信息基础 II **2023 秋**

Homework 2

学生: 刘浩 (202120120312) 时间: 9 月 15 日

1.1 实验要求

以 (2, 1) 为中心，(1, 0.5; 0.5, 2) 为协方差矩阵的二维高斯分布；随机生成 30 个点，使用 Linear Regression 找到近似的线性函数。（使用 close form solution 、GD、SGD 三种方法实现）。

1.2 实验原理

(1) 首先是闭式解算法 (close form solution)，在本题目中构建 30×1 的 Y 矩阵，30×2 的 X 矩阵，还有 2×1 的 θ 矩阵如下：

$$Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(30)} \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1^{(1)} \\ \vdots & \vdots \\ 1 & x_1^{(30)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

那么利用 $\bigtriangledown J(\theta) = 0$ 得到以下关于 θ 的表达式：

$$\theta = \left(X^T X\right)^{-1} \left(X^T Y\right)$$

因此我们只需要进行矩阵之间的运算即可得到 θ

(2)GD 算法, 我们需要计算 $\bigtriangledown J(\theta)$，其计算公式如下：

$$\bigtriangledown J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \bigtriangledown J_i(\theta)$$

进一步化简得：

$$\bigtriangledown J(\theta) = \frac{1}{N} X^T (X\theta - Y)$$

在训练过程中我们只需要更新 θ (其中 γ 为学习率):

$$\theta = \theta - \gamma \bigtriangledown J(\theta)$$

终止条件为：

$$\left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})\right\|_2 \leq \epsilon$$

(3)SGD 算法，与 GD 算法的区别是 SGD 只需要在数据集中随机选择点，计算该数据点的梯度再进行梯度下降，由原来的计算整个数据集的梯度，变为了计算数据点的梯度，我们只需要根据数据点梯度再通过学习率进行 θ 的更新即可：

$$J_i(\theta) = \frac{1}{2}\left(y^{(i)} - \theta^T \vec{x}^{(i)}\right)^2$$

1.3 实验步骤

首先是数据数据生成和保存，利用 numpy 库中的 random.multivariate_normal 函数生成 30 个符合要求的二维高斯分布点：

```
# 设置均值和协方差矩阵
mean = np.array([2, 1])
cov_matrix = np.array([[1, -0.5], [-0.5, 2]])
num_samples = 30
# 生成随机数据点,保证每次随机得到的结果相同
np.random.seed(0)

random_points = np.random.multivariate_normal(mean, cov_matrix, num_samples)
```

然后进行数据预处理，提取出对应的 x,y 坐标并且利用 np.column_stack 构造我们需要的增加偏置的 X 矩阵：

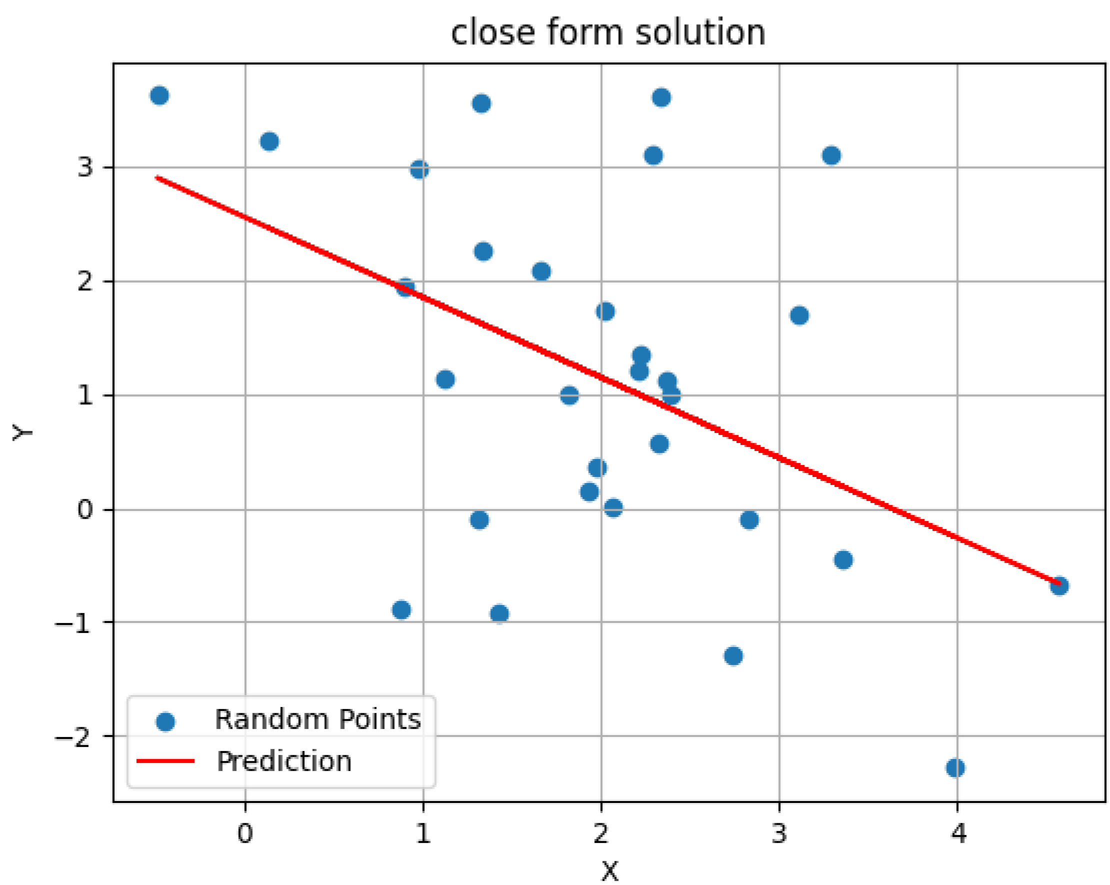
```
# 提取x和y坐标
x_val = random_points[:, 0]
y_val = random_points[:, 1]

# 构造矩阵
X_origin = np.array(x_val)
bias = np.ones(X_origin.shape)

# 增加偏置后的矩阵
X = np.column_stack((bias, x_val))
Y = np.array(y_val)
```

下面开始计算闭式解, 做法其实很简单，只需要代入给出的 θ 的公式即可：

```
# 闭式解算法
def close_form():
    # 计算闭式解

    # X_1 = X的转置*X-->逆
    X_1 = np.linalg.pinv(np.dot(X.T, X))
    theta = np.dot(np.dot(X_1, X.T), Y)
    y_hat = theta[0] + x_val * theta[1]

    # 绘制原始数据点
    plt.scatter(x_val, y_val, label='Random Points')
```

```
# 绘制预测线
plt.plot(x_val, y_hat, color='red', label='Prediction')
plt.title('close form solution')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

得到的结果如下图所示：



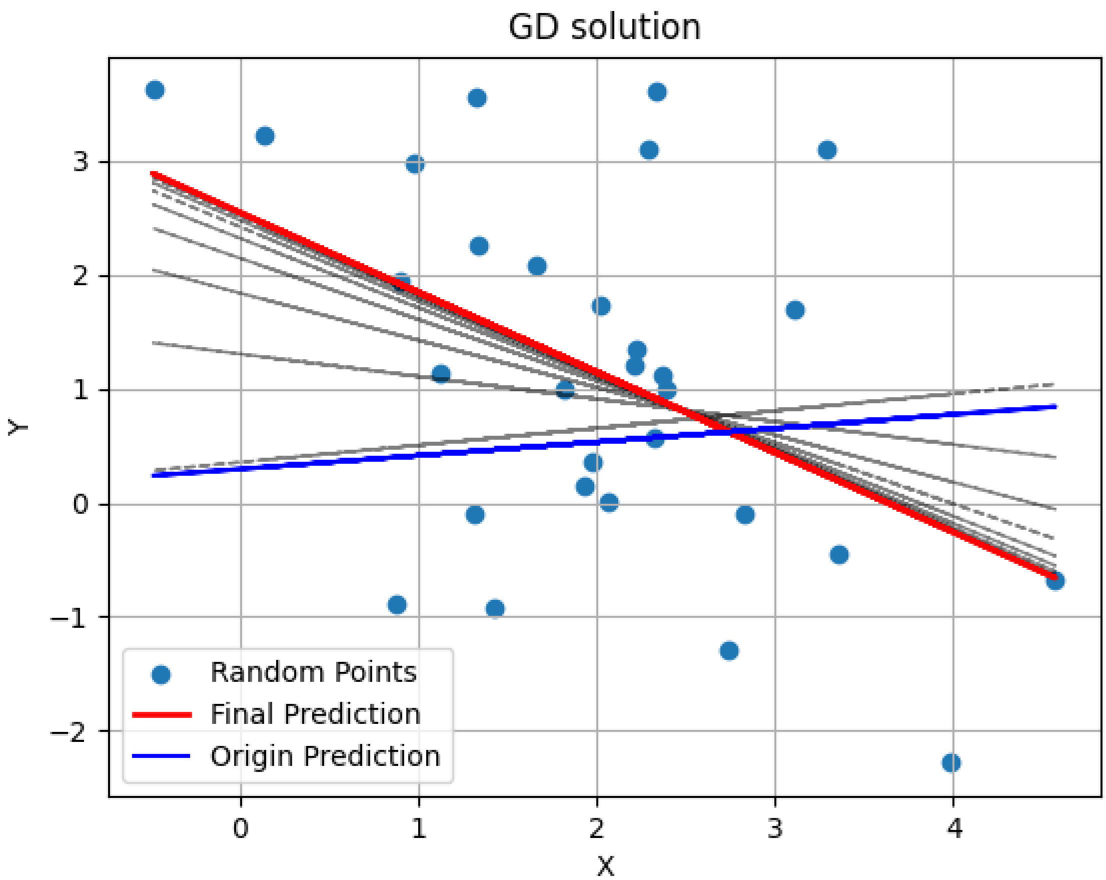
GD 算法则要计算梯度，首先我定义了梯度计算函数，同样按照实验原理中的梯度计算公式，代入即可：

```
# J的梯度
def gradJ(X, Y, theta):
    temp = -(Y - np.dot(X, theta))
    grad = np.dot(X.T, temp)
    return grad / len(X)
```

下面是 GD 主函数的构建，首先设置学习率为 0.1，随机初始化系数矩阵，训练次数为 600 次，每隔 30 次记录一次 θ，终止条件同样采取实验原理中的公式

```
# GD Solution
def GD():
    lr = 0.1
    theta = np.random.rand(2)
    epoch = 600
    batch = 30
    theta_list = []

    # 初始直线
    y_origin = theta[0] + x_val * theta[1]

    # 开始训练
    for i in range(epoch):
        theta = theta - lr * gradJ(X, Y, theta)
        grad_norm = np.linalg.norm(gradJ(X, Y, theta), ord=2)
        if i % batch == 0:
            theta_list.append(theta)
        if grad_norm < 0.001:
            break

    plt.scatter(x_val, y_val, label='Random Points')

    for i, theta_i in enumerate(theta_list):
        if i == len(theta_list) - 1:
            plt.plot(x_val, theta_i[0] + x_val * theta_i[1], color='red', label='Final Prediction',
                 linewidth=2)
        else:
            plt.plot(x_val, theta_i[0] + x_val * theta_i[1], color='black',alpha= 0.5, linestyle='--',
                 linewidth=1)

    plt.plot(x_val, y_origin, color='blue', label='Origin Prediction')
    plt.title('GD solution')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.grid(True)
    plt.show()
```

得到的结果为 (图中黑色虚线代表迭代的中间过程，红色代表最终预测直线，蓝色代表初始随机系数矩阵的直线)



最后是 SGD 算法，首先需要对数据集中的单个点进行求梯度的操作，为此我定义了 single_grad 进行单个点梯度的求解：

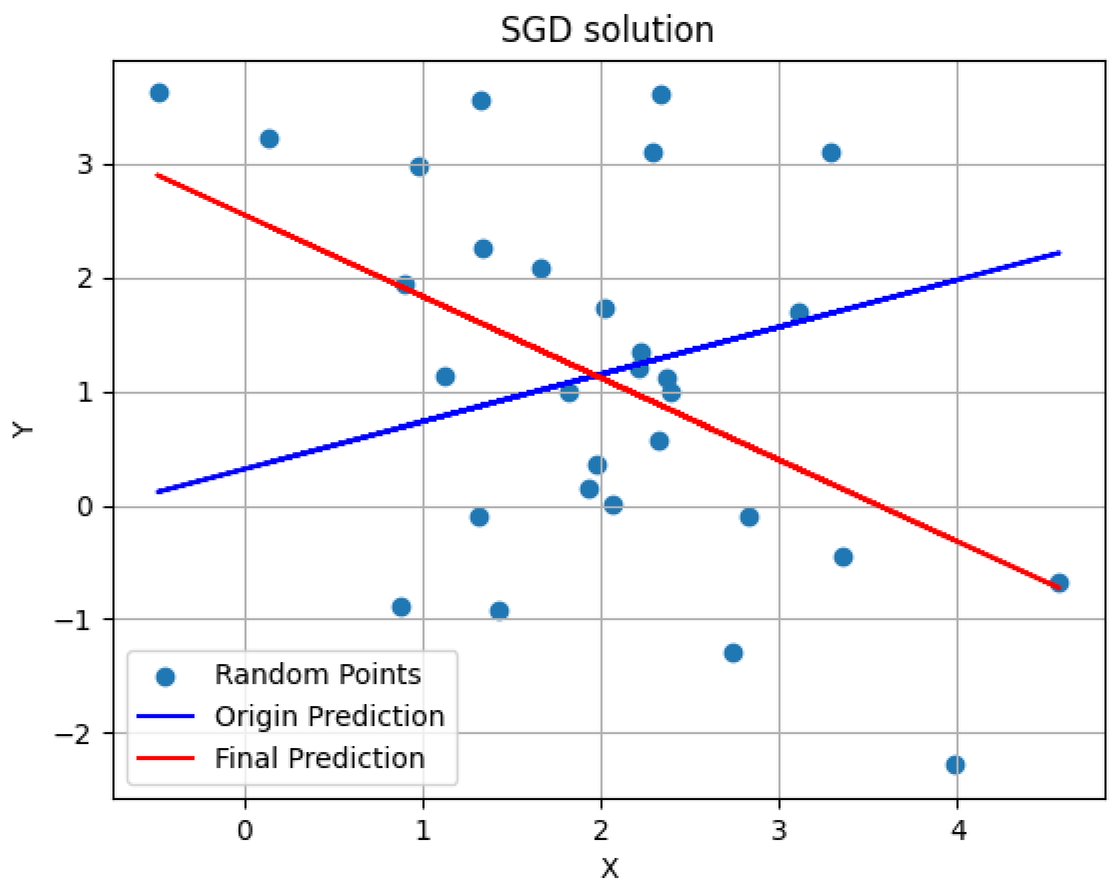
```
def single_grad(x, y, theta):
    grad = np.zeros(2)
    grad[0] = (theta[0] + theta[1] * x - y)
    grad[1] = x * (theta[0] + theta[1] * x - y)
    return grad
```

再利用 SGD 的基本原理，采取训练次数为 2000 次，学习率为 0.001 更新 θ

```
def SGD():
    lr = 0.001
    theta = np.random.rand(2)
    epoch = 2000
    y_origin = theta[0] + x_val * theta[1]
    # 开始训练
    for i in range(epoch):
        for (x, y) in zip(x_val, y_val):
            theta = theta - lr * single_grad(x, y, theta)
```

```
y_hat = theta[0] + x_val * theta[1]
plt.scatter(x_val, y_val, label='Random Points')
plt.plot(x_val, y_origin, color='blue', label='Origin Prediction')
plt.plot(x_val, y_hat, color='red', label='Final Prediction')
plt.title('SGD solution')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

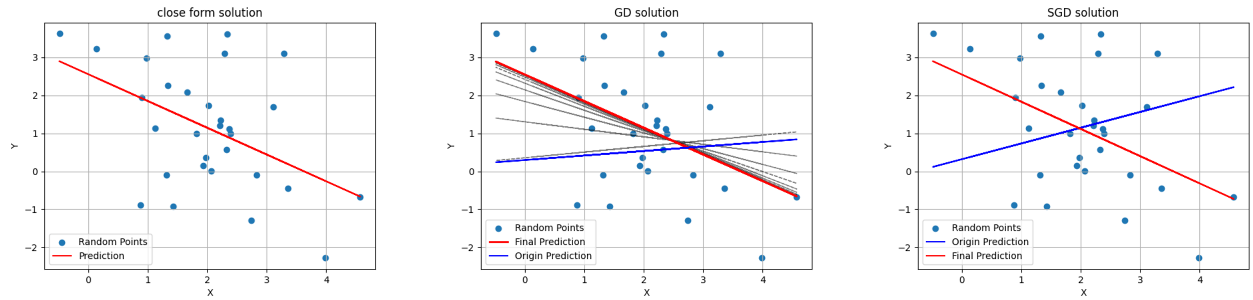
得到的结果如下：



其中红色代表最终预测直线，蓝色代表初始随机系数矩阵的直线

1.4 实验结果

程序结果一览：



(a) 闭式解　(b) 梯度下降　(c) 随机梯度下降

图 1.1: 实验结果

可以看到，三种方法预测出来的直线基本上是重合的，也验证了预测的准确性！

1.5 实验感想

在本次实验中，采取了三种方式进行线性回归，闭式解算法通常能够得到全局最小值，不存在迭代过程，但不是所有的问题都存在闭式解；而 GD 和 SGD 只需要不断根据梯度和学习率更新系数矩阵即可，无过多要求。GD 与 SGD 相比，GD 由于每次迭代都使用整个数据集，因此收敛速度较慢，SGD 运算次数较多但收敛速度较快；但是一般而言，GD 通常能够更稳定地收敛到全局最小值，而 SGD 可能会在训练过程中波动较大，难以达到全局最小值。

1.6 参考资料

[1] https://blog.csdn.net/m0_50117360/article/details/108743578

[2] https://blog.csdn.net/u012421852/article/details/79562067

[3] http://t.csdn.cn/zDZyp