# 2 Linear Regression *(40p)*

## 2.1 Enunț

Având o pasiune profundă și pentru **Metode Numerice**, Mihai este interesat atât de Învățarea Automată[4], cât și de Inteligența Artificială[5], și i-ar plăcea să le exploreze mai în detaliu (atât cât poate). Curios din fire, acesta începe să citească despre cum poate să proiecteze un model de învățare automată care să se antreneze pe baza unui set de date existent ce are o anumită dimensiune.

Cu ajutorul unui algoritm de Învățare Automată Supervizată[6], numit în literatura de specialitate **Linear Regression**, Mihai dorește să înțeleagă mai multe despre manipularea **predicțiilor** și a **erorilor**[7] ce pot să apară în prelucrarea computațională.

În esență, **Linear Regression** poate fi interpretat geometric drept o dreaptă (la ALGAED ați întâlnit noțiunea de *dreaptă de regresie*) care **minimizează** radicalul sumei pătratelor distanțelor punctelor (datelor) ce fac parte dintr-o mulțime de interes[8].

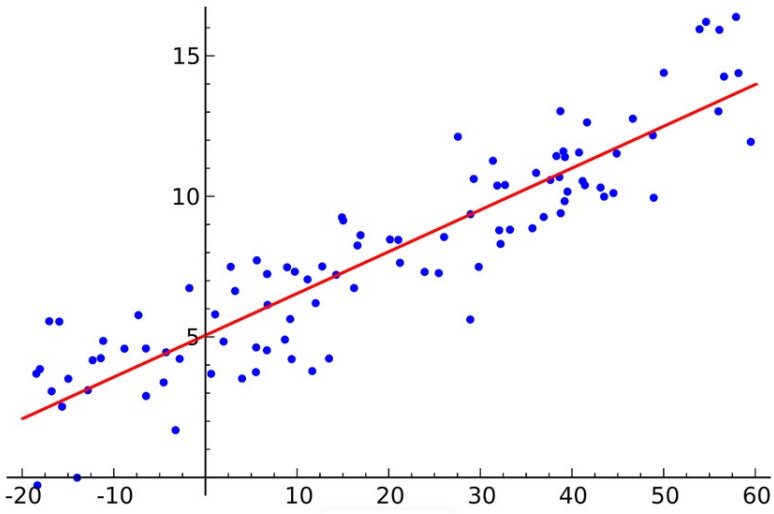


Figura 8: Reprezentarea grafică a unei drepte de regresie ce trece printr-un set de date.

Din punct de vedere funcțional, **Linear Regression** se ocupă de **micșorarea**, până într-o anumită limită, a **funcției de cost** și a **pierderii** (aceste concepte vor fi detaliate în paragrafele ce urmează). Evident, există mai multe tipuri de **Linear Regression** (precum regresia simplă, regresia multiplă și cea logistică).

În urma cercetărilor sale, Mihai se hotărăște să folosească **Multiple Linear Regression** pentru a putea face **predicții** cu privire la prețul apartamentelor din zona sa, întrucât nu mai dorește să locuiască cu ai lui, vrând să își manifeste independența față de ei.

O astfel de predicție poate fi scrisă sub forma unei funcții $h_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}$, cu $\boldsymbol{\theta} \in \mathbb{R}^{n+1}$, funcție ce se poate

[4] en. *Machine Learning*
[5] en. *Artificial Intelligence*
[6] en. *Supervised Machine Learning*
[7] Conceptele de *bias* și *variance*
[8] Acest fenomen este cunoscut și drept *aproximare în sensul celor mai mici pătrate* și va fi studiat în cadrul metodelor numerice funcționale (a doua parte a materiei).

defini după cum urmează:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n + \varepsilon$$

În scrierea anterioară, am folosit următoarele notații:

- $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ reprezintă valoarea **prezisă** pentru funcționalitățile $(x_1, x_2, \ldots, x_n)$ (acestea se mai numesc și **predictori** sau **features**);
- $\theta_1, \ldots, \theta_n \in \mathbb{R}$ reprezintă coeficienții specifici modelului de învățare automată (aceștia mai poartă denumirea de **weights**);
- $\theta_0 \in \mathbb{R}$ reprezintă valoarea lui $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ atunci când toți predictorii sunt 0, adică $\boldsymbol{x} = \mathbf{0}$ (în literatură poartă numele de **intercept**);
- $\varepsilon \in \mathbb{R}$ este eroarea (diferența **în modul**) dintre valoarea prezisă și cea actuală a lui $h_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Ei bine, acești coeficienți $\theta_0, \theta_1, \ldots, \theta_n$ ce formează $\boldsymbol{\theta}$ descriu cât de capabil este un model de învățare automată pentru a face predicții cât mai bune (apropiate de realitate) după primirea de date noi, ce nu au mai fost *văzute* de către acesta. Putem așadar să definim **funcția de cost**, o funcție ce returnează eroarea dintre valoarea actuală și cea prezisă, și să încercăm să o **minimizăm**.

Funcția de cost $J : \mathbb{R}^{n+1} \to \mathbb{R}$ va avea următoarea scriere:

$$J(\boldsymbol{\theta}) = J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left[ h_{\boldsymbol{\theta}} \left( \boldsymbol{x}^{(i)} \right) - y^{(i)} \right]^2$$

Am utilizat următoarele notații în scrierea de mai sus:

- $\theta_1, \ldots, \theta_n \in \mathbb{R}$ reprezintă coeficienții specifici modelului, la fel ca mai sus;
- $m \in \mathbb{N}^*$ reprezintă numărul de antrenamente[9];
- $\boldsymbol{x}^{(i)}$ reprezintă intrările pentru antrenamentele de ordin $i \in \mathbb{N}$[10], ceea ce înseamnă că $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$ este ipoteza (valoarea prezisă) pentru antrenamentul cu indexul $i$;
- $y^{(i)}$ reprezintă ieșirile pentru antrenamentele de ordin $i \in \mathbb{N}$.

**NU** confundați notația $\gamma^{(i)}$ cu ridicarea la putere sau cu derivarea! Facem referire strict la indexul (numărul) iterației curente.

### 2.1.1 Algoritmi de optimizare

Pe parcursul studiului său, Mihai a mai descoperit și faptul că există anumiți algoritmi de optimizare pentru a determina coeficienții modelului, și anume **metoda gradientului descendent**[11], respectiv **Normal Equation**.

**Metoda gradientului descendent** reprezintă o modalitate generală pentru optimizarea funcțiilor convexe (în cazul nostru, o vom aplica funcției de cost), ce poate determina minimul local al funcției de interes. Metoda utilizează o tehnică iterativă.

Având în vedere că funcția de cost $J(\boldsymbol{\theta})$ are un **minim global unic**, putem spune că orice minim local este, de asemenea, un minim global; cu alte cuvinte, funcția de cost este **convexă**, iar acest lucru ne garantează faptul că orice metodă de optimizare va converge către minimul global al funcției de cost.

Această metodă își efectuează pașii în funcție de gradientul funcției de cost și de valoarea aleasă pentru rata de învățare, notată la noi cu $\alpha \in \mathbb{R}$.

[9] en. *training samples*
[10] en. *$i^{th}$ training example*
[11] en. *Gradient Descent*

Amintim că prin gradientul funcției de cost înțelegem vectorul fomat din derivatele parțiale în raport cu $\theta_1, \ldots, \theta_n$. Cu alte cuvinte:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial J}{\partial \theta_n}(\boldsymbol{\theta}) \end{bmatrix}, \text{ unde } \frac{\partial J}{\partial \theta_j}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left[ h_{\boldsymbol{\theta}} \left( \boldsymbol{x}^{(i)} \right) - y^{(i)} \right] \cdot \boldsymbol{x}_j^{(i)}, \forall j \in \overline{1, n}$$

Transformarea pe care această metodă o propune este dată de relația:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J}{\partial \theta_j}(\boldsymbol{\theta}), \forall j \in \overline{1, n}$$

**Normal Equation** reprezintă o metodă care implică o ecuație directă pentru a determina coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$ specifici modelului de interes. Această tehnică este utilă în situația în care lucrăm cu seturi restrânse (mici) de date. Se cristalizează următoarea ecuație:

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T Y$$

unde:

- $X \in \mathbb{R}^{m \times n}$ reprezintă matricea ce stochează **m** vectori linie $\boldsymbol{x}^{(i)}$, $i \in 1, 2, \ldots, m$, fiecare vector linie având **n** valori specifici predictorilor.
- $Y \in \mathbb{R}^{m \times 1}$ reprezintă vectorul **coloană** ce reține **m** valori **actuale**.
- $\boldsymbol{\theta} \in \mathbb{R}^{n \times 1}$ reprezintă vectorul **coloană** ce reține **n** coeficienți $\theta_1, \ldots, \theta_n \in \mathbb{R}$ specifici modelului de învățare automată.

O problemă vizibilă cu acest algoritm este că determinarea inversei unei matrice implică un cost computațional ridicat pentru seturi mari de date. Pentru a mitiga această dificultate, vom folosi o altă metodă *(prezentată la curs)* pentru a rezolva sistemul, anume **metoda gradientului conjugat**[12].

Reamintim algoritmul în cauză:

**Conjugate Gradient Method**

**procedure** CONJUGATE_GRADIENT(A, b, x_0, tol, max_iter)
  $r^{(0)} \leftarrow b - Ax^{(0)}$
  $v^{(1)} \leftarrow r^{(0)}$
  $x \leftarrow x_0$
  $tol_{squared} \leftarrow tol^2$
  $k \leftarrow 1$
  **while** $k <= max_{iter}$ and ${r^{(\text{k-1})}}^T r^{(\text{k-1})} > tol_{squared}$ **do**
    $t_k \leftarrow \frac{{r^{(\text{k-1})}}^T r^{(\text{k - 1})}}{{v^{\text{k}}}^T A v^{\text{k}}}$
    $x^{(\text{k})} \leftarrow x^{(\text{k-1})} + t_k v^{(\text{k})}$
    $r^{(\text{k})} \leftarrow r^{(\text{k-1})} - t_k A v^{(\text{k})}$
    $s_k \leftarrow \frac{{r^{(\text{k})}}^T r^{(\text{k})}}{{r^{(\text{k-1})}}^T r^{(\text{k-1})}}$
    $v^{(\text{k+1})} \leftarrow r^{(\text{k})} + s_k v^{(\text{k})}$
    $k \leftarrow k + 1$
  **return** $x$

[12] **NU** uitați faptul că această metodă necesită ca matricea sistemului să fie **pozitiv definită**.

### 2.1.2 Regularizare

În domeniul Învățării Automate, **regularizarea** reprezintă o metodă ce poate fi aplicată unui model de învățare automată astfel încât acesta să devină mult mai **general**, adică să aibă eroarea de **variață** cât mai mică după introducerea de **noi date** în urma antrenamentului său.

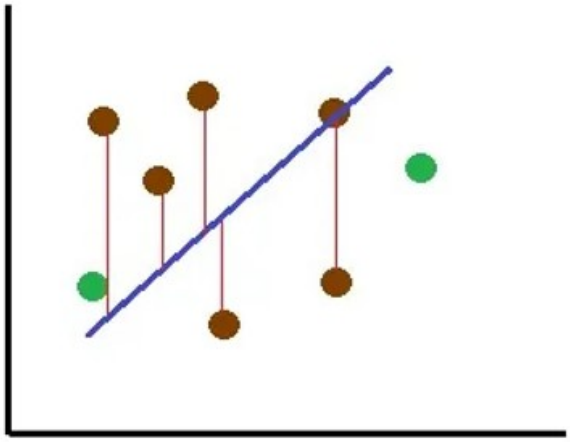

Figura 9: Reprezentarea grafică a unei drepte de regresie ce trece printr-un set de date de antrenament (punctele verzi) și printr-un set de date de testare (punctele maro). Se poate observa eroarea de variață (radicalul sumei pătratelor distanțelor punctelor maro)

Având în vedere cele menționate, Mihai este interesat în două tehnici de regularizare, **Regularizarea L1**, respectiv **Regularizarea L2**.

**Regularizarea L2**, denumită și **Ridge Regression**, se referă la a găsi o dreaptă de regresie care să treacă optim prin punctele care definesc setul de date de testare, introducând, însă, o mică eroare de bias. Cu alte cuvinte, dreapta găsită nu va minimiza pe deplin radicalul sumei pătratelor distanțelor punctelor din setul de date de antrenament.

În esență, această metodă se axează pe micșorarea coeficienților $\theta_0, \theta_1, \ldots, \theta_n \in \mathbb{R}$ astfel încât aceștia să fie apropiați de 0, efectul fiind **slăbirea** dependenței dintre $y^{(i)}$ și anumiți $x_1, x_2, \ldots, x_n$ din $\boldsymbol{x}^{(i)}$, adică ieșirea de ordin $i \in \mathbb{N}$ **va depinde mai puțin de predictori**.

Funcția regularizată de cost $J_{\boldsymbol{L2}} : \mathbb{R}^{n+1} \to \mathbb{R}$ va avea următoarea scriere:

$$J_{\boldsymbol{L2}}(\boldsymbol{\theta}) = J_{\boldsymbol{L2}}(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left[ h_{\boldsymbol{\theta}} \left( \boldsymbol{x}^{(i)} \right) - y^{(i)} \right]^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Unde:

- $\lambda \sum_{j=1}^{n} \theta_j^2$ reprezintă termenul specific regularizării L2;
- $\lambda \in \mathbb{R}_+$ este parametrul care controlează **puterea regularizării**, acesta se poate determina folosind tehnica cross-validation, însă noi îl vom oferi la partea de implementare.

**Regularizarea L1**, denumită și **Lasso Regression**, este similară cu regularizarea L2, cu excepția faptului că anumiți $\theta_0, \theta_1, \ldots, \theta_n \in \mathbb{R}$ pot fi chiar 0, adică **se poate elimina definitiv** dependența ieșirii de ordin $i \in \mathbb{N}$ de anumiți **predictori**. Scopul rămâne același, și anume micșorarea complexității modelului de învățare automată.

Funcția regularizată de cost $J_{\boldsymbol{L1}} : \mathbb{R}^{n+1} \to \mathbb{R}$ va adopta următoarea scriere:

$$J_{\boldsymbol{L1}}(\boldsymbol{\theta}) = J_{\boldsymbol{L1}}(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{m} \sum_{i=1}^{m} \left[ y^{(i)} - h_{\boldsymbol{\theta}} \left( \boldsymbol{x}^{(i)} \right) \right]^2 + \lambda \left\| \boldsymbol{\theta} \right\|_1$$

Unde:

- $\left\| \boldsymbol{\theta} \right\|_1$ reprezintă norma L1 a coeficienților modelului, adică $\left\| \boldsymbol{\theta} \right\|_1 = |\theta_0| + |\theta_1| + \cdots + |\theta_n|$.
- $\lambda \in \mathbb{R}_+$ este **parametrul** care controlează regularizarea.

### 2.1.3 Format CSV

Pentru realizarea funcției care implică metoda **gradientului descendent**, veți avea la dispoziție setul de date de antrenare în format CSV.

Pentru a exemplifica, ilustrăm tabelar primele 24 de intrări (doar 9 coloane din cele 13) din setul de date propus:

| Price | Area | Bedrooms | Bathrooms | Stories | Mainroad | Guestroom | Basement | Hot water |
|---|---|---|---|---|---|---|---|---|
| 13300000 | 7420 | 4 | 2 | 3 | yes | no | no | no |
| 12250000 | 8960 | 4 | 4 | 4 | yes | no | no | no |
| 12250000 | 9960 | 3 | 2 | 2 | yes | no | yes | no |
| 12215000 | 7500 | 4 | 2 | 2 | yes | no | yes | no |
| 11410000 | 7420 | 4 | 1 | 2 | yes | yes | yes | no |
| 10850000 | 7500 | 3 | 3 | 1 | yes | no | yes | no |
| 10150000 | 8580 | 4 | 3 | 4 | yes | no | no | no |
| 10150000 | 16200 | 5 | 3 | 2 | yes | no | no | no |
| 9870000 | 8100 | 4 | 1 | 2 | yes | yes | yes | no |
| 9800000 | 5750 | 3 | 2 | 4 | yes | yes | no | no |
| 9800000 | 13200 | 3 | 1 | 2 | yes | no | yes | no |
| 9681000 | 6000 | 4 | 3 | 2 | yes | yes | yes | yes |
| 9310000 | 6550 | 4 | 2 | 2 | yes | no | no | no |
| 9240000 | 3500 | 4 | 2 | 2 | yes | no | no | yes |
| 9240000 | 7800 | 3 | 2 | 2 | yes | no | no | no |
| 9100000 | 6000 | 4 | 1 | 2 | yes | no | yes | no |
| 9100000 | 6600 | 4 | 2 | 2 | yes | yes | yes | no |
| 8960000 | 8500 | 3 | 2 | 4 | yes | no | no | no |
| 8890000 | 4600 | 3 | 2 | 2 | yes | yes | no | no |
| 8855000 | 6420 | 3 | 2 | 2 | yes | no | no | no |
| 8750000 | 4320 | 3 | 1 | 2 | yes | no | yes | yes |
| 8680000 | 7155 | 3 | 2 | 1 | yes | yes | yes | no |
| 8645000 | 8050 | 3 | 1 | 1 | yes | yes | yes | no |

În acest caz, ieșirea (variabila $y$) reprezintă coloana **Price**, iar **predictorii** $x_1, x_2, \ldots, x_{12}$ sunt toate celelalte coloane.

## 2.2 Cerințe

Având în vedere expunerea suportului teoretic și problema ce se dorește a fi rezolvată, aveți de implementat următoarele funcții:

- `function [Y, InitialMatrix] = parse_data_set_file(file_path)`

  Funcția `parse_data_set_file` va primi o cale relativă către un fișier text unde se află datele pentru un set oarecare.

  Formatul fișierului de intrare va fi acesta:

```
m n
Y_11 x_11 x_12 x_13 ... x_1n
Y_21 x_21 x_22 x_23 ... x_2n
Y_31 x_31 x_32 x_33 ... x_3n
...
Y_m1 x_m1 x_m2 x_m3 ... x_mn
```

  În acest caz, $n$ este numărul de predictori, iar $m$ se refera la numărul vectorilor de predictori $x_1, x_2, \ldots, x_n$ și la dimensiunea vectorului Y de ieșire. *InitialMatrix* reprezintă o matrice cu tipuri de date **distincte**!, adică stochează atât tipuri numerice, cât și string-uri. Pentru a gestiona acest lucru, puteți folosi tipul **Cell** din **Octave**.

- `function [FeatureMatrix] = prepare_for_regression(InitialMatrix)`

  Funcția *prepare_for_regression* modelează matricea anterioară astfel încât să conțină doar tipuri **numerice**. Cu alte cuvinte, fiecare poziție din matrice ce conține string-ul 'yes' se înlocuiește cu tipul numeric (numărul) 1, iar fiecare poziție ce conține string-ul 'no' se înlocuiește cu tipul numeric (numărul) 0. Pentru pozițiile ce au aceste valori *'semi-furnished'*, *'unfurnished'* sau *'furnished'*, acestea se vor **descompune** în două poziții cu valori numerice de 0 și 1.
  Fie următoarele cazuri:
  - Dacă poziția are valoarea *'semi-furnished'*, atunci se va descompune în două poziții cu valorile 1 și 0.
  - Dacă poziția are valoarea *'unfurnished'*, atunci se va descompune în două poziții cu valorile 0 și 1.
  - Dacă poziția are valoarea *'furnished'*, atunci se va descompune în două poziții cu valorile 0 și 0.

  Exemplu:

```
no  0 yes semi-furnished
no  2 no  semi-furnished
yes 1 yes unfurnished
yes 2 no  furnished
yes 2 no  furnished
yes 1 yes semi-furnished
no  2 no  semi-furnished
```

  Înlocuind toate string-urile cu tipuri numerice, matricea de mai sus se va transforma în:

```
0 0 1 1 0
0 2 0 1 0
1 1 1 0 1
1 2 0 0 0
1 2 0 0 0
1 1 1 1 0
0 2 0 1 0
```

După înlocuirea tuturor pozițiilor cu valori numerice, rezultatul obținut trebuie salvat în variabila de ieșire *FeatureMatrix*. Se observă că s-a mai adăugat o coloană, prin urmare s-a mărit numărul de predictori cu 1.

- `function [Error] = linear_regression_cost_function(Theta, Y, FeatureMatrix)`

  Funcția *linear_ regression_ cost_ function* implementează funcția de cost, așa cum a fost descrisă în secțiunea teoretică, folosind cei doi vectori și o matrice:
  - **Theta**, care reprezintă un vector **coloană** format din coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$.
  - **FeatureMatrix**, care reprezintă o matrice ce reține valorile unor predictori (adică o linie **i** din această matrice reprezintă $\boldsymbol{x}^{(i)}$ descris în suportul teoretic).
  - **Y**, care reprezintă un vector **coloană** ce conține **valorile actuale**, adică **ieșirile** ce au un anumit ordin.

  Pentru simplificarea implementării, puteți omite termenul care indică eroarea din cadrul funcției $h_{\boldsymbol{\theta}}(\boldsymbol{x})$, iar $\theta_0$ îl puteți considera **0**.
  Se garantează faptul că dimensiunile argumentelor sunt **compatibile** pentru a prelucra funcția de cost.

- `function [InitialMatrix, Y] = parse_csv_file(file_path)`

  Funcția `parse_csv_file` va primi o cale relativă către fișierul .csv unde se află datele pentru setul **propus**.
  Formatul (parțial) al acestui fișier se află la pagina 18.
  Există funcții Octave pentru a parsa, cu ușurință, astfel de fișiere.

- `function [Theta] = gradient_descent(FeatureMatrix, Y, n, m, alpha, iter)`

  Funcția *gradient_ descent* calculează, folosind tehnica **gradientului descendent**, coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$ după efectuarea celor *iter* pași. Ca mai sus, $\theta_0 = 0$ (îl considerăm 0).
  De asemenea, vectorul de predictori $\boldsymbol{x}^{(i)}$ reprezintă linia **i** din matricea *FeatureMatrix*.
  Considerați această aproximație inițitală: $\theta_1 = 0, \theta_2 = 0, \ldots, \theta_n = 0$.
  Această funcție **se va testa** folosind setul de date din fișierul .csv (există funcții Octave pentru a parsa, cu ușurință, astfel de fișiere).

- `function [Theta] = normal_equation(FeaturesMatrix, Y, tol, iter)`

  Funcția *normal_ equation* calculează, cu ajutorul metodei **gradientului conjugat**, coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$. De asemenea, $\theta_0 = 0$.
  Această funcție trebuie să returneze un vector *Theta* cu toți coeficienții calculați.
  Dacă matricea *sistemului* nu este **pozitiv definită**, atunci *Theta* o să stocheze doar valori de 0 și o să fie returnat direct. Se garantează faptul că *iter* va fi ales în mod corespunzător.

- `function [Error] = lasso_regression_cost_function(Theta, Y, FeMatrix, lambda)`

Funcția *lasso_ regression_ cost_ function* implementează funcția de cost, așa cum a fost descrisă în secțiunea teoretică, folosind cei doi vectori, o matrice și un scalar de la intrare:

- **Theta**, care reprezintă un vector **coloană** format din coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$.
- **FeMatrix**, care reprezintă o matrice ce reține valorile unor predictori (adică o linie **i** din această matrice reprezintă $\boldsymbol{x}^{(i)}$ descris în suportul teoretic).
- **Y**, care reprezintă un vector **coloană** ce conține **valorile actuale**, adică **ieșirile** ce au un anumit ordin.
- $\lambda$, care reprezintă parametrul ce controlează **regularizarea**.

Pentru simplificarea implementării, puteți omite termenul care indică eroarea din cadrul funcției $h_{\boldsymbol{\theta}}(\boldsymbol{x})$, iar $\theta_0$ îl puteți considera **0**.
Se garantează faptul că dimensiunile argumentelor sunt **compatibile** pentru a prelucra funcția de cost.

- `function [Error] = ridge_regression_cost_function(Theta, Y, FeMatrix, lambda)`

Funcția *ridge_ regression_ cost_ function* implementează funcția de cost, așa cum a fost descrisă în secțiunea teoretică, folosind cei doi vectori, o matrice și un scalar de la intrare:

- **Theta**, care reprezintă un vector **coloană** format din coeficienții $\theta_1, \ldots, \theta_n \in \mathbb{R}$.
- **FeMatrix**, care reprezintă o matrice ce reține valorile unor predictori (adică o linie **i** din această matrice reprezintă $\boldsymbol{x}^{(i)}$ descris în suportul teoretic).
- **Y**, care reprezintă un vector **coloană** ce conține **valorile actuale**, adică **ieșirile** ce au un anumit ordin.
- $\lambda$, care reprezintă parametrul ce controlează **regularizarea**.

Pentru simplificarea implementării, puteți omite termenul care indică eroarea din cadrul funcției $h_{\boldsymbol{\theta}}(\boldsymbol{x})$, iar $\theta_0$ îl puteți considera **0**.
Se garantează faptul că dimensiunile argumentelor sunt **compatibile** pentru a prelucra funcția de cost.