

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 : `if 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 .

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V2_num1.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]`.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V3_num1.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 3×3 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 `kmax`, pentru $n = 1, \dots, N$, și valoarea `nmax` 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`: if `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 $N = 30$.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V4_num1.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]`.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V5_num1.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 $[100, 200]$.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V6_num1.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 $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ și $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. 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`: `if 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 $N = 500$.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V7_num1.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)`, cu `if b(i) == 1`, măriți k cu 1.
5. Testați funcția pe $n = 13$, 0 și 64.

Nume și prenume: _____

Grupa: _____

Timp de lucru: 20 de minute. Sarcina 1 se scrie în scriptul `lab_V8_num1.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. Cu `if v(m) == val`, puneți `poz = m`; cu `elseif 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]`.

Sisteme triunghiulare și factorizarea LU

1. **Sisteme inferior triunghiulare.** Pentru $L \in \mathbb{R}^{n \times n}$ inferior triunghiulară cu $\ell_{ii} \neq 0$, sistemul $Lx = b$ se rezolvă prin substituție directă:

$$x_i = \frac{1}{\ell_{ii}} \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right), \quad i = 1, 2, \dots, n.$$

```
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, opresc 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(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 necunoscutele 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
```

2. Sisteme superior triunghiulare. Pentru $U \in \mathbb{R}^{n \times n}$ superior triunghiulară cu $u_{ii} \neq 0$, sistemul $Ux = b$ se rezolvă prin substituție inversă:

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^n u_{ij} x_j \right), \quad i = n, n-1, \dots, 1.$$

```
function x = subst_superior(U, b)
% SUBST_SUPERIOR Rezolva sistemul U*x = b, cu U superior triunghiulara,
% prin substituție 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, opresc 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
```

3. Factorizarea LU (Doolittle). Dacă $A \in \mathbb{R}^{n \times n}$ și submatricele principale $A_k = A(1:k, 1:k)$, $k = 1, \dots, n-1$, sunt inversabile, atunci există o unică factorizare $A = LU$, cu L inferior triunghiulară având $\ell_{ii} = 1$ și U superior triunghiulară. La pasul k se calculează ponderile și matricea de ponderi (matricea Gauss)

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \dots, n, \quad M_k = I_n - m^{(k)} e_k^T, \quad A^{(k+1)} = M_k A^{(k)},$$

unde $m^{(k)} = (0, \dots, 0, m_{k+1,k}, \dots, m_{nk})^T$. După $n-1$ pași, $U = A^{(n)} = M_{n-1} \cdots M_1 A$, iar

$$L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}, \quad \ell_{ik} = m_{ik} \ (i > k), \quad \ell_{kk} = 1,$$

adică L se construiește direct din ponderi, fără a calcula inverse.

```
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 construiește 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, opresc 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

```

4. **Test.** Pentru $A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix}$ și $b = (4, 10, 24)^T$ se obțin

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

Sistemul $Ax = b$ se rezolvă în doi pași: $Ly = b$ (substituție directă), apoi $Ux = y$ (substituție inversă).

```
% TEST_LU Verificarea functiilor subst_inferior, subst_superior si factorizare_LU.

% Stergem variabilele din Workspace si continutul Command Window
clear; clc;

% Matricea sistemului si vectorul termenilor liberi (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);
% Afisam factorii L si U
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:
% pasul 1: L*y = b, prin substitutie directa
y = subst_inferior(L, b);
% pasul 2: U*x = y, prin substitutie inversa
x = subst_superior(U, y);
% Afisam solutia si reziduul norm(A*x - b) (ideal 0)
disp('x ='); disp(x);
fprintf('norm(A*x - b) = %g\n', norm(A*x - b));

% Matrice pentru care factorizarea LU nu exista: A(1,1) = 0
B = [0 1; 1 1];
% try ... catch prinde eroarea, ca scriptul sa nu se opreasca
try
    factorizare_LU(B);
catch err
    % Afisam mesajul de eroare generat de functie
    disp(err.message);
end
```