1. 给定一组数据(1, 9); (1.1, 10.5);(2, 18); (3, 28); (3.2, 30); (4, 37); (5, 48); (1.2, 10);

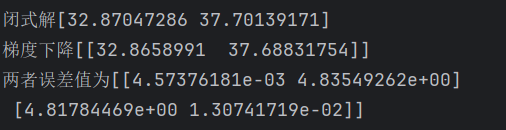
试用一元线性回归预测：当x取值为3.5 和 4时，y的值。

注：需要用闭式解和梯度下降两者方法，给出相应代码；对比两者之间的误差；

**代码部分：**

import numpy as np  
  
data = np.array([(1, 9), (1.1, 10.5), (2, 18), (3, 28), (3.2, 30), (4, 37), (5, 48), (1.2, 10)])  
  
X = data[:, 0]  
y = data[:, 1]  
  
# 截距项  
X\_b = np.c\_[np.ones((len(X), 1)), X]  
# 计算闭式解  
theta\_best = np.linalg.inv(X\_b.T.dot(X\_b)).dot(X\_b.T).dot(y)  
  
X\_new = np.array([[3.5], [4]])  
X\_new\_b = np.c\_[np.ones((2, 1)), X\_new] # 截距项  
y\_predict = X\_new\_b.dot(theta\_best)  
print(f"闭式解{y\_predict}")  
  
# 学习率  
alpha = 0.01  
# 迭代次数  
iterations = 1000  
m = len(X\_b)  
# 随机初始化参数  
theta = np.random.randn(2, 1)  
for iteration in range(iterations):  
 gradients = 2/m \* X\_b.T.dot(X\_b.dot(theta) - y.reshape(-1, 1))  
 theta = theta - alpha \* gradients  
  
# 预测  
y\_predict\_ = X\_new\_b.dot(theta)  
print(f"梯度下降{y\_predict\_.T}")  
  
print(f"两者误差值为{abs(y\_predict\_ - y\_predict.flatten())}")

运行结果如下：



1. 给定一组数据，

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | x3 | y |
| 1 | 2 | 3 | 20 |
| 2 | 3 | 3 | 25 |
| 3 | 2 | 2 | 21 |
| 4 | 2 | 3 | 28 |
| 2 | 3 | 2 | 22 |
| 1 | 2 | 4 | 23 |
| 3 | 3 | 2 | 25 |
| 4 | 4 | 2 | 29 |
| 5 | 5 | 4 | 43 |

试用标准方程法和梯度下降法寻找最优的三元线性模型；试求三个属性x1,x2,x3均取值为3时，y的预测值是多少？

**代码部分：**

import numpy as np  
  
data = np.array([  
 [1, 2, 3, 20],  
 [2, 3, 3, 25],  
 [3, 2, 2, 21],  
 [4, 2, 3, 28],  
 [2, 3, 2, 22],  
 [1, 2, 4, 23],  
 [3, 3, 2, 25],  
 [4, 4, 2, 29],  
 [5, 5, 4, 43]  
])  
  
X = data[:, :3]  
y = data[:, 3]  
  
X\_b = np.c\_[np.ones((X.shape[0], 1)), X]  
  
theta\_normal = np.linalg.inv(X\_b.T.dot(X\_b)).dot(X\_b.T).dot(y)  
  
  
def predict(X, theta):  
 X\_b = np.c\_[np.ones((X.shape[0], 1)), X]  
 return X\_b.dot(theta)  
  
  
x\_test = np.array([[3, 3, 3]])  
y\_pred\_normal = predict(x\_test, theta\_normal)  
  
  
def compute\_cost(X, y, theta):  
 m = len(y)  
 return (1 / (2 \* m)) \* np.sum((X.dot(theta) - y) \*\* 2)  
  
  
def gradient\_descent(X, y, theta, learning\_rate, num\_iterations):  
 m = len(y)  
 cost\_history = np.zeros(num\_iterations)  
  
 for i in range(num\_iterations):  
 gradients = (1 / m) \* X.T.dot(X.dot(theta) - y)  
 theta = theta - learning\_rate \* gradients  
 cost\_history[i] = compute\_cost(X, y, theta)  
  
 return theta, cost\_history  
  
  
theta\_initial = np.zeros(X\_b.shape[1])  
learning\_rate = 0.01  
num\_iterations = 1000  
  
theta\_gd, cost\_history = gradient\_descent(X\_b, y, theta\_initial, learning\_rate, num\_iterations)  
  
y\_pred\_gd = predict(x\_test, theta\_gd)  
  
print(f"标准方程法{y\_pred\_normal}")  
print(f"梯度下降法{y\_pred\_gd}")

运行结果：

