大作业：优化算法实验

21307099 李英骏

liyj323@mail2.sysu.edu.cn

目录

1 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 $b_i = A_i\mathbf{x} + \mathbf{e}_i$，其中 b_i 为 5 维的测量值，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%，$\mathbf{e}_i$ 为 5 维的测量噪声。从所有 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：

$$\min_{\mathbf{x}} \left(\frac{1}{2} \sum_{i=1}^{10} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2 + \lambda \|\mathbf{x}\|_1 \right) \tag{1}$$

其中 $\lambda > 0$ 为正则化参数。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。

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 算法设计

2.1 邻近点梯度法

2.1.1 算法介绍

邻近算子 *(proximity operator):*

$$\mathrm{prox}_f(x) = \arg\min_{y\in\mathbb{R}^n} f(y) + \frac{1}{2}\|x-y\|_2^2, \tag{2}$$

投影算子 *(projection operator):*

$$\mathrm{proj}_X(x) = \arg\min_{y\in X} \|y-x\|_2^2, \tag{3}$$

软门限法 *(soft thresholding):*

$\arg\min_X \|X-B\|_2 + \lambda\|X\|_1$ 的解为

$$\mathrm{soft}(B,\lambda/2) = \begin{cases} B+\lambda/2, & \text{if } B < -\lambda/2 \\ 0, & \text{if } |B| \le \lambda/2 \\ B-\lambda/2, & \text{if } B > \lambda/2 \end{cases} \tag{4}$$

即

$$\mathrm{soft}(B,\lambda/2) = \mathrm{sign}(B) * \max(|B|-\lambda/2, 0) \tag{5}$$

若函数 f_0 有结构非光滑，则

$$\min_x f_0(x) = s(x) + r(x) \tag{6}$$

其中 $s(x)$ 是光滑部分，$r(x)$ 是非光滑部分。
则对光滑的 $s(x)$ 部分可以做梯度下降：

$$x^{k+\frac{1}{2}} = x^k - \alpha\cdot\nabla s(x^k) \tag{7}$$

对非光滑的 $r(x)$ 做邻近点投影

$$\min_x r(x) + \frac{1}{2\alpha}\| x-\hat{x} \|, \tag{8}$$

从而得到递推方程

$$x^{k+1} = \arg\min_x r(x) + \frac{1}{2\alpha}\|x - x^{k+\frac{1}{2}}\|_2^2 \tag{9}$$

2.1.2 具体算法说明

将原问题 (??) 代入上述式 (??)(??), 可得：

$$s(x) = \frac{1}{2}\sum_{i=1}^{10} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2 \tag{10}$$

$$r(x) = \lambda\|\mathbf{x}\|_1 \tag{11}$$

则式 (??) 特化为：

$$x^{k+\frac{1}{2}} = x^k - \alpha * \sum_{i=1}^{10} \mathbf{A_i^T}(\mathbf{A}_i x^k - \mathbf{b}_i) \tag{12}$$

式 (??) 特化为：

$$x^{k+1} = \arg\min_x(\lambda\|x\|_1 + \frac{1}{2\alpha}\|x - x^{k+\frac{1}{2}}\|_2^2) \tag{13}$$

其软门限法优化步为

$$x^{k+1} = \text{sign}(x^{x+\frac{1}{2}}) * \max(|x^{x+\frac{1}{2}}| - \alpha * \lambda, 0) \tag{14}$$

2.1.3 代码实现

```
class ProximalGradientDescent:
    def __init__(self, alpha, lambda_val, stop=1e-5, max_iter
        =100000):
        self.alpha = alpha
        self.lambda_val = lambda_val
        self.stop = stop
        self.max_iter = max_iter

    def gradient_step(self, A, b, x_k):
        gradient = np.sum([A_i.T @ ((A_i @ x_k) - b_i)
                           for A_i, b_i in zip(A, b)], axis=0)
        return x_k - self.alpha * gradient

    def soft_thresholding_1(self, x_k_half):
        for i in range(len(x_k_half)):
            if x_k_half[i] < - self.lambda_val*self.alpha:
                x_k_half[i] = x_k_half[i] + self.lambda_val*self.
                    alpha
            elif x_k_half[i] > self.lambda_val*self.alpha:
                x_k_half[i] = x_k_half[i]
                - self.lambda_val*self.alpha
```

```
20                else:
21                    x_k_half[i] = 0.0
22                return x_k_half
23
24        def check_convergence(self, x_k, x_k_ud):
25            return np.linalg.norm((x_k_ud - x_k), 2) < self.stop
26
27        def fit(self, A, b, x_true):
28            Xk = np.zeros_like(x_true)
29            X_opt_step = np.zeros(
30            (self.max_iter, x_true.shape[0], x_true.shape[1]))
31            count = 0
32            while count < self.max_iter:
33                Xk_half = self.gradient_step(A, b, Xk)
34                Xk_ud = self.soft_thresholding_1(Xk_half)
35                #print(Xk_ud.shape)
36                print("Iter ", count+1, ":", np.linalg.norm((Xk_ud -
                      x_true), 2))
37
38                X_opt_step[count] = Xk_ud
39                if self.check_convergence(Xk, Xk_ud):
40                    print("DONE")
41                    break
42
43                Xk = Xk_ud
44                count += 1
45
46            return X_opt_step[:count + 1,:]
```

代码的 8-11 行, gradient_step 函数计算了式 (??), 13-20 行计算了式 (??),
24-25 行的函数用于检验终止条件，即两次迭代的结果的 2-norm 距离小于 10^{-5}。
fit 函数为主要迭代部分。
将软门限法的实现向量化如下 (即式 (??))，经实验可提升 40% 左右的性能。

```
1     def soft_thresholding_1(self, x_k_half):
2         x_k_half = np.sign(x_k_half) * np.maximum((np.abs(x_k_half)-self
              .alpha*self.lambda_val),0)
3         return x_k_half
```

2.2 交替方向乘子法

2.2.1 算法介绍

对于优化问题

$$\min f_1(x) + f_2(x) \tag{15}$$

$$s.t. Ax + By = 0 \tag{16}$$

则其优化步为：

$$x^{k+1} = \arg\min_x \mathrm{L}_\rho\left(x, y^k, \lambda^k\right) \tag{17}$$

$$y^{k+1} = \arg\min_y \mathrm{L}_\rho\left(x^{k+1}, y, \lambda^k\right) \tag{18}$$

$$\lambda^{k+1} = \lambda^k + \rho\left(\mathrm{A}x^{k+1} + \mathrm{B}y^{k+1}\right) \tag{19}$$

2.2.2 具体算法说明

将原问题将原问题 (**??**) 代入 (即加入约束 $x - y = 0$)，有

$$f_1(x) = \frac{1}{2}\sum_{i=1}^{10} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2 \tag{20}$$

$$f_2(y) = \lambda\|y\|_1 \tag{21}$$

其增广拉氏函数为：

$$L_c(x, y, t) = \frac{1}{2}\sum_{i=1}^{10} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2 + \lambda\|y\|_1 + < t, x - y > + \frac{\rho}{2}\|x - y\|_2^2 \tag{22}$$

得迭代方程 (此处用 t 代表上面的 λ，防止写码的时候和原问题的正则项权重 λ 搞混淆)：

$$x^{k+1} = \left(\sum_{i=1}^{10} A_i^T A_i + \rho I\right)^{-1}\left(\sum_{i=1}^{10} A_i^T b + \rho y^k - t^k\right) \tag{23}$$

$$y^{k+1} = \arg\min_y \lambda\|y\|_1 + \frac{\rho}{2}\|y - x^{k+1} - \frac{t^k}{\rho}\|_2^2 \tag{24}$$

$$t^{k+1} = t^k + \rho(x^{k+1} - y^{k+1}) \tag{25}$$

其中 x 项部分是光滑的，直接求偏导即得，y 项部分需用软门限法 (**??**)，优化步如下：

$$y^{k+1} = \mathbf{sign}(x^{k+1} + \frac{t^k}{\rho}) * \max(|x^{k+1} + \frac{t^k}{\rho}| - \frac{1}{2}\cdot\frac{2}{c}\lambda) \tag{26}$$

$$= \mathbf{sign}(x^{k+1} + \frac{t^k}{\rho}) * \max(|x^{k+1} + \frac{t^k}{\rho}| - \frac{\lambda}{\rho}) \tag{27}$$

2.2.3 代码实现

```
    class ADMM:
        def __init__(self, rho, lamda, x_dim, b_dim, stop=1e-5, max_iter
            =100000):
            self.rho = rho
            self.lamda = lamda
            self.stop = stop
            self.max_iter = max_iter
            self.x_dim = x_dim
            self.b_dim = b_dim

        def update_x(self, A, b, Y, T):
            A_T = self.rho * np.eye(self.x_dim)
            A_b = np.zeros((self.x_dim, 1))

            for i in range(10):
                A_T += A[i].T @ A[i]
                A_b += A[i].T @ b[i]

            A_T_inv = np.linalg.inv(A_T)
            X_update = A_T_inv @ (A_b + self.rho * Y - T)
            return X_update

        def update_y(self, X, T):
            Y_update = np.sign(X + T / self.rho) * np.maximum(np.abs(X +
            T / self.rho) - self.lamda / self.rho, 0)
            return Y_update

        def update_t(self, T, X, Y):
            T_update = T + self.rho * (X - Y)
            return T_update

        def check_convergence(self, x_k, x_k_ud):
            return np.linalg.norm((x_k_ud - x_k), 2) < self.stop
```

```
33          def fit(self, A, b, x_true):
34              # 初始化 X, Y, T
35              X = np.zeros((self.x_dim, 1))
36              Y = np.zeros((self.x_dim, 1))
37              T = np.zeros((self.x_dim, 1))
38              count = 0
39              X_opt_step = np.zeros(
40                  (self.max_iter, x_true.shape[0], x_true.shape[1]))
41
42              while count < self.max_iter:
43                  X_update = self.update_x(A, b, Y, T)
44                  Y_update = self.update_y(X_update, T)
45                  T_update = self.update_t(T, X_update, Y_update)
46
47                  X_opt_step[count] = X_update
48                  print("Iter ", count+1, ":", np.linalg.norm((X_update -
                        x_true), 2))
49                  if self.check_convergence(X_update, X):
50                      break
51
52                  # 更新 X, Y, T
53                  X, Y, T = X_update, Y_update, T_update
54                  count += 1
55              return X_opt_step[:count + 1, :]
```

10-20 行的 update_x 函数实现了式 (??), 22-25 行的 update_y 实现了式 (??), 27-29 行的 update_t 实现了式 (??),fit 函数为主要迭代部分，按顺序依次调用三个函数并记录每步优化结果。

2.3 次梯度法

2.3.1 算法介绍

$$x^{k+1} = x^k - \alpha^k * g_0(x^k), g_0 \in \partial f_0(x) \tag{28}$$

2.3.2 具体算法说明

对于原问题 (??)，g_0 有以下三种情形：

$$g_0(x) = \begin{cases} \sum_{i=1}^{10} A_i^T(A_i x - b) + \lambda, & \text{if } x > 0 \\ \sum_{i=1}^{10} A_i^T(A_i x - b) + \lambda * \text{rand}\left[-1, 1\right], & \text{if } x = 0 \\ \sum_{i=1}^{10} A_i^T(A_i x - b) - \lambda, & \text{if } x < 0 \end{cases} \tag{29}$$

我选择 α 为变长步长

$$\alpha^{k+1} = \frac{\alpha^k}{\sqrt{step + 1}} \tag{30}$$

2.3.3 代码实现

```
1   class SubgradientMethod:
2       def __init__(self, lamda, alpha, x_dim, b_dim, stop=1e-5,
            max_iter=80000):
3           self.lamda = lamda
4           self.alpha = alpha
5           self.stop = stop
6           self.max_iter = max_iter
7           self.x_dim = x_dim
8           self.b_dim = b_dim
9
10      def check_convergence(self, x_k, x_k_ud):
11          return np.linalg.norm((x_k_ud - x_k), 2) < self.stop
```

上方函数主要用于初始化和检测终止条件。

```
12          def fit(self, A, b, x_true):
13              X_temp = np.zeros((self.x_dim, 1))
14              count = 0
15              X_opt_step = np.zeros(
16                  (self.max_iter, x_true.shape[0], x_true.shape[1]))
17
18              while count < self.max_iter:
19                  g = np.zeros((self.x_dim, 1))
20                  for i in range(len(A)): # 遍历A中的每个矩阵A_i
21                      g += A[i].T @ (A[i] @ X_temp - b[i].reshape(-1, 1))
22
23                  X_new = np.array([np.random.uniform(-1, 1) if x ==
24                                    0 else np.sign(x) for x in X_temp]).reshape
                                      (-1, 1)
25                  g += self.lamda * X_new
26                  alphak = self.alpha / np.sqrt(count + 1)
27                  X_update = X_temp - alphak * g
28                  print("Iter ", count+1, ":", np.linalg.norm((X_update -
                        x_true), 2))
29                  X_opt_step[count] = X_update
30                  if self.check_convergence(X_update, X_temp):
31                      break
32
33                  X_temp = X_update
34                  count += 1
35
36              return X_opt_step[:count + 1, :]
```

fit 函数的 20-27 行实现了式 (??) 的功能，其中 26 行实现了式 (??) 的功能。

3 其他代码实现

3.1 数据生成

```python
def generate_data(x_dim, b_dim, sparsity, seed=42):
    np.random.seed(seed)

    x_true = np.zeros((x_dim,1))
    non_zero_indices = np.random.choice(
        x_dim, int(sparsity * x_dim), replace=False)
    x_true[non_zero_indices,0] = np.random.randn(int(sparsity *
        x_dim))
    # generate A_i
    A = [np.random.randn(b_dim, x_dim) for _ in range(10)]

    # generate noise e_i
    e = [np.random.randn(b_dim,1) * np.sqrt(0.1) for _ in range(10)]

    # generate b_i
    b = [A_i @ x_true + e_i for A_i, e_i in zip(A, e)]

    return x_true, A, e, b
```

我锁定了 seed 以保证可重复性，并且在 data_generator.py 中验证了均值和方差满足条件。如下图

```
(intro2dl) PS C:\Users\liyj\Desktop\DL\Intro2DL\ML_and_DataMining\code> & C
ning/code/opt_project/project1/code/data_generator.py
(200, 1) (10, 5, 200) (10, 5, 1) (10, 5, 1)
Actual Sparsity: 5e-05
Mean of non-zero elements in x_true: 0.0007310430481612477
Variance of non-zero elements in x_true: 1.0042737979250353
Mean of A: 0.00020964266502502508, Variance of A: 0.9991911641162454
Mean of e: -0.0019840159017425686, Variance of e: 0.09778076976557797
(intro2dl) PS C:\Users\liyj\Desktop\DL\Intro2DL\ML_and_DataMining\code>
```

图 1: 数据生成器测试

4 数值实验及结果分析

4.1 邻近点梯度法

取 $\alpha = 0.001$

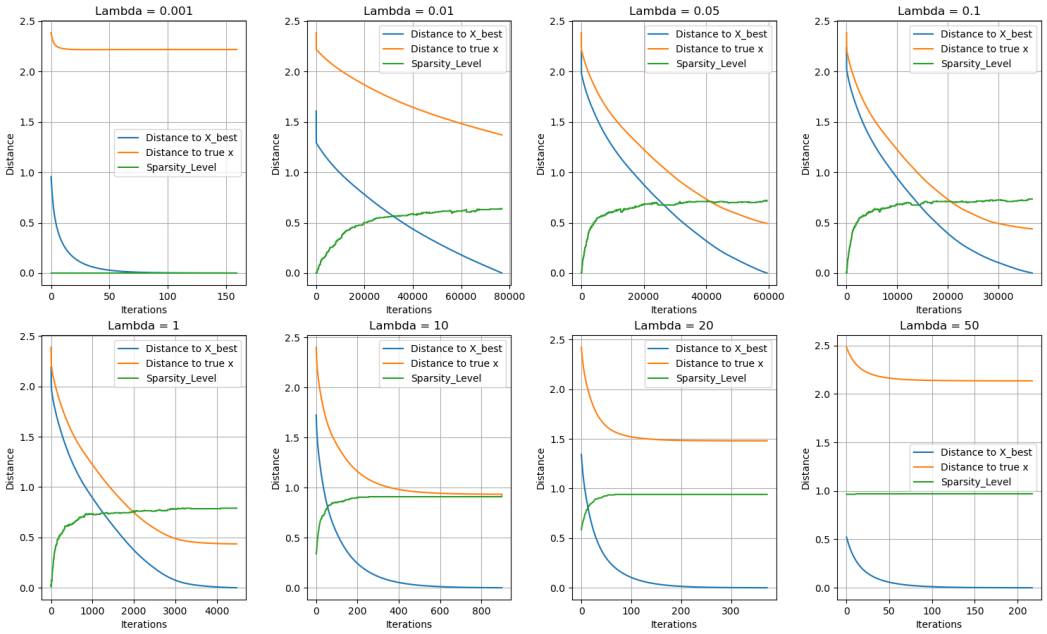

图 2: 邻近点梯度法

观察蓝色曲线 (与最优解的 2-norm 距离，可以认为反映了收敛速度) 可以看出，随着 λ 增大，函数的收敛速度加快。这可能是因为稀疏化正则项的优化速度更快，当它权重更大时自然收敛得更快。

同时观察橙色曲线 (与真实值的距离) 可以发现，在合适的 λ 范围内 (大约在 0.1-1)，函数可以被优化到最接近真实的 X，此时可以认为最小二乘项的权重和稀疏正则化项的权重比例是合适的

另外，观察绿色曲线可以很自然地发现，当 λ 较小 (0.001) 时，结果几乎是稠密的；而随着 λ 增大，结果越来越稀疏；$\lambda > 10$ 后，结果几乎是完全稀疏的。且稀疏程度随着迭代次数提高。

4.2 交叉方向乘子法

我们选定 $\rho = 0.005$, 观察如下式 (**??**) 可以发现：

$$L_c(x, y, t) = \frac{1}{2}\sum_{i=1}^{10} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2 + \lambda\|y\|_1 + < t, x - y > + \frac{\rho}{2}\|x - y\|_2^2$$

当 λ 较大时，函数更好地约束了 y 的稀疏度，但响应地 $\|x - y\|_2^2$ 的权重减少，即减弱了对条件 $x - y = 0$ 的约束，反而会降低 x 的稀疏程度。实验如下：

4.2.1 当 λ 相比 ρ 较大时

以向量中零元素的数量计算稀疏度（标准算法）

实验结果见下图 (**??**)。

1. 观察橙、蓝曲线可以看出，随着 λ 增加，算法收敛得更慢，在过大时甚至不收敛。
2. 观察橙色曲线，可以发现在 $lambda \leq 1$ 时，算法都可以收敛到一个较为接近 x_true 的结果上，但更大的 λ 会导致最小二乘项的权重下降，从而使迭代振荡得更加剧烈。
3. 观察绿色曲线，可以发现当以零元素的数量计算稀疏度时，矩阵几乎是不稀疏的（稀疏度 ≈ 0），这是因为稀疏化正则项和软门限（这是直接产生严格 0 的主要渠道）都作用在 y 上，通过 $\|x - y\|_2^2$ 很难产生出严格的 0 值。

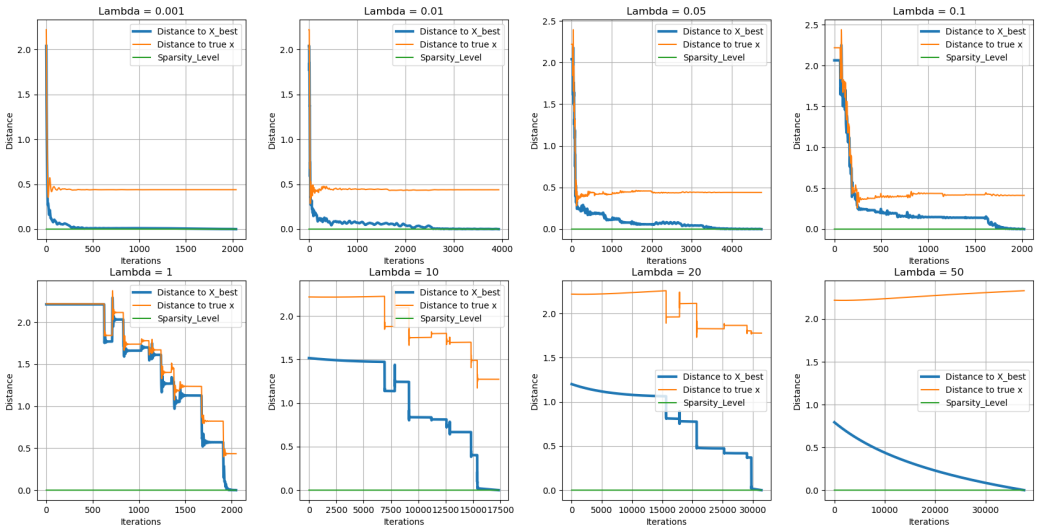

图 3: 交叉方向乘子法 2.0

以向量中 $\leq 10^{-3}$ 项的数量计算稀疏度（非标准算法）

实验结果见下图 (**??**)，可以看出这个度量较好地体现了矩阵的稀疏程度，尽管不是严格的稀疏度。它验证了上述说法：即在 λ 和 ρ 的比例合适时，我们可以获得较好的稀疏度，λ 过大时，稀疏度会明显下降。

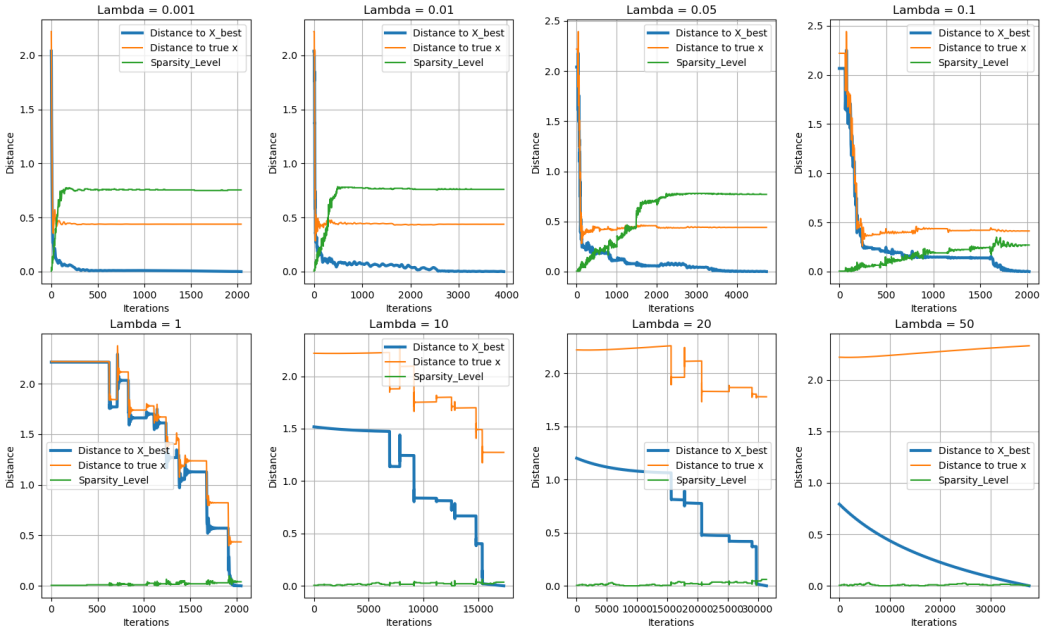

图 4: 交叉方向乘子法 2.1

4.2.2 当 λ 相比 ρ 较小时

以向量中零元素的数量计算稀疏度（标准算法）

实验结果见下图 (**??**)。

1. 观察橙、蓝曲线可以看出，随着 λ 增加，算法收敛得更快了。
2. 观察橙色曲线，可以发现 $lambda$ 取这些值时，算法都可以收敛到一个较为接近 x_true 的结果上，但更大的 λ 会导致最小二乘项的权重下降，从而使迭代振荡得更加剧烈。
3. 观察绿色曲线，可以发现当以零元素的数量计算稀疏度时，矩阵几乎是不稀疏的（稀疏度 ≈ 0），这是因为稀疏化正则项和软门限（这是直接产生严格 0 的主要渠道）都作用在 y 上，通过 $\|x-y\|_2^2$ 很难产生出严格的 0 值。

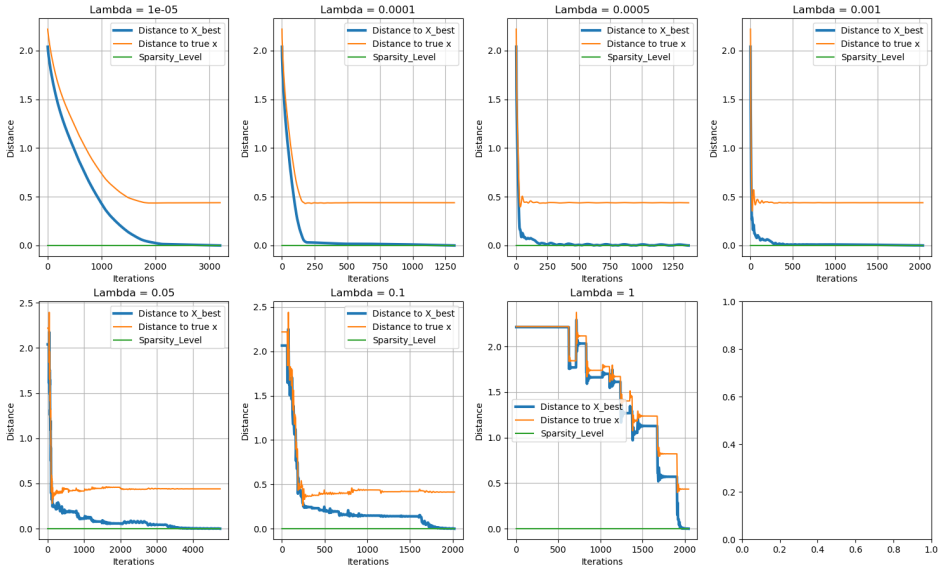

图 5: 交叉方向乘子法 2.2

以向量中 $\leq 10^{-2}$，$\leq 10^{-3}$，$\leq 10^{-4}$ 项的数量计算稀疏度（非标准算法）

实验结果见下图 (**??**)。图中三条稀疏度曲线分别是 Sparsity_Level_0：$\leq 10^{-2}$，Sparsity_Level_1：$\leq 10^{-3}$，Sparsity_Level_2：$\leq 10^{-4}$.

可以看出，在 $lambda \leq 0.001$ 时，优化结果和它的稀疏程度都较好，而当 $\lambda \geq 0.05$ 即 ρ 时，$\leq 10^{-4}$ 的项的数量显著下降。

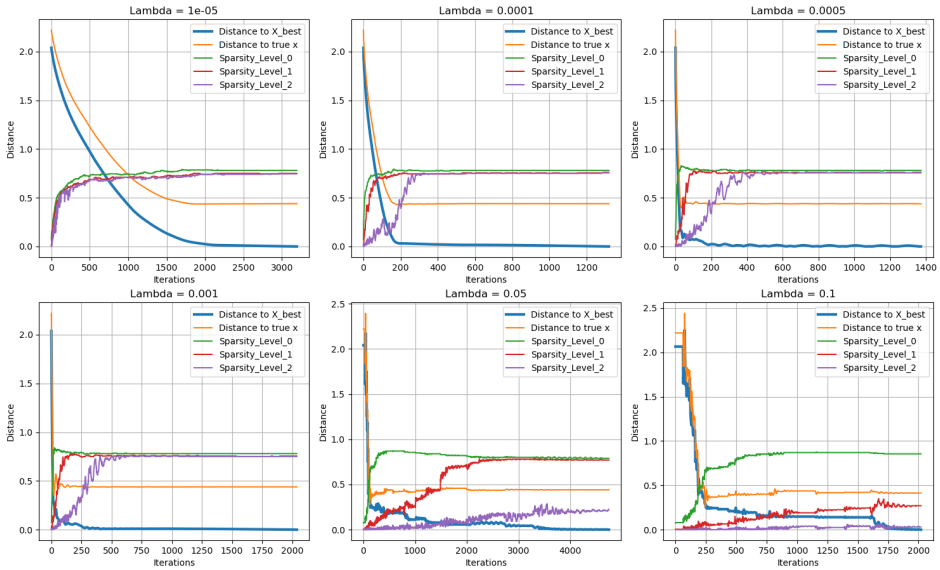

图 6: 交叉方向乘子法 2.3

4.3 次梯度法

4.3.1 取 $\alpha = 0.0001$

实验结果如图：

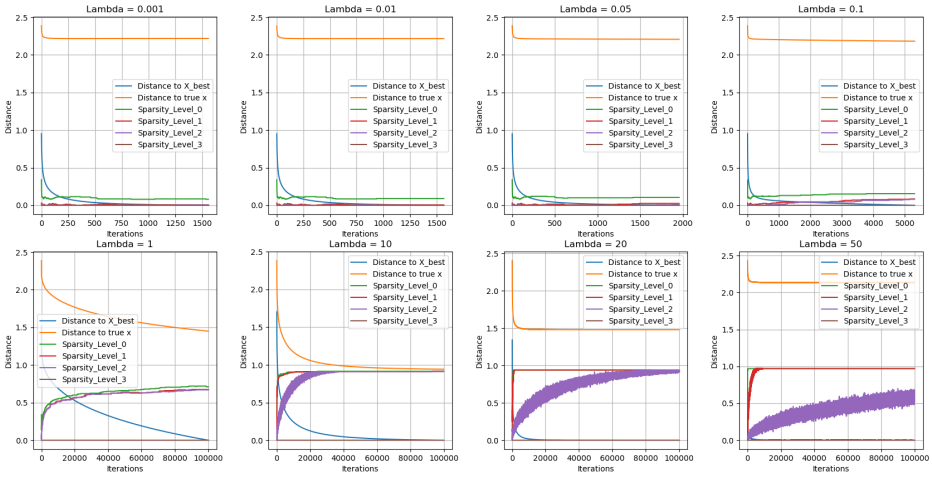

图 7: 次梯度法

图中三条稀疏度曲线分别是 Sparsity_Level_0：$\leq 10^{-2}$，Sparsity_Level_1：$\leq 10^{-3}$，Sparsity_Level_2：$\leq 10^{-4}$，Sparsity_Level_3：$= 0$，观察蓝色曲线可以发现，随着 λ 的增加，算法收敛先变慢后变快，在 λ 为 1 时最慢。观察橙色曲线，可以发现，在 λ 取 1-10 左右时，函数可以收敛到最接近 X_true

观察剩余四色曲线，可以发现，λ 取 1-20 时，所得值的稀疏程度较高，且均集中在 $0 \sim 10^{-4}$。注意到次梯度法中，严格等于 0 的元素也是几乎不存在的（棕色的 level3 曲线一直几乎为 0）。

另外可以注意到，即使 X 与 X_true 和 X_best 的距离已经几乎不变，$\leq 10^{-4}$ 的项的数量仍然在一直增加，这表明算法正在一个以 X_true 范数球面附近搜索。