优化算法实验

一、问题描述

考虑一个 10 节点的分布式系统。节点 i 有线性测量 $\mathbf{b}_i = \mathbf{A}_i\mathbf{x} + \mathbf{e}_i$，其中 $\mathbf{b}_i$ 为 5 维的测量值，$\mathbf{A}_i$ 为 5×200 维的测量矩阵，$\mathbf{x}$ 为 200 维的未知稀疏向量且稀疏度为 5，e_i 为 5 维的测量噪声。从所有 $\mathbf{b}_i$ 与 A_i 中恢复 $\mathbf{x}$ 的一范数正则化最小二乘模型如下：

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\|A_1\boldsymbol{x} - b_1\|_2^2 + \cdots + \frac{1}{2}\|A_{10}\boldsymbol{x} - \boldsymbol{b}_{10}\|_2^2 + \lambda\|\boldsymbol{x}\|_1,$$

其中 $\lambda > 0$ 为正则化参数。请设计下述算法求解该问题：

1. 邻近点梯度法；
2. 交替方向乘子法；
3. 次梯度法；

在实验中，设 $\mathbf{x}$ 的真值中的非零元素服从均值为 0 方差为 1 的高斯分布，A_i 中的元素服从均值为 0 方差为 1 的高斯分布，$\mathbf{e}_i$ 中的元素服从均值为 0 方差为 0.1 的高斯分布。对于每种算法，请给出每步计算结果与真值的距离以及每步计算结果与最优解的距离。此外，请讨论正则化参数 λ 对计算结果的影响。

二、算法设计

1. 邻近点梯度法

令$\mathrm{f_i(x^k)} = \frac{1}{2}||\mathrm{A_i x - B_i}||^2$， $\mathrm{g(x)} = \lambda||\mathrm{x}||_1$，将优化问题转化为如下迭代方法：

$$\mathrm{x^{k+\frac{1}{2}}} = \mathrm{x^k} - \alpha(\sum_{\mathrm{i=1}}^{10} \nabla \mathrm{f_i(x^k)})$$

$$\mathrm{x^{k+1}} = \underset{\mathrm{x}}{\operatorname{argmin}}\, \mathrm{g(x)} + \frac{1}{2\alpha}||\mathrm{x - x^{k+\frac{1}{2}}}||^2$$

其中将计算$\nabla \mathrm{f_i(x^k)}$作为各个子节点任务，$\nabla \mathrm{f_i(x^k)} = (\mathrm{A_i x - B_i})^{\mathrm{T}}\mathrm{A_i}$

主节点负责求和$\nabla \mathrm{f_i(x^k)}$，求解优化问题得到$\mathrm{x^{k+1}}$，使用软门限法求解

$$\mathrm{x^{k+1}} = \begin{cases} \mathrm{x^{k+\frac{1}{2}}} - \lambda\alpha, & \mathrm{x^{k+\frac{1}{2}}} > \lambda\alpha \\ \mathrm{x^{k+\frac{1}{2}}} + \lambda\alpha, & \mathrm{x^{k+\frac{1}{2}}} < \lambda\alpha \\ 0, & -\lambda\alpha \leqslant \mathrm{x^{k+\frac{1}{2}}} \leqslant \lambda\alpha \end{cases}$$

这里是对向量 x 的每个维度分离计算

2. 交替方向乘子法

将原优化问题转化为$\min_{\mathrm{x,y}} \sum_{\mathrm{i=1}}^{10} \frac{1}{2}||\mathrm{A_i x_i - B_i}||^2 + \lambda||\mathrm{y}||_1 + \sum_{\mathrm{i=1}}^{10} \mathrm{v_i}^{\mathrm{T}}(\mathrm{x_i - y}) + \frac{\mathrm{C}}{2}||\mathrm{x - y}||^2$

令$\mathrm{f_i(x_i^k)} = \frac{1}{2}||\mathrm{A_i x_i - B_i}||^2$，$\mathrm{g(y)} = \lambda||\mathrm{y}||_1$即可使用交替方向乘子法：

$$x_i^{k+1} = \underset{x}{\operatorname{argmin}}\, f_i(x_i) + \left(v_i^k\right)^T x_i + \frac{C}{2}||x_i - y^k||^2$$

$$y^{k+1} = \underset{y}{\operatorname{argmin}}\, g(y) - \sum_{i=1}^{10}\left(v_i^k\right)^T y + \frac{C}{2}\sum_{i=1}^{10}||\,y - x_i^{k+1}||^2$$

$$v_i^{k+1} = v_i^k + C(x_i^{k+1} - y^{k+1}), i = 1,\ldots 10$$

其中求解 y 的优化问题同样使用软门限法：

$$x_i^{k+1} = \left(A_i^T A_i + C\right)^{-1}(Cy^k + A_i^T B_i - v_i^k)$$

$$y^{k+1} = \begin{cases} \dfrac{\lambda + \sum_{i=1}^{10} v_i^k + C(\sum_{i=1}^{10} x_i^{k+1})}{10C}, & \sum_{i=1}^{10} v_i^k + C\left(\sum_{i=1}^{10} x_i^{k+1}\right) < -\lambda \\ \dfrac{-\lambda + \sum_{i=1}^{10} v_i^k + C(\sum_{i=1}^{10} x_i^{k+1})}{10C}, & \sum_{i=1}^{10} v_i^k + C\left(\sum_{i=1}^{10} x_i^{k+1}\right) > \lambda \\ 0, & -\lambda \leqslant \sum_{i=1}^{10} v_i^k + C\left(\sum_{i=1}^{10} x_i^{k+1}\right) \leqslant \lambda \end{cases}$$

3. 次梯度法

直接使用迭代公式：

$$x^{k+1} = x^k - \alpha d_k$$

其中$d_k \in \partial(\sum_{i=1}^{10} \frac{1}{2}||A_i x^k - B_i||^2 + \lambda||x^k||_1)$，得到：

$$d_k = \begin{cases} \sum_{i=1}^{10} A_i^T(A_i x^k - B_i) + \lambda, & x^k > 0 \\ \sum_{i=1}^{10} A_i^T(A_i x^k - B_i) - \lambda, & x^k < 0 \\ \sum_{i=1}^{10} A_i^T(A_i x^k - B_i), & x^k = 0 \end{cases}$$

三、数值实验

初始化参数：真实值 x 为稀疏度为 5 的 200 维矩阵，e 为 10 个服从均值为 0 方差为 0.1 高斯分布的 5 维矩阵，A 为 10 个服从均值为 0 方差为 1 高斯分布的 5x200 维矩阵，B 为通过真实值 x 和测量矩阵 A 相乘加上噪声 e 的线性测量矩阵。

```
if __name__ == "__main__":
    lamb = 0.01 # 正则项参数
    X_true = np.zeros(200)
    index_X = random.sample(range(200), 5)
    for index in index_X:
        X_true[index] = np.random.normal(0, 1)
    E = np.random.normal(0, 0.1, 5*10).reshape(10,5)
    A = np.random.normal(0, 1, 5*200*10).reshape(10,5,200)
    B = np.zeros((10,5))
    for i in range(10):
        B[i] = A[i] @ X_true + E[i]
    distance_real,distance_best = [],[]
```

1. 邻近点梯度法

```python
# 邻近点梯度法
    alpha = 0.001
    X_pred = np.zeros(200)
    while 1:
        delta_X = 0 # 梯度和
        loss = 0
        for i in range(10):
            pred_B = A[i] @ X_pred
            delta_X += (pred_B - B[i]) @ A[i]
        X_half = X_pred - alpha*delta_X
        X_next = np.where(X_half > alpha*lamb, X_half - alpha*lamb, np.where(X_half < alpha*lamb, X_half + alpha*lamb, 0))
        print(np.linalg.norm(X_pred - X_next, ord=2))
        if np.linalg.norm(X_pred - X_next, ord=2) < 10e-5: break
        X_pred = X_next
        distance_real.append(np.linalg.norm(X_next - X_true, ord=2))
        distance_best.append(X_next)
```

2. 交替方向乘子法

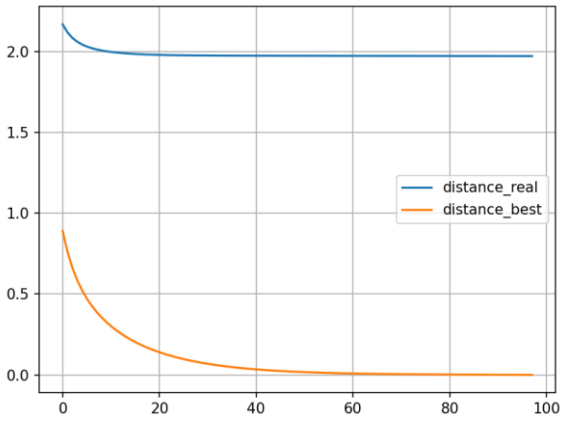
```python
v = np.zeros((10,200))
C = 0.05
X_pred = np.zeros((10,200))
Y_pred = np.zeros(200)
while 1:
    X_next = np.zeros((10,200))
    loss = 0
    temp = 0
    for i in range(10):
        X_next[i] = np.linalg.inv(A[i].T @ A[i] + C* np.eye(200, 200))@(C*Y_pred+A[i].T@B[i]-v[i])
        temp += v[i] + C*X_next[i]
    Y_next = np.where(temp > lamb, (temp - lamb)/(10*C), np.where(temp < -lamb, (temp + lamb)/(10*C), 0))
    if np.linalg.norm(Y_pred - Y_next, ord=2) < 10e-4:break
    for i in range(10):
        v[i] += C*(X_next[i]-Y_next)
    X_pred = X_next
    Y_pred = Y_next
    distance_real.append(np.linalg.norm(Y_next - X_true, ord=2))
    distance_best.append(Y_next)
```

3. 次梯度法

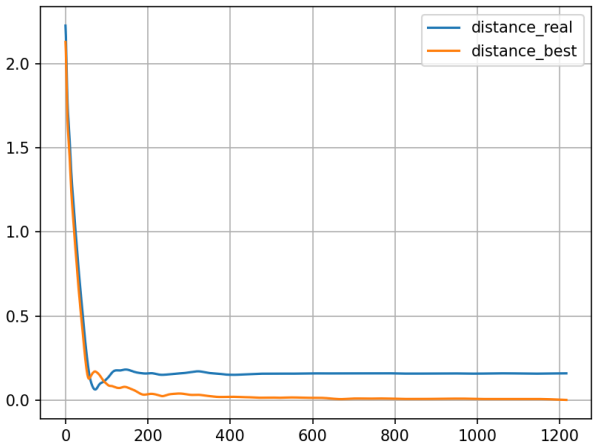
```python
alpha = 0.001
X_pred = np.zeros(200)
while 1:
    delta_X = 0 # 梯度和
    loss = 0
    for i in range(10):
        pred_B = A[i] @ X_pred
        delta_X += (pred_B - B[i]) @ A[i]
    X_next = X_pred - alpha*(delta_X + lamb * np.sign(X_pred))
    if np.linalg.norm(X_pred - X_next, ord=2) < 10e-5: break
    X_pred = X_next
    distance_real.append(np.linalg.norm(X_next - X_true, ord=2))
    distance_best.append(X_next)
```

四、结果分析

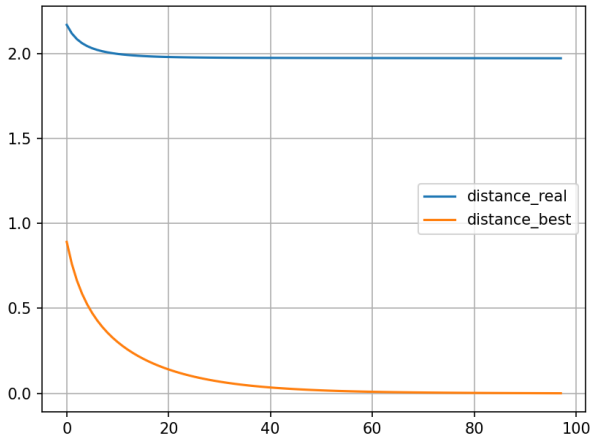
1. 邻近点梯度法



2. 交替方向乘子法



3. 次梯度法



由结果可视化可看出：

邻近点梯度法：与真实值和最优值的距离在迭代收敛前就很早趋于收敛，具有较快的收敛速度，但最优解和真实值之间具有较大距离。

交替方向乘子法：有快速收敛和平缓收敛两个阶段，收敛需要相对较多的迭代轮数，但求得的最优解和真实值之前的差距最小，能够很好地贴近真实值。

次梯度法：总体的曲线收敛速度和效果都与邻近点梯度法几乎相同，因为二者本质上都是计算梯度以及使用软门限进行更新迭代。不同之处在于邻近点梯度法能够将梯度的计算任务交给从节点的同时主节点来求解优化问题，平分各个节点之间的负载。

正则项参数 λ 对计算结果的影响：

λ 越大，正则项的比重也大，也代表着软门限越大，得到的估计值 x 中的零元素就越多，解更稀疏，越接近真实的 x，但也相对比较难收敛，因为在稀疏的情况下也意味着很难与预测值保持较低的误差。而较小的 λ 使得模型保持一定的复杂性，能够在训练中适应更多的细节。如右图是将 λ 调大 4 倍后的使用交替方向乘子法的训练结果，可以看出 λ 增大后迭代需要的轮数也变多（在这种情况下使用另外两种方法很难收敛）。