梯度下降法做线性回归:

估计函数:

$$\mathrm{h}(x) = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots$$

误差函数:

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

在选定线性回归模型后，只需要确定参数θ，就可以将模型用来预测。然而θ需要在 J(θ)最小的情况下才能确定。因此问题归结为求极小值问题，使用梯度下降法。梯度下降法最大的问题是求得有可能是全局极小值，这与初始点的选取有关。

梯度下降法是按面的流程进行:

1）首先对 θ 初始化，这个可以是随机的也让 θ 是一个全零的向量。

2）改变 θ 的值，使得 J(θ) 按梯度下降的方向减少

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}$$

单变量实现:

```
# coding: utf-8
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def plotData(x, y):
    plt.title('mytest')
    plt.xlabel('Population of City in 10,000s')
    plt.ylabel('Profit in $10,000s')
     plt.plot(x, y, 'rx')
     plt.show()

def gradientDescent(x, y, theta, alpha, num_iters):
    m = len(y)
    J_history = np.zeros((num_iters, 2))
    for i in range(num_iters):
        theta -= (alpha/m)*(np.dot(x.T, (np.dot(x, theta) - y)))
        J_history[i] = computeCost(x, y, theta)
    return theta, J_history

def computeCost(x, y, theta):
    m = len(y)
    return sum(np.square(np.dot(x, theta)-y))/(2*m)

def main():
    # 读入数据
    mydata = pd.read_csv("ex1data1.txt", header=None)
    x = np.array(mydata[0])
    y = np.array(mydata[1])
    m = len(y)
    # 画出数据图像
    plotData(x,y)
    theta = np.zeros((2, 1))
    x = np.c_[np.ones((m, 1)), x]
    y = y.reshape((len(y), 1))
    J = computeCost(x, y, theta)
    iterations = 1500
    alpha = 0.01
    theta, J_history = gradientDescent(x, y, theta, alpha, iterations)
    print theta
```

```
    plt.plot(J_history,'r')
    plt.show()



if __name__ == "__main__":
    main()
```

多变量实现:

```
# coding: utf-8
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def plotData(x, y):
    plt.title('mytest')
    plt.xlabel('Population of City in 10,000s')
    plt.ylabel('Profit in $10,000s')
    plt.plot(x, y, 'rx')
    plt.show()

def gradientDescent(x, y, theta, alpha, num_iters):
    m = len(y)
    # 用来存储历史损失
    J_history = np.zeros((num_iters, 2))
    for i in range(num_iters):
        # 梯度下降法求解
        theta -= (alpha / m) * (np.dot(x.T, (np.dot(x, theta) - y)))
        J_history[i] = computeCost(x, y, theta)
    return theta, J_history

def computeCost(x, y, theta):
    m = len(y)
    # 计算损失均方误差
    return sum(np.square(np.dot(x, theta) - y)) / (2 * m)

def featureNormalize(x):
    # 求均值与方差，注意 axis 的设置，控制求均值的方向（维度），如果不设置则为整体的均值/方差
```

```python
    mu = x.mean(axis=0)
    sigma = x.std(axis=0)
    return (x - mu)/sigma


def main():
    # 读入数据
    mydata = pd.read_csv("ex1data2.txt", header=None)
    # 选取前两列（Dataframe 如何切片）
    x = np.array(mydata.ix[:, 0:1])
    y = np.array(mydata[2])
    m = len(y)
    # 画出数据图像
    # plotData(x, y)
    theta = np.zeros((3, 1))
    x = featureNormalize(x)
    # numpy 添加一列
    x = np.c_[np.ones((m, 1)), x]
    # 变成列向量
    y = y.reshape((len(y), 1))
    J = computeCost(x, y, theta)
    iterations = 400
    alpha = 0.01
    theta, J_history = gradientDescent(x, y, theta, alpha, iterations)
    print theta
    plt.plot(J_history, 'r')
    plt.show()


if __name__ == "__main__":
    main()
```