Tema Laborator 6/7

Task 1:

% Definirea sistemului Ax = b

A = [1, -1; 0, 1];

b = [2; 3];

% Soluția exactă folosind metoda directă

x\_exact = A \ b;

% Inițializare

x = [0; 0]; % Soluție inițială

num\_iterations = 5; % Număr de iterații

% Stocăm evoluția soluției pentru plot

x1\_values = zeros(num\_iterations, 1);

x2\_values = zeros(num\_iterations, 1);

error\_values = zeros(num\_iterations, 1);

fprintf('Iterație | x1 | x2 | Eroare\n');

fprintf('-----------------------------------------\n');

for k = 1:num\_iterations

for i = 1:size(A,1) % Iterăm prin fiecare ecuație

a\_i = A(i, :); % Selectăm o linie din A

b\_i = b(i); % Selectăm componenta corespunzătoare din b

x = x + (b\_i - a\_i \* x) / (norm(a\_i)^2) \* a\_i'; % Aplicăm regula Kaczmarz

end

% Calculul erorii

error = norm(x - x\_exact);

% Salvăm valorile pentru grafic

x1\_values(k) = x(1);

x2\_values(k) = x(2);

error\_values(k) = error;

8

% Afișare rezultat

fprintf('%4d | %.4f | %.4f | %.6f\n', k, x(1), x(2), error);

end

% ======= Reprezentare grafică =======

% 1. Graficul evoluției soluției x1 și x2

figure;

plot(1:num\_iterations, x1\_values, '-bo', 'LineWidth', 2, 'MarkerSize', 8);

hold on;

plot(1:num\_iterations, x2\_values, '-ro', 'LineWidth', 2, 'MarkerSize', 8);

xlabel('Iterația');

ylabel('Valoare soluției');

title('Evoluția soluției x\_1 și x\_2 în algoritmul Kaczmarz');

legend('x\_1', 'x\_2', 'Location', 'best');

grid on;

hold off;

% 2. Graficul erorii la fiecare iterație

figure;

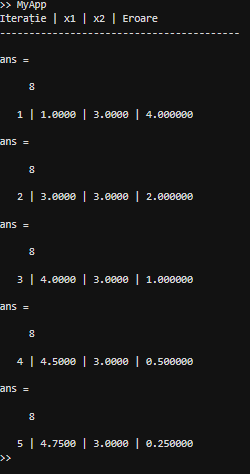
plot(1:num\_iterations, error\_values, '-ko', 'LineWidth', 2, 'MarkerSize', 8);

xlabel('Iterația');

ylabel('Eroare ||x\_k - x^\*||');

title('Convergența erorii în algoritmul Kaczmarz');

grid on;



Task 2:

% Definim matricea A

A = [2, 3, 1;

4, 7, 3;

6, 18, 5];

% Dimensiunea matricei A

n = size(A, 1);

% Inițializăm L ca matrice identitate și U ca A

L = eye(n);

U = A;

% Aplicăm eliminarea Gauss pentru factorizarea LU

for k = 1:n-1

for i = k+1:n

% Calculăm factorul de eliminare

L(i, k) = U(i, k) / U(k, k);

% Actualizăm rândul i din U

U(i, :) = U(i, :) - L(i, k) \* U(k, :);

end

13

end

% Afișăm rezultatele

disp('Matricea L:');

disp(L);

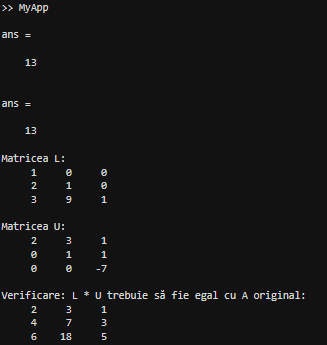
disp('Matricea U:');

disp(U);

% Verificare: A = L \* U

disp('Verificare: L \* U trebuie să fie egal cu A original:');

disp(L \* U);



Task 3:

function lu\_factorization()

% Definim matricea A si vectorul b

A = [4, 3, -1; 2, 1, 3; -6, -3, 4];

b = [5; 3; 2];

% Aplicam factorizarea LU

[L, U] = lu\_decomposition(A);

% Verificam daca LU = A

fprintf('Verificare: LU = A ?\n');

disp(L \* U);

% Rezolvam sistemul Ax = b folosind LU

y = forward\_substitution(L, b);

x = backward\_substitution(U, y);

% Afisam solutia sistemului

fprintf('Solutia sistemului Ax = b este:\n');

disp(x);

end

function [L, U] = lu\_decomposition(A)

n = size(A, 1);

L = eye(n);

U = A;

for k = 1:n-1

for i = k+1:n

L(i, k) = U(i, k) / U(k, k);

U(i, :) = U(i, :) - L(i, k) \* U(k, :);

end

end

end

function y = forward\_substitution(L, b)

n = length(b);

y = zeros(n, 1);

for i = 1:n

y(i) = b(i) - L(i, 1:i-1) \* y(1:i-1);

end

end

function x = backward\_substitution(U, y)

n = length(y);

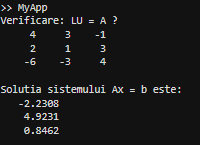
x = zeros(n, 1);

for i = n:-1:1

x(i) = (y(i) - U(i, i+1:n) \* x(i+1:n)) / U(i, i);

end

end



Task 4:

function dfp\_bfgs\_optimization()

% Definim functia obiectiv

f = @(x) (x(1) - 2)^4 + (x(1) - 2\*x(2))^2;

grad\_f = @(x) [4\*(x(1) - 2)^3 + 2\*(x(1) - 2\*x(2)); -4\*(x(1) - 2\*x(2))];

% Punct initial

x0 = [0; 3];

% Apelam metoda DFP

fprintf('Optimizare cu DFP:\n');

dfp\_method(f, grad\_f, x0);

% Apelam metoda BFGS

fprintf('\nOptimizare cu BFGS:\n');

bfgs\_method(f, grad\_f, x0);

end

function dfp\_method(f, grad\_f, x0)

tol = 1e-6;

max\_iter = 100;

n = length(x0);

H = eye(n);

x = x0;

iter = 0;

x\_hist = x;

while norm(grad\_f(x)) > tol && iter < max\_iter

p = -H \* grad\_f(x);

alpha = line\_search(f, grad\_f, x, p);

x\_new = x + alpha \* p;

s = x\_new - x;

y = grad\_f(x\_new) - grad\_f(x);

rho = 1 / (y' \* s);

H = H + rho \* (s \* s') - (H \* y \* y' \* H) / (y' \* H \* y);

x = x\_new;

iter = iter + 1;

x\_hist = [x\_hist, x];

end

plot\_trajectory(f, x\_hist, 'DFP');

end

function bfgs\_method(f, grad\_f, x0)

tol = 1e-6;

max\_iter = 100;

n = length(x0);

H = eye(n);

x = x0;

iter = 0;

x\_hist = x;

while norm(grad\_f(x)) > tol && iter < max\_iter

p = -H \* grad\_f(x);

alpha = line\_search(f, grad\_f, x, p);

x\_new = x + alpha \* p;

s = x\_new - x;

y = grad\_f(x\_new) - grad\_f(x);

rho = 1 / (y' \* s);

V = eye(n) - rho \* (s \* y');

H = V' \* H \* V + rho \* (s \* s');

x = x\_new;

iter = iter + 1;

x\_hist = [x\_hist, x];

end

plot\_trajectory(f, x\_hist, 'BFGS');

end

function alpha = line\_search(f, grad\_f, x, p)

alpha = 1;

c = 1e-4;

rho = 0.9;

while f(x + alpha \* p) > f(x) + c \* alpha \* grad\_f(x)' \* p

alpha = rho \* alpha;

end

end

function plot\_trajectory(f, x\_hist, method\_name)

[X, Y] = meshgrid(-1:0.1:4, -1:0.1:4);

Z = arrayfun(@(x, y) f([x; y]), X, Y);

figure;

contour(X, Y, Z, 50);

hold on;

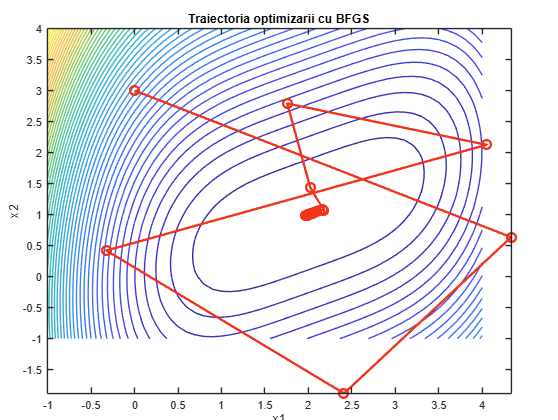
plot(x\_hist(1, :), x\_hist(2, :), 'ro-', 'LineWidth', 2, 'MarkerSize', 8);

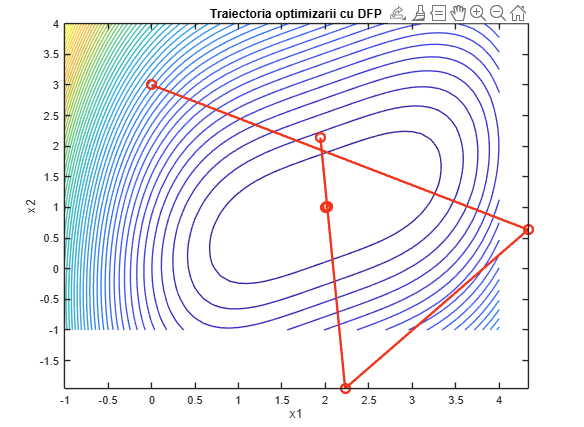
title(['Traiectoria optimizarii cu ', method\_name]);

xlabel('x1'); ylabel('x2');

hold off;

end





Task 5:

function bfgs\_exercise()

% Definim functia obiectiv

f = @(x) (x(1) - 3)^2 + (x(2) - 1)^4;

grad\_f = @(x) [2 \* (x(1) - 3); 4 \* (x(2) - 1)^3];

hessian\_f = @(x) [2, 0; 0, 12 \* (x(2) - 1)^2];

% Valori initiale

x0 = [1; 2];

H = eye(2); % Matricea identitate

alpha = 0.1;

% Aplicam metoda BFGS

[x\_opt, f\_opt] = bfgs\_method(f, grad\_f, x0, H, alpha);

% Afisam rezultatul

fprintf('Minimul gasit: x1 = %.6f, x2 = %.6f\n', x\_opt(1), x\_opt(2));

fprintf('Valoarea functiei in minim: f(x) = %.6f\n', f\_opt);

end

function [x, fval] = bfgs\_method(f, grad\_f, x0, H, alpha)

tol = 1e-6;

max\_iter = 100;

x = x0;

iter = 0;

x\_hist = x;

while norm(grad\_f(x)) > tol && iter < max\_iter

p = -H \* grad\_f(x);

x\_new = x + alpha \* p;

s = x\_new - x;

y = grad\_f(x\_new) - grad\_f(x);

rho = 1 / (y' \* s);

V = eye(2) - rho \* (s \* y');

H = V' \* H \* V + rho \* (s \* s');

x = x\_new;

iter = iter + 1;

x\_hist = [x\_hist, x];

end

fval = f(x);

% Reprezentam grafic traiectoria

plot\_trajectory(f, x\_hist);

end

function plot\_trajectory(f, x\_hist)

[X, Y] = meshgrid(-1:0.1:4, -1:0.1:4);

Z = arrayfun(@(x, y) f([x; y]), X, Y);

figure;

contour(X, Y, Z, 50);

hold on;

plot(x\_hist(1, :), x\_hist(2, :), 'ro-', 'LineWidth', 2, 'MarkerSize', 8);

title('Traiectoria optimizarii cu BFGS');

xlabel('x1'); ylabel('x2');

hold off;

end



