! Dacă calculăm reziduul, am vedea că el este totuși mic, dar pe noi ne interesează soluția. Prezentăm următoarele strategii:

- Pivotare parțială - GPP;
- Pivotare parțială cu pivot scalat - GPPS;
- Pivotare totală - GPT.

2.4.2 Pivotare parţială - GPP

Pivotarea parțială are loc numai prin permutarea liniilor. La pasul p al algoritmului se aduce în locul pivotului cel mai mare element din coloană de sub acesta.

Algoritmul în MATLAB se gândește exact la fel, numai că înaintea calculării coeficienților μ_{ip} se face permutarea liniilor.

```
A = [1 3 1 9; 1 1 -1 1; 3 11 8 35];

[m, n] = size(A);
maxP = min(m, n);

for p = 1 : maxP
  [~, idx] = max(A(p : m, p), [], "ComparisonMethod", "abs");
  idx = idx + p - 1;  % ne trebuie index-ul din toata coloana,
                      % nu doar de la p : m

  P = eye(m);
  P(p, p) = 0;
  P(idx, idx) = 0;
  P(p, idx) = 1;
  P(idx, p) = 1;

  A = P * A;

  T = eye(m);
  mu = A(p + 1 : m, p) / A(p, p);
  T(p + 1 : m, p) = -mu;

  A = T * A;
end

disp(A);
```

Este de observat că această metodă de pivotare încă nu rezolvă eroarea din sistemul de mai sus, nu se efectuează nicio permutare. Totuși, în general, este suficientă pentru a face erorile rezonabile. Dacă pivotul ajunge să fie 0, atunci matricea este singulară.

2.5 Pivotare parţială cu pivot scalat - GPPS

Această tehnică este similară cu precedenta, doar că definim la început un factor de scalare pentru fiecare linie i, factor de forma:

$$s_i = \max_{j=p:n} \{|a_{ij}|\} \quad \text{sau} \quad s_i = \sum_{j=p}^{n} |a_{ij}|$$

Dacă $s_i = 0$ atunci matricea este singulară.

Prin această metodă interschimbăm liniile în funcție de elementul cel mai mare relativ la celelelte elemente de pe linie. Practic, la fiecare pas p, vom căuta întregul i_p astfel încât:

$$\frac{|a_{i_p p}|}{s_{i_p}} = \max_{i=p:n} \frac{|a_{ip}|}{s_i}$$

De exemplu, considerând sistemul $\begin{bmatrix} 1 & 10000 \\ 1 & 0.0001 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10000 \\ 1 \end{bmatrix}$, folosind GPP nu ar trebui sa interschimbăm liniile, dar avem exact aceeași problemă cu stabilitatea numerică. În consecință, interschimbăm liniile pentru că 1 este foarte mic relativ la 10000.

Algoritmul în MATLAB este similar cu cel de la GPP, dar introducem calculul factorului de scalare.

```
A = [1 3 1 9; 1 1 -1 1; 3 11 8 35];

[m, n] = size(A);
maxP = min(m, n);

for p = 1 : maxP
    s_factors = max(A(p : m, p : n - 1), [], 2, "ComparisonMethod", "abs");
    fractions = A(p : m, p) ./ s_factors;
    [~, idx] = max(fractions, [], "ComparisonMethod", "abs");
    idx = idx + p - 1;  % ne trebuie index-ul din toata coloana,
                        % nu doar de la p : m

    P = eye(m);
    P(p, p) = 0;
    P(idx, idx) = 0;
    P(p, idx) = 1;
    P(idx, p) = 1;

    A = P * A;

    T = eye(m);
    mu = A(p + 1 : m, p) / A(p, p);
    T(p + 1 : m, p) = -mu;

    A = T * A;
end

disp(A);
```

2.6 Pivotare totală - GPT

Cea mai bună stabilitate numerică se obține atunci când pivotul este cel mai mare element în modul, din toata submatricea rămasă. Această tehnică se numește pivotare totală. Totuși, în practică nu se folosește deoarece trebuie ca la fiecare iterație să parcurgi toată submatricea. Practic, costurile sunt mai mari decât beneficiile.

Pentru că acum trebuie să permutăm și coloanele, trebuie să ținem minte aceste permutări. Acest lucru se datorează faptului că se schimbă ordinea necunoscutelor. Pentru aceste permutări, ne folosim tot de o matrice P numai că o vom aplica la dreapta.

Algoritmul în MATLAB este similar cu cel de la GPP, doar că modificăm căutarea maximului.

```
A = [1 3 1 9; 1 1 -1 1; 3 11 8 35];

[m, n] = size(A);
maxP = min(m, n);

rightPerms = eye(n);

for p = 1 : maxP
  [~, idx] = max(A(p : m, p : n - 1), [], "all", "ComparisonMethod", "abs");
  [row, col] = ind2sub(size(A(p : m, p : n - 1)), idx);
  row = row + p - 1;
  col = col + p - 1;

  P = eye(n);
  P(p, p) = 0;
  P(col, col) = 0;
  P(p, col) = 1;
  P(col, p) = 1;

  rightPerms = rightPerms * P;
  A = A * P;

  P = eye(m);
  P(p, p) = 0;
  P(row, row) = 0;
  P(p, row) = 1;
  P(row, p) = 1;

  A = P * A;

  T = eye(m);
  mu = A(p + 1 : m, p) / A(p, p);
  T(p + 1 : m, p) = -mu;

  A = T * A;
end

disp(A);
disp(rightPerms(1 : 3, 1 : 3) * (A(1 : 3,1 : 3) \ A(:, 4)));
```

Pentru că A este matricea extinsă trebuie să avem grijă cu indicii. Ultima comandă de `display` demonstrează restabilirea ordinii inițiale a necunoscutelor.

2.7 Algoritmul Thomas pentru rezolvarea sistemului 3-diagonal

Un sistem tridiagonal are ecuații de forma:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i, \quad a_0 = 0, \quad c_n = 0$$

Iar sub forma matriceală:

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

Un astfel de sistem este un caz special al eliminării Gaussiene și poate fi rezolvat în doar $O(n)$ operații. Astfel de matrici apar în special la calculul spline-urilor cubice.

Fie primele 2 ecuații ale unui sistem tridiagonal:

$$\begin{aligned} b_1 x_1 + c_1 x_2 &= d_1 \\ a_2 x_1 + b_2 x_2 + c_2 x_3 &= d_2 \end{aligned}$$

Facem primul pas din eliminarea gaussiană, cu $\mu = \frac{a_2}{b_1}$:

$$\begin{aligned} b_1 x_1 + c_1 x_2 &= d_1 \\ (b_1 - \mu c_1) x_2 + c_2 x_3 &= d_2 - \mu d_1 \end{aligned}$$

Necunoscuta x_1 a fost eliminată, și am rămas cu o ecuație similară cu prima. Acest procedeu s-ar continua până la final, deci putem defini coeficienții recursiv (coeficienții primesi ecuații nu se modifică):

$$\begin{aligned} \mu &= \frac{a_i}{b_{i-1}} \\ b_i &= b_i - \mu c_{i-1} \\ d_i &= d_i - \mu d_{i-1} \end{aligned}$$

În final, soluția sistemului o găsim prin substituție înapoi:

$$\begin{aligned} x_n &= \frac{d_n}{b_n} \\ x_i &= \frac{d_i - c_i x_{i+1}}{b_i} \end{aligned}$$

Algoritmul în MATLAB îl construim folosind 4 vectori corespunzători coeficienților și termenilor liberi.

```
a = [0 -2 -3 -1];
b = [2 4 5 3];
c = [-1 -1 -2 0];
d = [5 6 7 8];

n = length(d);

x = zeros(n, 1);

for i = 2 : n
    mu = a(i) / b(i - 1);
    b(i) = b(i) - mu * c(i - 1);
    d(i) = b(i) - mu * d(i - 1);
end

x(n) = d(n) / b(n);
for i = (n - 1) : -1 : 1
    x(i) = (d(i) - c(i) * x(i + 1)) / b(i);
end

disp(x);
```

Pentru analiza numerică, este suficient să analizăm ecuația:

$$b_i' = b_i - \frac{a_i c_{i-1}}{b_{i-1}}$$

După cum am observat în laboratoarele trecute, dacă $|b_{i-1}|$ este foarte mic, erorile se pot amplifica atunci când îl calculăm pe μ. În consecință, ne-am dori ca $|b_i| > |c_i|$. Mai mult, ne dorim ca $|b_i| > |a_i|$. O matrice care satisface aceste condiții se numește *diagonal dominantă*. Această matrice este caracterizată de faptul ca elementele de pe diagonala principală sunt mai mari decât suma elementelor de pe linie, mai exact, în cazul nostru:

$$|b_i| \geq |a_i| + |c_i|$$

3 Probleme rezolvate

3.1 Problema 1

Fie sistemul de ecuații: $\begin{cases} 5.2 \cdot x_1 + 7.1 \cdot x_2 = 19.8 \\ 2.4 \cdot x_1 + 3.2 \cdot x_2 = 4.1 \end{cases}$ Rezolvaţi sistemul folosind eliminare gaussiană cu pivotare parţială.

3.2 Problema 5

Să se scrie un program OCTAVE pentru a implementa algoritmul Thomas de rezolvare a unui sistem 3-diagonal.

Soluţie:

```
a = [0 -2 -3 -1];
b = [2 4 5 3];
c = [-1 -1 -2 0];
```

```
d = [5 6 7 8];

n = length(d);

x = zeros(n, 1);

for i = 2 : n
    mu = a(i) / b(i - 1);
    b(i) = b(i) - mu * c(i - 1);
    d(i) = b(i) - mu * d(i - 1);
end

x(n) = d(n) / b(n);
for i = (n - 1) : -1 : 1
    x(i) = (d(i) - c(i) * x(i + 1)) / b(i);
end

disp(x);
```

Date de intrare:

$a = \begin{bmatrix} 2 & 9 & 2 & 3 & 6 \end{bmatrix}$, $b = \begin{bmatrix} 12 & 15 & 2 & 9 & 1 & 0 \end{bmatrix}$, $c = \begin{bmatrix} 10 & 3 & 9 & 1 & 4 \end{bmatrix}$, $d = \begin{bmatrix} 1 & 5 & 9 & 11 & 13 & 7 \end{bmatrix}$

Date de ieşire:

$x = \begin{bmatrix} 0.160494 & -0.092593 & 2.022634 & 0.643118 & 1.166667 & 2.475995 \end{bmatrix}$

4 Probleme

1. Construiţi o variantă modificată pentru algoritmul lui Thomas care să lucreze cu matricea:

$$A = \begin{bmatrix} b_1 & 0 & c_1 & & & & 0 \\ 0 & b_2 & 0 & c_2 & & & \\ a_3 & 0 & b_3 & 0 & c_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & a_{n-2} & 0 & b_{n-2} & 0 & c_{n-2} \\ & & & a_{n-1} & 0 & b_{n-1} & 0 \\ 0 & & & & a_n & 0 & b_n \end{bmatrix}$$

2. Fie sistemul de ecuaţii:

$$\begin{cases} 1.5 \cdot x_1 - 2.1 \cdot x_2 = 8.3 \\ -7.6 \cdot x_1 + 3.11 \cdot x_2 = 6.7 \end{cases}$$

   Rezolvați sistemul folosind eliminare GPP, GPPS, GPT.

Referințe

[1] Alexandru Negrescu. Algebra Liniară. O abordare prietenoasă, 2025, 15(2): 502-505. doi: 10.3934/naco.2023025