流程记录——基础题程序的部分1

（其他程序的思路大同小异）

线性回归的基本步骤：

创建\引用数据->对数据进行处理（分组）->拟合模型->输出图像

此程序的目的是通过梯度下降法研究其参数之间的关系，所以并没有输出K，也没有分出测试组来模拟输出（相当于整个数据都是训练组），输出的图像也只是各参数之间的关系。

```
# 创建数据
np.random.seed(0)
m = 100
x = np.random.rand(m,1)
y = 5 + 3 * x + np.random.rand(m,1)
```

x = np.random.rand(m,1)表示创建m个一维数据，在部分1中探究损失函数与迭代次数的关系，样本容量m固定为100，此处用m而不直接用100是为了便于修改和避免紧耦合。

```
# 拟合模型
K = 0
rate = 0.01
loss = []
for n in range(3000):
    y_hat = K * x
    small_loss = np.sum((y_hat-y)**2) / (2*m)
    loss.append(small_loss)
    K += rate / m * np.dot(x.T,(y-y_hat)) # 通过矩阵的乘法来实现多个乘积的累加。
    K = K.item()
```

1. y_hat相当于拟合函数，small_loss是损失函数，K是参数。
2. **关于K：** 在算法中更新参数的操作为：$\theta_i = \theta_i - \frac{\alpha}{m}\sum_{j=1}^{m}(h_{(\theta)}(x^{(j)}) - y^{(i)})x_i^{(j)}$但是在代码实现时，实际上是用一个累加来更新参数。因为算法中，思路是找一个适当的θ_i，然后在迭代中更正；在代码实现中，面对不同的问题，找一个适当的θ并不容易，并且对上式进行变化可得：$\theta_i = \theta_i + \frac{\alpha}{m}\sum_{j=1}^{m}(h_{(\theta)}(x^{(j)})y^{(i)})x_i^{(j)}$，所以定参数初值为0来进行累加也是合理的。
3. np.sum()将所有平方差项求和，相当于实现了+=的累加。
4. 这里将每一次的small_loss加到列表中，而在部分2和部分3，是将small_loss累加后在加入列表。因为部分1是要研究迭代次数与损失函数的关系，所以在每次迭代时就要把损失函数存下。