Task 2. 张量的基本操作与自动求导

姓名 窦国泉 学号 202211011001

时间 2024.3.5

0. 模型设定与数据生成

考虑一个线性回归模型：

$$Y = X\beta + \varepsilon,$$

其中$X \in \mathbb{R}^{100\times 5}$，$\varepsilon \in \mathbb{R}^{100\times 1}$，$\beta \in \mathbb{R}^{5\times 1}$。

In [6]:

```
### 生成数据的代码
import torch
import matplotlib.pyplot as plt

# 样本维度
n = 100
p = 5

# 生成数据
X = torch.randn(n, p) # 生成100x5的正态分布随机数矩阵
eps = 0.1 * torch.randn(n, 1) # 生成方差为0.01的一列服从正态分布的误差
beta = torch.rand(p, 1) # 生成服从（0，1）上均匀分布的beta
Y = X @ beta + eps # 通过计算得到Y
```

1. 使用张量的基本操作求解OLS

由多元统计知识可得，最小二乘估计的解析表达式为：

$$\widehat{\beta}_{ols} = (X^\top X)^{-1}(X^\top Y)$$

In [8]:

```
### 求解OLS的代码
### 请封装为函数，输入为X和Y，输出beta_ols

def get_beta_ols(X, Y):
    beta_ols = torch.inverse(X.T @ X) @ (X.T @ Y) # 通过表达式计算出beta_ols
    return beta_ols
```

2. 使用梯度下降算法求解

梯度下降算法的迭代公式为：

$$\beta_{t+1} = \beta_t - \alpha\nabla\mathcal{L}(\beta_t),$$

其中$\nabla\mathcal{L}(\beta_t) = (\frac{X^\top X}{n})\beta_t - (\frac{X^\top Y}{n})$.

In [9]:

```
### 求解梯度下降的代码
### 请封装为函数，输入为X和Y，输出beta_gd

def get_beta_gd(X, Y):
```

```python
    beta_gd = torch.zeros(p, 1)
    alpha = 0.01
    for i in range(1000):
        grad = ((X.T @ X) / n) @ beta_gd - ((X.T @ Y) / n)
        beta_gd = beta_gd - alpha * grad # 手动计算梯度，并迭代
    return beta_gd
```

3. 使用自动求导实现梯度下降算法

In [10]:

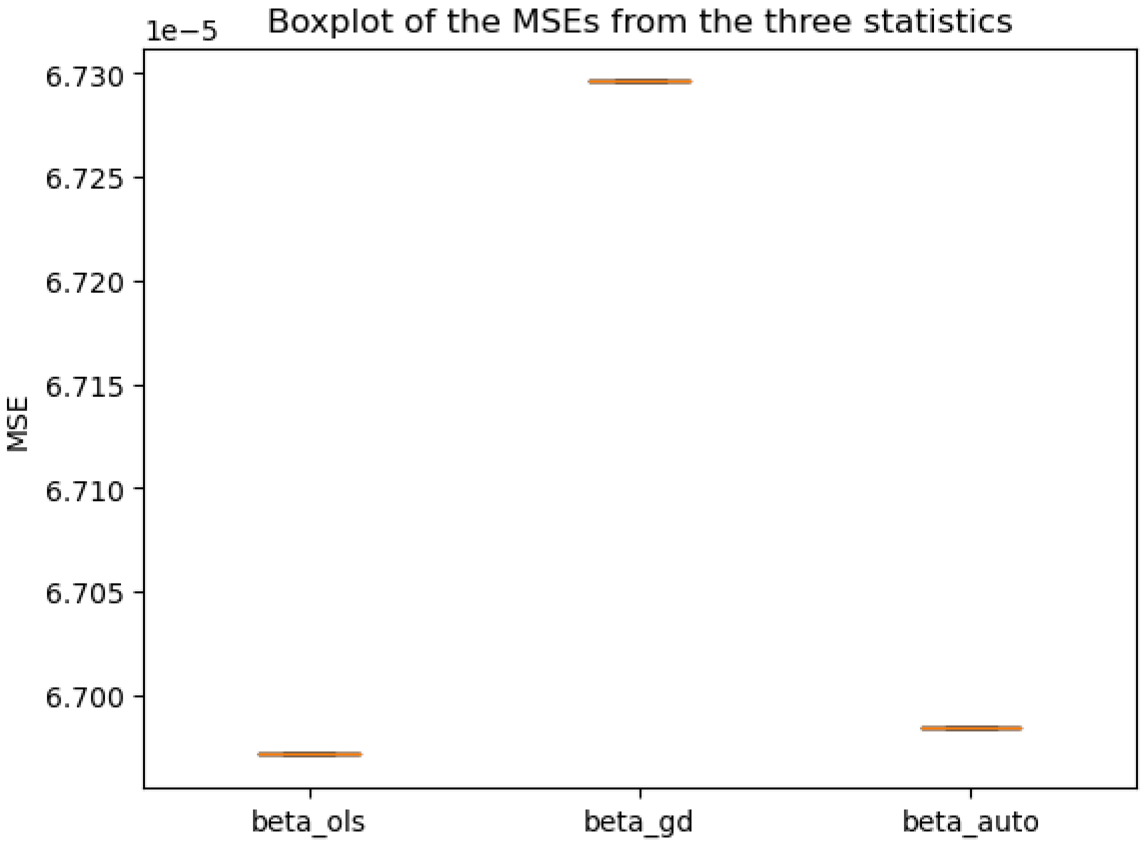
```python
def GD_autograd(X,Y, alpha = 0.01, T = 1000):

    n,p = X.size()
    beta_auto = torch.zeros(p,1,requires_grad=True)

    for t in range(T):

        # forward过程
        Y_hat = torch.mm(X,beta_auto)
        loss = torch.mean((Y_hat-Y).pow(2))
        # print(t,loss.data.numpy())

        # backword过程
        loss.backward()
        beta_auto.data = beta_auto.data - alpha*beta_auto.grad
        beta_auto.grad.fill_(0)

    return beta_auto.data
```

4. 对比实验

In [22]:

```python
# 使用循环进行多次实验(B次)，请保证beta在过程中不变。

    ## 重新生成X,eps和Y

    ## 计算beta_ols，并且和真值beta计算mse，记录结果

    ## 计算beta_gd，并且和真值beta计算mse，记录结果

    ## 计算beta_auto，并且和真值beta计算mse，记录结果

# 使用纪录的结果(共3*B个)，绘制盒型图(利用函数plt.boxplot)


B = 50 # 循环次数

# 设置存放三种估计量的初始空列表
mse1 = []
mse2 = []
mse3 = []

# 重新生成X, eps和Y
X = torch.randn(n, p)
eps = 0.1 * torch.randn(n, 1)
Y = X @ beta + eps

# 设置循环
for i in range(B):
    # 计算三个对beta的估计量
    ols = get_beta_ols(X, Y)
    gd = get_beta_gd(X, Y)
    auto = GD_autograd(X, Y)
```

```
    # 计算各自的均方误差
    ols_mse = torch.mean((ols - beta) ** 2)
    gd_mse = torch.mean((gd - beta) ** 2)
    auto_mse = torch.mean((auto - beta) ** 2)

    # 将计算出来的均方误差填入列表
    mse1.append(ols_mse)
    mse2.append(gd_mse)
    mse3.append(auto_mse)


# 画盒型图
plt.boxplot([mse1, mse2, mse3])
plt.title('Boxplot of the MSEs from the three statistics')
plt.ylabel('MSE')
plt.xticks([1, 2, 3], ['beta_ols', 'beta_gd', 'beta_auto'])

plt.show()
```



解释一下你的发现。

1. 观察到每个盒型图的方差很小，说明数据很为集中，表明三种估计方法都很有效、稳定
2. 三种估计方法的均方误差的数量级均为1e-5，都对beta有着很精确的估计
3. 尽管beta_gd的mse值看上去高于其他二者，但差距的数量级仅为1e-7。三种估计方法的稳定性和精确性都相差不大，难分伯仲，无法挑选出最好的估计方法