最优化大作业

姓名：林筱涵

学号：22336141

题目：

考虑一个 10 节点的分布式系统。节点 i 有线性测量 $\boldsymbol{b}_i = A_i\boldsymbol{x} + \boldsymbol{e}_i$，其中 $\boldsymbol{b}_i$ 为 5 维的测量值，A_i 为 5×200 维的测量矩阵，$\boldsymbol{x}$ 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 $\boldsymbol{b}_i$ 与 A_i 中恢复 $\boldsymbol{x}$ 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. 邻近梯度法；
2. 交替方向乘子法；
3. 次梯度法；

在实验中，设 $\boldsymbol{x}$ 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，$\boldsymbol{e}_i$ 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。

目标函数和数据生成

系统设定：考虑一个 10 节点的分布式系统，其中每个节点 i 通过线性模型 $b_i = A_i x + e_i$ 来进行测量。

- A_i：是一个 5×200 的矩阵，表示每个节点的设计矩阵。
- x：是一个 200 维的未知向量。
- e_i：是一个 5 维的测量噪声向量，服从均值为 0，标准差为 0.1 的高斯分布。
- 每个节点的测量值 b_i 是 A_i 和 x 的线性组合再加上噪声。

1.**生成矩阵** A：

```
A = np.array([np.random.normal(0, 1, (rows, cols)) for _ in range(samples)])
```

其中，row和col分别设定为 5 和 200 ，即每个矩阵包含 5 行 200 列，矩阵元素服从均值为 0，标准差为 1 的正态分布。
2.**生成稀疏的权重向量x:**

```
true_x = np.zeros(cols)
nonzero_indices = np.random.choice(cols, 5, replace=False)
true_x[nonzero_indices] = np.random.normal(0, 1, 5)
```

其中true_x 是一个长度为 200 的零向量。
随机选择 5 个位置，并赋值为从正态分布中采样的值，这样就得到了一个稀疏的向量x，其中只有 5 个非零元素。
3.**生成测量值b:**

```
b = np.array([A[j].dot(true_x) + np.random.normal(0, 0.1, rows) for j in range(samples)])
```

对于每个节点 i，测量值 b_i 由 A_i 和 x 的线性组合得到：A[j].dot(true_x)
然后添加噪声，噪声是服从均值为 0，标准差为 0.1 的高斯分布:np.random.normal(0, 0.1, rows)

实现方法

1.邻近梯度法

这是处理带有L1正则化的最常用方法。其基本思想是结合梯度下降和软阈值函数来更新权重。
步骤：
1.对于邻近梯度法：
对光滑部分做梯度下降：
$x^{k+\frac{1}{2}} = x^k - \alpha_k \nabla f(x^k)$
对非光滑部分使用邻近算子：
$x^{k+1} = \mathrm{prox}_{\alpha_k r}\left(x^{k+\frac{1}{2}}\right)$
即邻近梯度更新公式如下：

$x^{k+1} = \text{prox}_{\alpha_k r}\left(x^k - \alpha_k \nabla f(x^k)\right)$

2.对于目标函数

$\frac{1}{2}\sum_{j=1}^{10} \|A_j x - b_j\|_2^2$是该函数的光滑部分，是数据拟合项；

$\lambda\|x\|_1$是L1正则化项，促使解变得稀疏

3.对光滑部分做梯度下降

$\nabla f(x) = \sum_{j=1}^{10} A_j^T(A_j x - b_j)$

其更新公式：

$x^{k+\frac{1}{2}} = x^k - \alpha \nabla f(x^k)$

对非光滑部分使用邻近算子:

$x^{k+1} = \text{prox}_{\alpha\lambda}\left(x^{k+\frac{1}{2}}\right)$

其中软阈值算子的公式为：

$\text{soft\_threshold}(z, \tau) = \text{sign}(z) \cdot \max(|z| - \tau, 0)$

4.结合光滑部分的梯度下降和非光滑部分的邻近算子，得到最终的更新公式:

$x^{k+1} = \text{prox}_{\alpha\lambda}\left(x^k - \alpha \sum_{i=1}^{10} A_i^T(A_i x^k - b_i)\right)$

具体代码如下：

```
def proximal_gradient(A, b, reg_param, step_size=0.0001, max_iters=5000, tolerance=1e-5):
    start_time = time.time()
    num_samples, dim1, dim2 = A.shape
    x = np.zeros(dim2)
    history = []

    for _ in range(max_iters):
        prev_x = x.copy()

        # 梯度计算
        grad = np.sum([A[j].T.dot(A[j].dot(x) - b[j]) for j in range(num_samples)], axis=0)

        # 更新 x
        x = soft_threshold(x - step_size * grad, reg_param * step_size)
        history.append(x)

        # 检查收敛条件
        if np.linalg.norm(x - prev_x, ord=2) < tolerance:
            break

    elapsed_time = time.time() - start_time
    print(f'Proximal gradient finished in {elapsed_time:.2f}s')
    return x, history
```

2.ADMM

ADMM 将目标函数分解成两个子问题，并交替求解每个子问题。在这个问题中，我引入了一个辅助变量z来处理L1正则化项。

目标函数变为：

$\min_x \left(\frac{1}{2} \sum_{i=1}^{10} \|A_i x - b_i\|_2^2 + \lambda \|z\|_1 \right)$

约束条件变为：z = x

引入拉格朗日乘子u,其增广拉格朗日函数如下：

$\mathcal{L}(x, z, u) = \frac{1}{2} \sum_{i=1}^{10} \|A_i x - b_i\|_2^2 + \lambda \|x\|_1 + u^T (x - z) + \frac{\rho}{2} \|x - z\|_2^2$

步骤：

1.更新x:

$x = \arg\min_x \left(\frac{1}{2} \sum_{i=1}^{10} \|A_i x - b_i\|_2^2 + \frac{\rho}{2} \|x - z + u/\rho\|_2^2 \right)$

将其解出后得：

$x = \left(\sum_{i=1}^{10} A_i^T A_i + \rho I \right)^{-1} \left(\sum_{i=1}^{10} A_i^T b_i + \rho(z - u) \right)$

2.更新z

为了更新z，我们对拉格朗日函数L(x,z,u) 关于 z求导，并令其为零,得到以下式子：

$z^{k+1} = \text{soft\_threshold}(x^{k+1} + u^k/\rho, \lambda/\rho)$

软阈值操作soft_threshold的定义与上文的一致

3.更新u

$u^{k+1} = u^k + \rho(x^{k+1} - z^{k+1})$

具体代码如下：

```
def admm_solver(A, b, reg_param, penalty=1, max_iters=1000, tolerance=1e-5):
    start_time = time.time()
    num_samples, dim1, dim2 = A.shape
    x = np.zeros(dim2)
    z = np.zeros(dim2)
    u = np.zeros(dim2)
    history = []

    for _ in range(max_iters):
        prev_x = x.copy()

        # 更新 x
        x = np.linalg.inv(np.sum([A[j].T.dot(A[j]) for j in range(num_samples)], axis=0) +
        penalty * np.eye(dim2)).dot(np.sum([A[j].T.dot(b[j]) for j in range(num_samples)], axis=0)
        + penalty * z - u)

        # 更新 z
        z = soft_threshold(x + u / penalty, reg_param / penalty)

        # 更新 u
        u += penalty * (x - z)
        history.append(x)

        # 检查收敛条件
        if np.linalg.norm(x - prev_x, ord=2) < tolerance:
            break

    elapsed_time = time.time() - start_time
    print(f'ADMM completed in {elapsed_time:.2f}s')
    return x, history
```

3.次梯度

次梯度大部分和邻近算法相同，只是对于非光滑正则化项使用次梯度来计算。
次梯度对于L1范数的定义为：
$\partial \|x\|_1 = \text{sign}(x)$

$$\text{sign}(x_i) = \begin{cases} 1 & \text{if } x_i > 0 \\ 0 & \text{if } x_i = 0 \\ -1 & \text{if } x_i < 0 \end{cases}$$

1.对于光滑部分的梯度为：

$\nabla f(x) = \sum_{i=1}^{10} A_i^T (A_i x - b_i)$

2.对于正则化项次梯度：

$\lambda \cdot \text{sign}(x)$

故次梯度的组合为：

$\text{sub\_grad} = \lambda \cdot \text{sign}(x) + \sum_{i=1}^{10} A_i^T (A_i x - b_i)$

3.更新

在次梯度法中，更新公式是：

$x^{k+1} = x^k - \alpha \cdot \text{sub\_grad}$

4.判断收敛：

```
if np.linalg.norm(x - prev_x, ord=2) < tolerance:
    break
```

即如果x的变化小于tolerance，则停止迭代。

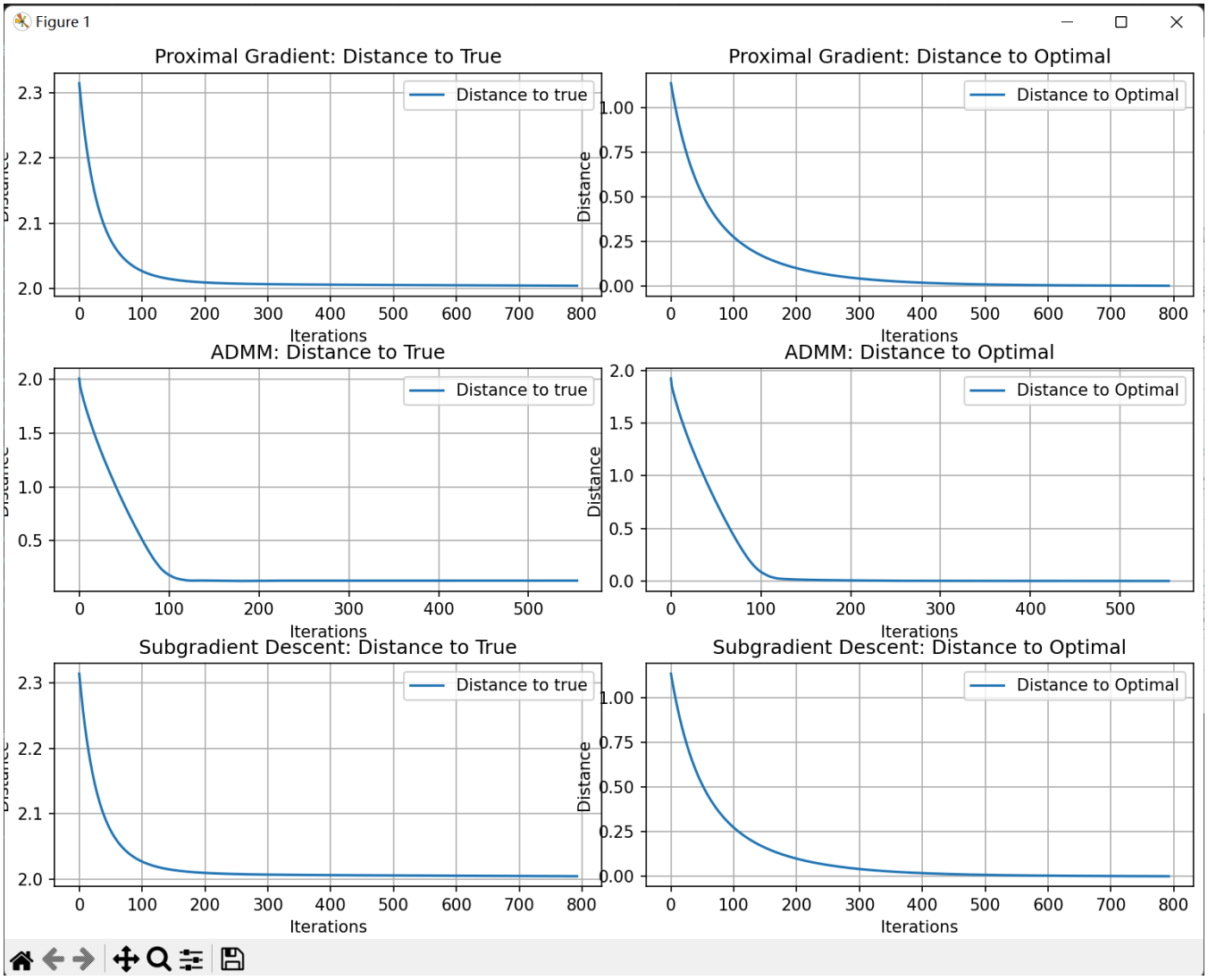
具体代码如下：

```
def subgradient_descent(A, b, reg_param, step_size=0.0001, max_iters=5000, tolerance=1e-5):
    start_time = time.time()
    num_samples, dim1, dim2 = A.shape
    x = np.zeros(dim2)
    history = []

    for _ in range(max_iters):
        prev_x = x.copy()

        # 计算次梯度
        sub_grad = reg_param * np.sign(x) + np.sum([A[j].T.dot(A[j].dot(x) - b[j])
                                                    for j in range(num_samples)], axis=0)

        # 更新 x
        x = x - step_size * sub_grad
        history.append(x)

        # 检查收敛条件
        if np.linalg.norm(x - prev_x, ord=2) < tolerance:
            break

    elapsed_time = time.time() - start_time
    print(f'Subgradient descent completed in {elapsed_time:.2f}s')
    return x, history
```

实现结果

绘制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。

1.设定参数值，对于这三个算法，邻近梯度法和次梯度法使用固定步长，设定a为0.0001，而对于ADMM，设定penalty 参数来控制更新的速率，设penalty=1；正则化参数为0.01。



可以看出：

1. 对于邻近算法，可看出其距离真实解的曲线和距离最优解的曲线 在某个点之后不再明显下降，说明邻近算法已经收敛并且达到了稳定解。

2. 对于交替乘子法，可看出距离真实解的曲线和距离最优解的曲线也是收敛的，且收敛速度比邻近梯度算法要快，且在处理带有约束和非光滑项的问题时效果较好。对于该算法其耗时最长，为3.58s

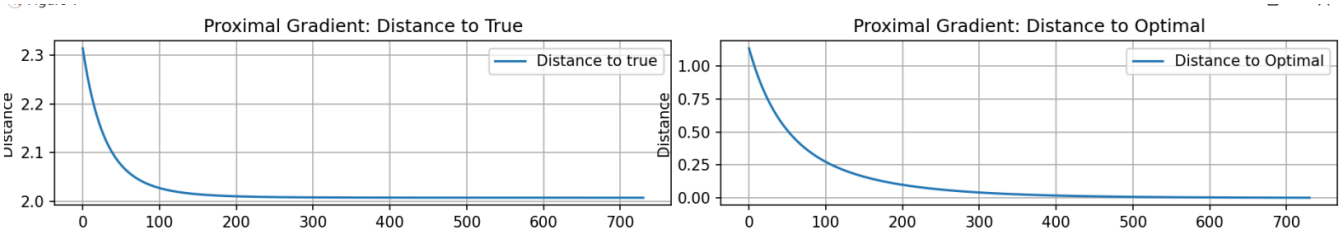
3. 次梯度法其距离真实解的曲线和距离最优解的曲线也是收敛的，但是没有ADMM效果好。因为次梯度法在处理L1正则化时相对简单，故该算法耗时最短，为0.02s

```
Proximal gradient finished in 0.03s
ADMM completed in 3.58s
Subgradient descent completed in 0.02s
```

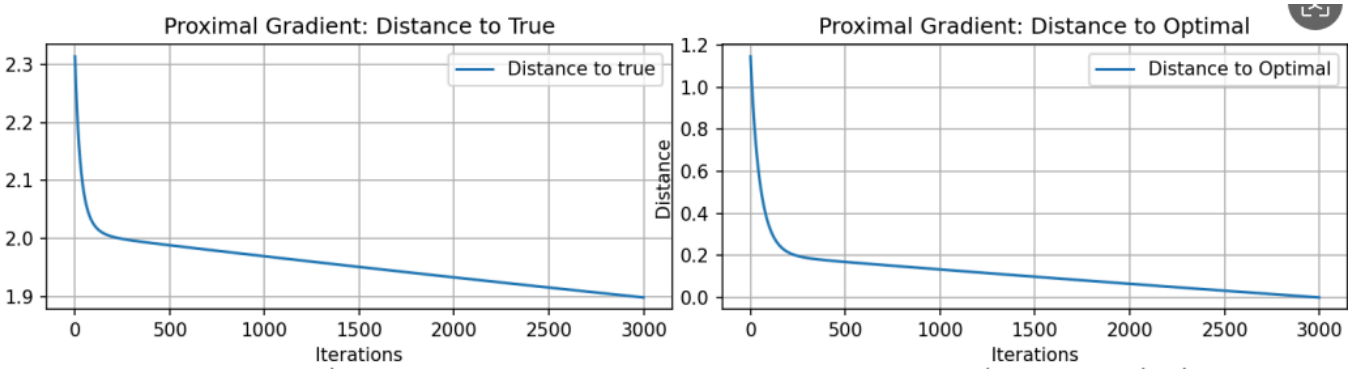
不同正则化参数的影响

- **1.邻近梯度法：**

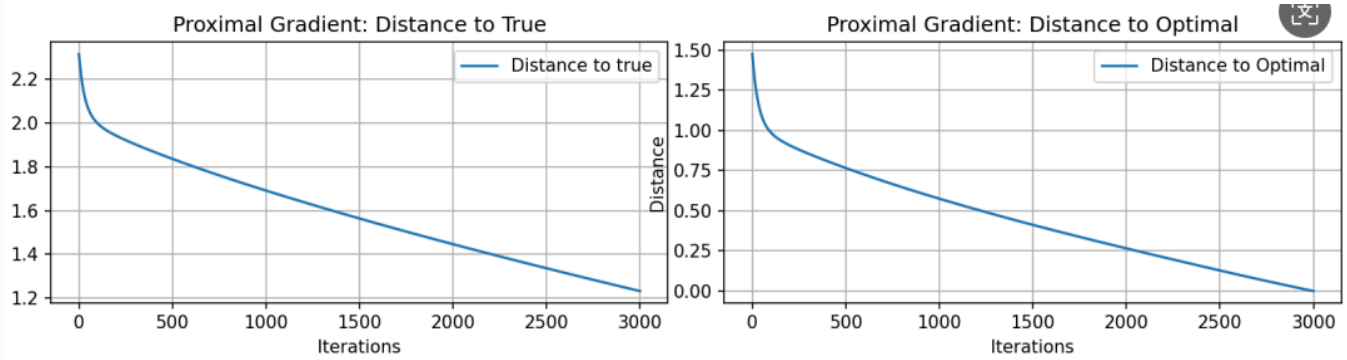
正则化参数=0.001:



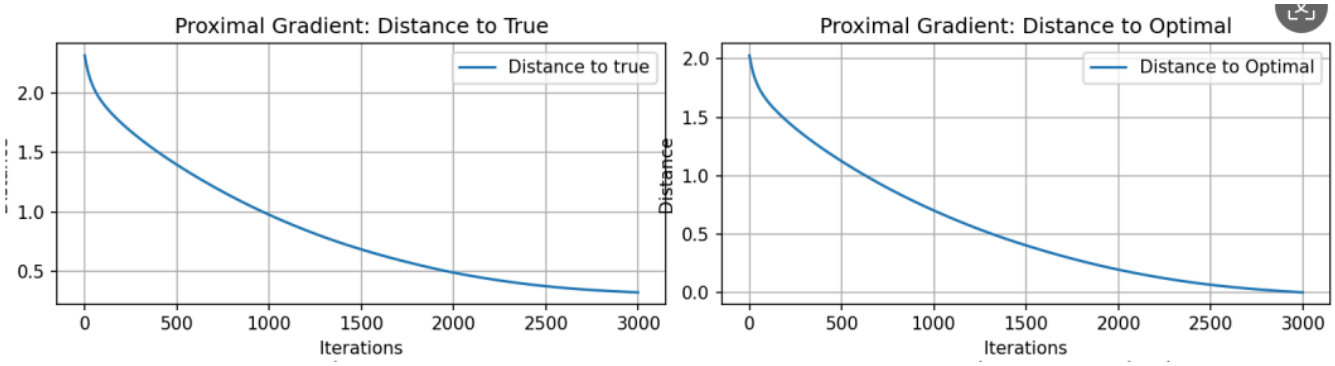
正则化参数=0.1:



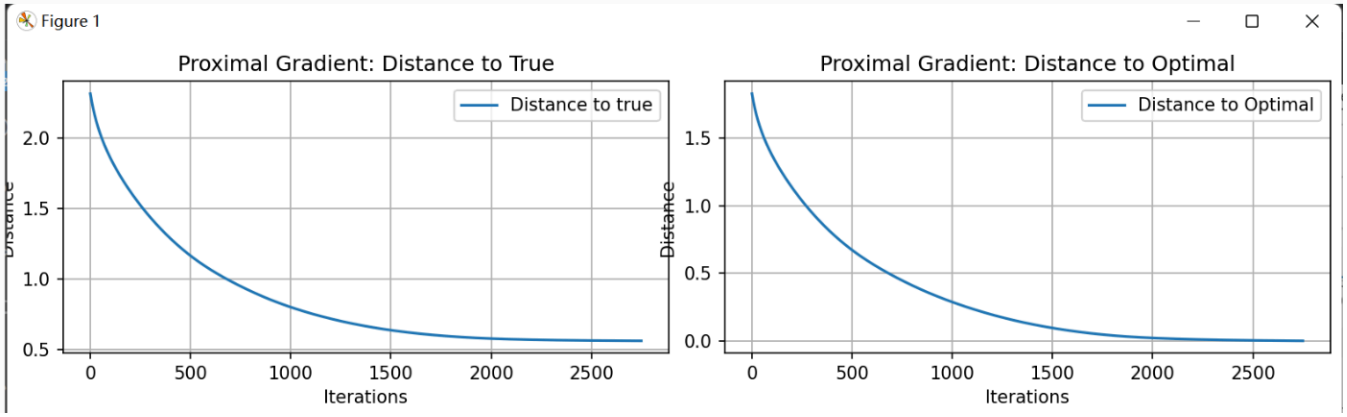
正则化参数=1:



正则化参数=5:



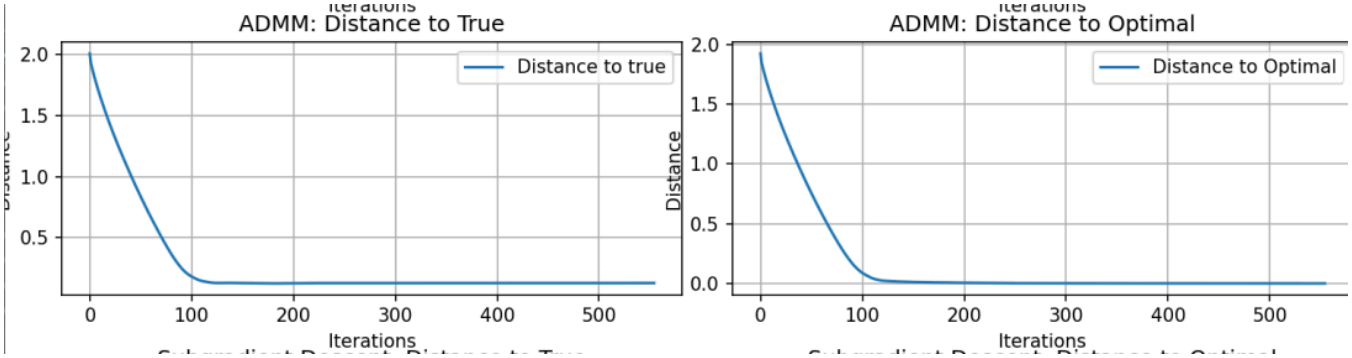
正则化参数=10:



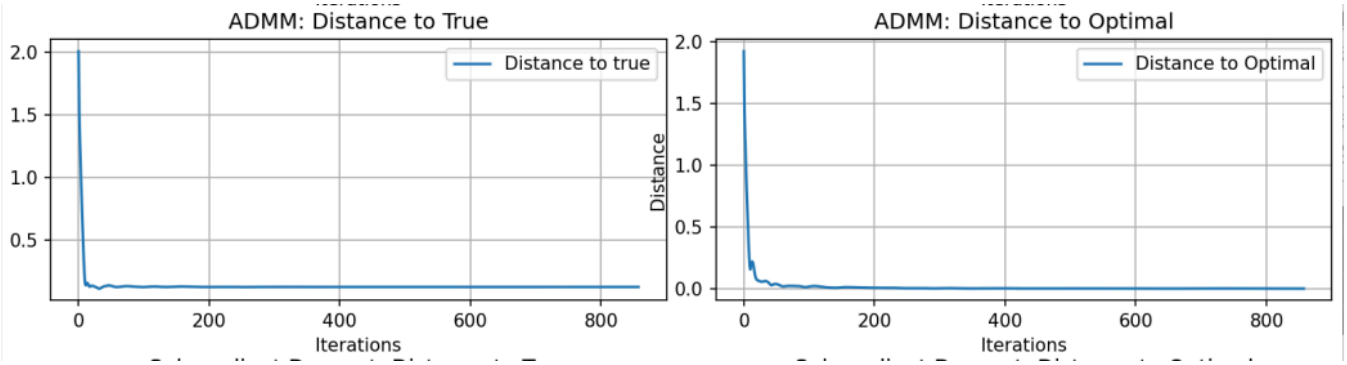
可以看出当正则化参数变大时，收敛速度减慢，但是在正则化参数更大时，例如5，10，其收敛速度要比正则化参数为0.1，1时要快，从0。001到10，邻近算法的最优解和真实值先接近后远离，其效果最好对应的正则化参数为5，这表明在选择正则化参数时，必须平衡解的稀疏性和逼近精度。较小的正则化参数能够得到更精确的解，而较大的正则化参数则可能导致过度稀疏化，从而丧失精度。

- **2.ADMM：**

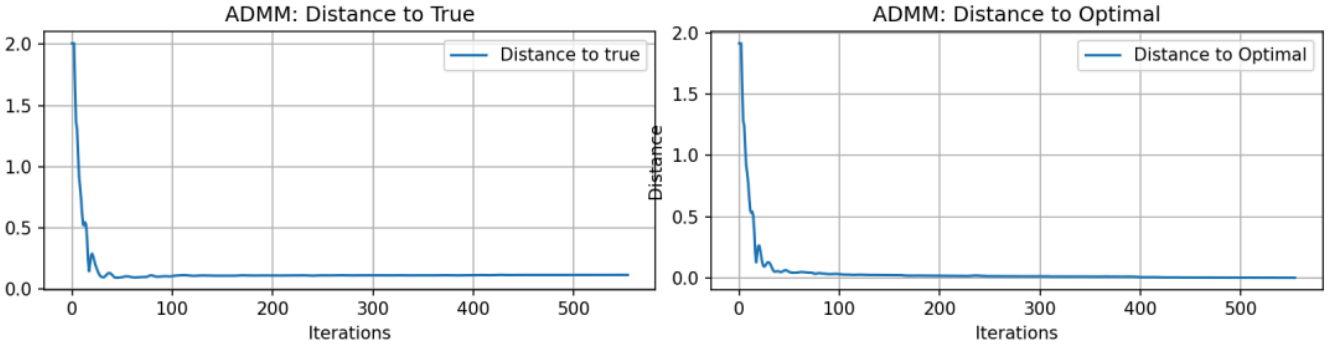
正则化参数=0.001:



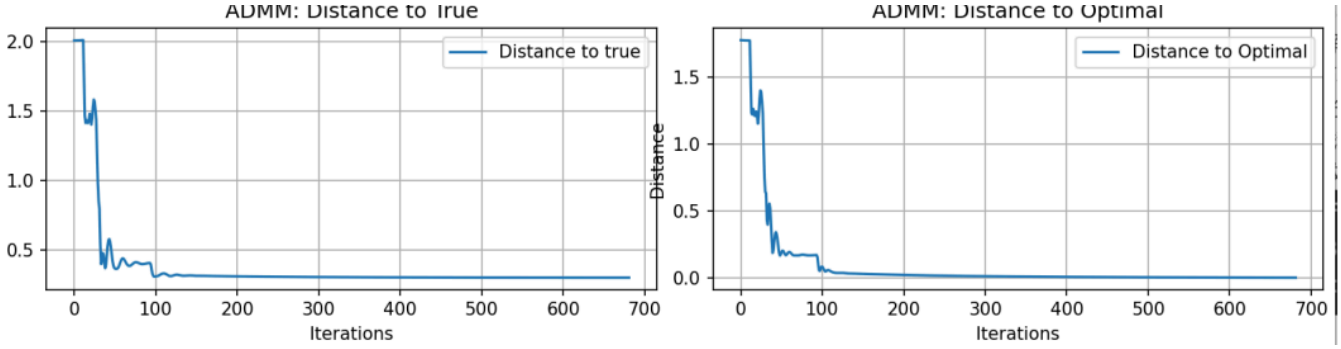
正则化参数=0.1:



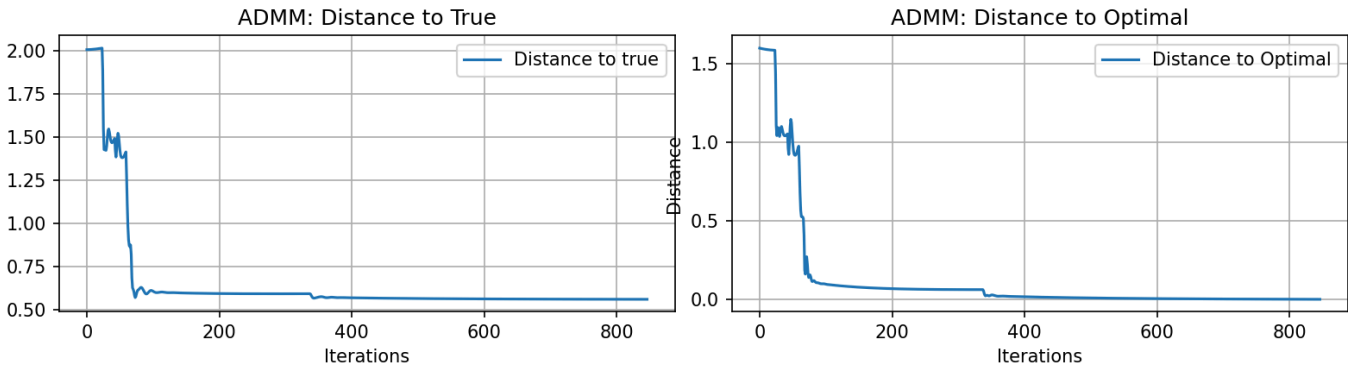
正则化参数=1:



正则化参数=5:



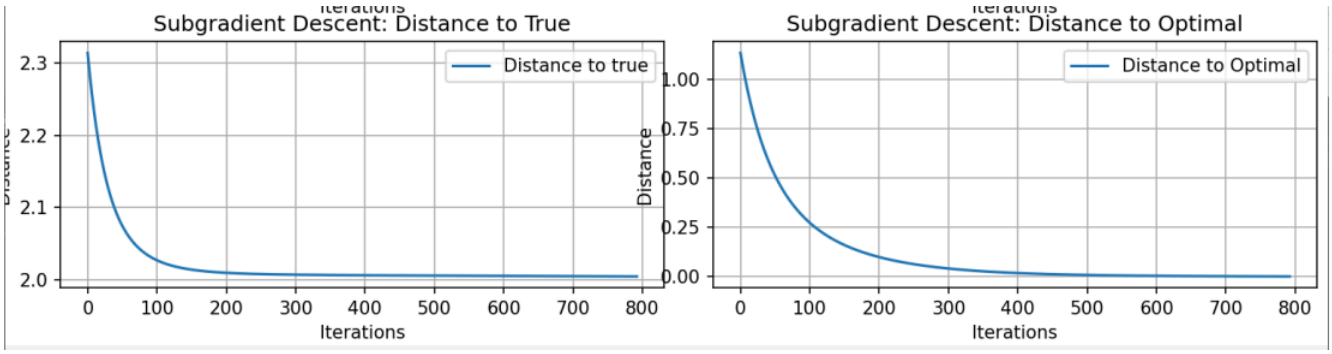
正则化参数=10:



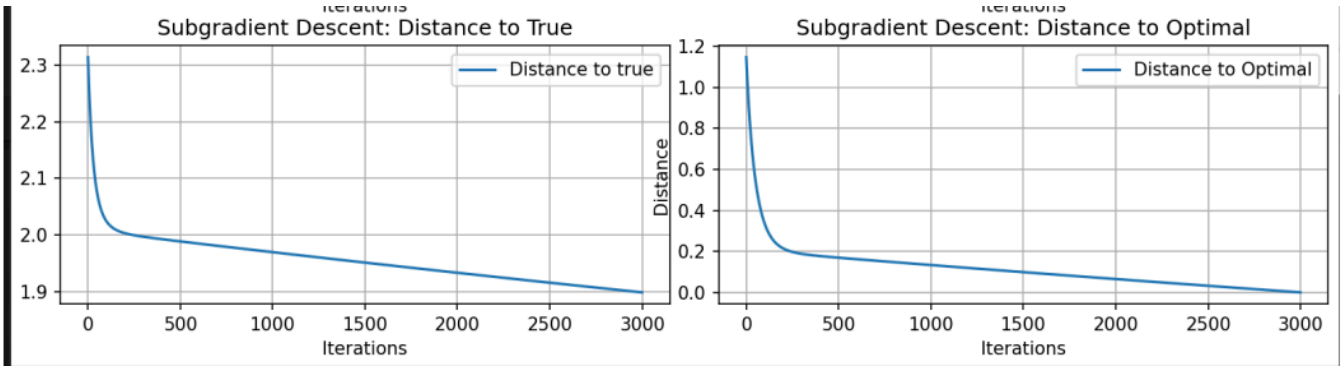
可以看出，随着正则化参数增大，收敛速度变快，但在正则化参数大于1时，收敛速度又开始减慢，在正则化参数较小时，其最优解和真实解比较接近，解的质量较高。随着正则化参数增大，其最优解和真实解的差距变大，例如5和10，较大的正则化参数（如 5, 10）收敛速度减慢，解偏离真实值和最优解，精度降低，表明正则化参数过大会导致过度稀疏化，解的质量明显下降。其最优正则化参数为0.1左右。

- **3.次梯度：**

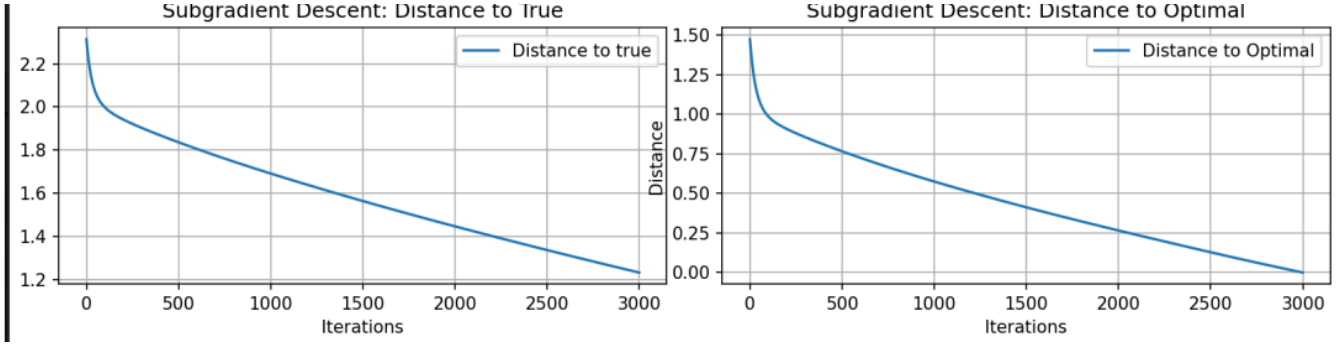
正则化参数=0.001:



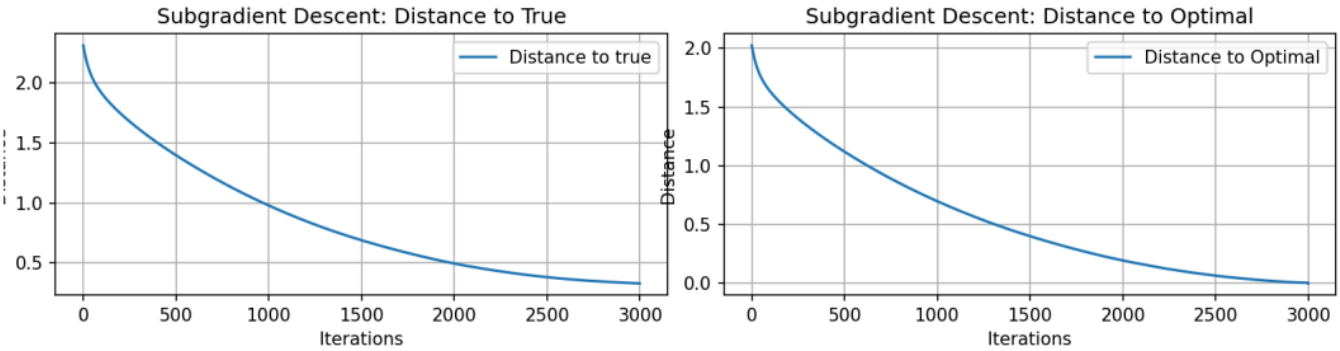
正则化参数=0.1:



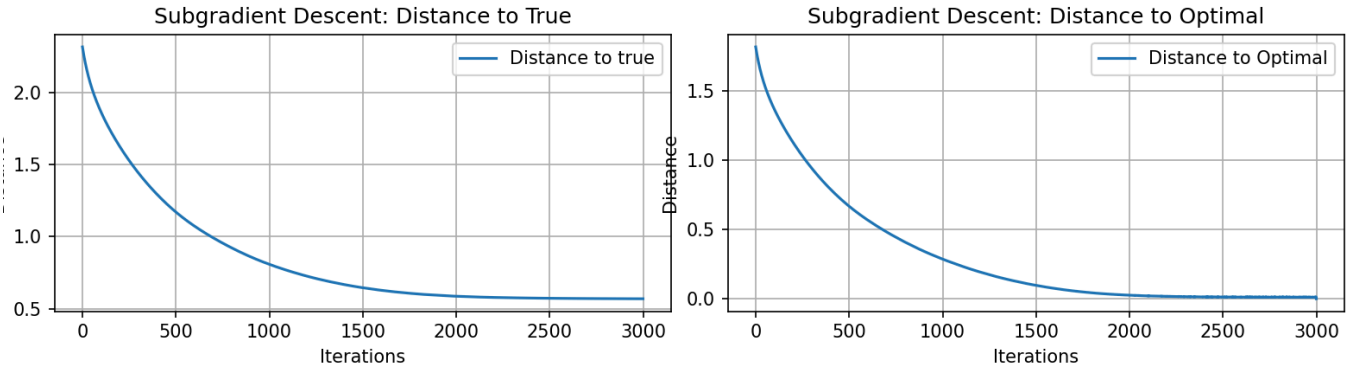
正则化参数=1:



正则化参数=5:



正则化参数=10:



可以看出，其结果和邻近算法类似，当正则化参数较小时（如0.001），其收敛速度比较快，解能够较好地逼近真实解，解的稀疏性较低，精度较高。随着正则化参数增大（0.1，1），可以看出，收敛速度降低，且解过于稀疏，导致与真实解和最优解之间的距离增加。但是当正则化参数再大时（如5,10），其收敛速度相较于1的时候会好一点，其最优正则化参数也为5。