# Calcul științific

Fișe de laborator, anul II

Laboratoarele 1–6

## Cuprins

**Laboratorul 1. Introducere în MATLAB (I): variabile, vectori, matrice, scripturi și funcții** **3**

1.1 Mediul MATLAB . . . 3
1.2 Variabile, expresii și operații aritmetice . . . 3
1.3 Vectori . . . 4
1.4 Matrice . . . 4
1.5 Accesarea și modificarea elementelor . . . 4
1.6 Concatenare, matrice pe blocuri, transpunere . . . 5
1.7 Operații matriciale și operații element cu element . . . 5
1.8 Scripturi și funcții . . . 5
1.9 Exemplu complet . . . 6
1.10 Exerciții . . . 7

**Laboratorul 2. Introducere în MATLAB (II): structuri de control și reprezentări grafice** **8**

2.1 Instrucțiunea `if`; operatori relaționali și logici . . . 8
2.2 Instrucțiunea `for` și vectorizarea . . . 8
2.3 Instrucțiunea `while`; `break` și `continue` . . . 9
2.4 Reprezentarea grafică a curbelor plane . . . 10
2.5 Curbe în spațiu și suprafețe . . . 10
2.6 Bucle și grafice . . . 11
2.7 Exerciții . . . 11

**Laboratorul 3. Fișe de lucru MATLAB. Sisteme inferior și superior triunghiulare** **13**

3.1 Sisteme liniare cu matrice triunghiulare . . . 22

**Laboratorul 4. Eliminarea Gauss și factorizarea LU** **26**

4.1 Eliminarea Gauss pe hârtie . . . 26
4.2 De la eliminarea Gauss la factorizarea LU . . . 27
4.3 Implementarea factorizării LU . . . 27
4.4 Forma compactă Doolittle . . . 30
4.5 Exerciții . . . 33

**Laboratorul 5. Factorizarea Cholesky** **34**

5.1 Breviar teoretic . . . 34
5.2 Exercițiu: verificarea condițiilor de aplicare . . . 35

5.3 Calcul pe hârtie . . . . . 35
5.4 Implementare . . . . . 35
5.5 Exerciții . . . . . 38

**Laboratorul 6. Factorizarea LU cu pivotare parțială. Inversa unei matrice. Aplicație: bara elastică** **40**
6.1 De ce este nevoie de pivotare . . . . . 40
6.2 Algoritmul și calculul pe hârtie . . . . . 40
6.3 Implementare în MATLAB . . . . . 41
6.4 Calculul inversei . . . . . 43
6.5 Aplicație: deformarea unei bare elastice . . . . . 45
Anexa 6.A. Codul aplicației interactive . . . . . 51

**Organizarea fișierelor MATLAB.** Exemplele din laboratoarele 1–2 și funcțiile și scripturile din laboratoarele 3–6 se află în directoarele `lab1`, ..., `lab6`, așezate unul lângă altul. Funcțiile scrise într-un laborator sunt folosite în laboratoarele următoare (de exemplu, `subst_inferior` și `subst_superior` din laboratorul 3 sunt folosite peste tot); scripturile de test adaugă automat directoarele necesare în *path*, cu `addpath`.

# Laboratorul 1

Introducere în MATLAB (I): variabile, vectori, matrice, scripturi și funcții

**Obiective.** La finalul laboratorului veți ști să lucrați în fereastra de comenzi și în editor, să construiți și să modificați vectori și matrice, să deosebiți operațiile matriciale de cele element cu element și să scrieți scripturi și funcții proprii.

## 1.1 Mediul MATLAB

MATLAB este un mediu de calcul numeric conceput pentru lucrul cu vectori, matrice, funcții și algoritmi numerici. Numele provine de la *MATrix LABoratory*: chiar și un număr real este privit ca o matrice de dimensiune $1 \times 1$. MATLAB se folosește în trei moduri:

- interactiv, prin comenzi scrise în *Command Window*;
- prin *scripturi*, adică fișiere `.m` care conțin succesiuni de comenzi;
- prin *funcții* definite de utilizator.

Zonele importante ale interfeței sunt: *Command Window* (introducerea și executarea comenzilor), *Workspace* (variabilele existente în sesiunea curentă), *Current Folder* (directorul de lucru), *Editor* (scrierea scripturilor și funcțiilor) și *Command History* (istoricul comenzilor). Dacă rezultatul unei comenzi nu este atribuit unei variabile, MATLAB îl memorează temporar în variabila `ans`:

```
2 + 3          % rezultatul 5 este memorat in ans
```

Comenzi folosite frecvent:

```
clc          % sterge continutul vizibil din Command Window
clear        % elimina variabilele din Workspace
close all    % inchide ferestrele grafice
who          % afiseaza variabilele existente
whos         % afiseaza variabilele si informatii despre ele
help sin     % ajutor scurt pentru functia sin
doc plot     % documentatie detaliata pentru plot
```

Semnul `;` de la finalul unei instrucțiuni suprimă afișarea rezultatului.

## 1.2 Variabile, expresii și operații aritmetice

O variabilă se definește prin atribuire; ea nu trebuie declarată în prealabil, deoarece MATLAB stabilește tipul și dimensiunea atunci când primește valoarea:

```
a = 3;
b = 5;
c = a + b;
```

MATLAB face distincție între litere mari și mici: după `a = 2; A = 5;` există două variabile diferite. Un nume de variabilă începe cu o literă și poate conține litere, cifre și caracterul `_`. Evitați nume care coincid cu funcții sau constante MATLAB (`sin`, `sum`, `max`, `pi`): o variabilă cu un astfel de nume *maschează* funcția corespunzătoare până la ștergerea ei cu `clear`.

Operatorii aritmetici elementari sunt `+ - * / ^`:

```
a = 4; b = 2;
s = a + b;  d = a - b;  p = a * b;  q = a / b;  r = a^b;
```

Funcții matematice uzuale: `sqrt(x)`, `abs(x)`, `exp(x)`, `log(x)` (logaritmul natural), `sin(x)`, `cos(x)`.

## 1.3 Vectori

Un vector linie se definește prin `v = [1 2 3 4]` sau `v = [1, 2, 3, 4]`, iar un vector coloană prin `w = [1; 2; 3; 4]`: spațiul sau virgula separă elemente de pe aceeași linie, iar punctul și virgula separă linii.

Operatorul `:` construiește vectori cu elemente echidistante, în forma `valoare_initiala : pas : valoare_finala`; dacă pasul lipsește, el este 1:

```
x = 1:5          % 1 2 3 4 5
x = 0:0.2:1      % 0 0.2 0.4 0.6 0.8 1
x = linspace(0,1,6)   % 6 valori uniform distribuite intre 0 si 1
```

Diferența practică: la `a:h:b` fixăm pasul $h$, iar la `linspace(a,b,m)` fixăm numărul $m$ de puncte (pasul rezultă $(b-a)/(m-1)$, iar capătul $b$ este întotdeauna inclus).

## 1.4 Matrice

O matrice se introduce pe linii, separate prin `;` sau prin trecerea la rând nou:

```
A = [1 2 3; 4 5 6; 7 8 9]
A = [1 2 3
     4 5 6
     7 8 9];
```

Ambele forme definesc matricea $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.

```
zeros(3,4)      % matrice nula 3 x 4
ones(2,5)       % matrice 2 x 5 cu toate elementele 1
eye(4)          % matricea unitate de ordin 4
rand(3,3)       % matrice 3 x 3 cu elemente aleatoare din (0,1)
D = diag([2 4 6])   % matrice diagonala cu 2, 4, 6 pe diagonala
```

Dacă argumentul lui `diag` este o matrice, `diag(A)` extrage diagonala principală.

**Dimensiuni.** `size(A)` întoarce numărul de linii și de coloane; `[m,n] = size(A)` le salvează separat. Pentru vectori se folosește și `length(v)`, iar `numel(A)` dă numărul total de elemente.

## 1.5 Accesarea și modificarea elementelor

Pentru `A = [1 2 3; 4 5 6; 7 8 9]`:

```
A(2,3)        % elementul de pe linia 2 si coloana 3 (adica 6)
A(2,:)        % linia a doua
A(:,3)        % coloana a treia
A(1:2,2:3)    % submatricea formata din liniile 1-2 si coloanele 2-3
v(end)        % ultimul element al vectorului v
A(end,:)      % ultima linie a lui A
```

Simbolul `:` singur înseamnă „toate liniile” sau „toate coloanele”, iar `end` desemnează ultimul indice. Modificări:

```
A(2,3) = 10;        % modificarea unui element
A(1,:) = [2 4 6];   % modificarea unei linii
A(2,:) = [];        % eliminarea liniei 2
A(:,3) = [];        % eliminarea coloanei 3
```

**Indexare liniară.** Elementele unei matrice pot fi accesate și printr-un singur indice; numerotarea se face *pe coloane*. Pentru `A = [1 2; 3 4]` avem `A(1) = 1`, `A(2) = 3`, `A(3) = 2`, `A(4) = 4`, iar `A(:)` transformă matricea în vectorul coloană $(1, 3, 2, 4)^T$.

## 1.6 Concatenare, matrice pe blocuri, transpunere

Parantezele drepte concatenează: orizontal cu spațiu (același număr de linii), vertical cu `;` (același număr de coloane).

```
a = [1 2 3]; b = [4 5];   c = [a b]      % [1 2 3 4 5]
a = [1 2 3]; b = [4 5 6]; C = [a; b]     % matrice 2 x 3
A = [1 2; 3 4]; B = [5 6; 7 8];
C = [A B];  D = [A; B];                  % 2 x 4, respectiv 4 x 2
```

Construcția pe blocuri este foarte utilă în algebra liniară numerică:

```
M = [eye(2) ones(2,1); zeros(1,2) 3]
```

produce $M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix}$.

Transpusa unei matrice reale se obține cu `A'`. Pentru matrice complexe, `A'` este transpusa *conjugată*, iar `A.'` este transpusa fără conjugare; pentru matrice reale cele două coincid.

## 1.7 Operații matriciale și operații element cu element

Adunarea și scăderea se fac element cu element (`A + B`, `A - B`). Pentru înmulțire, împărțire și putere, MATLAB distinge între operația matricială și cea element cu element, marcată prin punct:

```
A * B    % produs matricial (numarul de coloane ale lui A = numarul de linii ale lui B)
A .* B   % inmultire element cu element (aceleasi dimensiuni)
A / B    % impartire matriciala: A*inv(B)
A ./ B   % impartire element cu element
A^2      % putere matriciala: A*A (A patratica)
A.^2     % fiecare element la patrat
```

De aceea, funcția $f(x) = x^2 - 2x + 1$ evaluată pe un vector $x$ se scrie `y = x.^2 - 2*x + 1`, nu `y = x^2 - 2*x + 1` (a doua formă încearcă produsul matricial $x \cdot x$ și dă eroare pentru un vector linie).

Funcțiile uzuale se aplică fiecărei componente: pentru `x = [0 pi/2 pi]`, `sin(x)` este vectorul $(0, 1, 0)$. Atenție: exponențiala *matricială* este `expm(A)`, nu `exp(A)`.

Funcții frecvente pe vectori: `sum(v)`, `max(v)`, `min(v)`, `mean(v)`, `norm(v)` (norma euclidiană). Pentru matrice, unele funcții operează implicit pe coloane: `sum(A)` dă suma fiecărei coloane, iar suma tuturor elementelor este `sum(A,'all')` sau, în versiuni mai vechi, `sum(A(:))`.

## 1.8 Scripturi și funcții

**Scripturi.** Un script este un fișier `.m` cu o succesiune de comenzi; se rulează cu butonul *Run* sau scriind numele lui în Command Window. Exemplu, fișierul `exemplu.m`:

```
clear; clc;
a = 2;
b = 3;
c = a^2 + b^2;
fprintf('Rezultatul este %g\n', c);
```

Comentariile încep cu `%`. Un script lung se împarte în secțiuni care încep cu `%%` și care pot fi rulate separat din Editor:

```
%% Definirea datelor
%% Calculul valorilor
%% Afisarea rezultatelor
```

**Funcții.** Pentru calcule reutilizabile definim funcții. Numele fișierului trebuie, în mod obișnuit, să coincidă cu numele funcției principale. Fișierul `patrat.m`:

```
function y = patrat(x)
    y = x.^2;
end
```

se apelează prin `patrat(3)` sau `w = patrat([1 2 3 4])`. O funcție poate avea mai multe intrări și mai multe ieșiri:

```
function [s, p] = suma_produs(a, b)
    s = a + b;
    p = a * b;
end
```

apelată prin `[s, p] = suma_produs(2, 3)`.

**Script sau funcție?** Un script execută instrucțiunile în Workspace-ul curent (folosește și modifică variabilele existente). O funcție are propriul spațiu de lucru: primește date numai prin argumentele de intrare și returnează rezultate numai prin argumentele de ieșire. De aceea funcțiile sunt mai ușor de testat și de refolosit.

**Funcții anonime.** Pentru funcții simple, definite într-o singură expresie:

```
f = @(x) x.^2 - 2*x + 1;
f(3)                    % 4
y = f(0:0.1:2);         % evaluare pe un vector
g = @(x,y) x.^2 + y.^2;  % functie de doua variabile
```

**Directorul curent.** MATLAB caută fișierele în directorul curent și în directoarele din *path*: `pwd` (directorul curent), `dir` (conținutul lui), `cd` (schimbarea directorului), `path`. O funcție proprie poate fi apelată numai dacă fișierul ei se află într-unul dintre aceste directoare.

**Afișarea rezultatelor.** `disp(x)` afișează simplu, iar `fprintf` afișează formatat: în `fprintf('x = %.3f\n', x)`, `%.3f` înseamnă număr real cu 3 zecimale, `%d` întreg, `%g` forma cea mai scurtă, `%e` forma exponențială, iar `\n` trecerea la rând nou. Comenzile `format short` și `format long` schimbă doar afișarea, nu precizia internă a calculelor.

## 1.9 Exemplu complet

Calculăm valorile funcției $f(x) = x^2 - 2x + 2$ în 11 puncte echidistante din $[0, 2]$ și afișăm valoarea minimă:

```
clear; clc;
x = linspace(0,2,11);
y = x.^2 - 2*x + 2;
m = min(y);
fprintf('Valoarea minima este %.4f\n', m);
```

### 1.10 Exerciții

1. Construiți vectorul $v = (3, 6, 9, \ldots, 30)$ în două moduri, cu `:` și cu `linspace`. Afișați `length(v)`, al cincilea element, ultimele trei elemente și elementele de pe pozițiile impare.
2. Construiți matricea $A$ de ordin 4 cu $a_{ij} = i + j$ folosind `(1:4)'` și `1:4` (indicație: `(1:4)' + (1:4)`). Extrageți diagonala, linia a doua, coloana a treia și submatricea $2 \times 2$ din colțul dreapta-jos. Înlocuiți prima linie cu zerouri.
3. Fie $A = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 3 \end{smallmatrix}\right)$ și $B = \left(\begin{smallmatrix} 1 & 0 \\ -1 & 2 \end{smallmatrix}\right)$. Calculați `A*B`, `B*A`, `A.*B`, `A^2`, `A.^2`, `A'` și `A./B`. Explicați într-un comentariu de ce `A*B` $\neq$ `B*A` și ce valoare apare în `A./B` acolo unde $b_{ij} = 0$.
4. Construiți, pe blocuri, matricea $\left(\begin{smallmatrix} I_3 & 0 \\ 0 & 2I_2 \end{smallmatrix}\right)$ de ordin 5, unde $I_k$ este matricea unitate de ordin $k$, și verificați-i dimensiunea.
5. Scrieți funcția `[m, s] = medie_abatere(v)` care întoarce media $m = \frac{1}{n}\sum_{i=1}^{n} v_i$ și abaterea standard $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - m)^2}$ ale unui vector $v$ cu $n$ componente, folosind `sum` și operații element cu element. Comparați cu `mean(v)` și `std(v,1)`.
6. Definiți funcția anonimă $g(x) = e^{-x}\cos x$ și afișați cu `fprintf`, cu 6 zecimale, valorile ei în $x = 0, 0.5, 1, \ldots, 3$, câte una pe rând, în forma `g(0.50) = 0.532281`.
7. Scrieți un script cu trei secțiuni `%%` care: definește un vector de 5 temperaturi în grade Celsius; le transformă în Fahrenheit ($F = \frac{9}{5}C + 32$); afișează un tabel cu două coloane folosind `fprintf`.

# Laboratorul 2

Introducere în MATLAB (II): structuri de control și reprezentări grafice

**Obiective.** La finalul laboratorului veți ști să folosiți instrucțiunile `if`, `for`, `while`, să alegeți între o buclă și o formulare vectorizată și să reprezentați grafic curbe plane, curbe în spațiu și suprafețe.

## 2.1 Instrucțiunea `if`; operatori relaționali și logici

Instrucțiunea `if` execută un grup de comenzi numai dacă o condiție este adevărată:

```
x = -2;
if x >= 0
    disp('x este nenegativ');
else
    disp('x este negativ');
end
```

Pentru mai multe cazuri se folosește `elseif`:

```
if x > 0
    disp('x este pozitiv');
elseif x < 0
    disp('x este negativ');
else
    disp('x este zero');
end
```

Operatorii relaționali sunt `<`, `>`, `<=`, `>=`, `==` (egalitate), `~=` (diferit). Atenție la diferența dintre atribuire și testul de egalitate: `x = 3` atribuie, `x == 3` testează. Operatorii logici pentru condiții scalare sunt `&&` (și), `||` (sau) și `~` (negație):

```
x = 0.5;
if x >= 0 && x <= 1
    disp('x apartine intervalului [0,1]');
end
```

Cu `&&` și `||`, al doilea operand se evaluează numai dacă este necesar; de exemplu, în `k <= length(v) && v(k) > 0` nu se ajunge la `v(k)` când $k$ depășește lungimea lui $v$.

## 2.2 Instrucțiunea `for` și vectorizarea

Instrucțiunea `for` repetă un grup de comenzi pentru fiecare valoare dintr-un vector cunoscut dinainte:

```
for k = 1:5
    disp(k);
end
s = 0;                  % suma primelor zece numere naturale
for k = 1:10
    s = s + k;
end
for k = 0:0.5:2         % pasul poate fi orice numar real
    disp(k);
end
```

Același vector se poate obține cu o buclă sau vectorizat:

```
for k = 1:5             % prin bucla
    y(k) = k^2;
end
```

```
k = 1:5; y = k.^2;      % vectorizat
```

Când construim un vector într-o buclă, alocăm memoria înainte, cu `zeros`; altfel MATLAB mărește vectorul la fiecare pas, ceea ce încetinește mult codul:

```
n = 100;
y = zeros(1,n);
for k = 1:n
    y(k) = k^2;
end
```

**Bucle imbricate.** Pentru matrice putem folosi o buclă în altă buclă; fiecare element este calculat separat:

```
A = zeros(3,4);
for i = 1:3
    for j = 1:4
        A(i,j) = i + j;
    end
end
```

Când este posibil, se preferă o formulare vectorială sau matricială; în algoritmii de algebră liniară din laboratoarele următoare vom scrie însă buclele explicit, pentru a urmări exact formulele.

### 2.3 Instrucțiunea `while`; `break` și `continue`

Instrucțiunea `while` se folosește atunci când numărul de repetări nu este cunoscut dinainte; bucla continuă cât timp condiția este adevărată:

```
x = 1;
while x < 100
    x = 2*x;
end
```

În interiorul buclei trebuie modificată o variabilă care intervine în condiție; altfel se obține o buclă infinită (care se oprește cu *Ctrl+C*).

*Exemplu.* Determinăm numărul $n$ de termeni ai sumei $1 + \frac{1}{2} + \frac{1}{4} + \cdots$ necesari pentru ca suma să depășească 1.9. Variabila `termen` reține termenul curent, care se înjumătățește la fiecare pas:

```
s = 0; termen = 1; n = 0;
while s < 1.9
    s = s + termen;
    termen = termen/2;
    n = n + 1;
end
```

`break` întrerupe bucla, iar `continue` trece direct la următoarea iterație:

```
for k = 1:100
    if k^2 > 50
        break;          % se iese din bucla la k = 8
    end
end
for k = 1:10
    if mod(k,2) == 0
        continue;       % numerele pare sunt sarite
    end
    disp(k);            % se afiseaza numai numerele impare
end
```

## 2.4 Reprezentarea grafică a curbelor plane

Comanda `plot(x,y)` unește prin segmente punctele $(x_k, y_k)$, unde $x$ și $y$ sunt vectori de aceeași lungime; cu cât numărul de puncte este mai mare, cu atât curba apare mai netedă.

```
x = linspace(0,2*pi,200);
y = sin(x);
plot(x,y);
xlabel('x'); ylabel('sin(x)');
title('Functia sinus');
grid on;
```

```
plot(x,sin(x),x,cos(x));          % varianta directa
legend('sin(x)','cos(x)');

plot(x,sin(x));                   % varianta cu hold on
hold on;
plot(x,cos(x));
hold off;
legend('sin(x)','cos(x)'); grid on;
```

```
plot(x,y,'--')                    % alte stiluri: '-', ':', '-.', 'o', '-o'
plot(x,y,'LineWidth',1.5);        % grosimea liniei
xlim([0 2*pi]); ylim([-1.2 1.2]); % sau axis([0 2*pi -1.2 1.2])
```

**Mai multe grafice în aceeași fereastră.** `subplot(m,n,k)` împarte fereastra în $m$ linii și $n$ coloane de grafice și o selectează pe a $k$-a (numerotate pe linii):

```
subplot(2,1,1); plot(x,sin(x)); title('sin(x)');
subplot(2,1,2); plot(x,cos(x)); title('cos(x)');
```

**Curbe parametrizate.** Cercul unitate $x(t) = \cos t$, $y(t) = \sin t$, $t \in [0, 2\pi]$; `axis equal` păstrează aceeași scară pe cele două axe (altfel cercul apare ca elipsă):

```
t = linspace(0,2*pi,300);
plot(cos(t),sin(t)); axis equal; grid on;
```

## 2.5 Curbe în spațiu și suprafețe

O curbă în spațiu se reprezintă cu `plot3`. Elicea $x(t) = \cos t$, $y(t) = \sin t$, $z(t) = t$:

```
t = linspace(0,6*pi,500);
plot3(cos(t),sin(t),t); grid on;
xlabel('x'); ylabel('y'); zlabel('z');
```

Pentru o suprafață $z = f(x, y)$ construim mai întâi o rețea de puncte. Dacă $x$ are $m$ componente și $y$ are $p$ componente, `[X,Y] = meshgrid(x,y)` produce două matrice $p \times m$, cu $X(i, j) = x_j$ și $Y(i, j) = y_i$; astfel perechea $(X(i, j), Y(i, j))$ parcurge toate punctele rețelei, iar $f$ se evaluează element cu element:

```
x = linspace(-2,2,50);
y = linspace(-2,2,50);
```

```
[X,Y] = meshgrid(x,y);
Z = X.^2 + Y.^2;                    % operatorul .^ este esential
surf(X,Y,Z);                        % suprafata colorata
xlabel('x'); ylabel('y'); zlabel('z'); title('z = x^2 + y^2');
mesh(X,Y,Z);                        % doar reteaua (plasa)
```

`surf` evidențiază suprafața, iar `mesh` rețeaua de reprezentare. Curbele de nivel $f(x,y) = c$ se desenează cu `contour(X,Y,Z)` (pentru $z = x^2 + y^2$ ele sunt cercuri), iar `surfc(X,Y,Z)` combină suprafața cu proiecția curbelor de nivel.

*Exemplu complet.* Pentru $f(x,y) = e^{-(x^2+y^2)}\cos(3x)\sin(3y)$:

```
clear; clc; close all;
x = linspace(-2,2,100);
y = linspace(-2,2,100);
[X,Y] = meshgrid(x,y);
Z = exp(-(X.^2 + Y.^2)).*cos(3*X).*sin(3*Y);
surfc(X,Y,Z);
xlabel('x'); ylabel('y'); zlabel('f(x,y)');
title('Suprafata z = f(x,y)'); grid on;
```

## 2.6 Bucle și grafice

Valorile unei funcții pot fi calculate printr-o buclă,

```
x = linspace(0,5,200);
y = zeros(size(x));
for k = 1:length(x)
    y(k) = exp(-x(k))*sin(4*x(k));
end
plot(x,y); grid on;
```

dar forma vectorizată `y = exp(-x).*sin(4*x)` este mai simplă și mai rapidă.

**Idei de reținut.**

- MATLAB este orientat spre calculul vectorial și matricial; diferența dintre `*` și `.*`, respectiv dintre `^` și `.^`, este esențială.
- Scripturile grupează comenzi, iar funcțiile organizează calcule reutilizabile.
- `if`, `for` și `while` controlează ordinea de execuție.
- Pentru curbe folosim `plot`; pentru suprafețe folosim `meshgrid`, apoi `surf`, `mesh` sau `contour`.

## 2.7 Exerciții

1. Scrieți funcția `y = semn_pe_ramuri(x)` care, pentru un vector $x$, întoarce vectorul $y$ cu $y_k = -1$ dacă $x_k < 0$, $y_k = 0$ dacă $x_k = 0$ și $y_k = x_k^2$ dacă $x_k > 0$. Folosiți o buclă `for`, alocarea cu `zeros` și `if/elseif/else`. Desenați rezultatul pentru `x = linspace(-1,1,201)`.
2. Aproximați $e = \sum_{k\geq 0} \frac{1}{k!}$ cu `while`, adunând termeni până când termenul curent devine mai mic decât $10^{-10}$ (termenul $k$ se obține din termenul $k-1$ împărțind la $k$). Afișați numărul de termeni și eroarea `abs(s - exp(1))`.
3. Construiți cu două bucle imbricate tabla înmulțirii $9\times 9$, apoi construiți-o vectorizat, cu `(1:9)' * (1:9)`, și verificați că cele două matrice coincid (`isequal`).
4. Folosind `break`, găsiți cel mai mic $n$ pentru care $\sum_{k=1}^{n} \frac{1}{k} > 3$.

5. Desenați în aceeași figură, cu `hold on`, funcțiile $\sin x$, $\sin 2x$ și $\sin 3x$ pe $[0, 2\pi]$, cu stiluri diferite, legendă, etichete și grilă.
6. Desenați curba Lissajous $x(t) = \sin 3t$, $y(t) = \sin 2t$, $t \in [0, 2\pi]$, cu `axis equal`.
7. Pentru $f(x, y) = x^2 - y^2$ pe $[-2, 2]^2$ desenați, cu `subplot(1,2,k)`, suprafața (`surf`) și curbele de nivel (`contour`). Ce tip de punct este originea?

# Laboratorul 3

## Fișe de lucru MATLAB. Sisteme inferior și superior triunghiulare

**Structura laboratorului.** În prima parte, fiecare student rezolvă, în 20 de minute, una dintre cele opt fișe de lucru de mai jos (câte o pagină, cu sarcini pe pași fixați). În a doua parte rezolvăm sisteme liniare cu matrice triunghiulare, pe hârtie și în MATLAB; funcțiile scrise acum vor fi folosite în toate laboratoarele următoare.

## Calcul științific – fișă de lucru MATLAB **Varianta 1**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V1_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Creați `x = linspace(0,2,11)`, definiți `f = @(x) x.^2 - 2*x + 2` și calculați `y = f(x)`.
2. Afișați cu `fprintf` valoarea minimă a lui `y`, cu 4 zecimale.
3. Desenați graficul lui $f$ pe $[0,2]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `p = urmatorul_prim(n)`, care returnează cel mai mic număr prim strict mai mare decât $n$.

1. Inițializați `p = n + 1` și `gasit = false`.
2. Cu `while ~gasit`, repetați pașii 3–5.
3. Puneți `prim = true`. Cu `for d = 2:p-1`, verificați divizorii lui $p$: dacă `mod(p,d) == 0`, atunci `prim = false`.
4. Cu `if prim && p > 1`, puneți `gasit = true`.
5. Altfel, măriți $p$ cu 1.
6. Testați funcția pe $n = 10$, 13 și 100.

## Calcul științific – fișă de lucru MATLAB **Varianta 2**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V2_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Creați `v = 1:10` și `w = v.^2`, apoi afișați `w(end)` și `sum(w)`.
2. Construiți `M = [eye(2) ones(2,1); zeros(1,2) 3]` și afișați `size(M)`.
3. Pe `x = linspace(0,2*pi,200)`, desenați graficul lui $\sin x$ cu `plot(x, y)`.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `s = suma_cifre_vector(v)`, care returnează suma cifrelor tuturor elementelor strict pozitive ale vectorului de întregi $v$.

1. Inițializați `s = 0`.
2. Cu `for k = 1:length(v)`, parcurgeți elementele lui $v$.
3. Cu `if v(k) > 0`, prelucrați doar elementele pozitive. Puneți `x = v(k)`.
4. Cu `while x > 0`, adunați la $s$ ultima cifră, `mod(x,10)`, apoi eliminați-o: `x = floor(x/10)`.
5. Testați funcția pe `[12 -7 305 0]` și pe `[999 1 10]`.

## Calcul științific – fișă de lucru MATLAB **Varianta 3**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V3_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Construiți cu două bucle `for` imbricate matricea $3 \times 3$ cu `A(i,j) = i*j`, pornind de la `A = zeros(3,3)`.
2. Afișați `diag(A)` și ultima coloană, `A(:,end)`.
3. Desenați graficul lui $y = \cos(2x)$ pe $[0, \pi]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `[kmax, nmax] = collatz_max(N)`. Regula Collatz înlocuiește $n$ cu $n/2$ dacă $n$ este par și cu $3n + 1$ dacă $n$ este impar, până se ajunge la 1. Funcția returnează cel mai mare număr de pași $k_{\max}$, pentru $n = 1, \ldots, N$, și valoarea $n_{\max}$ la care se atinge.

1. Inițializați `kmax = 0` și `nmax = 1`.
2. Cu `for n = 1:N`, puneți `x = n` și `k = 0`.
3. Cu `while x ~= 1`: dacă `mod(x,2) == 0`, atunci `x = x/2`, altfel `x = 3*x + 1`; apoi `k = k + 1`.
4. După bucla `while`, cu `if k > kmax`, actualizați `kmax = k` și `nmax = n`.
5. Testați funcția pe $N = 10$ și $N = 30$.

## Calcul științific – fișă de lucru MATLAB **Varianta 4**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V4_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Fie `a = [1 2 3]` și `b = [4 5 6]`. Construiți `C = [a; b]` și afișați `size(C)`.
2. Calculați `a .* b` și produsul scalar `a * b'`.
3. Desenați graficul lui $y = e^{-x}\sin(4x)$ pe $[0, 5]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `c = cmmdc_vector(v)`, care returnează cel mai mare divizor comun al elementelor vectorului $v$ de numere naturale nenule, cu algoritmul lui Euclid.

1. Cu `if isempty(v)`, opriți execuția cu `error('Vector gol')`.
2. Puneți `c = v(1)`.
3. Cu `for k = 2:length(v)`, puneți `a = c` și `b = v(k)`.
4. Cu `while b ~= 0`, calculați `r = mod(a,b)`, apoi `a = b` și `b = r`. La ieșire, `c = a`.
5. Testați funcția pe `[84 126 210]` și pe `[17 34 51]`.

## Calcul științific – fișă de lucru MATLAB **Varianta 5**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V5_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Construiți `D = diag([2 4 6])` și `M = [D ones(3,1)]`, apoi afișați `size(M)`.
2. Afișați `sum(M)` și `M(2,:)`.
3. Desenați graficul lui $y = \sqrt{x}$ pe $[0, 4]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `nr = numara_palindroame(a, b)`, care returnează câte numere naturale din intervalul $[a, b]$ sunt palindroame, adică se citesc la fel în ambele sensuri (de exemplu 121).

1. Cu `if a > b`, opriți execuția cu `error`. Inițializați `nr = 0`.
2. Cu `for n = a:b`, puneți `x = n` și `r = 0`.
3. Cu `while x > 0`, construiți răsturnatul: `r = 10*r + mod(x,10)`, apoi `x = floor(x/10)`.
4. Cu `if r == n`, măriți `nr` cu 1.
5. Testați funcția pe $[1, 100]$ și pe $[100, 200]$.

## Calcul științific – fișă de lucru MATLAB **Varianta 6**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V6_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Fie `A = [1 2; 3 4]` și `B = [0 1; 1 0]`. Calculați `A*B` și `A.*B`.
2. Calculați `A^2` și `A.^2` și notați într-un comentariu diferența.
3. Desenați graficul lui $y = x^3$ pe $[-1, 1]$ cu `plot(x, y)`, folosind 100 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `P = numere_perfecte(N)`, care returnează vectorul numerelor perfecte cel mult egale cu $N$. Un număr este perfect dacă este egal cu suma divizorilor săi mai mici decât el, de exemplu $6 = 1 + 2 + 3$.

1. Inițializați `P = []`.
2. Cu `for n = 2:N`, puneți `s = 0` și `d = 1`.
3. Cu `while d < n`: dacă `mod(n,d) == 0`, atunci `s = s + d`; apoi `d = d + 1`.
4. După bucla `while`, cu `if s == n`, adăugați $n$ la $P$: `P = [P n]`.
5. Testați funcția pe $N = 30$ și $N = 500$.

## Calcul științific – fișă de lucru MATLAB **Varianta 7**

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V7_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Creați `v = 0:0.25:2` și afișați `length(v)` și `v(2:2:end)`.
2. Definiți `f = @(x) exp(x) - 2` și afișați `max(f(v))` cu `fprintf`, cu 3 zecimale.
3. Desenați graficul lui $f$ pe $[0, 2]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `[b, k] = binar(n)`, care returnează vectorul $b$ al cifrelor binare ale numărului natural $n$ (cea mai semnificativă cifră prima) și numărul $k$ de cifre 1.

1. Cu `if n == 0`, puneți `b = 0`, `k = 0` și ieșiți cu `return`.
2. Puneți `b = []`.
3. Cu `while n > 0`, adăugați restul la stânga: `b = [mod(n,2) b]`, apoi `n = floor(n/2)`.
4. Puneți `k = 0`. Cu `for i = 1:length(b)` și `if b(i) == 1`, măriți $k$ cu 1.
5. Testați funcția pe $n = 13$, 0 și 64.

## Calcul științific – fișă de lucru MATLAB — Varianta 8

Nume și prenume: _____ Grupa: _____

*Timp de lucru: 20 de minute.* Sarcina 1 se scrie în scriptul `lab_V8_nume.m`, cu câte o secțiune `%%` pentru fiecare pas. Funcția din sarcina 2 se scrie într-un fișier separat, cu același nume ca funcția, și se testează din script. Nu folosiți funcții MATLAB care rezolvă direct problema (de exemplu `isprime`, `gcd`, `dec2bin`, `find`). Punctaj: sarcina 1 – 4 p, sarcina 2 – 5 p, oficiu – 1 p.

**Sarcina 1 (script, 4 p).**

1. Pentru `t = pi/6`, construiți `R = [cos(t) -sin(t); sin(t) cos(t)]`.
2. Calculați `R'*R` și `R*[1; 0]`.
3. Desenați graficul lui $y = 1/(1+x^2)$ pe $[-3, 3]$ cu `plot(x, y)`, folosind 200 de puncte.

**Sarcina 2 (funcție, 5 p).** Scrieți funcția `poz = cauta_binar(v, val)`, care caută `val` într-un vector $v$ sortat crescător și returnează poziția lui, sau 0 dacă `val` nu apare în $v$.

1. Cu `for i = 1:length(v)-1` și `if v(i) > v(i+1)`, opriți execuția cu `error('Vectorul nu este sortat')`.
2. Puneți `st = 1`, `dr = length(v)` și `poz = 0`.
3. Cu `while st <= dr && poz == 0`, calculați `m = floor((st + dr)/2)`.
4. Dacă `v(m) == val`, puneți `poz = m`; altfel, dacă `v(m) < val`, puneți `st = m + 1`; altfel, puneți `dr = m - 1`.
5. Testați funcția pe `v = [1 3 5 7 9 11]` cu `val = 7` și `val = 4`, și pe vectorul nesortat `[3 1 2]`.

## 3.1 Sisteme liniare cu matrice triunghiulare

Fie $n \geq 1$ un număr natural, $A \in \mathbb{R}^{n\times n}$ o matrice pătratică și $b \in \mathbb{R}^n$ un vector. Notăm cu $a_{ij}$ elementul lui $A$ de pe linia $i$ și coloana $j$ și cu $b_i$ componenta $i$ a lui $b$. Sistemul $Ax = b$, cu necunoscuta $x \in \mathbb{R}^n$, este ușor de rezolvat în două cazuri particulare, care stau la baza tuturor metodelor directe din laboratoarele următoare:

- $L = (\ell_{ij})$ este *inferior triunghiulară* dacă $\ell_{ij} = 0$ pentru $j > i$ (deasupra diagonalei sunt numai zerouri);
- $U = (u_{ij})$ este *superior triunghiulară* dacă $u_{ij} = 0$ pentru $i > j$ (sub diagonală sunt numai zerouri).

Determinantul unei matrice triunghiulare este produsul elementelor diagonale, deci un sistem triunghiular are soluție unică dacă și numai dacă toate elementele diagonale sunt nenule.

**Substituția directă (înainte).** Fie $L$ inferior triunghiulară cu $\ell_{ii} \neq 0$. Ecuația $i$ a sistemului $Lx = b$ este

$$\ell_{i1}x_1 + \ell_{i2}x_2 + \cdots + \ell_{i,i-1}x_{i-1} + \ell_{ii}x_i = b_i,$$

deoarece termenii cu $j > i$ lipsesc. Prima ecuație conține numai $x_1$; dacă am calculat deja $x_1, \ldots, x_{i-1}$, ecuația $i$ conține o singură necunoscută nouă, $x_i$. Izolând-o, obținem

$$x_i = \frac{1}{\ell_{ii}}\Big(b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j\Big), \qquad i = 1, 2, \ldots, n, \tag{3.1}$$

unde, pentru $i = 1$, suma este vidă (egală cu 0).

**Substituția inversă (înapoi).** Fie $U$ superior triunghiulară cu $u_{ii} \neq 0$. Acum ultima ecuație conține numai $x_n$, iar ecuația $i$ este $u_{ii}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{in}x_n = b_i$. Parcurgând ecuațiile de jos în sus, obținem

$$x_i = \frac{1}{u_{ii}}\Big(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\Big), \qquad i = n, n-1, \ldots, 1. \tag{3.2}$$

**Costul.** Pentru $x_i$ se fac $i-1$ înmulțiri, $i-1$ scăderi și o împărțire, deci în total $\sum_{i=1}^{n}(2i-1) = n^2$ operații. Rezolvarea unui sistem triunghiular este deci mult mai ieftină decât cea a unui sistem oarecare, care, după cum vom vedea, costă de ordinul $n^3$ operații.

### Calcul pe hârtie

(a) Rezolvați prin substituție directă

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ -1 & 2 & 4 \end{pmatrix} x = \begin{pmatrix} 2 \\ 7 \\ -1 \end{pmatrix},$$

scriind la fiecare pas formula (3.1) cu numerele înlocuite.

(b) Rezolvați prin substituție inversă

$$\begin{pmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & -2 \end{pmatrix} x = \begin{pmatrix} -3 \\ 0 \\ -6 \end{pmatrix}.$$

(c) Verificați ambele soluții prin înlocuire în sistem.

### Implementare în MATLAB

Fiecare funcție verifică întâi că datele de intrare permit aplicarea metodei: matricea este pătratică, $b$ are dimensiunea potrivită, matricea este triunghiulară de tipul cerut, iar elementele diagonale sunt nenule. Dacă o condiție nu este îndeplinită, `error` oprește execuția cu un mesaj clar, în loc să întoarcă un rezultat fără sens.

Pentru verificarea formei triunghiulare se folosesc `triu(L,1)`, care păstrează numai elementele de deasupra diagonalei (restul devin 0), și `tril(U,-1)`, care le păstrează numai pe cele de sub diagonală. Expresia `any(any(M ~= 0))` este adevărată dacă $M$ are cel puțin un element nenul: `M ~= 0` este o matrice logică, primul `any` lucrează pe coloane, al doilea pe vectorul rezultat.

**Funcția** `subst_inferior` (fișierul `subst_inferior.m`), care implementează (3.1):

```
function x = subst_inferior(L, b)
% SUBST_INFERIOR Rezolva sistemul L*x = b, cu L inferior triunghiulara,
% prin substitutie directa (inainte):
%   x(i) = ( b(i) - sum_{j<i} L(i,j)*x(j) ) / L(i,i), i = 1,...,n.

% Numarul de linii n si numarul de coloane m ale lui L
[n, m] = size(L);

% Daca L nu este patratica, oprim executia cu un mesaj de eroare
if n ~= m
    error('Matricea L trebuie sa fie patratica.');
end

% Vectorul termenilor liberi trebuie sa aiba n componente
if length(b) ~= n
    error('Dimensiunea lui b nu se potriveste cu L.');
end

% triu(L,1) pastreaza doar elementele de deasupra diagonalei;
% daca vreunul este nenul, L nu este inferior triunghiulara
if any(any(triu(L, 1) ~= 0))
    error('Matricea L nu este inferior triunghiulara.');
end

% Alocam vectorul coloana al solutiei (n x 1), initial cu zerouri
x = zeros(n, 1);

% Calculam necunoscutele in ordinea x(1), x(2), ..., x(n)
for i = 1:n
    % Un element nul pe diagonala face imposibila impartirea
    if L(i,i) == 0
        error('L(%d,%d) = 0: sistemul nu are solutie unica.', i, i);
    end
    % s va contine suma L(i,1)*x(1) + ... + L(i,i-1)*x(i-1)
    s = 0;
    % Adunam contributiile necunoscutelor deja calculate
    for j = 1:i-1
        s = s + L(i,j) * x(j);
    end
    % Din ecuatia i obtinem necunoscuta x(i)
    x(i) = (b(i) - s) / L(i,i);
end
end
```

**Funcția** `subst_superior` (fișierul `subst_superior.m`), care implementează (3.2); singurele diferențe sunt testul de formă și ordinea în care se parcurg necunoscutele:

```
function x = subst_superior(U, b)
% SUBST_SUPERIOR Rezolva sistemul U*x = b, cu U superior triunghiulara,
% prin substitutie inversa (inapoi):
%   x(i) = ( b(i) - sum_{j>i} U(i,j)*x(j) ) / U(i,i), i = n,...,1.

% Numarul de linii n si numarul de coloane m ale lui U
[n, m] = size(U);

% Daca U nu este patratica, oprim executia cu un mesaj de eroare
if n ~= m
    error('Matricea U trebuie sa fie patratica.');
end

% Vectorul termenilor liberi trebuie sa aiba n componente
if length(b) ~= n
    error('Dimensiunea lui b nu se potriveste cu U.');
end

% tril(U,-1) pastreaza doar elementele de sub diagonala;
% daca vreunul este nenul, U nu este superior triunghiulara
if any(any(tril(U, -1) ~= 0))
    error('Matricea U nu este superior triunghiulara.');
end

% Alocam vectorul coloana al solutiei (n x 1), initial cu zerouri
x = zeros(n, 1);

% Calculam necunoscutele de la ultima la prima: x(n), x(n-1), ..., x(1)
for i = n:-1:1
    % Un element nul pe diagonala face imposibila impartirea
    if U(i,i) == 0
        error('U(%d,%d) = 0: sistemul nu are solutie unica.', i, i);
    end
    % s va contine suma U(i,i+1)*x(i+1) + ... + U(i,n)*x(n)
    s = 0;
    % Adunam contributiile necunoscutelor deja calculate
    for j = i+1:n
        s = s + U(i,j) * x(j);
    end
    % Din ecuatia i obtinem necunoscuta x(i)
    x(i) = (b(i) - s) / U(i,i);
end
end
```

**Scriptul de test** `test_triunghiulare.m` rezolvă exemplele de pe hârtie, calculează reziduurile și verifică mesajele de eroare. Construcția `try ... catch err ... end` execută instrucțiunile din `try`; dacă apare o eroare, execuția nu se oprește, ci continuă în `catch`, unde `err.message` conține mesajul erorii.

```
% TEST_TRIUNGHIULARE Verificarea functiilor subst_inferior si subst_superior
% pe exemplele calculate pe hartie.
clear; clc;

%% Sistem inferior triunghiular (solutia exacta: [1; 2; -1])
L = [2 0 0; 1 3 0; -1 2 4];
b = [2; 7; -1];
x = subst_inferior(L, b);
disp('x ='); disp(x);
% Reziduul norm(L*x - b) masoara cat de bine verifica x sistemul (ideal 0)
fprintf('norm(L*x - b) = %g\n', norm(L*x - b));
```

```
%% Sistem superior triunghiular (solutia exacta: [2; -1; 3])
U = [1 2 -1; 0 3 1; 0 0 -2];
c = [-3; 0; -6];
z = subst_superior(U, c);
disp('z ='); disp(z);
fprintf('norm(U*z - c) = %g\n', norm(U*z - c));

%% Comparatie cu operatorul \ al MATLAB
fprintf('diferenta fata de L\\b: %g\n', norm(x - L\b));

%% Verificarea mesajelor de eroare
try
    subst_inferior([1 2; 3 4], [1; 1]);   % matrice care nu e triunghiulara
catch err
    disp(err.message);
end
try
    subst_superior([1 2; 0 0], [1; 1]);   % element diagonal nul
catch err
    disp(err.message);
end
```

**Exerciții**

1. Rulați scriptul și comparați soluțiile cu cele obținute pe hârtie.
2. Înlocuiți bucla interioară din `subst_inferior` cu produsul scalar `s = L(i,1:i-1) * x(1:i-1);` și verificați că rezultatul nu se schimbă. Explicați de ce formula funcționează și pentru $i = 1$ (indicație: `1:0` este un vector gol, iar produsul unui vector $1 \times 0$ cu unul $0 \times 1$ este 0). Faceți aceeași modificare în `subst_superior`.
3. Ce se întâmplă dacă $b$ este dat ca vector linie? Ce dimensiune are soluția? De ce?
4. Adăugați la începutul ambelor funcții verificările: `L` și `b` sunt numerice (`isnumeric`), iar `b` este vector (`isvector`). Testați-le cu un șir de caractere.
5. Pentru $n \in \{500, 1000, 2000, 4000\}$ generați `L = tril(rand(n)) + n*eye(n)` și `b = rand(n,1)`. Măsurați cu `tic`/`toc` timpul lui `subst_inferior` și al operatorului `L\b`. De câte ori crește timpul când $n$ se dublează? Comparați cu costul de $n^2$ operații.
6. Folosind `subst_superior`, calculați inversa matricei $U$ de la punctul (b): coloana $j$ a lui $U^{-1}$ este soluția sistemului $Ux = e_j$, unde $e_j$ este coloana $j$ a matricei unitate (`I = eye(3); I(:,j)`). Verificați că $U^{-1}$ este tot superior triunghiulară.

**Rezultate de control.** (a) $x = (1, 2, -1)^T$; (b) $x = (2, -1, 3)^T$.

# Laboratorul 4

Eliminarea Gauss și factorizarea LU

**Obiective.** Rezolvăm pe hârtie un sistem prin eliminare Gauss, observăm că ponderile eliminării formează o matrice $L$ cu $A = LU$, implementăm factorizarea LU în MATLAB și apoi o calculăm direct, prin forma compactă Doolittle.

Notații: $n$ este dimensiunea sistemului, $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $I_n$ este matricea unitate de ordin $n$, iar $e_k$ este coloana $k$ a lui $I_n$. Prin $A_k = A(1{:}k, 1{:}k)$ notăm submatricea principală de ordin $k$ (primele $k$ linii și primele $k$ coloane ale lui $A$), iar $\det A_k$ este minorul principal de ordin $k$.

## 4.1 Eliminarea Gauss pe hârtie

**Ideea.** Transformăm sistemul $Ax = b$ într-un sistem echivalent cu matrice superior triunghiulară, pe care îl rezolvăm apoi prin substituție inversă (laboratorul 3). La pasul $k$ folosim ecuația $k$ (linia *pivot*) ca să eliminăm necunoscuta $x_k$ din ecuațiile $k+1, \ldots, n$: din linia $i$ scădem linia $k$ înmulțită cu *ponderea*

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad i = k+1, \ldots, n,$$

unde $a_{ij}^{(k)}$ sunt elementele sistemului obținut după primii $k-1$ pași, iar $a_{kk}^{(k)}$ este *pivotul*. Ponderea este aleasă astfel încât elementul de pe poziția $(i,k)$ să devină $a_{ik}^{(k)} - m_{ik}a_{kk}^{(k)} = 0$. Aceleași operații se fac și asupra termenilor liberi, deci lucrăm pe matricea extinsă $[A \,|\, b]$.

**Exemplu rezolvat.** Rezolvăm sistemul

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 2 & 5 & 3 & 3 \\ -1 & 1 & 4 & 6 \end{array}\right].$$

*Pasul 1* (pivot $a_{11} = 1$). Ponderile sunt $m_{21} = 2/1 = 2$ și $m_{31} = -1/1 = -1$, deci $L_2 \leftarrow L_2 - 2L_1$ și $L_3 \leftarrow L_3 + L_1$ (unde $L_i$ desemnează linia $i$ a matricei extinse):

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 7 \end{array}\right].$$

*Pasul 2* (pivot 1, pe poziția $(2,2)$). Ponderea este $m_{32} = 3/1 = 3$, deci $L_3 \leftarrow L_3 - 3L_2$:

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 4 \end{array}\right].$$

*Substituția inversă.* Din ultima ecuație $x_3 = 4/2 = 2$; din a doua $x_2 = 1 - x_3 = -1$; din prima $x_1 = 1 - 2x_2 - x_3 = 1 + 2 - 2 = 1$. Deci $x = (1, -1, 2)^T$, iar verificarea în sistemul inițial confirmă rezultatul.

**Observație.** Ponderile calculate, $m_{21} = 2$, $m_{31} = -1$, $m_{32} = 3$, nu sunt doar intermediare: așezate sub diagonala unei matrice cu 1 pe diagonală, ele dau

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 3 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix},$$

unde $U$ este matricea finală a eliminării. Verificați prin înmulțire că $LU = A$. Secțiunea următoare explică de ce.

## 4.2 De la eliminarea Gauss la factorizarea LU

Pasul $k$ al eliminării se scrie ca o înmulțire la stânga cu o matrice. Fie vectorul ponderilor $m^{(k)} = (0, \ldots, 0, m_{k+1,k}, \ldots, m_{nk})^T$ (primele $k$ componente sunt nule) și *matricea Gauss* (de ponderi)

$$M_k = I_n - m^{(k)} e_k^T.$$

Produsul $m^{(k)} e_k^T$ are vectorul $m^{(k)}$ pe coloana $k$ și zerouri în rest, deci $M_k$ diferă de $I_n$ numai prin elementele $-m_{ik}$ de sub diagonală, în coloana $k$. Înmulțirea $M_k A^{(k)}$ scade din fiecare linie $i > k$ linia $k$ înmulțită cu $m_{ik}$, adică exact operația de la pasul $k$:

$$A^{(1)} = A, \qquad A^{(k+1)} = M_k A^{(k)}, \qquad k = 1, \ldots, n-1.$$

După $n-1$ pași, $U = A^{(n)} = M_{n-1} \cdots M_1 A$ este superior triunghiulară. Matricele $M_k$ sunt inversabile, cu $M_k^{-1} = I_n + m^{(k)} e_k^T$ (deoarece $e_k^T m^{(k)} = 0$), deci

$$A = LU, \qquad L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}.$$

Se verifică direct că acest produs are 1 pe diagonală și $\ell_{ik} = m_{ik}$ pentru $i > k$: $L$ se construiește direct din ponderi, fără a calcula inverse.

**Existența.** Eliminarea se poate face fără schimbări de linii dacă toți pivoții $a_{kk}^{(k)}$, $k = 1, \ldots, n-1$, sunt nenuli. Deoarece operațiile de la primii $k$ pași nu schimbă minorul principal de ordin $k$, avem $\det A_k = a_{11}^{(1)} a_{22}^{(2)} \cdots a_{kk}^{(k)}$. Rezultă:

*Dacă* $\det A_k \neq 0$ *pentru* $k = 1, \ldots, n-1$*, atunci există o unică factorizare* $A = LU$*, cu* $L$ *inferior triunghiulară cu* $\ell_{kk} = 1$ *și* $U$ *superior triunghiulară.*

După factorizare, sistemul $Ax = b$ se scrie $L(Ux) = b$. Notând $y = Ux$, îl rezolvăm în doi pași: $Ly = b$ (substituție directă), apoi $Ux = y$ (substituție inversă).

### Calcul pe hârtie

Se consideră

$$A = \begin{pmatrix} 1 & 1 & 2 & -1 \\ 2 & 4 & 5 & -1 \\ -1 & 1 & 2 & 4 \\ 3 & -1 & 10 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} -3 \\ -5 \\ 5 \\ -7 \end{pmatrix}.$$

(a) Calculați ponderile pasului 1, scrieți matricea $M_1$ și calculați $A^{(2)} = M_1 A$.

(b) Repetați pentru pașii 2 și 3, obținând $M_2$, $A^{(3)}$, $M_3$ și $U = A^{(4)}$.

(c) Scrieți $L$ din ponderi și verificați prin înmulțire că $LU = A$.

(d) Calculați $\det A = u_{11} u_{22} u_{33} u_{44}$ și minorii principali $\det A_1, \det A_2, \det A_3$ din pivoți.

(e) Rezolvați $Ly = b$, apoi $Ux = y$, și verificați soluția.

## 4.3 Implementarea factorizării LU

Funcția `factorizare_LU` urmează exact construcția de mai sus. Ea verifică întâi că $A$ este pătratică și că primele $n-1$ submatrice principale sunt inversabile (prin determinanți), apoi construiește la fiecare pas vectorul ponderilor, matricea $M_k$ și produsul $M_k A^{(k)}$. Liniile marcate în comentarii corespund formulelor din secțiunea anterioară.

```
function [L, U] = factorizare_LU(A)
% FACTORIZARE_LU Factorizarea A = L*U (Doolittle), cu L inferior
% triunghiulara avand 1 pe diagonala si U superior triunghiulara.
%
% Factorizarea exista (si este unica) daca A este patratica si primele
% n-1 submatrice principale (diagonale) A(1:k,1:k), k = 1,...,n-1,
% sunt inversabile.
%
% La pasul k se calculeaza ponderile (multiplicatorii)
%   m(i) = A(i,k) / A(k,k), i = k+1,...,n,
% si matricea de ponderi (matricea Gauss)
%   M_k = I - m * e_k',
% cu care se inmulteste matricea de la pasul precedent:
%   A^(k+1) = M_k * A^(k).
% Dupa n-1 pasi, U = M_{n-1} * ... * M_1 * A, iar
%   L = inv(M_1) * ... * inv(M_{n-1})
% se construieste direct din ponderi: L(i,k) = m(i).

% ---- 1. Matricea trebuie sa fie patratica ----
% Numarul de linii n si numarul de coloane p ale lui A
[n, p] = size(A);
% Daca A nu este patratica, oprim executia
if n ~= p
    error('Matricea A trebuie sa fie patratica.');
end

% ---- 2. Primele n-1 submatrice principale trebuie sa fie inversabile ----
% Pragul sub care consideram determinantul egal cu zero
tol = 1e-12;
% Verificam pe rand submatricele A(1:1,1:1), A(1:2,1:2), ..., A(1:n-1,1:n-1)
for k = 1:n-1
    % A(1:k,1:k) este coltul stanga-sus de dimensiune k x k;
    % daca determinantul lui este (practic) zero, factorizarea nu exista
    if abs(det(A(1:k,1:k))) < tol
        error(['Submatricea principala A(1:%d,1:%d) nu este inversabila: ' ...
               'factorizarea LU nu exista.'], k, k);
    end
end

% ---- 3. Eliminarea Gauss cu matrice de ponderi ----
% L porneste ca matricea unitate (are deja 1 pe diagonala)
L = eye(n);
% U porneste ca A; la fiecare pas o transformam: U = A^(k)
U = A;
% Pasul k anuleaza elementele de sub diagonala din coloana k
for k = 1:n-1
    % Vectorul ponderilor m (n x 1); primele k componente raman 0
    m = zeros(n, 1);
    % Ponderea pentru fiecare linie i de sub pivotul U(k,k)
    for i = k+1:n
        m(i) = U(i,k) / U(k,k);
    end
    % ek = al k-lea vector din baza canonica (coloana k a matricei unitate)
    ek = zeros(n, 1);
    ek(k) = 1;
    % Matricea de ponderi de la pasul k: M_k = I - m*ek'
    % (m*ek' are vectorul m pe coloana k si zerouri in rest)
    Mk = eye(n) - m * ek';
    % Inmultim matricea de la pasul precedent: A^(k+1) = M_k * A^(k)
    U = Mk * U;
```

```
    % Punem zerouri exacte sub pivot, eliminand erorile de rotunjire
    U(k+1:n, k) = 0;
    % Constructia directa a lui L: coloana k primeste ponderile
    L(k+1:n, k) = m(k+1:n);
end
end
```

Câteva detalii de implementare:

- `error(['...' '...'], k, k)`: parantezele drepte concatenează două șiruri (mesajul este lung), iar `...` continuă instrucțiunea pe rândul următor;
- `U(k+1:n, k) = 0` pune zerouri exacte sub pivot: teoretic ele sunt deja 0, dar în aritmetica în virgulă mobilă pot rămâne valori de ordinul $10^{-16}$;
- la final, `L(k+1:n, k) = m(k+1:n)` scrie ponderile pe coloana $k$ a lui $L$.

**Scriptul de test** `test_LU.m` folosește și funcțiile `subst_inferior` și `subst_superior` din laboratorul 3 (copiați-le în directorul de lucru). Pe exemplul $A = \left(\begin{smallmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{smallmatrix}\right)$, $b = (4, 10, 24)^T$ se obține

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}, \qquad x = (1, 1, 1)^T.$$

```
% TEST_LU Verificarea functiilor factorizare_LU si doolittle_compact
% si rezolvarea sistemelor A*x = b.
clear; clc;
% Functiile din laboratoarele anterioare se afla in directoarele vecine
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));

%% Exemplul din anexa (solutia exacta este [1;1;1])
A = [2 1 1; 4 3 3; 8 7 9];
b = [4; 10; 24];
% Calculam factorizarea A = L*U
[L, U] = factorizare_LU(A);
disp('L ='); disp(L);
disp('U ='); disp(U);
% norm(L*U - A) masoara cat de bine reface produsul L*U matricea A (ideal 0)
fprintf('norm(L*U - A) = %g\n', norm(L*U - A));
% A*x = b <=> L*(U*x) = b. Notam y = U*x si rezolvam in doi pasi:
y = subst_inferior(L, b);      % pasul 1: L*y = b
x = subst_superior(U, y);      % pasul 2: U*x = y
disp('x ='); disp(x);
fprintf('norm(A*x - b) = %g\n', norm(A*x - b));

%% Exemplul 4x4 calculat pe hartie (solutia exacta este [1;0;-1;2])
A = [1 1 2 -1; 2 4 5 -1; -1 1 2 4; 3 -1 10 0];
b = [-3; -5; 5; -7];
[L1, U1] = factorizare_LU(A);
[L2, U2] = doolittle_compact(A);
% Factorizarea este unica, deci cele doua metode dau aceiasi factori
fprintf('diferenta intre factori: %g, %g\n', norm(L1 - L2), norm(U1 - U2));
x = subst_superior(U2, subst_inferior(L2, b));
disp('x ='); disp(x);

%% Matrice pentru care factorizarea LU nu exista: A(1,1) = 0
B = [0 1; 1 1];
try
    factorizare_LU(B);
```

```
catch err
    disp(err.message);
end
try
    doolittle_compact(B);
catch err
    disp(err.message);
end
```

## 4.4 Forma compactă Doolittle

**Motivație.** Funcția `factorizare_LU` construiește și înmulțește explicit matricele $M_k$; fiecare produs $M_k A^{(k)}$ costă aproximativ $2n^3$ operații, deci întreaga factorizare costă de ordinul $n^4$, deși majoritatea elementelor lui $M_k$ sunt 0 sau 1. În plus, verificarea prin determinanți adaugă încă de ordinul $n^4$ operații. Forma compactă obține elementele lui $L$ și $U$ direct din egalitatea $A = LU$, cu aproximativ $\frac{2}{3}n^3$ operații, și verifică existența pe parcurs, prin pivoți.

**Formulele.** Egalitatea $A = LU$ scrisă pe componente este $a_{ij} = \sum_{p=1}^{n} \ell_{ip} u_{pj}$. Fixăm linia $k$ și luăm $j \geq k$. Deoarece $\ell_{kp} = 0$ pentru $p > k$ și $\ell_{kk} = 1$, suma se oprește la $p = k$, iar ultimul termen este chiar $u_{kj}$. Izolându-l, obținem **linia $k$ a lui $U$**:

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} \ell_{kp}\, u_{pj}, \qquad j = k, k+1, \ldots, n. \tag{4.1}$$

Fixăm acum coloana $k$ și luăm $i > k$. Deoarece $u_{pk} = 0$ pentru $p > k$, suma din $a_{ik} = \sum_p \ell_{ip} u_{pk}$ se oprește la $p = k$, iar ultimul termen este $\ell_{ik} u_{kk}$. Izolând $\ell_{ik}$, obținem **coloana $k$ a lui $L$**:

$$\ell_{ik} = \frac{1}{u_{kk}} \Big( a_{ik} - \sum_{p=1}^{k-1} \ell_{ip}\, u_{pk} \Big), \qquad i = k+1, \ldots, n. \tag{4.2}$$

Ordinea calculului este: linia 1 a lui $U$, coloana 1 a lui $L$, linia 2 a lui $U$, coloana 2 a lui $L$ etc.; fiecare formulă folosește numai elemente calculate anterior. Pentru $k = 1$ sumele sunt vide: prima linie a lui $U$ este prima linie a lui $A$, iar $\ell_{i1} = a_{i1}/a_{11}$. Deoarece $\det A_k = u_{11} u_{22} \cdots u_{kk}$, un pivot $u_{kk} = 0$ cu $k < n$ înseamnă exact că minorul principal de ordin $k$ este nul: testul de existență se face gratuit, imediat după calculul pivotului.

### Calcul pe hârtie

Refaceți factorizarea matricei $A$ de la secțiunea 4.2 prin formulele (4.1)–(4.2), în ordinea indicată, scriind la fiecare pas formula și numerele înlocuite. Verificați că obțineți aceiași factori $L$ și $U$ (factorizarea este unică) și comparați numărul de operații făcute în cele două variante.

### Implementare

Funcția `doolittle_compact` începe cu verificarea parametrului de intrare: exact un argument (`nargin`), matrice numerică reală nevidă, pătratică, fără `NaN` sau `Inf`. Primul argument al lui `error`, de forma `'functie:motiv'`, este un identificator al erorii, util la testare (`err.identifier`). Pragul `tol = 1e-12*norm(A,inf)` este relativ la mărimea elementelor lui $A$, deci nu depinde de unitățile de măsură.

```
function [L, U] = doolittle_compact(A)
% DOOLITTLE_COMPACT Factorizarea A = L*U prin metoda compacta Doolittle:
% L inferior triunghiulara cu 1 pe diagonala, U superior triunghiulara.
%
```

```
% Elementele se obtin direct din egalitatea A = L*U, alternand:
%   linia k a lui U:   U(k,j) = A(k,j) - sum_{p<k} L(k,p)*U(p,j),   j = k,...,n,
%   coloana k a lui L: L(i,k) = ( A(i,k) - sum_{p<k} L(i,p)*U(p,k) ) / U(k,k),
%                      i = k+1,...,n.
% Conditia de existenta (minorii principali de ordin 1,...,n-1 nenuli) se
% verifica pe parcurs, deoarece det(A(1:k,1:k)) = U(1,1)*U(2,2)*...*U(k,k).

% ---- 1. Verificarea parametrului de intrare ----
% Functia primeste exact un argument
if nargin ~= 1
    error('doolittle_compact:nrArg', 'Functia primeste exact un argument.');
end
% A trebuie sa fie o matrice numerica reala, nevida
if ~isnumeric(A) || ~isreal(A) || isempty(A)
    error('doolittle_compact:tip', 'A trebuie sa fie o matrice numerica reala nevida.')
    ;
end
% A trebuie sa fie patratica
[n, p] = size(A);
if ~ismatrix(A) || n ~= p
    error('doolittle_compact:nepatratica', ...
          'A trebuie sa fie patratica (este %d x %d).', n, p);
end
% A nu poate contine NaN sau Inf
if ~all(isfinite(A(:)))
    error('doolittle_compact:nefinit', 'A contine NaN sau Inf.');
end

% ---- 2. Initializare ----
L = eye(n);                 % diagonala lui L este deja 1
U = zeros(n);
% Prag relativ: un pivot mai mic decat tol este considerat nul
tol = 1e-12 * norm(A, inf);

% ---- 3. Calculul alternat: linia k din U, apoi coloana k din L ----
for k = 1:n
    % Linia k a lui U (elementele U(k,k), ..., U(k,n))
    for j = k:n
        s = 0;
        for q = 1:k-1
            s = s + L(k,q) * U(q,j);
        end
        U(k,j) = A(k,j) - s;
    end
    % Daca nu suntem pe ultima linie, pivotul U(k,k) trebuie sa fie nenul:
    % altfel minorul principal de ordin k este nul si metoda se opreste
    if k < n && abs(U(k,k)) < tol
        error('doolittle_compact:pivot', ...
              ['Pivotul U(%d,%d) este (practic) nul: minorul principal ' ...
               'de ordin %d este nul.'], k, k, k);
    end
    % Coloana k a lui L (elementele L(k+1,k), ..., L(n,k))
    for i = k+1:n
        s = 0;
        for q = 1:k-1
            s = s + L(i,q) * U(q,k);
        end
        L(i,k) = (A(i,k) - s) / U(k,k);
    end
end
```

```
end
```

**Indicație (vectorizare).** Sumele din (4.1)–(4.2) se pot scrie ca produse matriciale, eliminând buclele după `j`, `i` și `q`:

```
U(k,k:n)   = A(k,k:n) - L(k,1:k-1)*U(1:k-1,k:n);
L(k+1:n,k) = (A(k+1:n,k) - L(k+1:n,1:k-1)*U(1:k-1,k)) / U(k,k);
```

Pentru $k = 1$, `L(1,1:0)` are dimensiunea $1 \times 0$, iar produsul cu o matrice $0 \times n$ este matricea nulă $1 \times n$, deci formula funcționează fără caz separat.

### Testare și durata de execuție

Scriptul `timp_LU.m` măsoară timpul celor două implementări și al funcției MATLAB `lu`, pe matrice strict diagonal dominante `A = rand(n) + n*eye(n)` (pentru ele toți minorii principali sunt nenuli). Dreapta de regresie a lui $\log t$ în funcție de $\log n$, calculată cu `polyfit`, are panta $c$ dacă $t \approx Cn^c$.

```
% TIMP_LU Durata factorizarii LU in functie de dimensiunea n.
% factorizare_LU inmulteste explicit cu matricele M_k (cost ~ n^3 pe pas),
% de aceea o testam pe dimensiuni mai mici decat forma compacta.
clear; clc; close all;

%% 1. factorizare_LU (cu matrice de ponderi M_k)
ng = [25 50 100 200];
t_gauss = zeros(size(ng));
for s = 1:numel(ng)
    n = ng(s);
    % Matrice strict diagonal dominanta: toti minorii principali sunt nenuli
    A = rand(n) + n*eye(n);
    tic; [L, U] = factorizare_LU(A); t_gauss(s) = toc;
end

%% 2. doolittle_compact si functia MATLAB lu
nc = [50 100 200 400 800];
t_comp = zeros(size(nc));
t_lu   = zeros(size(nc));
for s = 1:numel(nc)
    n = nc(s);
    A = rand(n) + n*eye(n);
    tic; [L, U] = doolittle_compact(A); t_comp(s) = toc;
    tic; [Lm, Um, P] = lu(A);           t_lu(s)   = toc;
    fprintf('n = %4d: compact %8.4f s, lu %8.5f s, eroare relativa %.1e\n', ...
            n, t_comp(s), t_lu(s), norm(A - L*U, inf)/norm(A, inf));
end

%% 3. Grafic in scara dublu logaritmica si pantele
loglog(ng, t_gauss, 'o-', nc, t_comp, 's-', nc, t_lu, 'd-', 'LineWidth', 1.5);
grid on; xlabel('n'); ylabel('timp (s)');
legend({'factorizare_LU', 'doolittle_compact', 'lu'}, ...
       'Interpreter', 'none', 'Location', 'northwest');
title('Durata factorizarii LU');

% Panta dreptei log(t) = c(1)*log(n) + c(2) estimeaza exponentul din O(n^c(1))
cg = polyfit(log(ng(2:end)), log(t_gauss(2:end)), 1);
cc = polyfit(log(nc(2:end)), log(t_comp(2:end)), 1);
fprintf('panta factorizare_LU: %.2f, panta doolittle_compact: %.2f\n', cg(1), cc(1));
```

**Atenție:** `[L,U,P] = lu(A)` face factorizarea cu pivotare parțială (laboratorul 6), $PA = LU$; factorii ei diferă, în general, de cei Doolittle, deci comparăm timpul și reziduul, nu factorii.

### 4.5 Exerciții

1. Rulați `test_LU.m` și comparați factorii cu cei obținuți pe hârtie.
2. Rulați `timp_LU.m`. Ce pante obțineți? Explicați de ce panta pentru `factorizare_LU` este apropiată de 4, iar pentru `doolittle_compact` de 3. De ce `lu` este mult mai rapidă, deși are același ordin de complexitate ca forma compactă?
3. Modificați `factorizare_LU` astfel încât să nu mai construiască $M_k$: la pasul $k$, pentru fiecare $i > k$, actualizați direct `U(i,k:n) = U(i,k:n) - m(i)*U(k,k:n)`. Înlocuiți verificarea prin determinanți cu testul pivotului. Remăsurați timpul.
4. Înlocuiți buclele din `doolittle_compact` cu forma vectorizată din indicație și comparați timpii.
5. Verificați mesajele de eroare ale lui `doolittle_compact` pentru: o matrice $3 \times 4$; un șir de caractere; o matrice care conține `NaN`; $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, care este inversabilă, dar are $\det A_1 = 0$.
6. Scrieți funcția `x = rezolva_LU(A, b)` care verifică $b$ (vector numeric cu `size(A,1)` componente), apelează `doolittle_compact` și cele două substituții. Testați-o pe exemplul de la 4.2.

**Rezultate de control (secțiunea 4.2).**

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 3 & -2 & 2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 1 & 2 & -1 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad y = \begin{pmatrix} -3 \\ 1 \\ 1 \\ 2 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 2 \end{pmatrix},$$

$\det A_1 = 1$, $\det A_2 = 2$, $\det A_3 = 6$, $\det A = 6$.

# Laboratorul 5
## Factorizarea Cholesky

**Obiective.** Verificăm pe exemple condițiile în care se aplică factorizarea Cholesky, o calculăm pe hârtie pentru o matrice $4 \times 4$, apoi o implementăm și o folosim la rezolvarea sistemelor liniare.

Notații: $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, $A^T$ este transpusa lui $A$, $A_k = A(1{:}k, 1{:}k)$ este submatricea principală de ordin $k$, iar $L = (\ell_{ij})$ este o matrice inferior triunghiulară.

### 5.1 Breviar teoretic

**Condițiile de aplicare.** Matricea $A$ este *simetrică* dacă $A = A^T$, adică $a_{ij} = a_{ji}$, și *pozitiv definită* dacă

$$v^T A v = \sum_{i,j=1}^{n} a_{ij} v_i v_j > 0 \quad \text{pentru orice } v \in \mathbb{R}^n,\ v \neq 0.$$

Verificarea directă a definiției este dificilă; în practică folosim următoarele criterii.

- *Criteriul lui Sylvester:* o matrice simetrică este pozitiv definită dacă și numai dacă $\det A_k > 0$ pentru $k = 1, \ldots, n$.
- *Condiție necesară:* dacă $A$ este pozitiv definită, atunci $a_{kk} = e_k^T A e_k > 0$ pentru orice $k$, unde $e_k$ este coloana $k$ a matricei unitate. Un singur element diagonal $\leq 0$ arată deci că $A$ *nu* este pozitiv definită; reciproca este falsă.
- *Condiție suficientă:* o matrice simetrică, cu elemente diagonale pozitive și strict diagonal dominantă ($|a_{ii}| > \sum_{j\neq i} |a_{ij}|$ pentru orice $i$) este pozitiv definită.

**Factorizarea.** *Dacă $A$ este simetrică și pozitiv definită, atunci există o unică matrice $L$ inferior triunghiulară, cu $\ell_{kk} > 0$, astfel încât $A = LL^T$.* Reciproc, dacă $A = LL^T$ cu $L$ inversabilă, atunci $A$ este simetrică și $v^T A v = \|L^T v\|^2 > 0$ pentru $v \neq 0$.

**De unde provin formulele.** Componenta $(i, k)$ a lui $LL^T$ este $\sum_p \ell_{ip}\ell_{kp}$, iar suma se oprește la $p = \min\{i, k\}$, deoarece $L$ este inferior triunghiulară. Pentru $i = k$, ultimul termen este $\ell_{kk}^2$; izolându-l și luând rădăcina pozitivă, obținem **elementul diagonal**

$$\ell_{kk} = \sqrt{a_{kk} - \sum_{p=1}^{k-1} \ell_{kp}^2}\,. \tag{5.1}$$

Pentru $i > k$, ultimul termen este $\ell_{ik}\ell_{kk}$, deci **restul coloanei** $k$ este

$$\ell_{ik} = \frac{1}{\ell_{kk}}\Big(a_{ik} - \sum_{p=1}^{k-1} \ell_{ip}\,\ell_{kp}\Big), \qquad i = k+1, \ldots, n. \tag{5.2}$$

Matricea $L$ se calculează coloană cu coloană. Deoarece $\det A_k = (\ell_{11}\ell_{22}\cdots\ell_{kk})^2$, expresia de sub radical din (5.1) este strict pozitivă la toți pașii exact atunci când toți minorii principali sunt strict pozitivi. Prin urmare, algoritmul însuși verifică pozitivitatea: dacă la un pas expresia de sub radical este $\leq 0$, matricea nu este pozitiv definită și metoda se oprește.

**Costul și rezolvarea sistemelor.** Factorizarea folosește numai triunghiul inferior al lui $A$ și costă aproximativ $\frac{1}{3}n^3$ operații, jumătate din costul factorizării LU. Sistemul $Ax = b$ devine $L(L^T x) = b$ și se rezolvă prin $Ly = b$ (substituție directă), apoi $L^T x = y$ (substituție inversă), cu funcțiile din laboratorul 3.

### 5.2 Exercițiu: verificarea condițiilor de aplicare

Pentru fiecare dintre matricele de mai jos stabiliți, pe hârtie, dacă se poate aplica factorizarea Cholesky. Verificați întâi simetria, apoi semnul elementelor diagonale, apoi minorii principali (criteriul lui Sylvester). Acolo unde factorizarea există, calculați $L$.

$$M_1 = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix},$$

$$M_4 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}, \quad M_5 = \begin{pmatrix} 4 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 5 \end{pmatrix}, \quad M_6 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{pmatrix}.$$

Pentru $M_2$ și $M_5$ începeți factorizarea cu formulele (5.1)–(5.2): la ce pas și de ce se oprește? Ce legătură există între valoarea de sub radical și minorul principal corespunzător? Matricea $M_3$ va reapărea în laboratorul 6, la deformarea unei bare elastice.

### 5.3 Calcul pe hârtie

Se consideră

$$A = \begin{pmatrix} 4 & 2 & -2 & 2 \\ 2 & 10 & 5 & 1 \\ -2 & 5 & 6 & -2 \\ 2 & 1 & -2 & 6 \end{pmatrix}, \qquad b = \begin{pmatrix} 8 \\ -2 \\ -10 \\ 10 \end{pmatrix}.$$

(a) Verificați că $A$ este simetrică și are elementele diagonale pozitive.

(b) Calculați coloanele lui $L$, pentru $k = 1, 2, 3, 4$, cu (5.1) și (5.2). Notați la fiecare pas valoarea de sub radical și verificați că este pozitivă.

(c) Verificați prin înmulțire că $LL^T = A$.

(d) Deduceți minorii principali din relația $\det A_k = (\ell_{11} \cdots \ell_{kk})^2$ și verificați criteriul lui Sylvester.

(e) Rezolvați $Ly = b$, apoi $L^T x = y$, și verificați soluția.

### 5.4 Implementare

**Funcția `cholesky_fact`.** Verificările de la început corespund condițiilor din breviar, de la cele mai ieftine la cele mai scumpe:

1. un singur argument, matrice numerică reală nevidă, pătratică, fără `NaN`/`Inf`;
2. simetria, testată prin $\|A - A^T\|_\infty \le \text{tol}$, unde $\text{tol} = 10^{-12}\|A\|_\infty$ este un prag relativ. Nu folosim `isequal(A, A')`, deoarece o matrice simetrică „teoretic", obținută prin calcule, poate diferi de transpusa ei prin erori de rotunjire de ordinul $10^{-16}$;
3. condiția necesară $a_{kk} > 0$;
4. pozitivitatea propriu-zisă, verificată *în timpul* factorizării, prin semnul expresiei de sub radical. Nu calculăm $\det A_k$ pentru fiecare $k$: ar costa de ordinul $n^4$ operații, mai mult decât factorizarea însăși.

În calculul lui $L$, sumele din (5.1)–(5.2) se scriu `sum(L(k,1:k-1).^2)` și `sum(L(i,1:k-1) .* L(k,1:k-1))`; pentru $k = 1$, `1:0` este vid, iar `sum` de un vector vid este 0.

```
function L = cholesky_fact(A)
% CHOLESKY_FACT Factorizarea Cholesky A = L*L', cu L inferior triunghiulara
% avand elemente diagonale strict pozitive.
```

```
%
% Factorizarea exista (si este unica) daca si numai daca A este simetrica
% si pozitiv definita. Coloana k a lui L se calculeaza din egalitatea A = L*L':
%   L(k,k) = sqrt( A(k,k) - sum_{p<k} L(k,p)^2 ),
%   L(i,k) = ( A(i,k) - sum_{p<k} L(i,p)*L(k,p) ) / L(k,k),  i = k+1,...,n.
% Daca expresia de sub radical este <= 0, A nu este pozitiv definita.

% ---- 1. Verificarea parametrului de intrare ----
% Functia primeste exact un argument
if nargin ~= 1
    error('cholesky_fact:nrArg', 'Functia primeste exact un argument.');
end
% A trebuie sa fie o matrice numerica reala, nevida
if ~isnumeric(A) || ~isreal(A) || isempty(A)
    error('cholesky_fact:tip', 'A trebuie sa fie o matrice numerica reala nevida.');
end
% A trebuie sa fie patratica
[n, p] = size(A);
if ~ismatrix(A) || n ~= p
    error('cholesky_fact:nepatratica', ...
          'A trebuie sa fie patratica (este %d x %d).', n, p);
end
% A nu poate contine NaN sau Inf
if ~all(isfinite(A(:)))
    error('cholesky_fact:nefinit', 'A contine NaN sau Inf.');
end
% Prag relativ sub care un numar este considerat nul
tol = 1e-12 * norm(A, inf);
% A trebuie sa fie simetrica (pana la erori de rotunjire)
if norm(A - A', inf) > tol
    error('cholesky_fact:nesimetrica', 'A nu este simetrica.');
end
% Conditie necesara (ieftina) de pozitivitate: A(k,k) = e_k'*A*e_k > 0
if any(diag(A) <= 0)
    error('cholesky_fact:diagonala', ...
          'A are un element diagonal <= 0, deci nu este pozitiv definita.');
end

% ---- 2. Calculul lui L, coloana cu coloana ----
L = zeros(n);
for k = 1:n
    % Expresia de sub radical: A(k,k) - (L(k,1)^2 + ... + L(k,k-1)^2)
    s = A(k,k) - sum(L(k,1:k-1).^2);
    % Daca nu este strict pozitiva, minorul principal de ordin k este <= 0
    if s <= tol
        error('cholesky_fact:nepozitiva', ...
              ['La pasul %d expresia de sub radical este %g <= 0: ' ...
               'A nu este pozitiv definita.'], k, s);
    end
    L(k,k) = sqrt(s);
    % Elementele de sub diagonala din coloana k
    for i = k+1:n
        L(i,k) = (A(i,k) - sum(L(i,1:k-1) .* L(k,1:k-1))) / L(k,k);
    end
end
end
```

**Funcția `rezolva_cholesky`.** Verifică vectorul $b$ (verificările asupra lui $A$ le face `cholesky_fact`), apoi rezolvă cele două sisteme triunghiulare. Instrucțiunea `b = b(:)` transformă $b$ în vector coloană, indiferent cum a fost dat.

```
function x = rezolva_cholesky(A, b)
% REZOLVA_CHOLESKY Rezolva sistemul A*x = b, cu A simetrica si pozitiv
% definita, folosind factorizarea A = L*L':
%   L*y = b   (substitutie directa),
%   L'*x = y  (substitutie inversa).

% ---- 1. Verificarea parametrilor de intrare ----
if nargin ~= 2
    error('rezolva_cholesky:nrArg', 'Functia primeste exact doua argumente.');
end
% b trebuie sa fie un vector numeric real, cu elemente finite
if ~isnumeric(b) || ~isreal(b) || ~isvector(b) || ~all(isfinite(b))
    error('rezolva_cholesky:b', 'b trebuie sa fie un vector numeric real, finit.');
end
% Numarul de componente ale lui b trebuie sa fie egal cu numarul de linii ale lui A
if numel(b) ~= size(A, 1)
    error('rezolva_cholesky:dim', 'Dimensiunea lui b nu se potriveste cu A.');
end
% Lucram cu b ca vector coloana
b = b(:);

% ---- 2. Factorizarea (cholesky_fact verifica proprietatile lui A) ----
L = cholesky_fact(A);

% ---- 3. Cele doua sisteme triunghiulare ----
y = subst_inferior(L, b);
x = subst_superior(L', y);
end
```

**Apelarea.** Scriptul `test_cholesky.m` rezolvă exemplul de pe hârtie, compară cu funcția MATLAB `chol`, încearcă factorizarea matricelor $M_1, \ldots, M_5$ (afișând mesajul de eroare acolo unde ea nu există) și compară două moduri de a rezolva $m$ sisteme cu aceeași matrice. Funcția `chol(A)` întoarce matricea *superior* triunghiulară $R = L^T$, cu $A = R^T R$, iar `chol(A,'lower')` întoarce $L$. Matricea `A = B'*B + n*eye(n)` este simetrică și pozitiv definită, deoarece $v^T A v = \|Bv\|^2 + n\|v\|^2 > 0$ pentru $v \neq 0$. În `mat2str(v, 4)`, vectorul $v$ este transformat în text cu 4 cifre semnificative.

```
% TEST_CHOLESKY Verificarea functiilor cholesky_fact si rezolva_cholesky.
clear; clc;
% Functiile din laboratoarele anterioare se afla in directoarele vecine
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));
addpath(fullfile(aici, '..', 'lab4'));

%% Exemplul calculat pe hartie (solutia exacta este [1; 0; -1; 1])
A = [4 2 -2 2; 2 10 5 1; -2 5 6 -2; 2 1 -2 6];
b = [8; -2; -10; 10];
L = cholesky_fact(A);
disp('L ='); disp(L);
% Comparatie cu functia MATLAB chol (care intoarce R = L')
fprintf('norm(L*L'' - A) = %g, diferenta fata de chol: %g\n', ...
        norm(L*L' - A), norm(L - chol(A, 'lower')));
x = rezolva_cholesky(A, b);
disp('x ='); disp(x);
fprintf('norm(A*x - b) = %g\n', norm(A*x - b));
```

```
%% Matricele din exercitiul de verificare a conditiilor
M = { [4 2; 1 3], ...                    % nesimetrica
      [1 2 0; 2 3 1; 0 1 2], ...         % simetrica, minorul de ordin 2 negativ
      [2 -1 0; -1 2 -1; 0 -1 2], ...     % simetrica si pozitiv definita
      [1 1 1; 1 2 2; 1 2 3], ...         % simetrica si pozitiv definita
      [4 2 0; 2 1 0; 0 0 5] };           % simetrica, minorul de ordin 2 nul
for k = 1:numel(M)
    try
        Lk = cholesky_fact(M{k});
        fprintf('Matricea %d: factorizarea exista, diag(L) = %s\n', k, mat2str(diag(Lk)
    ', 4));
    catch err
        fprintf('Matricea %d: %s\n', k, err.message);
    end
end

%% Mai multi termeni liberi: factorizam o singura data
n = 400; m = 50;
B = rand(n); A = B'*B + n*eye(n);      % simetrica si pozitiv definita
T = rand(n, m);                         % m termeni liberi, pe coloane
tic;                                     % varianta (a): factorizare de m ori
for j = 1:m
    x = rezolva_cholesky(A, T(:,j));
end
ta = toc;
tic;                                     % varianta (b): o singura factorizare
L = cholesky_fact(A);
for j = 1:m
    x = subst_superior(L', subst_inferior(L, T(:,j)));
end
tb = toc;
fprintf('(a) %.3f s, (b) %.3f s, raport %.1f\n', ta, tb, ta/tb);
```

## 5.5 Exerciții

1. Rulați `test_cholesky.m` și comparați $L$ și $x$ cu rezultatele de pe hârtie, iar mesajele pentru $M_1, \ldots, M_5$ cu concluziile exercițiului 5.2.
2. Adăugați matricea $M_6$ în lista din script. Care verificare o respinge?
3. Pentru $n \in \{100, 200, 400, 800\}$ generați matrice simetrice și pozitiv definite ca în script și măsurați timpii lui `cholesky_fact` și `doolittle_compact` (laboratorul 4) pe aceleași matrice. Este raportul timpilor apropiat de 2? Reprezentați timpii cu `loglog`.
4. Explicați raportul timpilor obținut în script pentru variantele (a) și (b), folosind costurile: $\frac{1}{3}n^3$ pentru factorizare și $n^2$ pentru fiecare substituție.
5. Verificați că, pentru $L$ calculat de `cholesky_fact`, `prod(diag(L))^2` este egal cu `det(A)`. De ce este aceasta o metodă bună de a calcula determinantul unei matrice simetrice și pozitiv definite?
6. Scrieți funcția `este_pozitiv_definita(A)` care întoarce `true` sau `false` fără să oprească execuția, apelând `cholesky_fact` în interiorul unui bloc `try ... catch`. Testați-o pe $M_1, \ldots, M_6$.

**Rezultate de control.** Exercițiul 5.2: $M_1$ nu este simetrică; $M_2$: $\det A_2 = -1 < 0$, metoda se oprește la pasul 2 (sub radical $-1$); $M_3$: minorii $2, 3, 4$, $\operatorname{diag} L = (\sqrt{2}, \sqrt{3/2}, \sqrt{4/3})$; $M_4$: minorii

$1, 1, 1$, $L$ are toate elementele de pe diagonală și de sub ea egale cu 1; $M_5$: $\det A_2 = 0$, metoda se oprește la pasul 2 (sub radical 0); $M_6$: element diagonal negativ. Secțiunea 5.3:

$$L = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ -1 & 2 & 1 & 0 \\ 1 & 0 & -1 & 2 \end{pmatrix}, \qquad y = \begin{pmatrix} 4 \\ -2 \\ -2 \\ 2 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \end{pmatrix},$$

minorii principali 4, 36, 36, 144.

# Laboratorul 6

Factorizarea LU cu pivotare parțială. Inversa unei matrice. Aplicație: bara elastică

**Obiective.** Calculăm pe hârtie factorizarea LU cu pivotare parțială, o implementăm în MATLAB și o folosim la rezolvarea sistemelor și la calculul inversei. În final, folosim metodele din laboratoarele 3–6 pentru a simula deformarea unei bare elastice.

Notații: $A \in \mathbb{R}^{n\times n}$, $I_n$ este matricea unitate de ordin $n$, $e_j$ este coloana $j$ a lui $I_n$, iar $A^{(k)} = (a_{ij}^{(k)})$ este matricea obținută după primii $k-1$ pași ai eliminării (ca în laboratorul 4).

## 6.1 De ce este nevoie de pivotare

Eliminarea Gauss fără schimbări de linii (laboratorul 4) cere pivoți nenuli. Două exemple arată că această cerință nu este suficientă în practică.

- *Pivot nul.* Pentru $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ avem $\det A_1 = 0$, deci factorizarea $A = LU$ nu există, deși $A$ este inversabilă. Dacă însă schimbăm cele două ecuații între ele, matricea devine $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$, deja triunghiulară.
- *Pivot foarte mic.* Pentru $A = \left(\begin{smallmatrix} \varepsilon & 1 \\ 1 & 1 \end{smallmatrix}\right)$, $b = (1,2)^T$ și $\varepsilon = 10^{-17}$, soluția exactă este $x \approx (1,1)^T$. Fără schimbare de linii, ponderea este $1/\varepsilon = 10^{17}$, iar în aritmetica în virgulă mobilă (cu aproximativ 16 cifre semnificative) $1 - 10^{17}$ și $2 - 10^{17}$ se rotunjesc amândouă la $-10^{17}$. Se obține $x_2 = 1$, apoi $x_1 = (1 - x_2)/\varepsilon = 0$, complet greșit. Dacă schimbăm întâi liniile, ponderea este $\varepsilon$, iar rezultatul este corect.

În ambele cazuri problema este o pondere mare (sau infinită). *Pivotarea parțială* alege la fiecare pas ca pivot elementul de modul maxim din coloana curentă, de pe linia pivotului în jos; atunci toate ponderile verifică $|m_{ik}| \le 1$.

## 6.2 Algoritmul și calculul pe hârtie

O *matrice de permutare* $P$ se obține rearanjând liniile lui $I_n$; produsul $PA$ rearanjează în același mod liniile lui $A$. La pasul $k$, $k = 1, \ldots, n-1$:

1. alegem linia pivotului $q \ge k$ astfel încât $|a_{qk}^{(k)}| = \max_{k\le i\le n} |a_{ik}^{(k)}|$; dacă acest maxim este 0, matricea $A$ este singulară;
2. interschimbăm liniile $k$ și $q$ ale matricei curente;
3. calculăm ponderile $m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$, $i > k$, și eliminăm ca în laboratorul 4.

*Pentru orice matrice inversabilă $A$ există o matrice de permutare $P$, o matrice $L$ inferior triunghiulară cu $\ell_{kk} = 1$ și $|\ell_{ik}| \le 1$ și o matrice $U$ superior triunghiulară, astfel încât $PA = LU$.* Matricea $P$ înregistrează toate interschimbările de linii.

**Ce se întâmplă cu ponderile deja calculate.** Ponderea $m_{ik}$ aparține ecuației $i$: ea spune de câte ori a fost scăzută linia pivot din ecuația $i$. Când, la un pas ulterior, ecuația $i$ își schimbă poziția, ponderile ei calculate anterior (coloanele $1, \ldots, k-1$ ale lui $L$) se mută odată cu ea. De aceea, la pasul $k$ interschimbăm în $L$ numai elementele din coloanele $1, \ldots, k-1$.

**Rezolvarea sistemelor.** Din $PA = LU$ rezultă $Ax = b \iff PAx = Pb \iff L(Ux) = Pb$, deci rezolvăm $Ly = Pb$ și apoi $Ux = y$. Termenii liberi se permută la fel ca ecuațiile.

### Calcul pe hârtie

Se consideră

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 4 & -6 & 0 \\ -2 & 11 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ -2 \\ 11 \end{pmatrix}.$$

(a) *Pasul 1.* Găsiți pivotul din coloana 1, interschimbați liniile, calculați ponderile și eliminați. Notați permutarea făcută.

(b) *Pasul 2.* Găsiți pivotul din coloana 2 (liniile 2 și 3). Interschimbați liniile în matricea curentă și ponderile pasului 1 în $L$, apoi eliminați.

(c) Scrieți $P$, $L$ și $U$ și verificați prin înmulțire că $PA = LU$. Verificați că $|\ell_{ik}| \leq 1$.

(d) Calculați $\det A$. Indicație: $\det P = \pm 1$, după cum numărul de interschimbări este par sau impar, iar $\det L = 1$.

(e) Rezolvați $Ly = Pb$, apoi $Ux = y$.

(f) Calculați prima coloană a lui $A^{-1}$, rezolvând $Ax = e_1$ cu aceiași factori.

## 6.3 Implementare în MATLAB

**Funcția `lu_pivotare`.** După verificarea parametrului de intrare (aceleași condiții ca la `doolittle_compact`: pătratică, numerică, reală, finită), funcția pornește cu $U = A$, $L = I_n$, $P = I_n$ și execută cei trei pași ai algoritmului:

- `[vmax, r] = max(abs(U(k:n,k)))` întoarce valoarea maximă `vmax` a vectorului $|U(k{:}n, k)|$ și poziția ei `r` *în acest vector*. Vectorul începe la linia $k$, deci linia pivotului în $U$ este $q = r + k - 1$;
- `U([k q], :) = U([q k], :)` interschimbă liniile $k$ și $q$: în stânga, liniile $k$ și $q$ primesc, în această ordine, liniile $q$ și $k$ din dreapta. Aceeași interschimbare se face în $P$ și, numai pe coloanele `1:k-1`, în $L$;
- eliminarea actualizează linia $i$ numai pe coloanele $k, \ldots, n$, deoarece pe coloanele anterioare linia pivot are zerouri.

Factorizarea $PA = LU$ există și pentru matrice singulare, dar funcția este folosită la rezolvări de sisteme, de aceea se oprește cu mesaj de eroare dacă un pivot (inclusiv $u_{nn}$) este practic nul.

```
function [L, U, P] = lu_pivotare(A)
% LU_PIVOTARE Factorizarea LU cu pivotare partiala: P*A = L*U, unde
%   P este o matrice de permutare (liniile lui I, rearanjate),
%   L este inferior triunghiulara, cu 1 pe diagonala si |L(i,k)| <= 1,
%   U este superior triunghiulara.
%
% La pasul k:
%   1) alegem ca pivot elementul de modul maxim din coloana k, liniile k..n;
%   2) interschimbam linia pivotului cu linia k in U, in P si in partea
%      deja calculata a lui L (coloanele 1..k-1);
%   3) eliminam elementele de sub pivot cu ponderile L(i,k) = U(i,k)/U(k,k).
% Factorizarea exista pentru orice matrice patratica; functia se opreste
% daca A este (numeric) singulara, deoarece este folosita la rezolvari.

% ---- 1. Verificarea parametrului de intrare ----
if nargin ~= 1
    error('lu_pivotare:nrArg', 'Functia primeste exact un argument.');
end
if ~isnumeric(A) || ~isreal(A) || isempty(A)
    error('lu_pivotare:tip', 'A trebuie sa fie o matrice numerica reala nevida.');
```

```
end
[n, p] = size(A);
if ~ismatrix(A) || n ~= p
    error('lu_pivotare:nepatratica', 'A trebuie sa fie patratica (este %d x %d).', n, p
    );
end
if ~all(isfinite(A(:)))
    error('lu_pivotare:nefinit', 'A contine NaN sau Inf.');
end

% ---- 2. Initializare ----
U = A;                          % U va deveni superior triunghiulara
L = eye(n);                     % ponderile se scriu sub diagonala lui L
P = eye(n);                     % P inregistreaza interschimbarile de linii
tol = 1e-12 * norm(A, inf);  % prag relativ pentru pivot nul

% ---- 3. Eliminarea Gauss cu pivotare partiala ----
for k = 1:n-1
    % 3.1 Cautam pivotul: [vmax, r] = max(v) da valoarea maxima din v si
    % pozitia ei r in v; v = abs(U(k:n,k)) incepe la linia k, deci linia
    % pivotului in U este q = r + k - 1
    [vmax, r] = max(abs(U(k:n, k)));
    q = r + k - 1;
    % Daca toata coloana (de la linia k in jos) este nula, A este singulara
    if vmax < tol
        error('lu_pivotare:singulara', ...
              'A este (numeric) singulara: coloana %d nu are pivot nenul.', k);
    end
    % 3.2 Interschimbam liniile k si q (daca pivotul nu este deja pe linia k)
    if q ~= k
        U([k q], :)     = U([q k], :);
        P([k q], :)     = P([q k], :);
        L([k q], 1:k-1) = L([q k], 1:k-1);   % doar ponderile deja calculate
    end
    % 3.3 Eliminarea: linia i devine linia i - L(i,k) * linia k
    for i = k+1:n
        L(i,k) = U(i,k) / U(k,k);
        U(i,k:n) = U(i,k:n) - L(i,k) * U(k,k:n);
    end
    % Zerouri exacte sub pivot (eliminam erorile de rotunjire)
    U(k+1:n, k) = 0;
end
% Ultimul pivot nu a fost testat in bucla
if abs(U(n,n)) < tol
    error('lu_pivotare:singulara', 'A este (numeric) singulara: U(%d,%d) = 0.', n, n);
end
end
```

**Rezolvarea sistemelor.** Funcția `rezolva_lu_pivotare` verifică vectorul $b$ și aplică $Ly = Pb$, $Ux = y$, cu funcțiile `subst_inferior` și `subst_superior` din laboratorul 3:

```
function x = rezolva_lu_pivotare(A, b)
% REZOLVA_LU_PIVOTARE Rezolva A*x = b folosind P*A = L*U:
%   A*x = b  <=>  P*A*x = P*b  <=>  L*(U*x) = P*b,
% deci L*y = P*b (substitutie directa), apoi U*x = y (substitutie inversa).

if nargin ~= 2
    error('rezolva_lu_pivotare:nrArg', 'Functia primeste exact doua argumente.');
end
```

```
if ~isnumeric(b) || ~isreal(b) || ~isvector(b) || ~all(isfinite(b))
    error('rezolva_lu_pivotare:b', 'b trebuie sa fie un vector numeric real, finit.');
end
if numel(b) ~= size(A, 1)
    error('rezolva_lu_pivotare:dim', 'Dimensiunea lui b nu se potriveste cu A.');
end
b = b(:);

[L, U, P] = lu_pivotare(A);      % verifica proprietatile lui A
y = subst_inferior(L, P*b);
x = subst_superior(U, y);
end
```

## 6.4 Calculul inversei

Coloana $j$ a inversei, $X(:, j)$, verifică $AX(:, j) = e_j$, deoarece $AX = I_n$. Prin urmare inversa se obține rezolvând $n$ sisteme cu aceeași matrice $A$ și termenii liberi $e_1, \ldots, e_n$. Factorizarea se face o singură dată (aproximativ $\frac{2}{3}n^3$ operații), iar pentru fiecare coloană se rezolvă $Ly = Pe_j$ și $UX(:, j) = y$ (aproximativ $2n^2$ operații); costul total este de ordinul $\frac{8}{3}n^3$ operații. Observăm că $Pe_j$ este coloana $j$ a lui $P$.

```
function X = inversa_lu(A)
% INVERSA_LU Calculeaza inversa lui A folosind P*A = L*U.
% Coloana j a inversei, X(:,j), este solutia sistemului A*X(:,j) = e_j,
% unde e_j este coloana j a matricei unitate. Factorizarea se face o
% singura data; pentru fiecare coloana se rezolva doar doua sisteme
% triunghiulare: L*y = P*e_j si U*X(:,j) = y. Observam ca P*e_j = P(:,j).

[L, U, P] = lu_pivotare(A);      % verifica proprietatile lui A
n = size(A, 1);
X = zeros(n);
for j = 1:n
    y = subst_inferior(L, P(:, j));
    X(:, j) = subst_superior(U, y);
end
end
```

**Observație.** Pentru a rezolva $Ax = b$ nu se calculează $A^{-1}$ și apoi $A^{-1}b$: costul este de aproximativ patru ori mai mare decât al factorizării urmate de două substituții, iar erorile de rotunjire sunt, în general, mai mari. În MATLAB se folosește `x = A\b`, nu `inv(A)*b`.

**Testare.** Scriptul `test_pivotare.m` reface exemplul de pe hârtie, calculează inversa și o compară cu `inv(A)`, factorizează matricea $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ și reproduce exemplul cu pivot $10^{-17}$ din secțiunea 6.1. Folosește și `doolittle_compact` din laboratorul 4.

```
% TEST_PIVOTARE Verificarea functiilor lu_pivotare, rezolva_lu_pivotare
% si inversa_lu.
clear; clc;
% Functiile din laboratoarele anterioare se afla in directoarele vecine
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));
addpath(fullfile(aici, '..', 'lab4'));
addpath(fullfile(aici, '..', 'lab5'));

%% Exemplul calculat pe hartie (solutia exacta este [1; 1; 1])
A = [2 1 3; 4 -6 0; -2 11 2];
b = [6; -2; 11];
[L, U, P] = lu_pivotare(A);
```

```
disp('P ='); disp(P); disp('L ='); disp(L); disp('U ='); disp(U);
fprintf('norm(P*A - L*U) = %g\n', norm(P*A - L*U));
x = rezolva_lu_pivotare(A, b);
disp('x ='); disp(x);

%% Inversa: comparatie cu inv(A) si verificarea A*X = I
X = inversa_lu(A);
disp('64*X ='); disp(64*X);
fprintf('norm(A*X - I) = %g, diferenta fata de inv(A): %g\n', ...
        norm(A*X - eye(3)), norm(X - inv(A)));

%% Matrice pentru care Doolittle fara pivotare esueaza, dar pivotarea merge
B = [0 1; 1 1];
[L, U, P] = lu_pivotare(B);
fprintf('B: norm(P*B - L*U) = %g\n', norm(P*B - L*U));

%% De ce pivotare: pivot foarte mic, dar nenul
e = 1e-17;
C = [e 1; 1 1]; c = [1; 2];              % solutia exacta este aproximativ [1; 1]
% Eliminare fara pivotare, scrisa direct pentru 2 x 2 (pivotul este e):
l21 = C(2,1) / C(1,1);                    % pondere uriasa: 1e17
u22 = C(2,2) - l21 * C(1,2);              % 1 - 1e17, rotunjit la -1e17
y2  = c(2) - l21 * c(1);                  % 2 - 1e17, rotunjit la -1e17
x2  = y2 / u22;                           % = 1
x1  = (c(1) - C(1,2) * x2) / C(1,1);      % (1 - 1)/1e-17 = 0, gresit!
x0  = [x1; x2];
x1p = rezolva_lu_pivotare(C, c);          % cu pivotare: pivotul este 1
fprintf('fara pivotare: [%g %g], cu pivotare: [%g %g]\n', x0, x1p);

%% Matrice singulara: functia trebuie sa se opreasca cu mesaj
try
    lu_pivotare([1 2; 2 4]);
catch err
    disp(err.message);
end
```

### Exerciții

1. Rulați `test_pivotare.m` și comparați $P$, $L$, $U$, $x$ și prima coloană a lui $64A^{-1}$ cu rezultatele de pe hârtie.
2. Comparați factorii obținuți cu `[L,U,P] = lu(A)`. Ce returnează `[L,U] = lu(A)` (cu numai două ieșiri)? Este acel $L$ inferior triunghiular?
3. Scrieți funcția `d = determinant_lu(A)` care calculează $\det A$ din factorizarea $PA = LU$, numărând interschimbările de linii. Comparați cu `det(A)`.
4. Pentru matricea Hilbert `H = hilb(n)`, $n \in \{4, 8, 12\}$, calculați `X = inversa_lu(H)` și `norm(H*X - eye(n))`. Comparați cu `cond(H)`. Ce observați?
5. Pentru $n \in \{100, 200, 400\}$ comparați timpul pentru rezolvarea lui $Ax = b$ prin `rezolva_lu_pivotare` și prin `inversa_lu(A)*b`. Este raportul apropiat de 4?

**Rezultate de control (secțiunea 6.2).** La pasul 1 se interschimbă liniile 1 și 2 (pivot 4), la pasul 2 liniile 2 și 3 (pivot 8):

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 4 & -6 & 0 \\ 0 & 8 & 2 \\ 0 & 0 & 2 \end{pmatrix},$$

$\det A = 64$, $y = (-2, 10, 2)^T$, $x = (1, 1, 1)^T$, iar prima coloană a lui $A^{-1}$ este $\big(-\frac{3}{16}, -\frac{1}{8}, \frac{1}{2}\big)^T$; în întregime, $A^{-1} = \frac{1}{64}\begin{pmatrix} -12 & 31 & 18 \\ -8 & 10 & 12 \\ 32 & -24 & -16 \end{pmatrix}$.

## 6.5 Aplicație: deformarea unei bare elastice

**Modelul.** Considerăm o bară elastică de lungime $\ell_b$, așezată pe axa $Ox$ între $x = 0$ și $x = \ell_b$, cu ambele capete fixate. Asupra barei acționează o forță axială distribuită $f(x)$ (forță pe unitatea de lungime, pozitivă spre dreapta). Notăm cu $u(x)$ deplasarea axială a punctului care se afla inițial în $x$: după deformare, el se află în $x + u(x)$. Derivata $u'(x)$ este *deformația specifică*: $u' > 0$ înseamnă alungire locală, $u' < 0$ comprimare. Notăm cu $E$ modulul lui Young al materialului și cu $\mathcal{A}$ aria secțiunii; produsul $E\mathcal{A}$ este rigiditatea axială. Echilibrul unei porțiuni mici a barei conduce la problema la limită

$$-E\mathcal{A}\,u''(x) = f(x), \quad 0 < x < \ell_b, \qquad u(0) = u(\ell_b) = 0. \tag{6.1}$$

(Aceeași ecuație descrie săgeata unei corzi întinse sub o încărcare transversală.) Pentru $f$ constant, soluția exactă este $u(x) = \frac{f}{2E\mathcal{A}}x(\ell_b - x)$, însă pentru o forță oarecare o aproximăm numeric.

**Aproximarea derivatei a doua.** Din formula lui Taylor, pentru $u$ suficient de netedă,

$$u(x \pm h) = u(x) \pm hu'(x) + \frac{h^2}{2}u''(x) \pm \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)}(\xi_\pm),$$

cu $\xi_\pm$ între $x$ și $x \pm h$. Adunând cele două relații, termenii cu $u'$ și $u'''$ se reduc și obținem

$$\frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = u''(x) + \mathcal{O}(h^2).$$

Alegem un număr natural $N$ de noduri interioare, pasul $h = \ell_b/(N+1)$ și nodurile $x_i = ih$, $i = 0, 1, \ldots, N+1$; notăm cu $u_i$ aproximarea lui $u(x_i)$ și cu $f_i = f(x_i)$. Scriind ecuația (6.1) în fiecare nod interior și înlocuind $u''$ prin aproximarea de mai sus, obținem

$$\frac{E\mathcal{A}}{h^2}\big(-u_{i-1} + 2u_i - u_{i+1}\big) = f_i, \qquad i = 1, \ldots, N,$$

cu $u_0 = u_{N+1} = 0$ din condițiile la capete. Acesta este sistemul liniar $K\mathbf{u} = \mathbf{f}$, unde $\mathbf{u} = (u_1, \ldots, u_N)^T$, $\mathbf{f} = (f_1, \ldots, f_N)^T$ și

$$K = \frac{E\mathcal{A}}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Matricea $K$ este simetrică, iar minorii ei principali sunt $(E\mathcal{A}/h^2)^k(k+1) > 0$, $k = 1, \ldots, N$ (pentru $N = 3$ și $E\mathcal{A}/h^2 = 1$ este exact matricea $M_3$ din laboratorul 5), deci $K$ este pozitiv definită: sistemul se rezolvă prin factorizarea Cholesky. Se poate arăta că eroarea $\max_i |u(x_i) - u_i|$ este de ordinul $h^2$.

**Funcția `bara_elastica`.** Primește forța ca funcție anonimă vectorizată (de exemplu `@(x) sin(pi*x)`), numărul $N$ de noduri interioare și, opțional, rigiditatea `EA` și lungimea `Lb` (implicit 1; `nargin` dă numărul de argumente primite). Verificările de la început asigură că problema are sens: $f$ este funcție, $N \geq 2$ este natural, $E\mathcal{A} > 0$, $\ell_b > 0$, iar $f$ întoarce valori finite pentru toate nodurile. Funcția `isa(f, 'function_handle')` testează dacă $f$ este o funcție anonimă. Matricea $K$ se construiește cu `diag(e,1)` și `diag(e,-1)`, care pun vectorul $e$ pe prima supradiagonală, respectiv subdiagonală.

```
function [x, u, K] = bara_elastica(f, N, EA, Lb)
% BARA_ELASTICA Deplasarile unei bare elastice 1D, fixate la ambele capete,
% sub o forta axiala distribuita f:
%   -EA * u''(x) = f(x),   0 < x < Lb,   u(0) = u(Lb) = 0.
% Derivata a doua se aproximeaza prin diferente finite centrate,
%   u''(x_i) ~ ( u(x_{i-1}) - 2*u(x_i) + u(x_{i+1}) ) / h^2,
% pe nodurile x_i = i*h, i = 0,...,N+1, cu h = Lb/(N+1). Se obtine
% sistemul K*U = F, cu K = (EA/h^2)*tridiag(-1, 2, -1), care este
% simetrica si pozitiv definita; il rezolvam prin factorizarea Cholesky.
%
% Intrari: f  - functie (function handle) vectorizata, de exemplu @(x) sin(pi*x)
%          N  - numarul de noduri interioare (N >= 2)
%          EA - rigiditatea axiala (modulul lui Young * aria sectiunii), implicit 1
%          Lb - lungimea barei, implicit 1
% Iesiri:  x  - nodurile 0 = x_0 < x_1 < ... < x_{N+1} = Lb (vector coloana)
%          u  - deplasarile in noduri, cu u(1) = u(end) = 0
%          K  - matricea sistemului (N x N)

% ---- 1. Valori implicite si verificarea parametrilor de intrare ----
if nargin < 2
    error('bara_elastica:nrArg', 'Sunt necesare cel putin doua argumente: f si N.');
end
if nargin < 3, EA = 1; end
if nargin < 4, Lb = 1; end
if ~isa(f, 'function_handle')
    error('bara_elastica:f', 'f trebuie sa fie o functie, de exemplu @(x) sin(pi*x).');
end
if ~isscalar(N) || ~isnumeric(N) || N ~= round(N) || N < 2
    error('bara_elastica:N', 'N trebuie sa fie un numar natural, N >= 2.');
end
if ~isscalar(EA) || ~isnumeric(EA) || ~isfinite(EA) || EA <= 0
    error('bara_elastica:EA', 'EA trebuie sa fie un numar real strict pozitiv.');
end
if ~isscalar(Lb) || ~isnumeric(Lb) || ~isfinite(Lb) || Lb <= 0
    error('bara_elastica:Lb', 'Lb trebuie sa fie un numar real strict pozitiv.');
end

% ---- 2. Nodurile interioare si valorile fortei ----
h  = Lb / (N + 1);           % pasul retelei
xi = h * (1:N)';             % nodurile interioare x_1, ..., x_N
F  = f(xi);
% O functie constanta scrisa @(x) 2 intoarce un singur numar
if isscalar(F)
    F = F * ones(N, 1);
end
if ~isequal(size(F), size(xi)) || ~all(isfinite(F))
    error('bara_elastica:valori_f', ...
          ['f(x) trebuie sa intoarca un vector finit de aceeasi dimensiune ' ...
           'cu x (folositi .*, ./, .^).']);
end

% ---- 3. Matricea sistemului: (EA/h^2) * tridiag(-1, 2, -1) ----
e = ones(N - 1, 1);
K = (EA / h^2) * (2*eye(N) - diag(e, 1) - diag(e, -1));

% ---- 4. Rezolvarea prin Cholesky si adaugarea conditiilor la capete ----
ui = rezolva_cholesky(K, F);
x = [0; xi; Lb];
u = [0; ui; 0];
end
```

**Exemple: cum se deformează bara când se schimbă $f$.** Scriptul `bara_exemple.m` compară soluția numerică cu cea exactă pentru $f = 1$, desenează deplasarea pentru patru tipuri de forțe, urmărește o forță localizată

$$f(x) = q\, e^{-(x-a)^2/(2w^2)}$$

(un „clopot” de înălțime $q$, centrat în $a$, de lățime $w$) care se mută de-a lungul barei (figura 1), verifică liniaritatea ($f \mapsto 2f$ dublează $u$) și estimează ordinul de convergență. Desenul barei deformate este făcut de funcția `deseneaza_bara` (anexa 6.A).

```
% BARA_EXEMPLE Cum se deformeaza o bara elastica fixata la capete cand
% se modifica forta distribuita f. Ecuatia: -EA*u'' = f, u(0) = u(1) = 0.
clear; clc; close all;
% Functiile din laboratoarele anterioare se afla in directoarele vecine
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));
addpath(fullfile(aici, '..', 'lab5'));
EA = 1; Lb = 1; N = 99;              % h = 1/100

%% 1. Forta constanta: comparatie cu solutia exacta u(x) = x(1-x)/(2EA)
[x, u] = bara_elastica(@(x) ones(size(x)), N, EA, Lb);
uex = x .* (Lb - x) / (2*EA);
figure;
plot(x, u, 'o', x, uex, '-', 'LineWidth', 1.2, 'MarkerSize', 3);
grid on; xlabel('x'); ylabel('u(x)');
legend('diferente finite', 'solutia exacta');
title('f = 1');
fprintf('Eroarea maxima pentru f = 1: %.2e\n', max(abs(u - uex)));

%% 2. Patru tipuri de forte: f (sus) si deplasarea u (jos)
forte = { @(x) ones(size(x)), ...
          @(x) 2*x - 1, ...
          @(x) sin(2*pi*x), ...
          @(x) 4*exp(-(x - 0.3).^2 / (2*0.05^2)) };
nume  = {'f = 1', 'f = 2x - 1', 'f = sin(2\pi x)', 'forta localizata in x = 0.3'};
figure;
for k = 1:4
    [x, u] = bara_elastica(forte{k}, N, EA, Lb);
    subplot(2, 4, k);
    plot(x, forte{k}(x), 'r', 'LineWidth', 1.2); grid on; title(nume{k});
    subplot(2, 4, k + 4);
    plot(x, u, 'b', 'LineWidth', 1.2); grid on; xlabel('x');
    if k == 1, ylabel('u(x)'); end
end

%% 3. Forta localizata care se muta: familia de deplasari
figure;
pozitii = 0.15:0.1:0.85;
culori = parula(numel(pozitii));
subplot(2, 1, 1); hold on; subplot(2, 1, 2); hold on;
for k = 1:numel(pozitii)
    a = pozitii(k);
    f = @(x) 4*exp(-(x - a).^2 / (2*0.05^2));
    [x, u] = bara_elastica(f, N, EA, Lb);
    subplot(2, 1, 1); plot(x, f(x), 'Color', culori(k,:), 'LineWidth', 1.2);
    subplot(2, 1, 2); plot(x, u, 'Color', culori(k,:), 'LineWidth', 1.2);
end
subplot(2, 1, 1); grid on; ylabel('f(x)'); title('Forta localizata in x = a, a = 0.15,
    ..., 0.85');
subplot(2, 1, 2); grid on; ylabel('u(x)'); xlabel('x');
```

```
%% 4. Liniaritate: dublarea fortei dubleaza deplasarea
figure; hold on;
for q = 1:4
    [x, u] = bara_elastica(@(x) q*sin(pi*x), N, EA, Lb);
    plot(x, u, 'LineWidth', 1.2);
end
grid on; xlabel('x'); ylabel('u(x)');
legend('q = 1', 'q = 2', 'q = 3', 'q = 4'); title('f = q sin(\pi x)');

%% 5. Bara deformata, colorata dupa deformatia specifica
figure;
[x, u] = bara_elastica(@(x) 4*exp(-(x - 0.3).^2 / (2*0.05^2)), N, EA, Lb);
deseneaza_bara(gca, x, u, 4);
colorbar; title('Forta localizata in x = 0.3 (deplasari amplificate de 4 ori)');

%% 6. Ordinul de convergenta: f = pi^2 sin(pi x), u exact = sin(pi x)/EA
Nv = [9 19 39 79 159];
err = zeros(size(Nv)); hv = Lb ./ (Nv + 1);
for k = 1:numel(Nv)
    [x, u] = bara_elastica(@(x) pi^2*sin(pi*x), Nv(k), EA, Lb);
    err(k) = max(abs(u - sin(pi*x)/EA));
end
figure;
loglog(hv, err, 'o-', hv, err(1)*(hv/hv(1)).^2, '--', 'LineWidth', 1.2);
grid on; xlabel('h'); ylabel('eroarea maxima');
legend('eroare', 'panta 2', 'Location', 'northwest');
c = polyfit(log(hv), log(err), 1);
fprintf('Ordinul de convergenta estimat: %.2f\n', c(1));
```

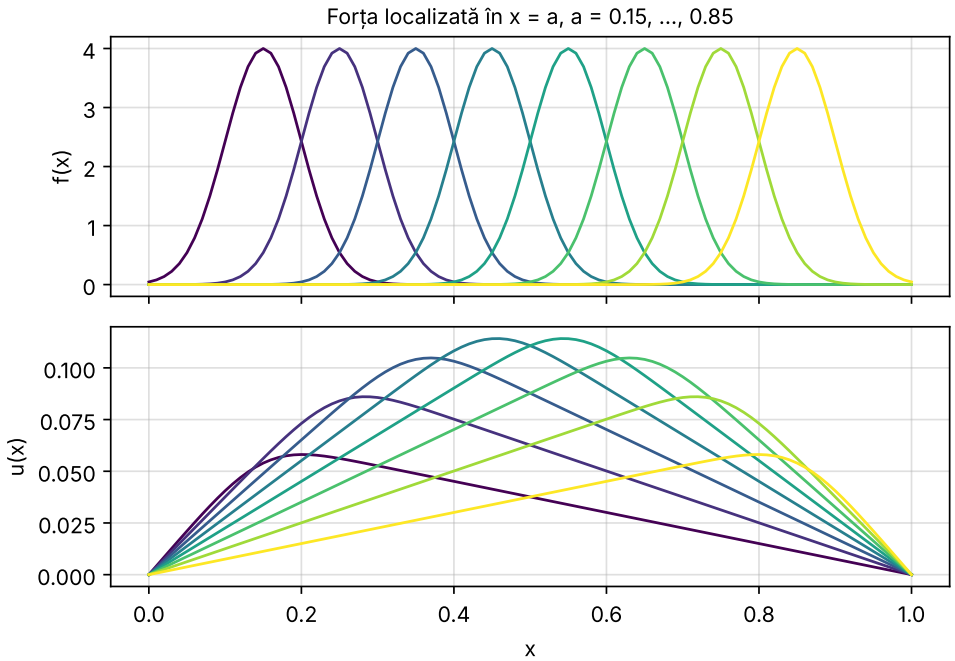



Figura 1: Forța localizată $f(x) = 4e^{-(x-a)^2/(2\cdot 0.05^2)}$ pentru $a = 0.15, \ldots, 0.85$ (sus) și deplasările corespunzătoare (jos), cu $E\mathcal{A} = 1$, $\ell_b = 1$. Partea barei din stânga forței se alungește ($u' > 0$), iar cea din dreapta se comprimă ($u' < 0$); deplasarea maximă apare sub forță.

**Aplicația interactivă.** Funcția `bara_interactiva` (codul complet în anexa 6.A) deschide o fereastră cu trei grafice – forța $f$, deplasarea $u$ și bara (nedeformată jos, deformată sus, colorată după

$u'$) – și cu controale pentru: tipul forței (constantă, liniară, sinusoidală, localizată, două forțe opuse), amplitudinea $q$, poziția $a$, lățimea $w$, rigiditatea $E\mathcal{A}$, factorul $s$ de amplificare a desenului și numărul $N$ de noduri. Butonul *Fixează curba* păstrează deplasarea curentă (gri) pentru comparație. Figura 2 reproduce conținutul ferestrei pentru două forțe opuse.

Aplicația folosește direct ideea din laboratorul 5 despre mai mulți termeni liberi: $K = E\mathcal{A}\,K_1$, unde $K_1$ este matricea pentru $E\mathcal{A} = 1$. Pentru $N$ fixat, $K_1 = LL^T$ se factorizează o singură dată; la fiecare mișcare a unui cursor se rezolvă numai $Ly = \mathbf{f}$, $L^T\mathbf{v} = y$ și se ia $\mathbf{u} = \mathbf{v}/E\mathcal{A}$, adică aproximativ $2N^2$ operații. De aceea graficele se actualizează instantaneu, chiar și pentru $N = 200$. Cursoarele se creează cu `uicontrol('Style', 'slider', ...)`, iar funcțiile imbricate (definite în interiorul funcției principale, înaintea ultimului `end`) au acces la variabilele acesteia.

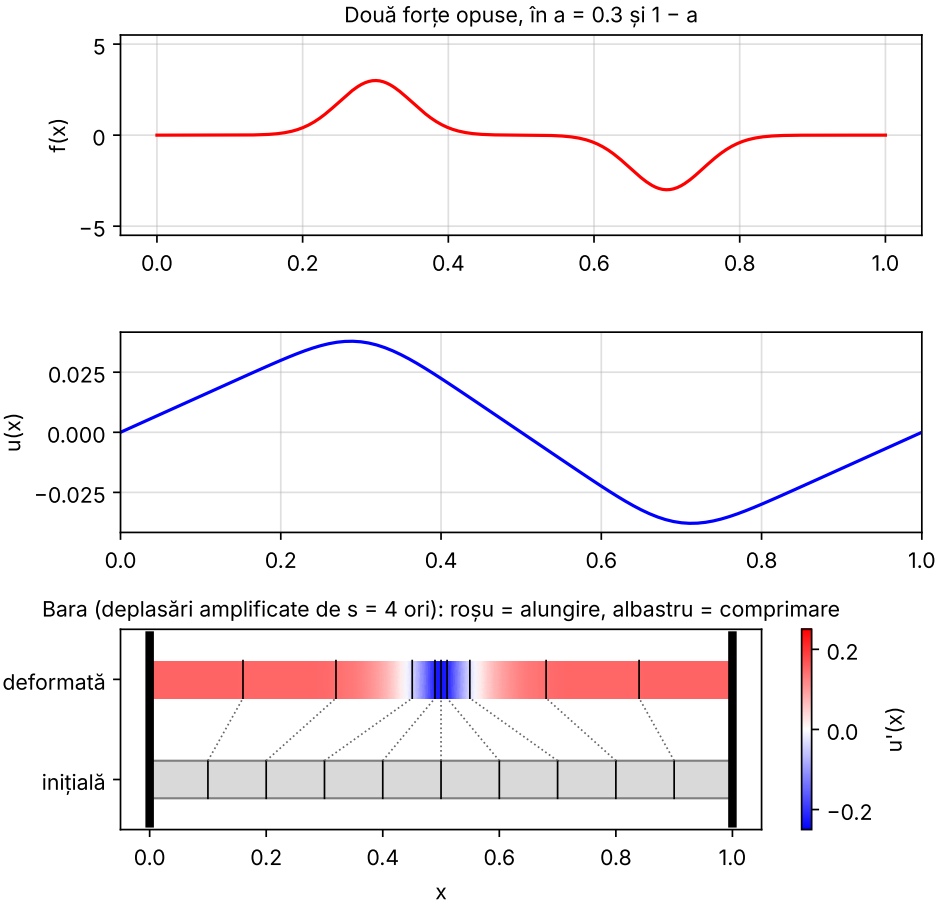



Figura 2: Conținutul ferestrei `bara_interactiva` pentru două forțe opuse, $q = 3$, $a = 0.3$, $w = 0.05$: forța, deplasarea și bara, cu deplasările amplificate de $s = 4$ ori. Liniile punctate unesc pozițiile inițiale și finale ale acelorași puncte materiale.

**Animația.** Scriptul `bara_animatie.m` mută forța localizată de la $a = 0.1$ la $a = 0.9$ și înapoi și redesenează la fiecare cadru; `set(h, 'YData', ...)` schimbă datele unei curbe deja desenate, iar `drawnow` actualizează imediat fereastra. Cu `salveaza_gif = true`, animația se salvează în fișierul `bara.gif`, care poate fi inclus într-o prezentare.

```
% BARA_ANIMATIE Animatie: o forta localizata se deplaseaza de-a lungul
% barei, iar deformarea se recalculeaza la fiecare cadru. Matricea K nu
% depinde de f, deci factorizarea Cholesky se face o singura data; la
% fiecare cadru se rezolva doar doua sisteme triunghiulare.
clear; clc; close all;
% Functiile din laboratoarele anterioare se afla in directoarele vecine
```

```
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));
addpath(fullfile(aici, '..', 'lab5'));
EA = 1; Lb = 1; N = 79;
s = 2.5;                                % factorul de amplificare a deplasarilor
salveaza_gif = false;                   % true: salveaza animatia in bara.gif

% Matricea K si factorizarea ei (o singura data)
[x, ~, K] = bara_elastica(@(x) zeros(size(x)), N, EA, Lb);
L  = cholesky_fact(K);
xi = x(2:end-1);                        % nodurile interioare

% Pozitiile succesive ale fortei: dus si intors
pozitii = [linspace(0.1, 0.9, 80), linspace(0.9, 0.1, 80)];
forta = @(t, a) 4*exp(-(t - a).^2 / (2*0.05^2));

fig = figure('Color', 'w', 'Position', [100 100 800 650]);
ax1 = subplot(3, 1, 1); hf = plot(ax1, x, forta(x, pozitii(1)), 'r', 'LineWidth', 1.5);
grid(ax1, 'on'); ylabel(ax1, 'f(x)'); axis(ax1, [0 Lb -0.5 4.5]);
ax2 = subplot(3, 1, 2); hu = plot(ax2, x, 0*x, 'b', 'LineWidth', 1.5);
grid(ax2, 'on'); ylabel(ax2, 'u(x)'); axis(ax2, [0 Lb -0.12 0.12]);
ax3 = subplot(3, 1, 3);

for k = 1:numel(pozitii)
    a = pozitii(k);
    F = forta(xi, a);
    ui = subst_superior(L', subst_inferior(L, F));   % K*ui = F
    u = [0; ui; 0];
    set(hf, 'YData', forta(x, a));
    set(hu, 'YData', u);
    title(ax1, sprintf('Forta localizata in x = %.2f', a));
    deseneaza_bara(ax3, x, u, s, 0.4);
    drawnow;
    if salveaza_gif
        [img, harta] = rgb2ind(frame2im(getframe(fig)), 128);
        if k == 1
            imwrite(img, harta, 'bara.gif', 'gif', 'LoopCount', Inf, 'DelayTime', 0.04)
    ;
        else
            imwrite(img, harta, 'bara.gif', 'gif', 'WriteMode', 'append', 'DelayTime',
    0.04);
        end
    else
        pause(0.03);
    end
end
```

### Exerciții

1. Rulați `bara_exemple.m`. Ce eroare obțineți pentru $f = 1$? De ce este ea de ordinul erorilor de rotunjire, deși aproximarea derivatei a doua are eroare $\mathcal{O}(h^2)$? (Indicație: $u^{(4)} = 0$.)
2. Ce ordin de convergență estimează secțiunea 6 a scriptului? Este în acord cu analiza Taylor?
3. În aplicația interactivă, alegeți forța localizată și mutați-o cu cursorul $a$. Unde este deplasarea maximă? Ce parte a barei se alungește și ce parte se comprimă? Verificați că pentru $a = 0.5$ deplasarea este simetrică.
4. Pentru forța constantă, dublați $q$, apoi dublați $EA$. Cum se modifică $\max|u|$? Explicați pe

baza ecuației (6.1).

5. Alegeți „două forțe opuse”. De ce $u$ se anulează și la mijlocul barei? Ce simetrie are $u$?
6. Folosiți *Fixează curba* pentru $N = 10$, apoi treceți la $N = 200$. Cât de diferite sunt cele două curbe pentru forța localizată cu $w = 0.02$? Explicați prin raportul dintre $w$ și $h$.
7. Mărind $s$, desenul barei deformate se poate „plia”. Arătați că punctele $x + su(x)$ rămân în ordine crescătoare dacă $s \max |u'| < 1$.
8. Rezolvați sistemul $K\mathbf{u} = \mathbf{f}$ și cu `rezolva_lu_pivotare`. Se fac interschimbări de linii? De ce? (Indicație: $K$ este diagonal dominantă.) Comparați timpii celor două metode pentru $N = 400$.
9. Modificați `bara_elastica` pentru o bară fixată numai în $x = 0$, liberă în $x = \ell_b$: $u'(\ell_b) = 0$, aproximată prin $u_{N+1} = u_N$. Ce se schimbă în ultima linie a lui $K$? Mai este $K$ simetrică și pozitiv definită?

## Anexa 6.A. Codul aplicației interactive și al desenului barei

**Funcția `deseneaza_bara`.** Bara deformată este desenată cu `surface` ca o bandă subțire ale cărei culori se interpolează între noduri (`'FaceColor', 'interp'`); deformația specifică $u'$ este aproximată cu `gradient(u, x)`, prin diferențe finite. Harta de culori trece de la albastru (comprimare) la alb și roșu (alungire).

```
function deseneaza_bara(ax, x, u, s, epsmax)
% DESENEAZA_BARA Deseneaza in axele ax bara nedeformata (jos) si bara
% deformata (sus). Punctul material aflat initial in x ajunge in x + s*u(x);
% factorul s > 0 amplifica deplasarile, ca sa fie vizibile. Culoarea barei
% deformate arata deformatia specifica eps = u'(x): rosu = alungire (eps > 0),
% albastru = comprimare (eps < 0), alb = nedeformat.

if nargin < 5, epsmax = max(abs(gradient(u, x))) + eps; end
x = x(:); u = u(:);
n = numel(x); Lb = x(end);
xd  = x + s * u;                    % pozitiile deformate
def = gradient(u, x);               % u'(x) aproximat prin diferente finite

cla(ax); hold(ax, 'on');
% Bara nedeformata (gri)
patch([0 Lb Lb 0], [-0.55 -0.55 -0.25 -0.25], [0.85 0.85 0.85], ...
      'EdgeColor', [0.5 0.5 0.5], 'Parent', ax);
% Bara deformata, colorata dupa deformatia specifica
surface([xd'; xd'], [0.25*ones(1,n); 0.55*ones(1,n)], zeros(2,n), [def'; def'], ...
        'FaceColor', 'interp', 'EdgeColor', 'none', 'Parent', ax);
% Aceleasi puncte materiale pe cele doua bare (11 repere)
for k = round(linspace(1, n, 11))
    plot(ax, [x(k) x(k)], [-0.55 -0.25], 'k-');
    plot(ax, [x(k) xd(k)], [-0.25 0.25], ':', 'Color', [0.4 0.4 0.4]);
    plot(ax, [xd(k) xd(k)], [0.25 0.55], 'k-');
end
% Peretii care fixeaza capetele
plot(ax, [0 0], [-0.75 0.75], 'k-', 'LineWidth', 4);
plot(ax, [Lb Lb], [-0.75 0.75], 'k-', 'LineWidth', 4);
hold(ax, 'off');

% Harta de culori albastru - alb - rosu
m = 64; t = linspace(0, 1, m)';
cm = [t t ones(m,1); ones(m,1) flipud(t) flipud(t)];
colormap(ax, cm);
set(ax, 'CLim', [-epsmax epsmax], 'YTick', [-0.4 0.4], ...
```

```
        'YTickLabel', {'initiala', 'deformata'});
axis(ax, [-0.05*Lb 1.05*Lb -0.8 0.8]);
xlabel(ax, 'x');
end
```

```
function bara_interactiva
% BARA_INTERACTIVA Aplicatie interactiva: deformarea unei bare elastice
% fixate la capete, -EA*u'' = f, u(0) = u(1) = 0, cand se modifica forta f.
% Din meniuri si cursoare se aleg tipul fortei, amplitudinea q, pozitia a,
% latimea w, rigiditatea EA, numarul de noduri N si factorul s de
% amplificare a desenului.
%
% Observatie de calcul: matricea sistemului este K = EA*K1, unde K1 este
% matricea pentru EA = 1. Factorizam K1 = L*L' o singura data (pentru N
% fixat); la fiecare miscare a unui cursor rezolvam doar L*y = F,
% L'*v = y si luam u = v/EA, adica aproximativ 2*N^2 operatii.

% Functiile din laboratoarele anterioare se afla in directoarele vecine
aici = fileparts(mfilename('fullpath'));
addpath(fullfile(aici, '..', 'lab3'));
addpath(fullfile(aici, '..', 'lab5'));

Lb = 1;                                % lungimea barei
qmax = 5;                              % amplitudinea maxima a fortei
valN = [10 20 50 100 200];             % valorile posibile pentru N
xf = linspace(0, Lb, 400)';            % retea fina, doar pentru desenul lui f
curbe = [];                            % curbele fixate pentru comparatie

% ---- Fereastra si axele ----
fig = figure('Name', 'Bara elastica 1D', 'NumberTitle', 'off', ...
             'Color', 'w', 'Position', [80 80 1100 680]);
ax1 = axes('Parent', fig, 'Position', [0.07 0.71 0.58 0.23]);
ax2 = axes('Parent', fig, 'Position', [0.07 0.40 0.58 0.23]);
ax3 = axes('Parent', fig, 'Position', [0.07 0.07 0.58 0.23]);
hold(ax2, 'on');

% ---- Controalele din dreapta ----
y0 = 0.93; dy = 0.075; xc = 0.70; lc = 0.27;
eticheta = @(txt, k) uicontrol(fig, 'Style', 'text', 'Units', 'normalized', ...
    'Position', [xc y0-k*dy+0.025 lc 0.03], 'String', txt, ...
    'HorizontalAlignment', 'left', 'BackgroundColor', 'w', 'FontSize', 10);
cursor = @(mn, mx, v, k) uicontrol(fig, 'Style', 'slider', 'Units', 'normalized', ...
    'Position', [xc y0-k*dy lc 0.03], 'Min', mn, 'Max', mx, 'Value', v);

eticheta('Tipul fortei f', 0);
popF = uicontrol(fig, 'Style', 'popupmenu', 'Units', 'normalized', ...
    'Position', [xc y0-0*dy-0.005 lc 0.03], 'FontSize', 10, 'String', ...
    {'constanta: q', 'liniara: q(2x-1)', 'sinusoidala: q sin(pi x)', ...
     'sinusoidala: q sin(2 pi x)', 'localizata in x = a', ...
     'doua forte opuse, in a si 1-a'}, 'Value', 5);
etQ = eticheta('', 1);  slQ = cursor(-qmax, qmax, 3, 1);
etA = eticheta('', 2);  slA = cursor(0.05, 0.95, 0.3, 2);
etW = eticheta('', 3);  slW = cursor(0.02, 0.3, 0.05, 3);
etE = eticheta('', 4);  slE = cursor(1, 5, 1, 4);
etS = eticheta('', 5);  slS = cursor(0, 5, 1, 5);
eticheta('Numarul de noduri interioare N', 6);
popN = uicontrol(fig, 'Style', 'popupmenu', 'Units', 'normalized', ...
    'Position', [xc y0-6*dy-0.005 lc 0.03], 'FontSize', 10, ...
    'String', {'10', '20', '50', '100', '200'}, 'Value', 3);
uicontrol(fig, 'Style', 'pushbutton', 'Units', 'normalized', ...
    'Position', [xc y0-7.3*dy 0.13 0.045], 'String', 'Fixeaza curba', ...
    'FontSize', 10, 'Callback', @(s, e) fixeaza());
uicontrol(fig, 'Style', 'pushbutton', 'Units', 'normalized', ...
```

```
    'Position', [xc+0.14 y0-7.3*dy 0.13 0.045], 'String', 'Sterge curbele', ...
    'FontSize', 10, 'Callback', @(s, e) sterge());
info = uicontrol(fig, 'Style', 'text', 'Units', 'normalized', ...
    'Position', [xc 0.07 lc 0.20], 'BackgroundColor', 'w', 'FontSize', 10, ...
    'HorizontalAlignment', 'left', 'String', '');

% Fiecare control recalculeaza solutia; cursoarele si in timpul tragerii
set([popF slQ slA slW slE slS], 'Callback', @(s, e) actualizeaza());
set(popN, 'Callback', @(s, e) schimbaN());
for h = [slQ slA slW slE slS]
    try
        addlistener(h, 'ContinuousValueChange', @(s, e) actualizeaza());
    catch
        % versiunile care nu au acest eveniment actualizeaza la eliberare
    end
end

% ---- Graficele (create o singura data, apoi doar actualizate) ----
hf = plot(ax1, xf, 0*xf, 'r', 'LineWidth', 1.5);
grid(ax1, 'on'); ylabel(ax1, 'f(x)'); axis(ax1, [0 Lb -1.1*qmax 1.1*qmax]);
title(ax1, 'Forta axiala distribuita');
hu = plot(ax2, [0 Lb], [0 0], 'b.-', 'LineWidth', 1.5, 'MarkerSize', 10);
grid(ax2, 'on'); ylabel(ax2, 'u(x)'); axis(ax2, [0 Lb -0.7 0.7]);
title(ax2, 'Deplasarea u(x) (axe fixe, ca sa se vada cum se schimba)');

% ---- Pregatirea pentru N initial si primul calcul ----
[x, xi, L] = pregateste(valN(get(popN, 'Value')));
u = 0*x;
actualizeaza();

    % ===== Functii imbricate (folosesc variabilele de mai sus) =====
    function [x, xi, L] = pregateste(N)
        % Reteaua si factorizarea Cholesky a matricei pentru EA = 1
        [x, ~, K1] = bara_elastica(@(t) zeros(size(t)), N, 1, Lb);
        xi = x(2:end-1);
        L = cholesky_fact(K1);
    end

    function F = forta(t)
        % Valorile fortei alese in meniu, in punctele t
        q = get(slQ, 'Value'); a = get(slA, 'Value'); w = get(slW, 'Value');
        g = @(c) exp(-(t - c).^2 / (2*w^2));       % "clopot" centrat in c
        switch get(popF, 'Value')
            case 1, F = q * ones(size(t));
            case 2, F = q * (2*t/Lb - 1);
            case 3, F = q * sin(pi*t/Lb);
            case 4, F = q * sin(2*pi*t/Lb);
            case 5, F = q * g(a);
            case 6, F = q * (g(a) - g(Lb - a));
        end
    end

    function actualizeaza()
        EA = get(slE, 'Value'); s = get(slS, 'Value');
        set(etQ, 'String', sprintf('Amplitudinea q = %.2f', get(slQ, 'Value')));
        set(etA, 'String', sprintf('Pozitia a = %.2f', get(slA, 'Value')));
        set(etW, 'String', sprintf('Latimea w = %.3f', get(slW, 'Value')));
        set(etE, 'String', sprintf('Rigiditatea EA = %.2f', EA));
        set(etS, 'String', sprintf('Amplificarea desenului s = %.2f', s));
        % Rezolvarea: K1*v = F prin cele doua substitutii, apoi u = v/EA
        v = subst_superior(L', subst_inferior(L, forta(xi)));
        u = [0; v; 0] / EA;
        % Actualizarea graficelor
        set(hf, 'YData', forta(xf));
        set(hu, 'XData', x, 'YData', u);
```

```
        if numel(x) > 60, set(hu, 'Marker', 'none'); else, set(hu, 'Marker', '.'); end
        deseneaza_bara(ax3, x, u, s, 2.5);
        title(ax3, 'Bara: rosu = alungire, albastru = comprimare');
        % Informatii numerice
        [umax, k] = max(abs(u));
        % Daca s*max|u'| >= 1, punctele desenate se suprapun (desenul se
        % "pliaza"); este doar un efect al amplificarii, nu al modelului
        epsmax = max(abs(gradient(u, x)));
        if s * epsmax >= 1
            aviz = sprintf('\nATENTIE: s*max|u''| = %.2f >= 1, micsorati s.', s*epsmax);
        else
            aviz = '';
        end
        set(info, 'String', sprintf(['N = %d, h = %.4f\n' ...
            'max |u| = %.4f, atins in x = %.3f\n' ...
            'forta totala = %.4f\n' ...
            'Factorizarea se face o data pentru N fixat; ' ...
            'la fiecare miscare se rezolva doar doua sisteme triunghiulare.%s'], ...
            numel(xi), x(2) - x(1), umax, x(k), trapz(xf, forta(xf)), aviz));
    end

    function schimbaN()
        [x, xi, L] = pregateste(valN(get(popN, 'Value')));
        actualizeaza();
    end

    function fixeaza()
        c = plot(ax2, x, u, '-', 'Color', [0.7 0.7 0.7], 'LineWidth', 1);
        curbe = [curbe c];
        uistack(hu, 'top');
    end

    function sterge()
        delete(curbe(isgraphics(curbe)));
        curbe = [];
    end
end
```

**Funcția** `bara_interactiva`.