Proiect Optimizări 1

Nume: Vlăduțu Alexandru Nicușor

Grupa: 323AA

1. Sarcina de învățare

Sarcina de învățare aleasă a fost regresia, întrucât aceasta se potrivește cel mai bine cu structura și tipul de date oferite de setul de date ales. Regresia permite prezicerea unor valori continue, cum este cazul sarcinii de încălzire într-o clădire. Pentru a evalua precizia modelului, am utilizat funcția de pierdere (loss) eroarea medie pătratică (Mean Squared Error - MSE):

$$L = (1/n) * \Sigma(y_i - \hat{y}_i)^2$$

unde y_i este valoarea reală, iar ŷ_i este valoarea prezisă de rețea. MSE este o alegere populară pentru regresie deoarece penalizează erorile mari mai puternic.

2. Baza de date

Baza de date utilizată: Energy Efficiency Dataset – UCI Machine Learning Repository
Referință teoretică: Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools – ScienceDirect

Setul conține 768 de exemple simulate în Ecotect, modelând 12 tipuri de clădiri cu variații în:
- suprafața vitrată
- distribuția acesteia
- orientare

Fiecare eșantion are 8 caracteristici de intrare și 2 ieșiri reale:
X1: Compactitate relativă
X2: Suprafață
X3: Zona peretelui
X4: Suprafața acoperișului
X5: Înălțime totală
X6: Orientare
X7: Zona de geamuri
X8: Distribuția suprafeței de geam

Y1: Sarcina de încălzire
Y2: Sarcina de răcire

Pentru acest proiect, am folosit doar Y1 (Heating Load) pentru simplificare și claritate.

3. Sarcina de minimizare

Am utilizat o rețea neuronală cu un singur strat ascuns (hidden layer) format din 16 neuroni. Setul de date a fost împărțit astfel:
- A_train: 615 măsurători (pentru antrenare)
- A_test: 153 măsurători (pentru testare)

Funcția de activare utilizată a fost :SigLin(x) = σ(ax) ∗ x, unde a = 0.2 și σ este funcția sigmoidă:

$$\sigma(z) = 1/\left(1 + e^{(-z)}\right)$$

Obiectivul este minimizarea funcției de pierdere față de greutățile de intrare și ieșire ale rețelei:
$min_{X,x} L(X, x)$

Actualizarea se face simultan pentru ambele tipuri de greutăți, conform formulelor derivate în backpropagation

$$\min_{x\epsilon\mathbb{R}^m, X\epsilon\mathbb{R}^{(n+1)xm}} L(e, g(\bar{A}X)x)$$

$$\min_{x\epsilon\mathbb{R}^m, X\epsilon\mathbb{R}^{(n+1)xm}} \frac{1}{2N}\|g(\bar{A}X)x - e\|^2$$

Functia obiectiv de minimizat este :

$$L(X, x) = \frac{1}{2N}\|g(\bar{A}X)x - e\|^2$$

- $A \in \mathbb{R}^{N\times n}$ baza de date, unde N este nr. de exemple și n este nr. de caracteristici ale unui exemplu
- $e \in \mathbb{R}^N$ vectorul referință (etichete).
- $\mathbb{1}_N \in \mathbb{R}^N$ vector plin cu unu.

$$\bar{A} = [A, \mathbb{1}_N]$$

4. Metode de optimizare

Am implementat două metode diferite: Gradient Descent și Levenberg–Marquardt.

4.1 Metoda Gradient Descent

Gradient Descent actualizează parametrii modelului în direcția opusă gradientului funcției de pierdere:
$x_{k+1} = x_k - \alpha \nabla L(x_k)$

Gradientul a fost calculat folosind derivatele parţiale ale funcţiei de pierdere faţă de fiecare greutate din reţea.

$$\frac{\partial \mathrm{L}}{\partial \mathrm{x}} = \frac{1}{\mathrm{N}} \mathrm{g}(\bar{\mathrm{A}}\mathrm{X})^{\mathrm{T}} (\mathrm{g}(\bar{\mathrm{A}}\mathrm{X})\mathrm{x} - \mathrm{e})$$

$$\frac{\partial L}{\partial X} = \frac{1}{N} \bar{A}^T [(g(\bar{A}X)x - e) \odot g'(\bar{A}X)x^T]$$

Pentru acest lucru, a fost necesar să derivăm şi funcţia de activare SigLin în raport cu inputul său:

$$d\big(SigLin(x)\big)/dx = \sigma(ax) + a * \sigma(ax) * \big(1 - \sigma(ax)\big) * x$$

$$g'(z) = \sigma'(z) - 0.2 = \sigma(z)(1 - \sigma(z)) - 0.2$$

Am folosit produsul Hadamard ($\odot$) pentru aplicarea derivatelor pe fiecare element.

Condiţia de oprire a algoritmului a fost norma diferenţei dintre gradientul vechi şi cel nou:

$$\left\| \nabla L_k - \nabla L_{\{k-1\}} \right\| < \varepsilon$$

4.2 Metoda Levenberg–Marquardt

Levenberg–Marquardt combină avantajele metodelor Gauss–Newton şi Gradient Descent. Formula de actualizare este:

$$x_{\{k+1\}} = x_k - \alpha_k \big((J_k)^{\mathrm{T}} J_k + \beta_k I_n \big)^{-1} {J_k}^{\mathrm{T}} F_k$$

unde J ($J(x) = \nabla F(x)$) este Jacobianul erorilor faţă de parametri, iar F este vectorul reziduurilor. În MATLAB, Jacobianul a fost estimat folosind metoda „i-trick”, care presupune modificarea unui singur parametru la un moment dat.

$$J(x)^T d = \mathcal{R}\left(\frac{F(x + itd) - F(x)}{it}\right)$$

Am folosit o regulă adaptivă pentru ajustarea lui α si β :

```
if val_loss_new < val_loss
    beta_lm = beta_lm / 1.5; % Crestem efectul metodei Newton
    alfa_lm = alfa_lm * 1.05;
else
    beta_lm = beta_lm * 1.5;
    alfa_lm = alfa_lm * 0.5; % Crestem efectul metodei Gradient

    % Revenire la vechii parametri -> directie nu prea buna
    X_lm_new = X_lm;
    x_lm_new = x_lm;
    y_pred_new = y_pred;
    val_loss_new = val_loss;
end
```

α: controlul caracterului GD

β: controlul caracterului GN

Condiția de oprire: ||∇L|| < ε

$$\|\nabla L\| = J^T * F$$

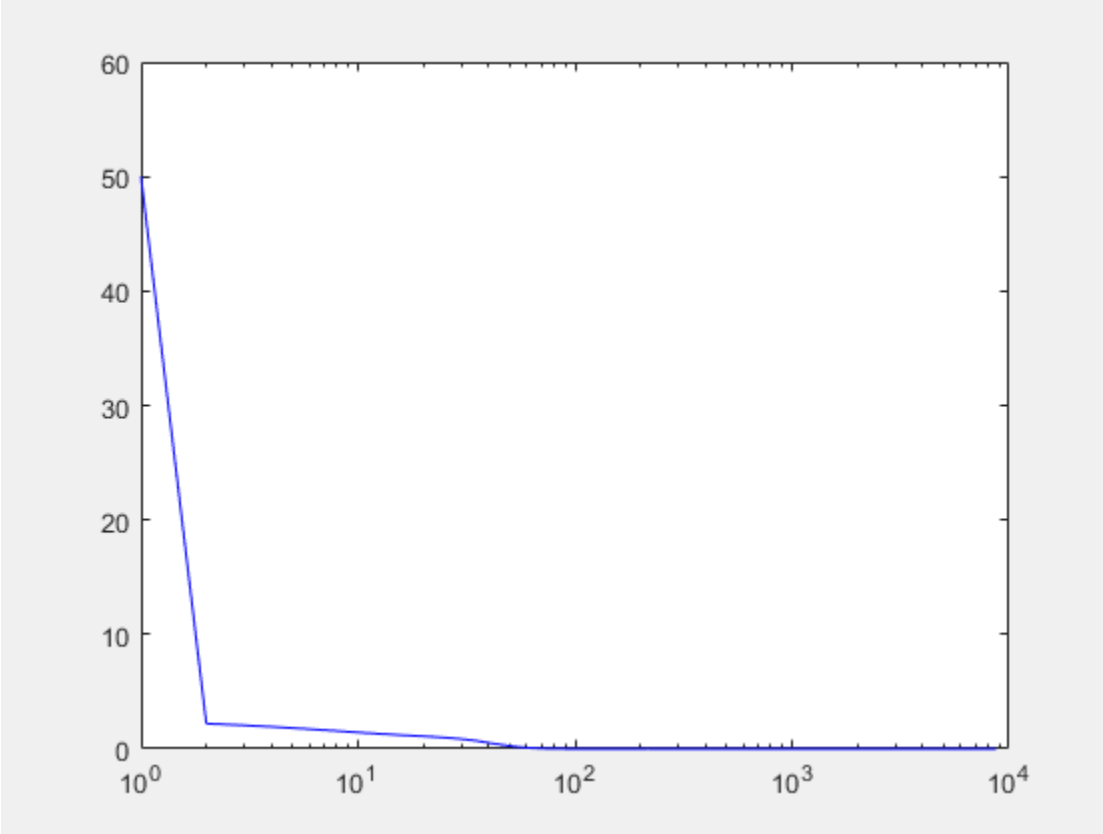
5. Rezultate

5.1 Comparație în funcție de numărul de iterații

5.1.1 Gradient Descent (GD)

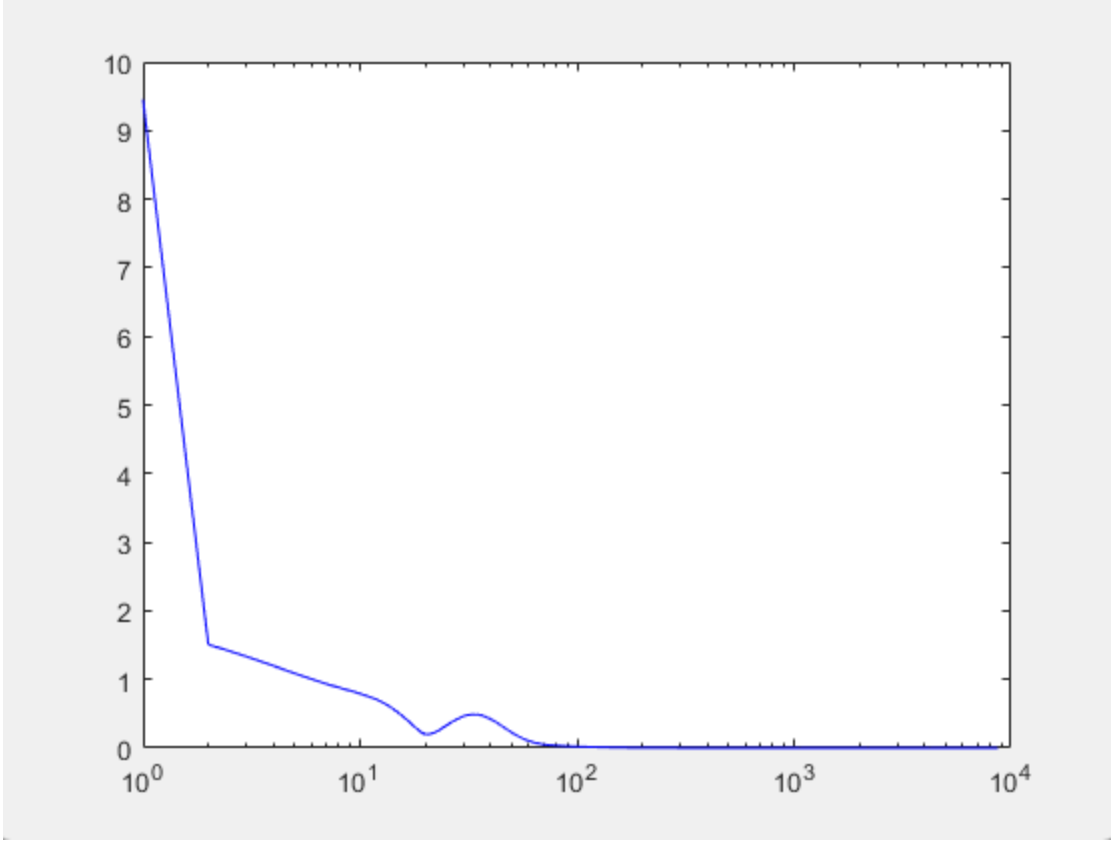
Referitor la numărul de iterații metoda GD necesita mai multe iterații:

```
Iteratia 8678: ||dgrad(X)|| = 9.998333e-06, ||dgrad(x)|| = 8.382795e-06
```



Figură 1. Weight Input

Metoda converge rapida pana in proximitatea punctului de minim, dar progresează foarte greu odată ce ajunge in jurul lui.



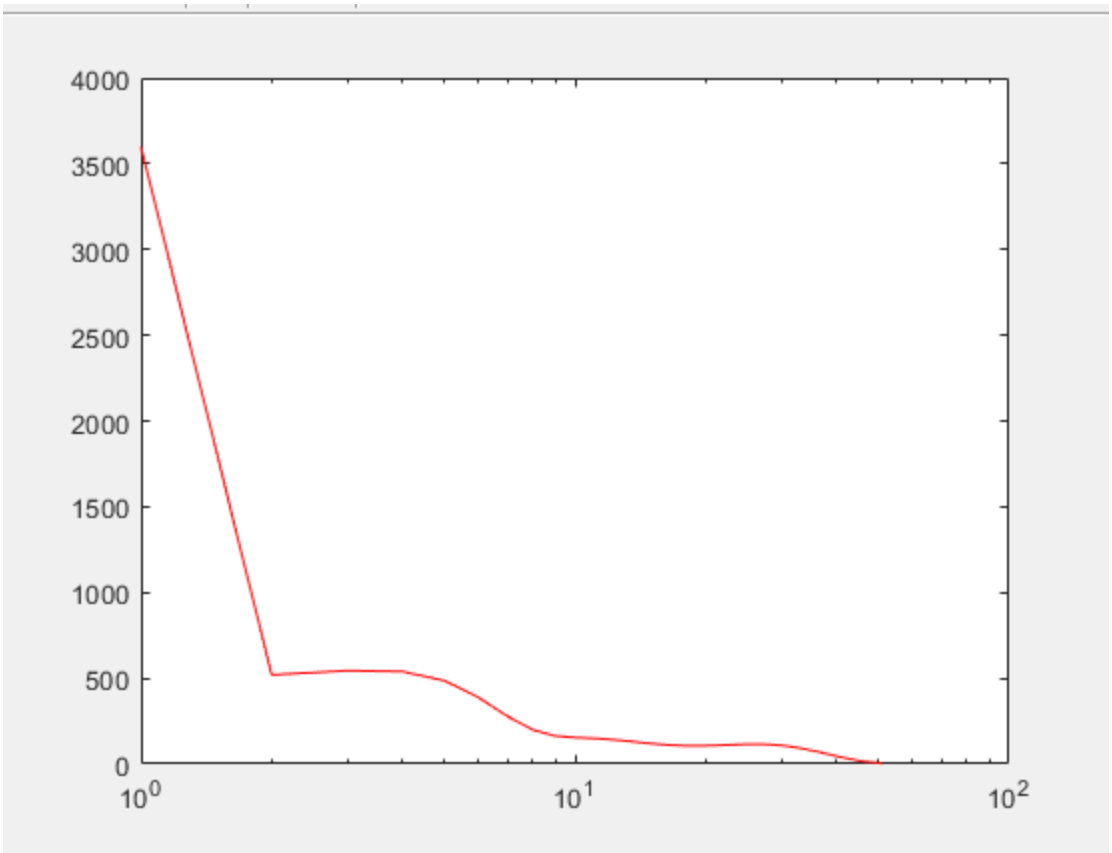
Figură 2. Weight Output

Metoda Gradient Descent necesită un număr semnificativ mai mare de iteraţii pentru a converge. Convergenţa iniţială este rapidă, însă devine lentă în apropierea minimului funcţiei obiectiv.
- Figura 1 – Evoluția greutăţilor de intrare: se observă o stabilizare lentă în apropierea minimului.
- Figura 2 – Evoluția greutăţilor de ieşire: fluctuaţiile sunt mai pronunţate din cauza optimizării simultane a tuturor greutăţilor.

5.1.2 Levenberg–Marquardt (LM)

Metoda LM este mai rapida din punctul de vedere al iteraţiilor .

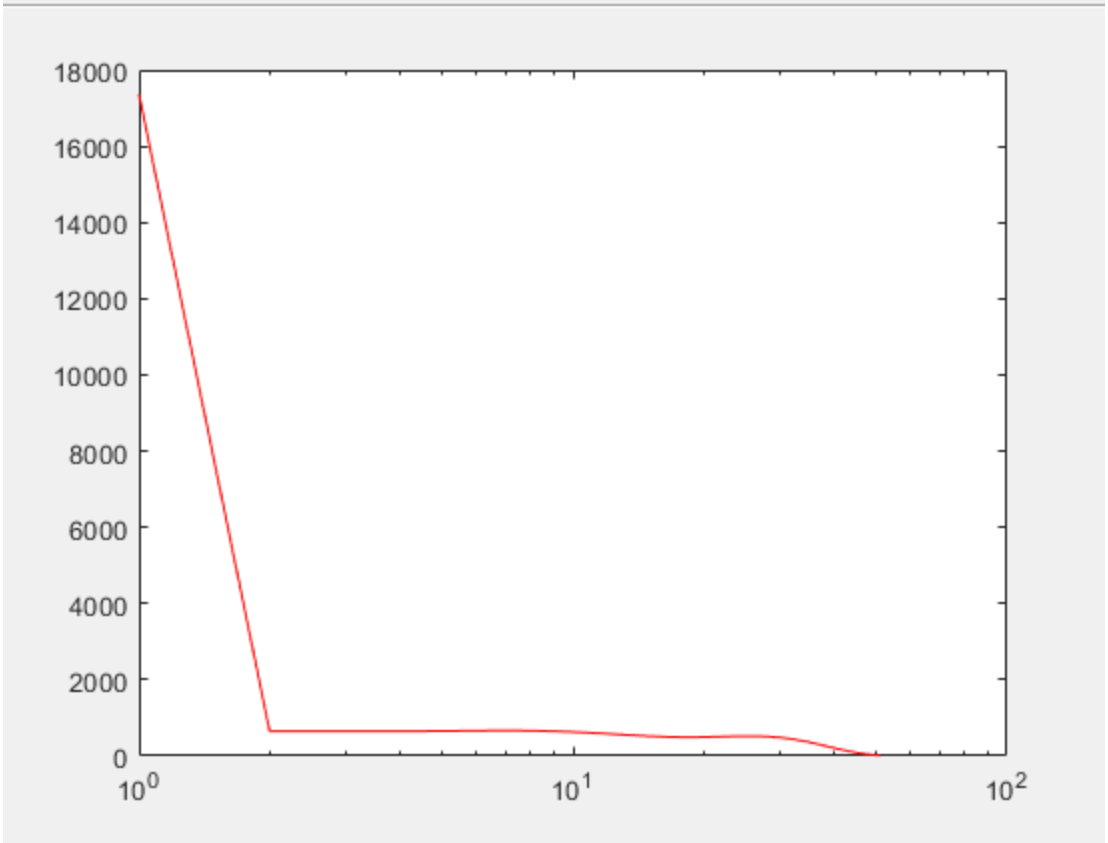
```
Iteratia 51: ||dgrad(X)|| = 0.000000e+00, ||dgrad(x)|| = 0.000000e+00
```



Figură 3. Weight Input

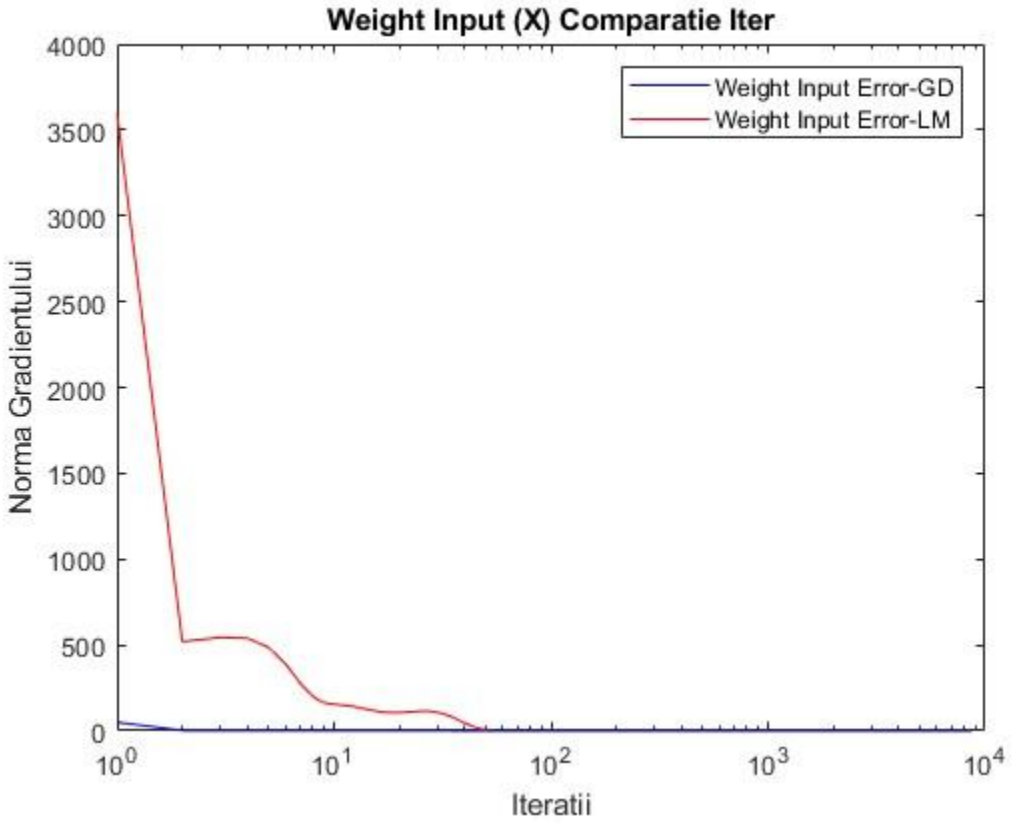
Această metodă ajunge în apropierea minimului mult mai rapid, în doar câteva iteraţii. Este eficientă în zonele apropiate de soluția optimă.

- Figura 3 – Evoluția greutăţilor de intrare: rapiditate şi stabilitate.
- Figura 4 – Evoluția greutăţilor de ieşire: convergenţă mai lină comparativ cu GD.

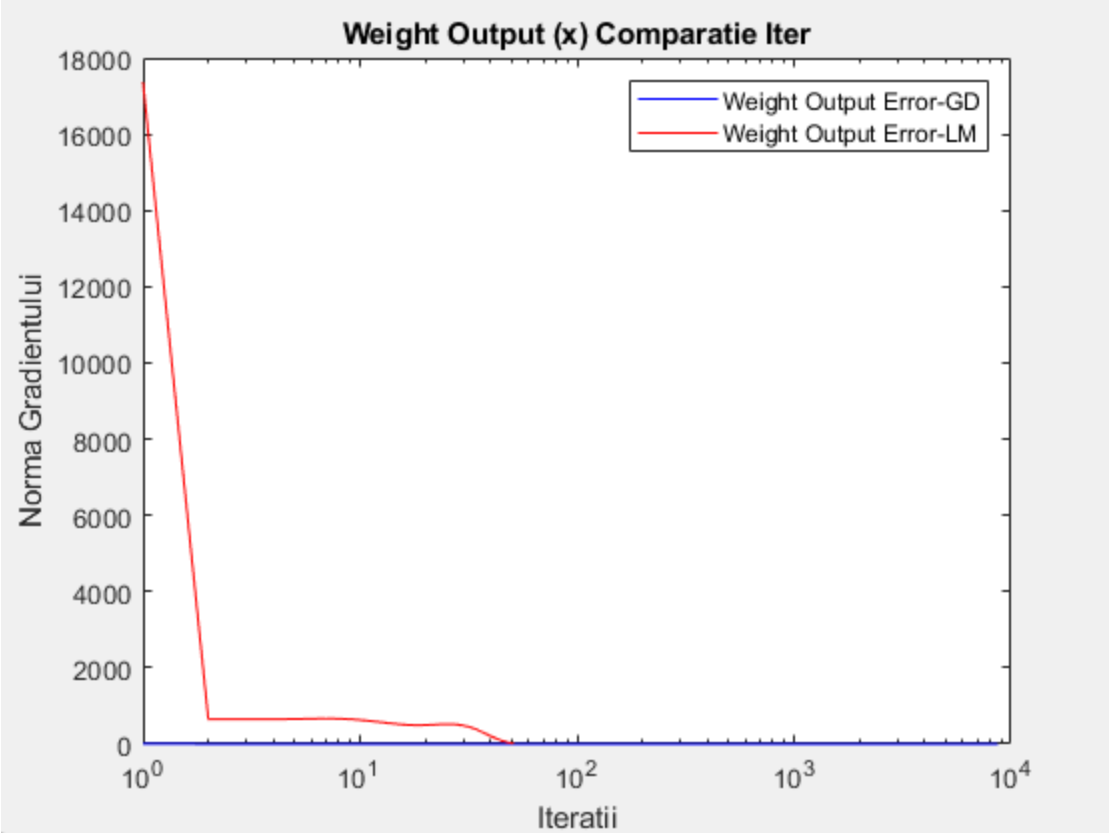


Figură 4. Weight Output

5.1.3 Comparație între metode



\

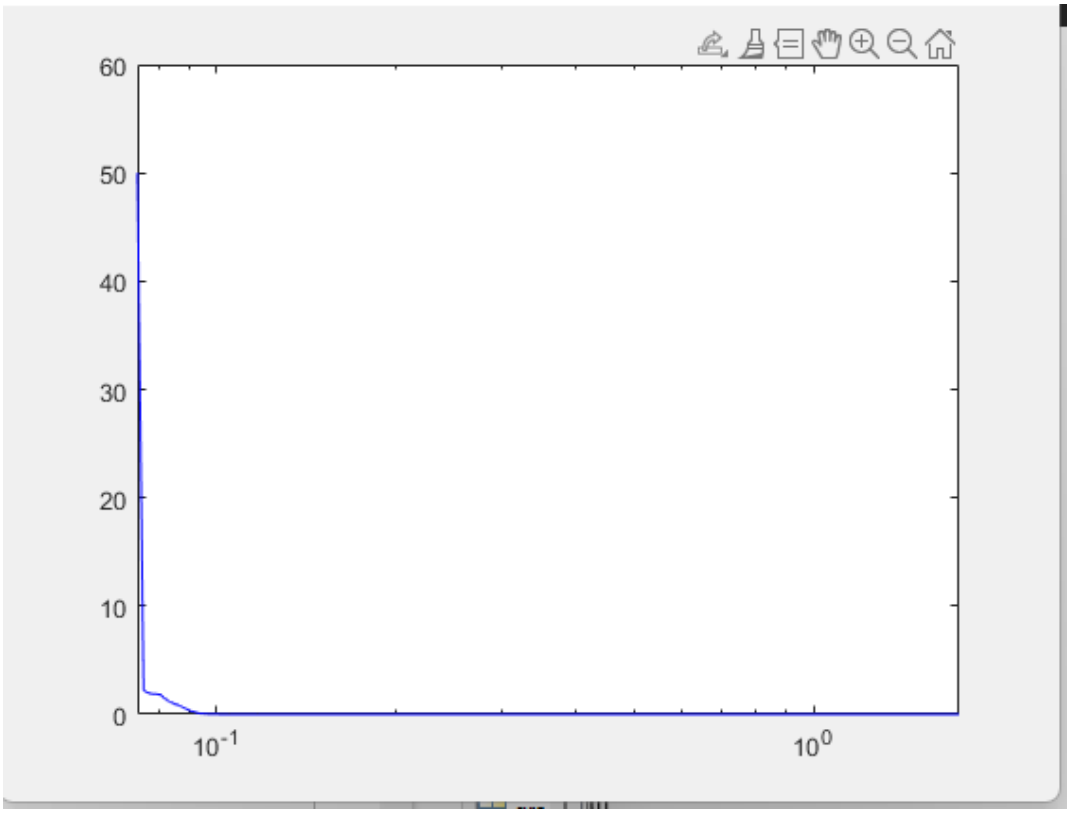


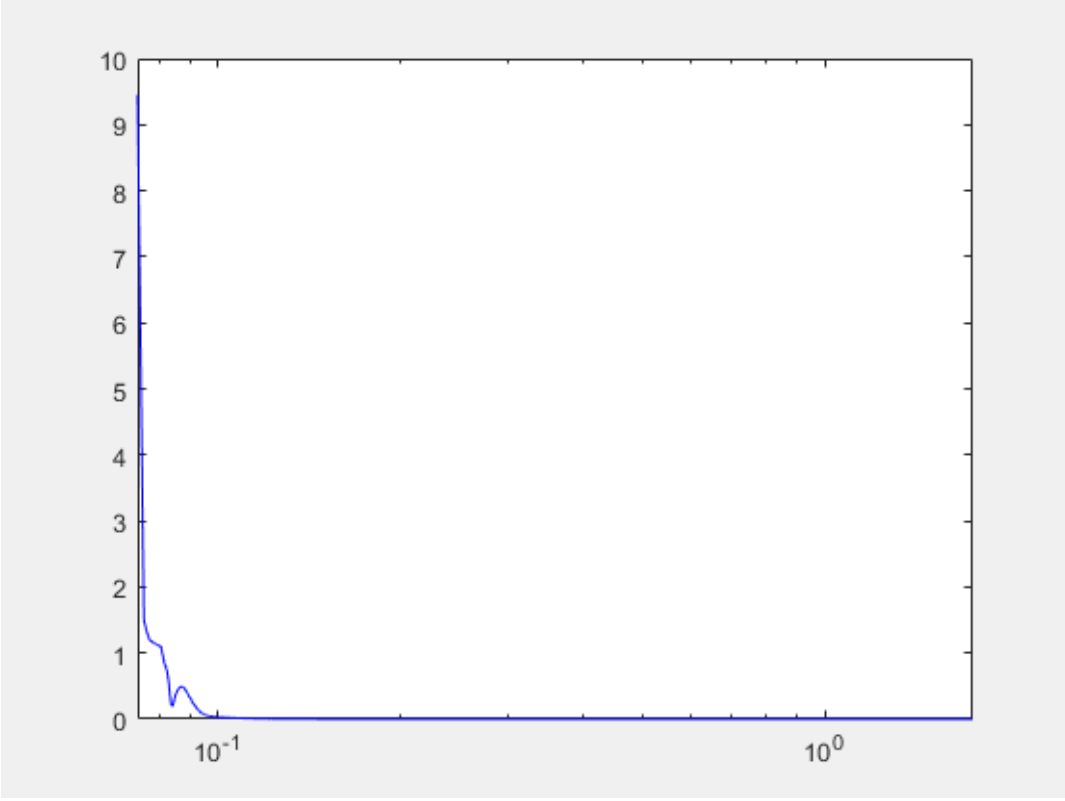
Din grafice reiese clar:
- GD progresează rapid la început dar stagnează în apropierea minimului.
- LM converge mai eficient şi mai stabil în regiunea minimului.

5.2 Comparație în funcție de timp

5.2.1 GD

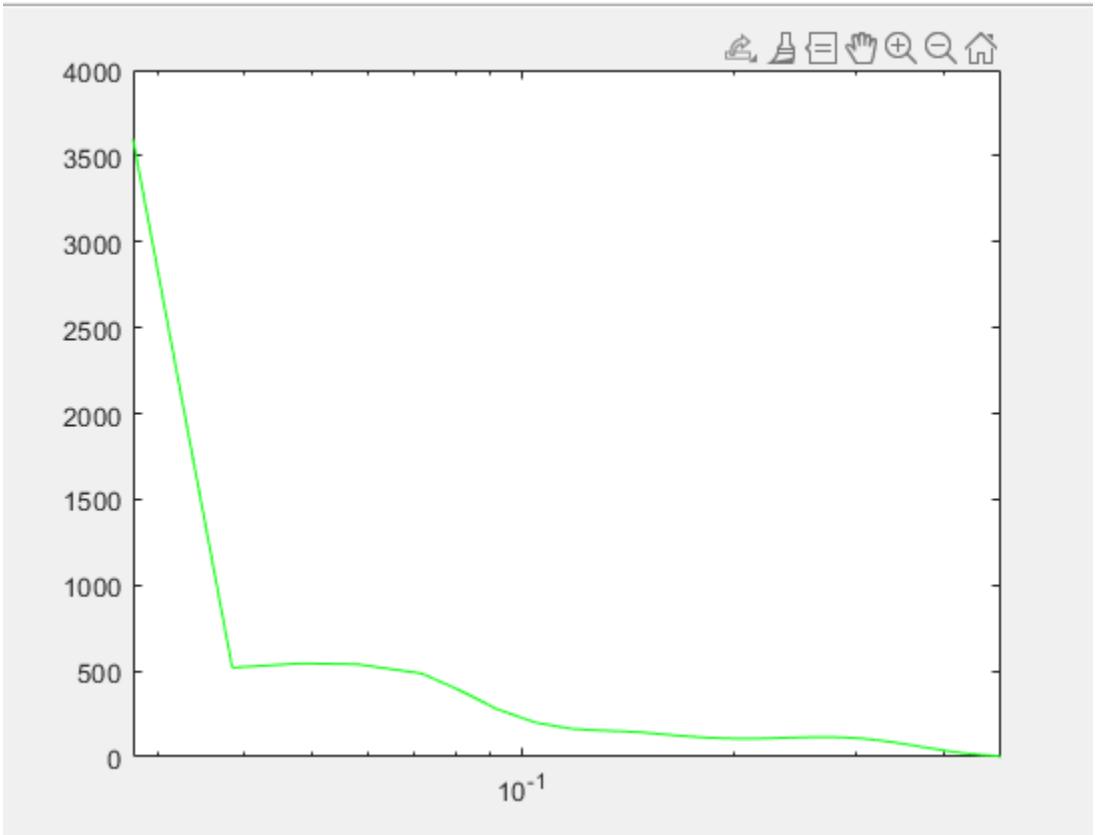
GD ajunge mai rapid în proximitatea minimului deoarece implică doar calculul gradientului, fără inversări de matrici sau estimări Jacobiene.

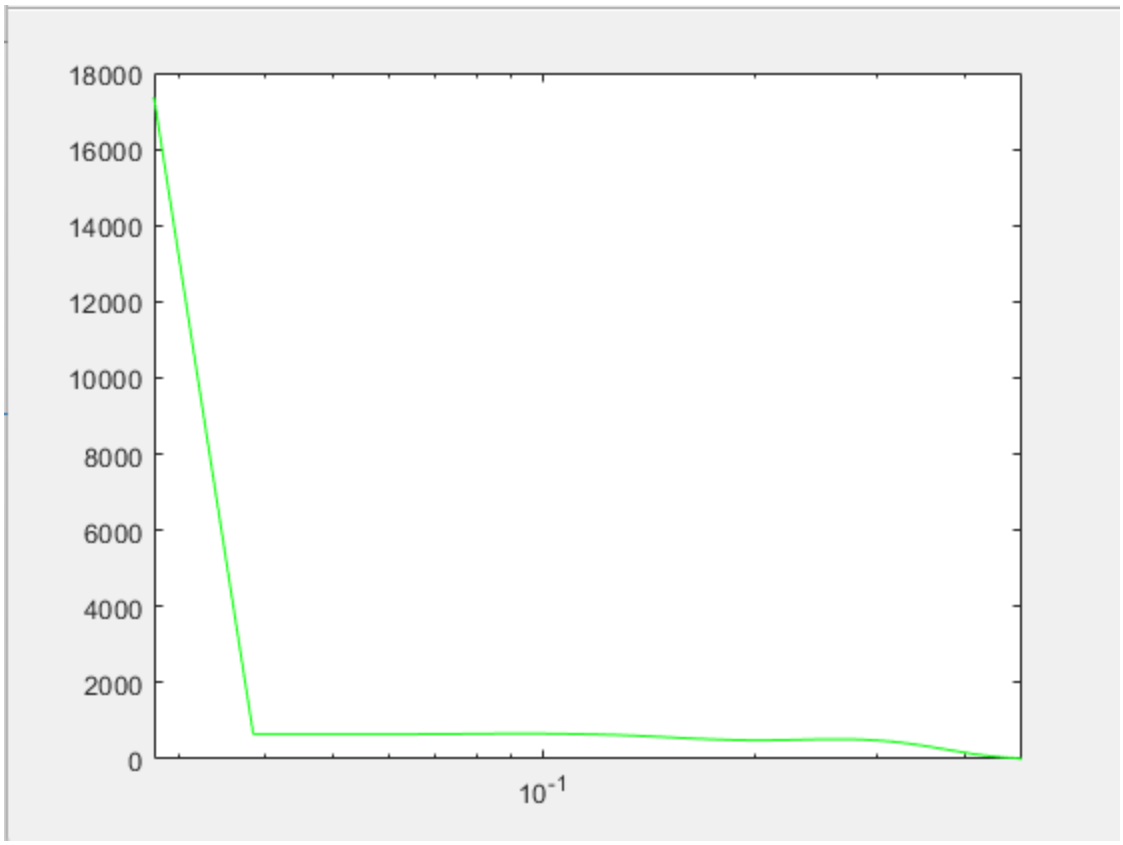

60
50
40
30
20
10
0
10^{-1}
10^{0}


10
9
8
7
6
5
4
3
2
1
0
10^{-1}
10^{0}

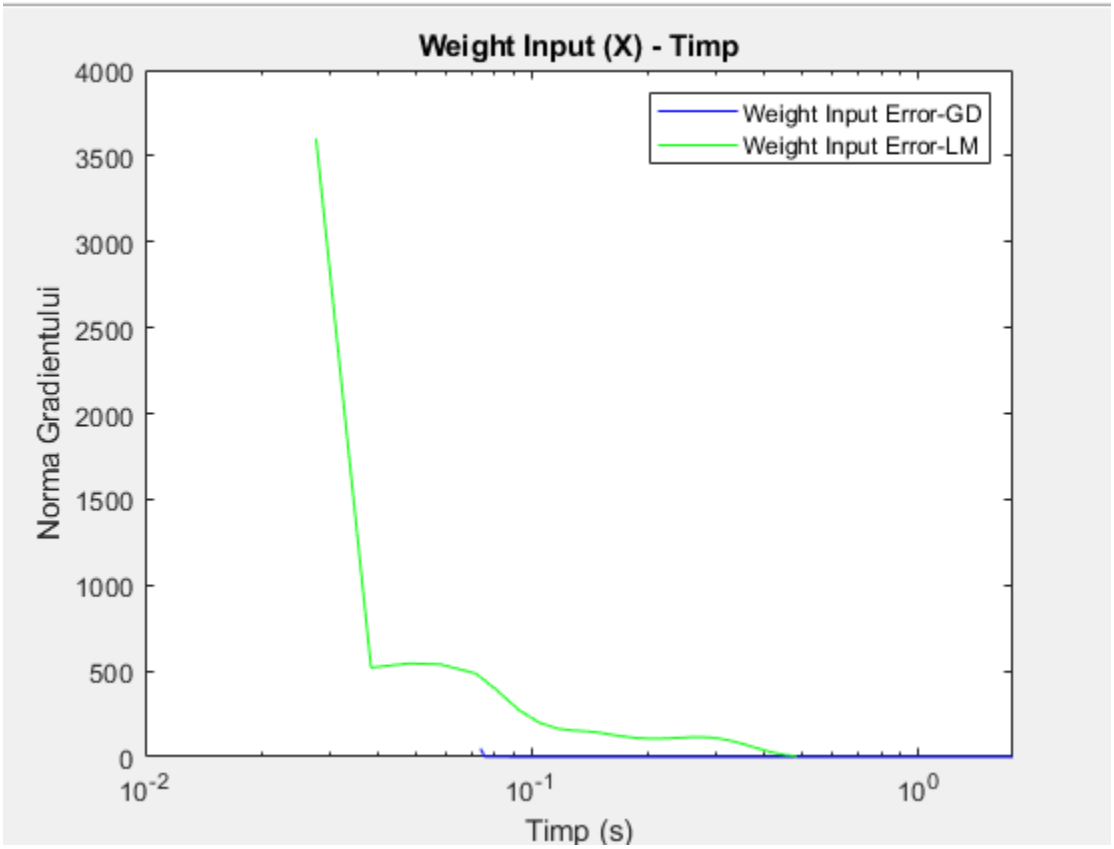
5.2.2 LM

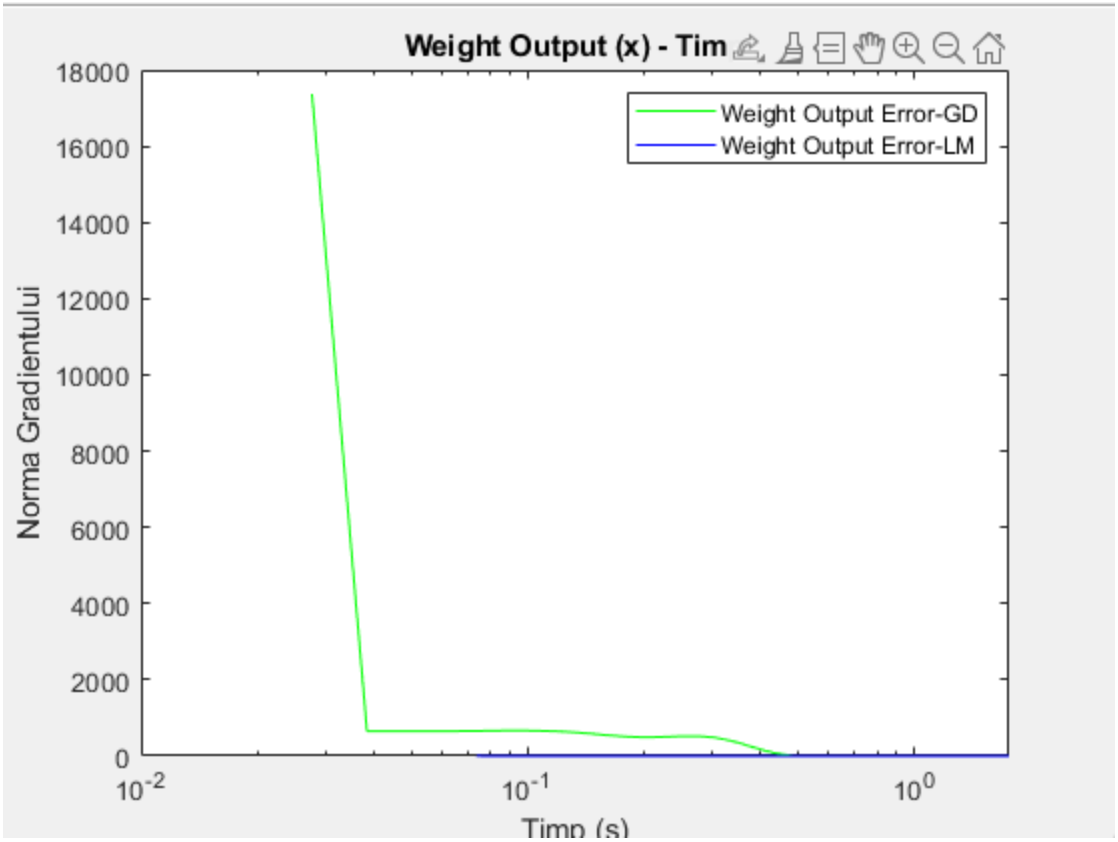
LM necesită mai mult timp de execuţie, în principal din cauza complexităţii calculului Jacobianului şi a inversării matriciale.




18000
16000
14000
12000
10000
8000
6000
4000
2000
0
10^{-1}

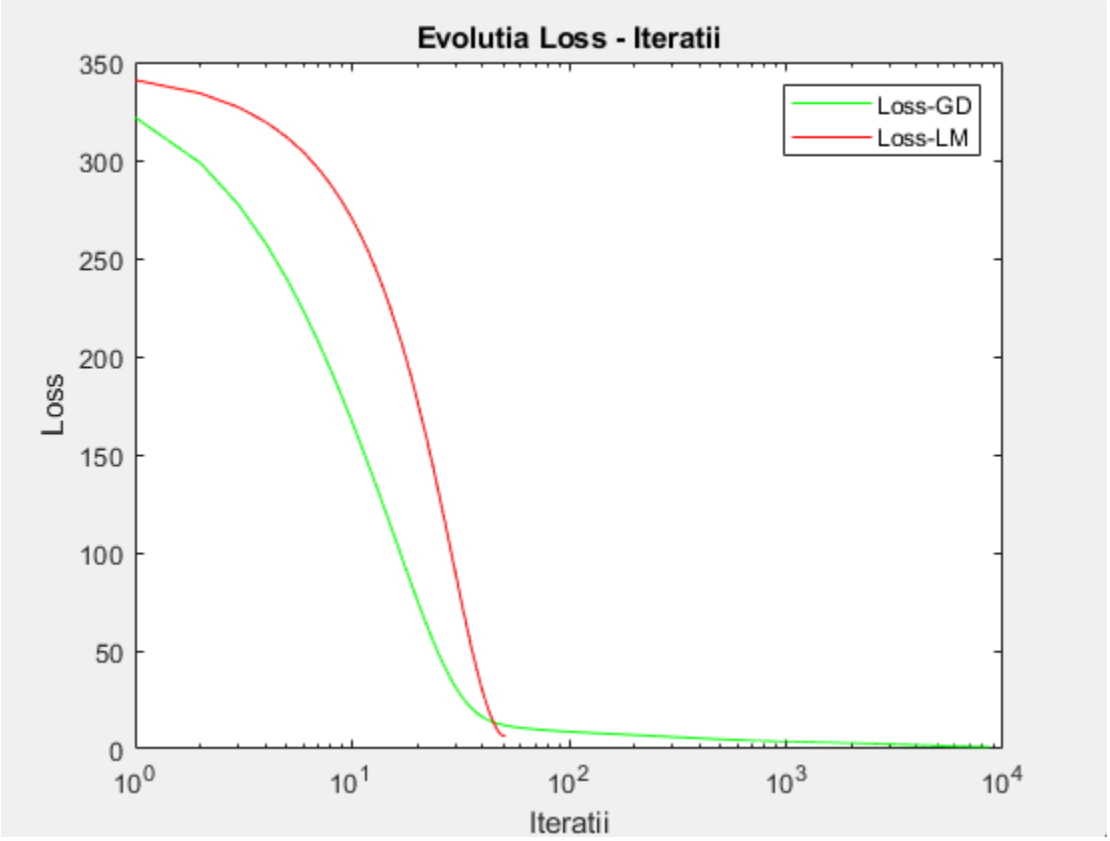
5.2.3 Comparație Metode



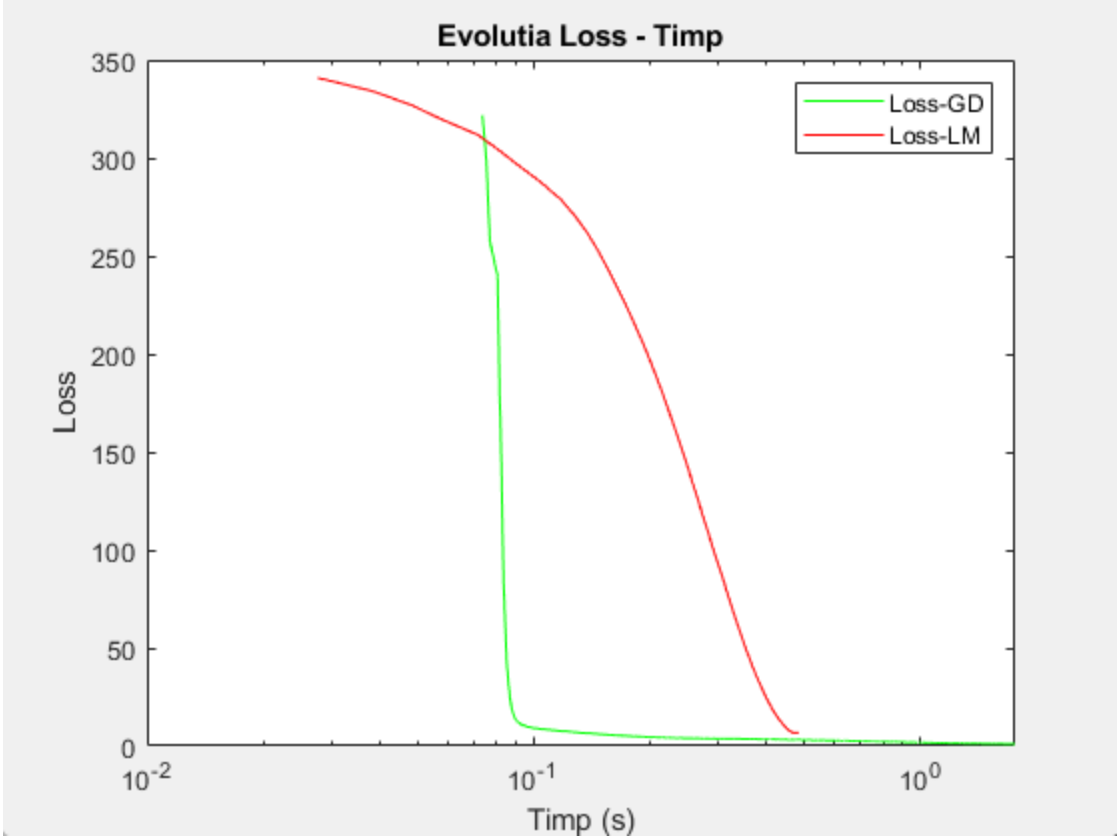


- GD: mai rapid ca timp de execuție, dar mai slab în zona minimului.
- LM: mai lent ca timp, dar mai eficient în optimizare.

5.3 Evoluția funcției obiectiv



LM are nevoie de mai puţine iteraţii pentru atingerea minimului.



Pentru LM atingerea minimului necesita un timp mai mare de execuţie.

- LM ajunge la valori mai mici ale funcţiei obiectiv în mai puţine iteraţii.
- Totuşi, timpul necesar este mai lung decât în cazul GD.

6. Evaluare performante

```
R^2 (Gradient Descent):         0.9793
R^2 (Levenberg-Marquardt):      0.8790
```

Metoda GD are o predicţie mai buna a datelor.

7. Concluzii

- Gradient Descent este o metodă robustă şi uşor de implementat, oferind performanţe bune, dar convergenţă lentă în apropierea minimului.
- Levenberg–Marquardt prezintă o convergenţă rapidă şi eficientă în jurul soluţiei optime, dar presupune un cost computaţional ridicat.

8. Anexa 1 – Matlab

8.1 Script.m

```
clc;
clear;

%% 1. Incarcare date
data = readtable('ENB2012_data.xlsx');
A = table2array(data(:, 1:8));     % Caracteristici (features)
HL = table2array(data(:, 9));      % Heating Load (target)

%% 2. Normalizare caracteristicilor
A = normalize(A);

%% 3. Impartire train/test (80% / 20%)
cv = cvpartition(size(A,1), 'HoldOut', 0.2);
A_train = A(training(cv), :);
HL_train = HL(training(cv));
A_test  = A(test(cv), :);
HL_test = HL(test(cv));

%% 4. Adaugare bias (coloana de 1) la final
A_train = [A_train, ones(size(A_train, 1), 1)];
A_test  = [A_test,  ones(size(A_test, 1),  1)];

%% 5. Initializare date
[n_measure, n_features] = size(A_train);  % n_features include bias
m_neurons = 16;

X = randn(n_features, m_neurons);     % greutati input → hidden
x = randn(m_neurons, 1);              % greutati hidden → output

%% 6. Gradient Descent

%Initializare date GD
eps_gd = 1e-5;
iter1 = 1;
maxIter1 = 150000;

x_gd = x;
x_gd_new = zeros(m_neurons, 1);
X_gd = X;
X_gd_new = zeros(n_features, m_neurons);

condition_gd = ones(2,1);
alfa_gd = 0.01;

eroare_gd_input = [];
eroare_gd_output = [];

loss_function = [];
```

```
time_gd = [];

grad_x_old_gd=0;
grad_X_old_gd=0;

while (condition_gd(1) > eps_gd || condition_gd(2) > eps_gd) && iter1 <
maxIter1

    tic;

    %Calcul gradienti
    grad_x_gd=gradient_x_big(A_train, X_gd, x_gd, HL_train);
    grad_X_gd=gradient_x(A_train, X_gd, x_gd, HL_train);

    %Aplicare metoda
    X_gd_new = X_gd - alfa_gd * grad_x_gd;
    x_gd_new = x_gd - alfa_gd * grad_X_gd;

    %Calcul conditie oprire
    condition_gd(1) = norm(grad_X_old_gd-grad_X_gd);
    condition_gd(2) = norm(grad_x_old_gd-grad_x_gd);

    fprintf("Iteratia %d: ||dgrad(X)|| = %.6e, ||dgrad(x)|| = %.6e\n", iter1,
condition_gd(1), condition_gd(2));

    eroare_gd_input  = [eroare_gd_input,  condition_gd(1)];
    eroare_gd_output = [eroare_gd_output, condition_gd(2)];

    % Evalueaza predictia cu X_gd_new și x_gd_new
    y_pred = siglin(A_train * X_gd_new) * x_gd_new;

    % Evalueare loss
    val_loss = loss(HL_train, y_pred);
    loss_function = [loss_function, val_loss];

    % Actualizare date
    iter1 = iter1 + 1;

    X_gd = X_gd_new;
    x_gd = x_gd_new;

    grad_x_old_gd=grad_x_gd;
    grad_X_old_gd=grad_X_gd;

    time_gd = [time_gd, toc];
end

%% Timp Total
total_time_gd=cumsum(time_gd);
%% 7. Metoda Levenberg-Marquardt
%Initializare date LM
eps_lm = 1e-3;
iter2 = 1;
maxIter2 = 150000;
```

```
x_lm = x;
x_lm_new = zeros(m_neurons, 1);
X_lm = X;
X_lm_new = zeros(n_features, m_neurons);

condition_lm = ones(2,1);
alfa_lm = 0.01;
beta_lm = 10;

eroare_lm_input = [];
eroare_lm_output = [];
loss_function_lm = [];
time_lm = [];

grad_x_old_lm = 0;
grad_X_old_lm = 0;

while (condition_lm(1) > eps_lm || condition_lm(2) > eps_lm) && iter2 <
maxIter2
    tic;

    % Loss initial
    y_pred = siglin(A_train * X_lm) * x_lm;
    val_loss = loss(HL_train, y_pred);

    % Jacobieni
    [F_x, J_x, F_X, J_X] = jacobian(@loss, X_lm, x_lm, A_train, HL_train);

    % Gradient LM
    grad_x_lm = J_x' * F_x;
    grad_X_lm = J_X' * F_X;

    % Update x
    x_lm_new = x_lm - alfa_lm * ((J_x' * J_x + beta_lm * eye(m_neurons)) \
grad_x_lm);

    % Update X
    X_lm_vec = X_lm(:);
    X_lm_vec = X_lm_vec - alfa_lm * ((J_X' * J_X + beta_lm * eye(n_features *
m_neurons)) \ grad_X_lm);
    X_lm_new = reshape(X_lm_vec, n_features, m_neurons);

    % Variatia gradient -> conditie oprire
    condition_lm(1) = norm(grad_X_old_lm - grad_X_lm);
    condition_lm(2) = norm(grad_x_old_lm - grad_x_lm);

    fprintf("Iteratia %d: ||dgrad(X)|| = %.6e, ||dgrad(x)|| = %.6e\n", iter2,
condition_lm(1), condition_lm(2));

    eroare_lm_input  = [eroare_lm_input, condition_lm(1)];
    eroare_lm_output = [eroare_lm_output, condition_lm(2)];

    % Evaluare loss dupa update
    y_pred_new = siglin(A_train * X_lm_new) * x_lm_new;
    val_loss_new = loss(HL_train, y_pred_new);
```

```
    % Adaptare beta și alfa
    if val_loss_new < val_loss
        beta_lm = beta_lm / 1.5; % Crestem efectul metodei Newton
        alfa_lm = alfa_lm * 1.05;
    else
        beta_lm = beta_lm * 1.5;
        alfa_lm = alfa_lm * 0.5; % Crestem efectul metodei Gradient

        % Revenire la vechea stare
        X_lm_new = X_lm;
        x_lm_new = x_lm;
        y_pred_new = y_pred;
        val_loss_new = val_loss;
    end

    % Update general
    loss_function_lm = [loss_function_lm, val_loss_new];
    X_lm = X_lm_new;
    x_lm = x_lm_new;

    grad_x_old_lm = grad_x_lm;
    grad_X_old_lm = grad_X_lm;

    time_lm = [time_lm, toc];
    iter2 = iter2 + 1;
end

%% Timp Total
total_time_lm=cumsum(time_lm);
%% 8. Grafic Eroare Weight Input & Output Iteratii

%Weight Input (X) Error Comparatie
figure('Name','Weight Input');
grid on;
semilogx(1:iter1-1,eroare_gd_input(1:iter1-1),'b');
hold on;
semilogx(1:iter2-1,eroare_lm_input(1:iter2-1),'r');
xlabel('Iteratii'); ylabel('Norma Gradientului');
legend('Weight Input Error-GD','Weight Input Error-LM');
hold off;
title('Weight Input (X) Comparatie Iter');

%Weight Output (x) Error Comparatie
figure('Name','Weight Outpu');
semilogx(1:iter1-1,eroare_gd_output(1:iter1-1),'b');
hold on
semilogx(1:iter2-1,eroare_lm_output(1:iter2-1),'r');
xlabel('Iteratii'); ylabel('Norma Gradientului');
legend('Weight Output Error-GD','Weight Output Error-LM');
hold off;
title('Weight Output (x) Comparatie Iter')

%% 9. Grafic Eroare Weight Input & Output Timp
```

```
figure;
grid on;
semilogx(total_time_gd,eroare_gd_input,'b');
hold on;
semilogx(total_time_lm,eroare_lm_input,'g');
hold off;
xlabel('Timp (s)'); ylabel('Norma Gradientului');
legend('Weight Input Error-GD','Weight Input Error-LM');
title('Weight Input (X) - Timp')


figure;
grid on;
semilogx(total_time_lm,eroare_lm_output,'g');
hold on;
semilogx(total_time_gd,eroare_gd_output,'b');

hold off;
xlabel('Timp (s)'); ylabel('Norma Gradientului');
legend('Weight Output Error-GD','Weight Output Error-LM');
title('Weight Output (x) - Timp')


%% 10. Evolutia Loss Gradient Descent Iteratii
figure
semilogx(1:iter1-1,loss_function(1:iter1-1),'g');
hold on;
semilogx(1:iter2-1,loss_function_lm(1:iter2-1),'r');
xlabel('Iteratii'); ylabel('Loss');
legend('Loss-GD','Loss-LM')
title('Evolutia Loss - Iteratii')
hold off;
%% 11. Evolutia Loss Gradient Descent Timp
figure
semilogx(total_time_gd,loss_function,'g');
hold on;
semilogx(total_time_lm,loss_function_lm,'r');
xlabel('Timp (s)'); ylabel('Loss');
legend('Loss-GD','Loss-LM')
title('Evolutia Loss - Timp')
hold off;

%% 12. Testare Gradient Descent

y_pred_gd = siglin(A_test * X_gd) * x_gd;
R_gd=R2(y_pred_gd,HL_test); % Performanta Gradient Descent

%% 13. Testare Levenberg-Marquardt

y_pred_lm = siglin(A_test * X_lm) * x_lm;
R_lm=R2(y_pred_lm,HL_test); % Performanta Levenberg-Marquardt

%% 14. Performante finale
fprintf("\nR^2 (Gradient Descent):        %.4f\n", R_gd);
fprintf("R^2 (Levenberg-Marquardt):     %.4f\n", R_lm);
```

9. Bibliografie

Gradient descent - Wikipedia

Baza de date : Energy Efficiency - UCI Machine Learning Repository

Articol teoretic : Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools - ScienceDirect

Levenberg–Marquardt algorithm - Wikipedia

Curs Optimizari – Ion Necoara 2022

Least-Squares (Model Fitting) Algorithms

Stanford ENGR108: Introduction to Applied Linear Algebra | 2020 | Lecture 51-VMLS Leven. Marq. algo

What is a Loss Function? Understanding How AI Models Learn

Activation Functions In Neural Networks Explained | Deep Learning Tutorial