Exercice de programmation n°1 :

Régression Linéaire

GAO BIN

11/11/2016

Introduction

Dans ce TP, on commerce par télécharger le Octave et on améliore le code pour bien marche.

1. Simple octave function

Dans cette exercice, on ajoute le code A=eye(5) dans le fichier warmUpExercise.m

```
function A = warmUpExercise()
%WARMUPEXERCISE Example function in octave
%   A = WARMUPEXERCISE() is an example function that returns the 5x5 identity matrix


% ============= YOUR CODE HERE ==============
% Instructions: Return the 5x5 identity matrix
%               In octave, we return values by defining which variables
%               represent the return values (at the top of the file)
%               and then set them accordingly.


A=eye(5);




% ===========================================


end
```

Après, pour observer les résultats de modifications, on tape <<ex1.m>> et on fait run ex1.m dans la <<fenêtre de commande>>, on peut obtenir une matrice 5x5 identité.

```
%% ==================== Part 1: Basic Function ====================
% Complete warmUpExercise.m
fprintf('Running warmUpExercise ... \n');
fprintf('5x5 Identity Matrix: \n');
warmUpExercise()
```

Le résultat :

```
Running warmUpExercise ...
5x5 Identity Matrix:
ans =

Diagonal Matrix

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1

Program paused. Press enter to continue.
```

Ci-dessus on appelle <<warmUpExercice>>. Si vous voulez quitter, taper ctrl-c va arrêter le programme.

1.1 Submitting Solutions

Après avoir terminé la première partie de l'exercice, on peut aller sur l'objectif et puis on peut améliorer le code pour arriver à la régression linéaire des points donnés, on a :

6.1101, 17.592
5.5277, 9.1302
8.5186, 13.662
7.0032, 11.854
5.8598, 6.8233
8.3829, 11.886
7.4764, 4.3483
8.5781, 12
6.4862, 6.5987
5.0546, 3.8166
5.7107, 3.2522
14.164, 15.505
5.734, 3.1551
8.4084, 7.2258
5.6407, 0.71618

Donc sur ces données, l'abscisse est la population d'une ville en dizaine de milliers d'habitants. L'ordonnée est le profit d'un camion de restauration dans cette ville en dizaine de milliers de dollars, le script ex1.m a déjà été mis en place pour charger ces données. Donc on doit afficher tous les points dans un graph en régression linéaire, on calcule l'équation de la doit qui se rapproche le plus de ces données.

2. Linear regression with one variable

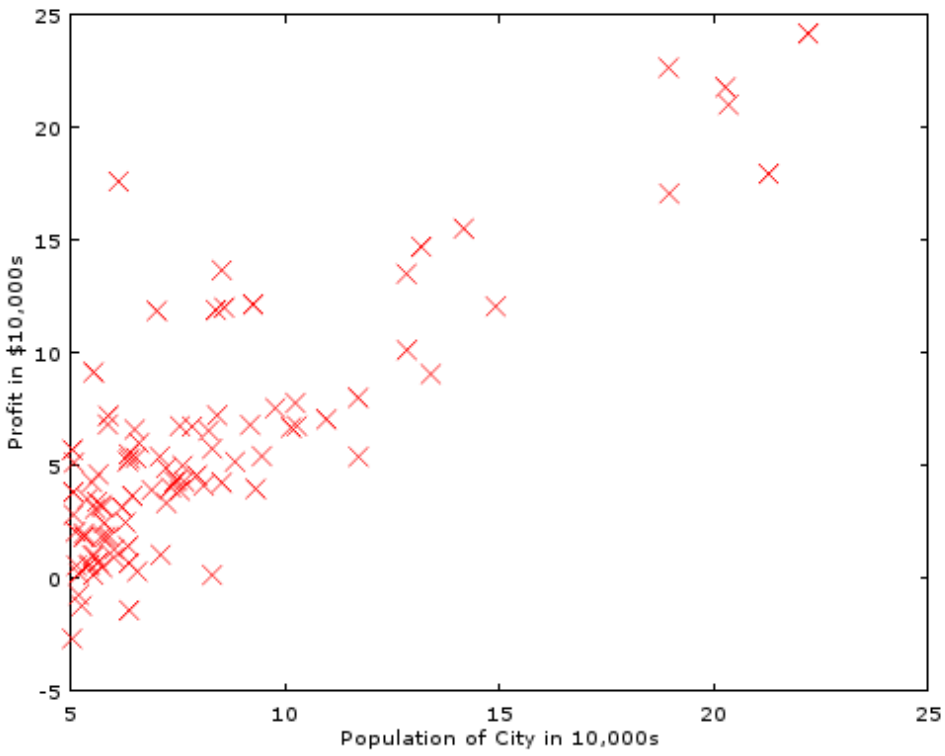
Le fichier ex1data1.txt contient l'ensemble de données pour notre problème de régression linéaire.

2.1 Plotting the Data

Et on affecte à X les valeurs de la première colonne et à Y les valeurs de la deuxième colonne. On a suivant :

```
function plotData(x, y)
%PLOTDATA Plots the data points x and y into a new figure
%   PLOTDATA(x,y) plots the data points and gives the figure axes labels of
%   population and profit.

% ====================== YOUR CODE HERE ======================
% Instructions: Plot the training data into a figure using the
%               "figure" and "plot" commands. Set the axes labels using
%               the "xlabel" and "ylabel" commands. Assume the
%               population and revenue data have been passed in
%               as the x and y arguments of this function.
%
% Hint: You can use the 'rx' option with plot to have the markers
%       appear as red crosses. Furthermore, you can make the
%       markers larger by using plot(..., 'rx', 'MarkerSize', 10);


plot(x,y,'rx','MarkerSize',10);
ylabel('Profit in $10,000s');
xlabel('Population of City in 10,000s');



% ============================================================

end
```

Donc quand on finit le code dans le fichier plotData.m, on peut obtenir les données X et Y dans un graph avec les légendes des axes x et y (ylabel et xlabel) :



Le script qui s'appelle PlotData, ce script pour créer un diagramme de dispersion des données.

2.2 Gradient Descent

Dans ce parti, On arrive à la régression linéaire on peut obtenir suivant :

```
%% ==================== Part 3: Gradient descent ====================
fprintf('Running Gradient Descent ...\n')

X = [ones(m, 1), data(:,1)]; % Add a column of ones to x
theta = zeros(2, 1); % initialize fitting parameters

% Some gradient descent settings
iterations = 1500;
alpha = 0.01;

% compute and display initial cost
computeCost(X, y, theta)

% run gradient descent
theta = gradientDescent(X, y, theta, alpha, iterations);

% print theta to screen
fprintf('Theta found by gradient descent: ');
fprintf('%f %f \n', theta(1), theta(2));

% Plot the linear fit
hold on; % keep previous plot visible
plot(X(:,2), X*theta, '-')
legend('Training data', 'Linear regression')
hold off % don't overlay any more plots on this figure

% Predict values for population sizes of 35,000 and 70,000
predict1 = [1, 3.5] *theta;
fprintf('For population = 35,000, we predict a profit of %f\n',...
    predict1*10000);
predict2 = [1, 7] * theta;
fprintf('For population = 70,000, we predict a profit of %f\n',...
    predict2*10000);

fprintf('Program paused. Press enter to continue.\n');
pause;
```

2.2.1 Update Equations

On a cost fonction :

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Et on a hypothèse h(x) :

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Ce sont les valeurs que m'ajuster pour réduire au minimum le coût J (θ). Une façon de le faire est d'utiliser l'algorithme de descente de gradient de lot. Dans la descente gradient de lot, chaque itération effectue la mise à jour.

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad (\text{simultaneously update } \theta_j \text{ for all } j).$$

A chaque étape de descente gradient, le paramètre theta conduit le coût J (θ) à un minimum.
On peut reconnaitre la formule du gradient dans Octave :

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
%   theta = GRADIENTDESENT(X, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters

    % ====================== YOUR CODE HERE ======================
    % Instructions: Perform a single gradient step on the parameter vector
    %               theta.
    %
    % Hint: While debugging, it can be useful to print out the values
    %       of the cost function (computeCost) and gradient here.
    %


     x = X(:,2);
     h = theta(1) + (theta(2)*x);
     theta0= theta(1) - alpha * (1/m) * sum(h-y);
     theta1  = theta(2) - alpha * (1/m) * sum((h - y) .* x);
     theta = [theta0; theta1];



    % ============================================================

    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);

end

end

```

2.2.2 Implementation

Dans les lignes suivantes, nous ajoutons une autre dimension à nos données pour accueillir le terme θ0 d'interception. On initialise également les paramètres initiaux à 0 et l'alpha taux d'apprentissage à 0,01.

2.2.3 Computing the cost J(θ)

Lorsque on effectue la descente gradient pour apprendre minimiser la fonction de coût J (θ), on remplit le code dans le fichier computeCost.m, on a :

```octave
function J = computeCost(X, y, theta)
%COMPUTECOST Compute cost for linear regression
%   J = COMPUTECOST(X, y, theta) computes the cost of using theta as the
%   parameter for linear regression to fit the data points in X and y

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost of a particular choice of theta
%               You should set J to the cost.

predictions = X*theta;
sqrErrors   = (predictions - y).^2;
J = 1/(2*m) * sum(sqrErrors);


% =========================================================================

end
```

```
Running warmUpExercise ...
5x5 Identity Matrix:
ans =

Diagonal Matrix

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1

Program paused. Press enter to continue.
Plotting Data ...
Program paused. Press enter to continue.
Running Gradient Descent ...
ans =  32.073
Theta found by gradient descent: -3.630291 1.166362
For population = 35,000, we predict a profit of 4519.767868
For population = 70,000, we predict a profit of 45342.450129
Program paused. Press enter to continue.
```

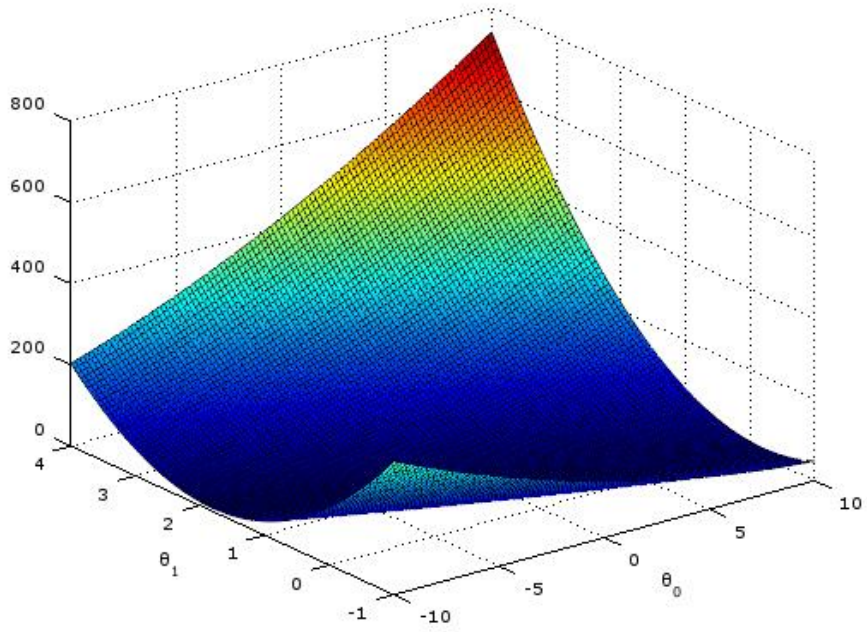
2.4 Visualizing J(θ)

Dans ce parti on a la programme suivant :

```
%% ============= Part 4: Visualizing J(theta_0, theta_1) =============
fprintf('Visualizing J(theta_0, theta_1) ...\n')

% Grid over which we will calculate J
theta0_vals = linspace(-10, 10, 100);
theta1_vals = linspace(-1, 4, 100);

% initialize J_vals to a matrix of 0's
J_vals = zeros(length(theta0_vals), length(theta1_vals));

% Fill out J_vals
for i = 1:length(theta0_vals)
    for j = 1:length(theta1_vals)
    t = [theta0_vals(i); theta1_vals(j)];
    J_vals(i,j) = computeCost(X, y, t);
    end
end


% Because of the way meshgrids work in the surf command, we need to
% transpose J_vals before calling surf, or else the axes will be flipped
J_vals = J_vals';
% Surface plot
figure;
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0'); ylabel('\theta_1');

% Contour plot
figure;
% Plot J_vals as 15 contours spaced logarithmically between 0.01 and 100
contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 3, 20))
xlabel('\theta_0'); ylabel('\theta_1');
hold on;
plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);

```

Après ces lignes sont exécutées,On crée une matrice de coût dans laquelle on rentre les valeurs de J (θ) avec les numéros de ligne et de colonnes correspondant aux deux theta.

Donc on a le graphe 3D avec les cordonnées θ0, θ1 et J (θ), on a suivant :



Le but de ces graphiques est montrer J (θ) avec les changements de θ0 et θ1, On avoir les deux paramètres pour lesquels on a un coût minimum, ce minimum est le point optimal pour θ0 et θ1, et chaque étape de descente gradient se rapproche de ce point.