# 1.代价函数

```
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.
```

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

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

1. 预测减去实际得到误差
2. 对所有误差的平方求和
3. 除以总的样本数量m，出于计算方便的考虑加上1/2

```
%
=========================================================================

end
```

# 2.梯度下降

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
%   theta = GRADIENTDESCENT(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.
    %
```

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

```
    a = theta(1);
    b = theta(2);
    a = a - alpha * (1/m) * sum((X (:, 1))' * (X * theta - y));
    b = b - alpha * (1/m) * sum((X (:, 2))' * (X * theta - y));
    theta = [a; b];
```

这里的重点在于a和b要同步更新，不能用a的新值去计算b

```
    % ============================================================

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

end

end
```

# 3.多元梯度下降

```
function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters)
%GRADIENTDESCENTMULTI Performs gradient descent to learn theta
%   theta = GRADIENTDESCENTMULTI(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 (computeCostMulti) and gradient here.
    %
    hypo = (1/m) * sum(repmat((X * theta - y), 1, size(X,2)) .* X);
    theta = theta - (alpha * hypo)';
```

这里有repmat函数来快速做出一个方便和矩阵X的每一项做乘法的矩阵，
这是一个巧妙的办法，避免了用for循环一个一个去计算theta的值

```
    % ============================================================

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

end

end
```