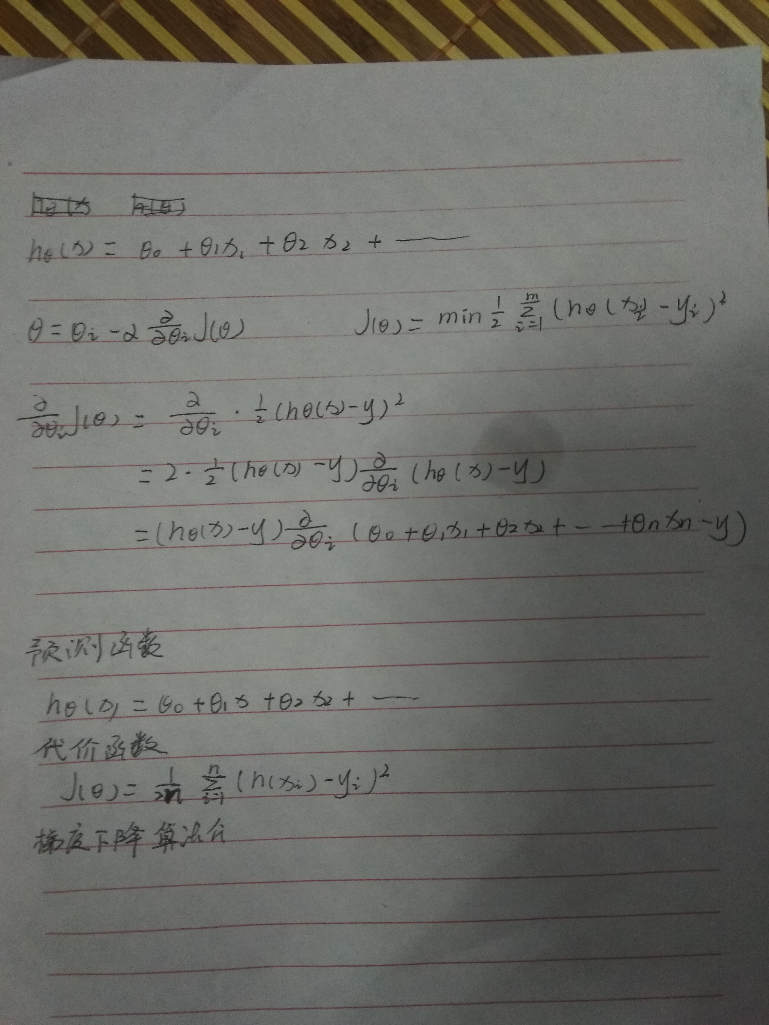
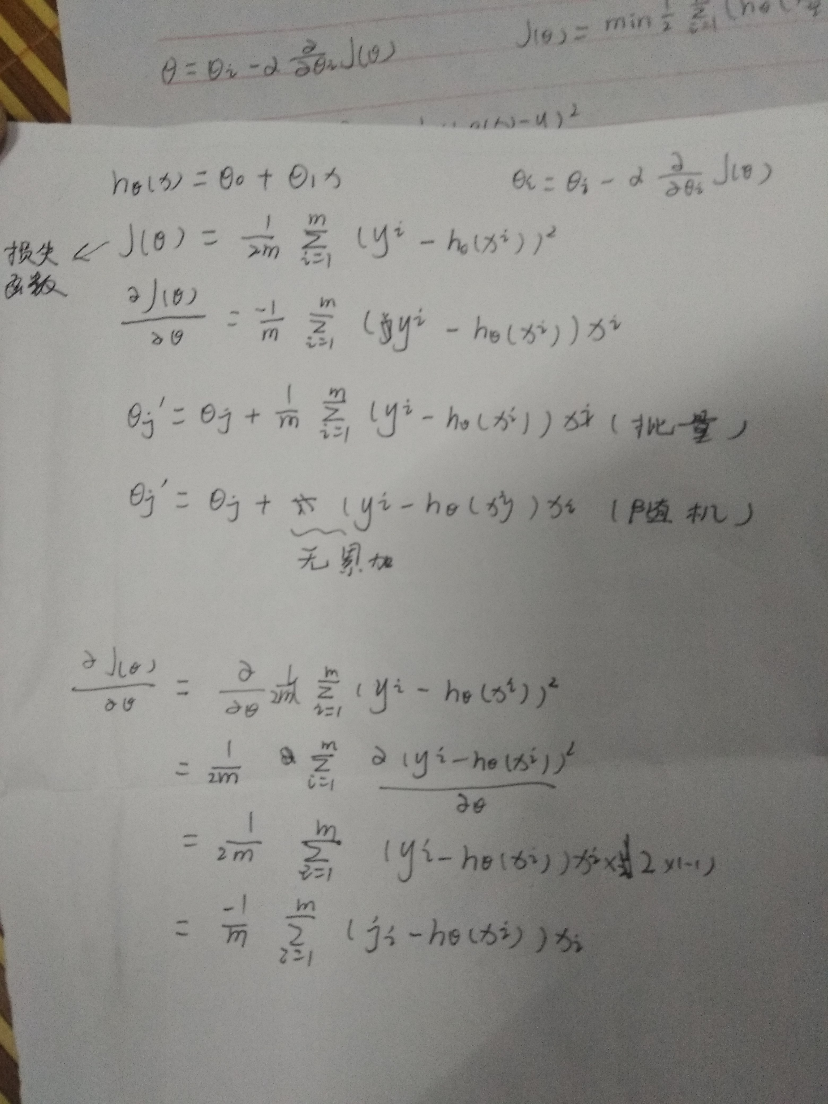
# 梯度下降

## 梯度下降实现

用到的公式的推导过程要写下来，无论是在纸上写（拍照），还是（最好是）电子文档写出来。

需要把下面两个数据集都要做实验



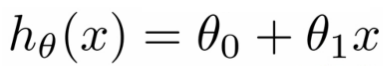


### 单变量线性回归问题

数据集1：房屋价格与面积（数据在下面表格中）

|  |  |  |
| --- | --- | --- |
| **序号** | **面积** | **价格** |
| 1 | 150 | 6450 |
| 2 | 200 | 7450 |
| 3 | 250 | 8450 |
| 4 | 300 | 9450 |
| 5 | 350 | 11450 |
| 6 | 400 | 15450 |
| 7 | 600 | 18450 |

使用梯度下降求解线性回归（求0，1）



要求：

1. 对关键代码进行注释；
2. 把每次迭代计算出来的theta0 ，theta1 画出来；
3. 分别用随机梯度和批量梯度实现，并在同一个图形中进行对比（颜色不同）。

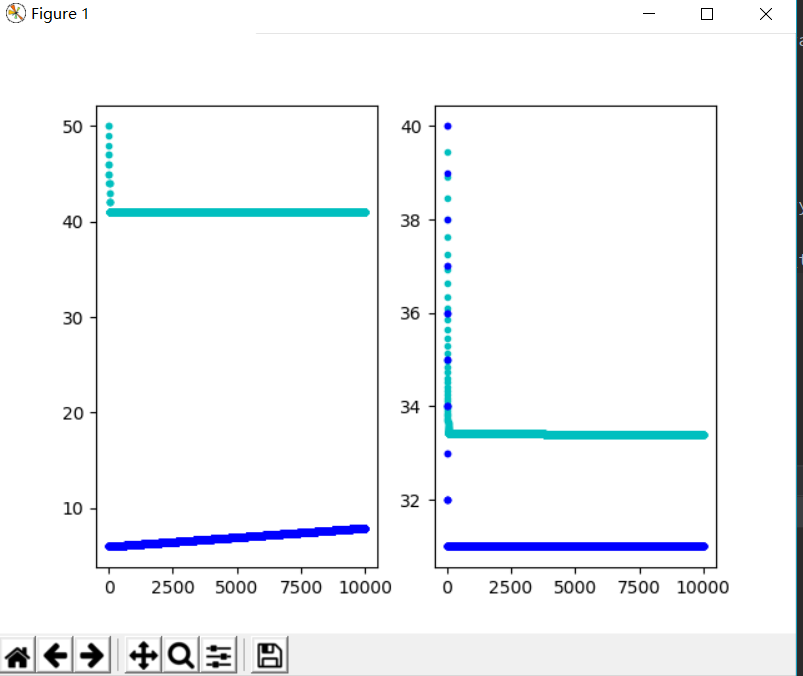
数据集2：实现功能与面上一致

|  |  |
| --- | --- |
| YearsExperience | Salary |
| 1.1 | 39343 |
| 1.3 | 46205 |
| 1.5 | 37731 |
| 2 | 43525 |
| 2.2 | 39891 |
| 2.9 | 56642 |
| 3 | 60150 |
| 3.2 | 54445 |
| 3.2 | 64445 |
| 3.7 | 57189 |
| 3.9 | 63218 |
| 4 | 55794 |
| 4 | 56957 |
| 4.1 | 57081 |
| 4.5 | 61111 |
| 4.9 | 67938 |
| 5.1 | 66029 |
| 5.3 | 83088 |
| 5.9 | 81363 |
| 6 | 93940 |
| 6.8 | 91738 |
| 7.1 | 98273 |
| 7.9 | 101302 |
| 8.2 | 113812 |
| 8.7 | 109431 |
| 9 | 105582 |
| 9.5 | 116969 |
| 9.6 | 112635 |
| 10.3 | 122391 |
| 10.5 | 121872 |

## 完整代码

数据集一：

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# 数据集大小  
m = 7  
#迭代次数  
iterations = 10000  
# 学习率  
alpha = 0.0000001  
  
X0 = np.ones((m, 1))  
X1 = np.array([150, 200, 250, 300, 350, 400, 600]).reshape(m, 1)  
X = np.hstack((X0, X1))  
print(X[6])  
y = np.array([  
 6450, 7450, 8450, 9450, 11450, 15450, 18450  
]).reshape(m, 1)  
  
def graFun\_Random(theta, X, y, temp):  
 diff = np.dot(X[temp], theta) - y[temp]  
 return diff\*X[temp]  
  
def gradient\_random(X, y, m):  
 theta = np.array([50,40]).reshape(2,1)#定义二维数组theta，其中包括theta0，theta1  
 history\_theta00 = np.zeros(iterations)  
 history\_theta11 = np.zeros(iterations)  
 for i in range(iterations):  
 temp = np.random.randint(0, m - 1)  
 gradient = graFun\_Random(theta, X, y, temp)  
 old = theta[1]  
 history\_theta00[i] = theta[0]  
 history\_theta11[i] = theta[1]  
 theta[0] = theta[0] - alpha \* gradient[0]  
 theta[1] = theta[1] - alpha \* gradient[1]  
  
 print('theta', theta)  
 return theta, history\_theta00, history\_theta11  
  
  
  
def gradient\_function(theta, X, y):  
 *'''求函数的梯度'''* diff = np.dot(X, theta) - y #dot函数求积，此处求预测值与真实值的差即误差函数 m\*1  
 return np.dot(np.transpose(X), diff)  
  
def gradient\_descent(X, y, alpha,iterations):  
 *'''  
 批量梯度下降  
 alpha：学习率  
 iterations：迭代次数  
 '''* theta = np.array([6, 40]).reshape(2, 1)  
 gradient = gradient\_function(theta, X, y)  
  
 #while(1000):  
 history\_theta0 = np.zeros(iterations)  
 history\_theta1 = np.zeros(iterations)  
 for i in range(iterations):  
 theta\_old = theta  
 #print('545454', theta)  
 theta = theta - alpha \* gradient  
 # print('gradient', gradient)  
 history\_theta0[i] = theta[0]  
 history\_theta1[i] = theta[1]  
 print('theta0 = ', theta[0], 'theta1 = ', theta[1])  
 gradient = gradient\_function(theta, X, y)  
  
 return theta, history\_theta0, history\_theta1  
  
optimal, history\_theta0, history\_theta1 = gradient\_descent(X, y, alpha,iterations)  
print('optimal:', optimal)  
optimal, history\_theta0, history\_theta1 = gradient\_descent(X, y, alpha,iterations)  
print('optimal:', optimal)  
optimal1, history\_theta00, history\_theta11 = gradient\_random(X, y , m)  
plt.ylabel('J(Theta)')  
plt.xlabel('Iterations')  
plt.subplot(121)  
plt.plot(range(iterations), history\_theta0, 'b.', range(iterations), history\_theta00, 'c.')  
plt.subplot(122)  
plt.plot(range(iterations),history\_theta1, 'c.', range(iterations),history\_theta11, 'b.')  
plt.show()



数据集二：

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# 数据集大小  
m = 30  
#迭代次数  
iterations = 10000  
# 学习率  
alpha = 0.001  
X0 = np.ones((m, 1))  
X1 = np.array([1.1, 1.3, 1.5, 2, 2.2, 2.9, 3, 3.2, 3.2, 3.7, 3.9, 4, 4,  
 4.1, 4.5, 4.9, 5.1, 5.3, 5.9, 6, 6.8, 7.1, 7.9, 8.2, 8.7,  
 9, 9.5, 9.6, 10.3, 10.5]).reshape(m, 1)  
X = np.hstack((X0, X1))  
print(X[6])  
y = np.array([39343, 46205, 37731, 43525, 39891, 56642, 60150, 54445, 64445, 57189, 63218,  
 55794, 56957, 57081, 61111, 67938, 66029, 83088, 81363, 93940, 91738,  
 98273, 101302, 113812, 109431, 105582, 116969, 112635, 122391, 121872  
  
]).reshape(m, 1)  
  
def graFun\_Random(theta, X, y, temp):  
 diff = np.dot(X[temp], theta) - y[temp]  
 return diff\*X[temp]  
  
def gradient\_random(X, y, m):  
 theta = np.array([100,10000]).reshape(2,1)#定义二维数组theta，其中包括theta0，theta1  
 history\_theta00 = np.zeros(iterations)  
 history\_theta11 = np.zeros(iterations)  
 for i in range(iterations):  
 temp = np.random.randint(0,m-1)  
 gradient = graFun\_Random(theta, X, y, temp)  
 old = theta[1]  
 history\_theta00[i] = theta[0]  
 history\_theta11[i] = theta[1]  
 theta[0] = theta[0] - alpha\*gradient[0]  
 theta[1] = theta[1] - alpha \* gradient[1]  
  
 print('theta', theta)  
 return theta, history\_theta00, history\_theta11  
  
  
  
def gradient\_function(theta, X, y):  
 *'''求函数的梯度'''* diff = np.dot(X, theta) - y #dot函数求积，此处求预测值与真实值的差即误差函数 m\*1  
 return np.dot(np.transpose(X), diff)  
  
def gradient\_descent(X, y, alpha,iterations):  
 *'''  
 批量梯度下降  
 alpha：学习率  
 iterations：迭代次数  
 '''* theta = np.array([100, 100000]).reshape(2, 1)  
 gradient = gradient\_function(theta, X, y)  
  
 #while(1000):  
 history\_theta0 = np.zeros(iterations)  
 history\_theta1 = np.zeros(iterations)  
 for i in range(iterations):  
 theta\_old = theta  
 #print('545454', theta)  
 theta = theta - alpha \* gradient  
 # print('gradient', gradient)  
 history\_theta0[i] = theta[0]  
 history\_theta1[i] = theta[1]  
 print('theta0 = ', theta[0], 'theta1 = ', theta[1])  
 gradient = gradient\_function(theta, X, y)  
  
 return theta, history\_theta0, history\_theta1  
  
optimal, history\_theta0, history\_theta1 = gradient\_descent(X, y, alpha,iterations)  
print('optimal:', optimal)  
optimal1, history\_theta00, history\_theta11 = gradient\_random(X, y , m)  
plt.ylabel('J(Theta)')  
plt.xlabel('Iterations')  
plt.subplot(121)  
plt.plot(range(iterations), history\_theta0, 'b.', range(iterations), history\_theta00, 'c.')  
plt.subplot(122)  
plt.plot(range(iterations),history\_theta1, 'c.', range(iterations),history\_theta11, 'b.')  
plt.show()