最优化理论 大作业：优化算法实验

21304219 刘文婧

问题描述

一个 10 节点的分布式系统：

- $\boldsymbol{x}$: 200 维的未知稀疏向量，稀疏度为 5（只有5个元素非 0）。
- 在每一个节点 i 上，使用测量矩阵 $A_i \in \mathbb{R}^{5\times 200}$ 测量得到 5 维测量值 $\boldsymbol{b}_i = A_i\boldsymbol{x} + \boldsymbol{e}_i$，其中 $\boldsymbol{e}_i$ 是 5 维的测量噪声。
- 根据所有测量值 $\boldsymbol{b}_i$, A_i，恢复未知向量 $\boldsymbol{x}$。由于测量噪声的存在，即要找到满足：使得 所有 $\|A_i\boldsymbol{x} - \boldsymbol{b}_i\|_2^2$ 尽量小的 $\boldsymbol{x}$，建立一范数正则化的最小二乘模型如下：

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

- 其中 $\lambda > 0$ 是正则化参数。一范数正则化项就是希望得到的解 $\boldsymbol{x}$ 尽量稀疏，λ 越大，该项影响越大，越倾向于生成更稀疏解。

实验假设与具体过程

- 首先生成真值 $\hat{\boldsymbol{x}}$，$\hat{\boldsymbol{x}}$ 中只有 5 个非 0 元素，且服从高斯分布 $\mathcal{N}(0,1)$。我们需要观察利用不同算法恢复得到的 $\boldsymbol{x}$ 和 实际真值 $\hat{\boldsymbol{x}}$ 的差距。
- 还要生成每个节点的测量矩阵 A_i，A_i 中的元素服从高斯分布 $\mathcal{N}(0,1)$；生成每个结点的测量噪声 $\boldsymbol{e}_i$，$\boldsymbol{e}_i$ 中的元素服从高斯分布 $\mathcal{N}(0,0.1)$。
- 通过画这些图来观察算法实验结果：
  - $\|\boldsymbol{x}^k - \hat{\boldsymbol{x}}\|_2$ 随迭代次数 k 的变化图像，其中 $\boldsymbol{x}^k$ 指算法第 k 次迭代得到的解。用来观察算法求解问题的效果。
  - $\|\boldsymbol{x}^k - \boldsymbol{x}^*\|_2$ 随迭代次数 k 的变化图像，其中 $\boldsymbol{x}^*$ 指一个稳定的优化算法所能得到的最优解。用来观察收敛性质。
  - 算法最终收敛到的解 $\boldsymbol{x}$ 的稀疏度 随 正则化参数 λ 变化的图像。用来讨论正则化参数 λ 对计算结果的影响。

算法设计

1.临近梯度法

选固定步长 $\alpha_k = \alpha$

对

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

每轮迭代的第 1 步，对可微的部分进行梯度下降：
记

$$s(\boldsymbol{x})=\frac{1}{2}\|A_1\boldsymbol{x}-\boldsymbol{b}_1\|_2^2+\cdots+\frac{1}{2}\|A_{10}\boldsymbol{x}-\boldsymbol{b}_{10}\|_2^2$$

有

$$\frac{\partial s(\boldsymbol{x})}{\partial \boldsymbol{x}}=A_1^\top(A_1\boldsymbol{x}-\boldsymbol{b}_1)+\cdots+A_{10}^\top(A_{10}\boldsymbol{x}-\boldsymbol{b}_{10})$$

$$\begin{aligned}\boldsymbol{x}^{k+\frac{1}{2}}=&\boldsymbol{x}^k-\alpha\nabla s(\boldsymbol{x}^k)\\=&\boldsymbol{x}^k-\alpha\left[A_1^\top(A_1\boldsymbol{x}^k-\boldsymbol{b}_1)+\cdots+A_{10}^\top(A_{10}\boldsymbol{x}^k-\boldsymbol{b}_{10})\right]\end{aligned}$$

每轮迭代的第 2 步，对另一部分求临近点投影：
记

$$r(\boldsymbol{x})=\lambda\|\boldsymbol{x}\|_1$$

$r(\boldsymbol{x})$ 在 $\boldsymbol{x}^{k+\frac{1}{2}}$ 处临近点投影：

$$\mathrm{prox}_r(\boldsymbol{x}^{k+\frac{1}{2}})=\arg\min_{\boldsymbol{x}}\left\{r(\boldsymbol{x})+\frac{1}{2\alpha}\|\boldsymbol{x}-\boldsymbol{x}^{k+\frac{1}{2}}\|_2^2\right\}$$

即

$$\boldsymbol{x}^{k+1}=\arg\min_{\boldsymbol{x}}\left\{\lambda\|\boldsymbol{x}\|_1+\frac{1}{2\alpha}\|\boldsymbol{x}-\boldsymbol{x}^{k+\frac{1}{2}}\|_2^2\right\}$$

这个问题是按维度可分的，在每一维上，求解：

$$\arg\min_{x_i}\left\{\lambda|x_i|+\frac{1}{2\alpha}(x_i-x_i^{k+\frac{1}{2}})^2\right\}$$

由KKT条件，最优解 x_i 满足：存在次梯度$=0$，如下：

$$\partial\lambda|x_i| = \begin{cases} \lambda, & x_i > 0 \\ [-\lambda, \lambda], & x_i = 0 \\ -\lambda, & x_i < 0 \end{cases}$$

$$\Rightarrow \begin{cases} x_i > 0, & \lambda + \frac{1}{\alpha}(x_i - x_i^{k+\frac{1}{2}}) = 0 \quad \Rightarrow \quad x_i = x_i^{k+\frac{1}{2}} - \alpha\lambda \\ x_i = 0, & 0 \in [-\lambda + \frac{1}{\alpha}(x_i - x_i^{k+\frac{1}{2}}), \lambda + \frac{1}{\alpha}(x_i - x_i^{k+\frac{1}{2}})] \quad \Rightarrow \quad x_i^{k+\frac{1}{2}} \in [-\alpha\lambda, \alpha\lambda] \\ x_i < 0, & -\lambda + \frac{1}{\alpha}(x_i - x_i^{k+\frac{1}{2}}) = 0 \quad \Rightarrow \quad x_i = x_i^{k+\frac{1}{2}} + \alpha\lambda \end{cases}$$

得到（软门限算法）

$$x_i^{k+1} = \begin{cases} x_i^{k+\frac{1}{2}} - \alpha\lambda, & \text{if} \quad x_i^{k+\frac{1}{2}} > \alpha\lambda \\ 0, & \text{if} \quad x_i^{k+\frac{1}{2}} \in [-\alpha\lambda, \alpha\lambda] \\ x_i^{k+\frac{1}{2}} + \alpha\lambda, & \text{if} \quad x_i^{k+\frac{1}{2}} < -\alpha\lambda \end{cases}$$

这就是临近点梯度法的迭代格式：

$$\boldsymbol{x}^{k+\frac{1}{2}} = \boldsymbol{x}^k - \alpha \left[A_1^\top (A_1 \boldsymbol{x}^k - \boldsymbol{b}_1) + \cdots + A_{10}^\top (A_{10} \boldsymbol{x}^k - \boldsymbol{b}_{10}) \right]$$

$$\boldsymbol{x}^{k+1} = \begin{cases} \boldsymbol{x}^{k+\frac{1}{2}} - \alpha\lambda, & \text{if} \quad \boldsymbol{x}^{k+\frac{1}{2}} > \alpha\lambda \\ 0, & \text{if} \quad \boldsymbol{x}^{k+\frac{1}{2}} \in [-\alpha\lambda, \alpha\lambda] \\ \boldsymbol{x}^{k+\frac{1}{2}} + \alpha\lambda, & \text{if} \quad \boldsymbol{x}^{k+\frac{1}{2}} < -\alpha\lambda \end{cases}$$

代码实现如下：

```
def proximal_gradient(x_0, A, b, iters, regu_lambda, step_size):
    """
    Proximal Gradient Method
    :param x_0: variable to be updated
    :param A: measurement matrix
    :param b: measurement result with noise
    :param iters: iterations num
    :param regu_lambda: regularization coefficient
    :param step_size: step size alpha
    :return: updated x, x_list
    """
    iter_list = []
    x = np.copy(x_0)

    for i in range(iters):
        grad = 0
        for j in range(10):
            grad += np.dot(A[j].T, np.dot(A[j], x) - b[j])

        x -= step_size * grad
        # soft threshold
        condlist = [x > step_size * regu_lambda, x < - step_size * regu_lambda]
        choicelist = [x - step_size * regu_lambda, x + step_size * regu_lambda]
        x = np.select(condlist, choicelist, 0)

        iter_list.append(np.copy(x))

    return x, iter_list
```

2.交替方向乘子法

把问题写成有约束优化问题

$$\begin{aligned}\min_{\boldsymbol{x},\boldsymbol{y}} \quad & \frac{1}{2}\|A_1\boldsymbol{x}-\boldsymbol{b}_1\|_2^2+\cdots+\frac{1}{2}\|A_{10}\boldsymbol{x}-\boldsymbol{b}_{10}\|_2^2+\lambda\|\boldsymbol{y}\|_1 \\ \text{s.t.} \quad & \boldsymbol{x}-\boldsymbol{y}=0\end{aligned}$$

那么拉格朗日增广函数：

$$L_c(\boldsymbol{x},\boldsymbol{y},\boldsymbol{v})=\frac{1}{2}\|A_1\boldsymbol{x}-\boldsymbol{b}_1\|_2^2+\cdots+\frac{1}{2}\|A_{10}\boldsymbol{x}-\boldsymbol{b}_{10}\|_2^2+\lambda\|\boldsymbol{y}\|_1+\langle\boldsymbol{x}-\boldsymbol{y},\boldsymbol{v}\rangle+\frac{c}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2$$

交替方向乘子法迭代过程：

$$
\begin{aligned}
\boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} L_c(\boldsymbol{x}, \boldsymbol{y}^k, \boldsymbol{v}^k) \\
&= \arg\min_{\boldsymbol{x}} \frac{1}{2}\|A_1\boldsymbol{x} - \boldsymbol{b}_1\|_2^2 + \cdots + \frac{1}{2}\|A_{10}\boldsymbol{x} - \boldsymbol{b}_{10}\|_2^2 + \langle \boldsymbol{x}, \boldsymbol{v}^k \rangle + \frac{c}{2}\|\boldsymbol{x} - \boldsymbol{y}^k\|_2^2
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{y}^{k+1} &= \arg\min_{\boldsymbol{y}} L_c(\boldsymbol{x}^{k+1}, \boldsymbol{y}, \boldsymbol{v}^k) \\
&= \arg\min_{\boldsymbol{y}} \lambda\|\boldsymbol{y}\|_1 + \langle -\boldsymbol{y}, \boldsymbol{v}^k \rangle + \frac{c}{2}\|\boldsymbol{x}^{k+1} - \boldsymbol{y}\|_2^2
\end{aligned}
$$

$$
\boldsymbol{v}^{k+1} = \boldsymbol{v}^k + c(\boldsymbol{x}^{k+1} - \boldsymbol{y}^{k+1})
$$

具体地求解，得到（此时求解 $\boldsymbol{y}^{k+1}$ 类似于临近点投影的求解，也是一个软门限算法）：
对 $\boldsymbol{x}^{k+1}$，由梯度 $= 0$：

$$
\begin{aligned}
& \sum_{i=1}^{10} A_i^\top (A_i\boldsymbol{x}^{k+1} - \boldsymbol{b}_i) + \boldsymbol{v}^k + c(\boldsymbol{x}^{k+1} - \boldsymbol{y}^k) = 0 \\
\Leftrightarrow \quad & \left(\sum_{i=1}^{10} A_i^\top A_i + cI\right)\boldsymbol{x}^{k+1} = \sum_{i=1}^{10} A_i^\top \boldsymbol{b}_i - \boldsymbol{v}^k + c\boldsymbol{y}^k \\
\Leftrightarrow \quad & \boldsymbol{x}^{k+1} = \left(\sum_{i=1}^{10} A_i^\top A_i + cI\right)^{-1}\left(\sum_{i=1}^{10} A_i^\top \boldsymbol{b}_i - \boldsymbol{v}^k + c\boldsymbol{y}^k\right)
\end{aligned}
$$

对 $\boldsymbol{y}^{k+1}$，

$$
\begin{aligned}
& 0 \in (\partial\lambda|y_i^{k+1}|) - v_i^k - c(x_i^{k+1} - y_i^{k+1}) \\
\Rightarrow \quad & y_i^{k+1} = \begin{cases} \frac{v_i^k - \lambda}{c} + x_i^{k+1}, & v_i^k + cx_i^{k+1} > \lambda \\ 0, & v_i^k + cx_i^{k+1} \in [-\lambda, \lambda] \\ \frac{v_i^k + \lambda}{c} + x_i^{k+1}, & v_i^k + cx_i^{k+1} < -\lambda \end{cases}
\end{aligned}
$$

于是交替方向乘子法的具体参数更新公式：

$$\boldsymbol{x}^{k+1} = \left(\sum_{i=1}^{10} A_i^\top A_i + cI \right)^{-1} \left(\sum_{i=1}^{10} A_i^\top \boldsymbol{b}_i - \boldsymbol{v}^k + c\boldsymbol{y}^k \right)$$

$$\boldsymbol{y}^{k+1} = \begin{cases} \frac{\boldsymbol{v}^k - \lambda}{c} + \boldsymbol{x}^{k+1}, & \boldsymbol{v}^k + c\boldsymbol{x}^{k+1} > \lambda \\ 0, & \boldsymbol{v}^k + c\boldsymbol{x}^{k+1} \in [-\lambda, \lambda] \\ \frac{\boldsymbol{v}^k + \lambda}{c} + \boldsymbol{x}^{k+1}, & \boldsymbol{v}^k + c\boldsymbol{x}^{k+1} < -\lambda \end{cases}$$

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k + c(\boldsymbol{x}^{k+1} - \boldsymbol{y}^{k+1})$$

代码实现如下：

```python
def ADMM(x_0, A, b, iters, regu_lambda, step_size):
    """
    Alternating Direction Method of Multipliers
    :param x_0: variable to be updated
    :param A: measurement matrix
    :param b: measurement result with noise
    :param iters: iterations num
    :param regu_lambda: regularization coefficient
    :param step_size: step size c
    :return: updated x, x_list
    """
    iter_list = []
    x = np.copy(x_0)
    y = np.copy(x)
    v = np.zeros(200)

    part1 = step_size * np.eye(200)
    for j in range(10):
        part1 += np.dot(A[j].T, A[j])
    part1 = np.linalg.inv(part1)

    for i in range(iters):
        # update x
        part2 = step_size * y - v
        for j in range(10):
            part2 += np.dot(A[j].T, b[j])

        x = np.dot(part1, part2)

        iter_list.append(np.copy(x))

        # update y
        condlist = [v + step_size * x > regu_lambda, v + step_size * x < - regu_lambda]
        choicelist = [(v - regu_lambda) / step_size + x, (v + regu_lambda) / step_size + x]
        y = np.select(condlist, choicelist, 0)

        # update v
        v += step_size * (x - y)

    return x, iter_list
```

3.次梯度法

考虑目标函数的次梯度：
$\boldsymbol{x}>0$ 时，$g(\boldsymbol{x})=A_1^\top(A_1\boldsymbol{x}-\boldsymbol{b}_1)+\cdots+A_{10}^\top(A_{10}\boldsymbol{x}-\boldsymbol{b}_{10})+\lambda$
$\boldsymbol{x}=0$ 时，$g(\boldsymbol{x})\in[-\lambda,\lambda]-A_1^\top\boldsymbol{b}_1-\cdots-A_{10}^\top\boldsymbol{b}_{10}$
$\boldsymbol{x}<0$ 时，$g(\boldsymbol{x})=A_1^\top(A_1\boldsymbol{x}-\boldsymbol{b}_1)+\cdots+A_{10}^\top(A_{10}\boldsymbol{x}-\boldsymbol{b}_{10})-\lambda$
每轮迭代中，有：

$$\boldsymbol{x}^{k+1}=\boldsymbol{x}^k-\alpha^k g(\boldsymbol{x}^k)$$

代码实现如下：

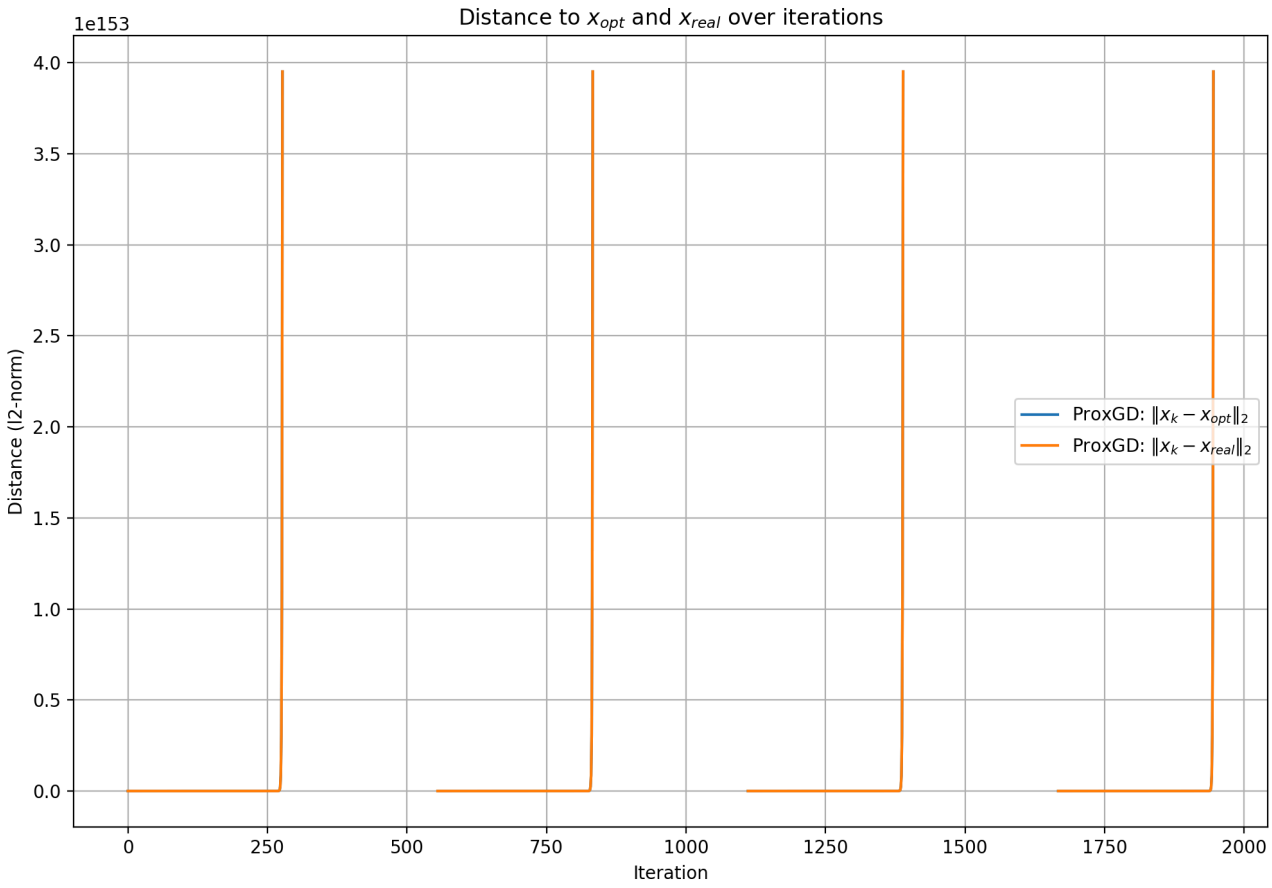
```
def subgradient(x_0, A, b, iters, regu_lambda, step_size):
    """
    Subgradient Method
    :param x_0: variable to be updated
    :param A: measurement matrix
    :param b: measurement result with noise
    :param iters: iterations num
    :param regu_lambda: regularization coefficient
    :param step_size: step size c
    :return: updated x, x_list
    """
    iter_list = []

    x = np.copy(x_0)

    for i in range(iters):
        grad_sum = 0
        for j in range(10):
            grad_sum += np.dot(A[j].T, np.dot(A[j], x) - b[j])
        condlist = [x > 0, x == 0, x < 0]
        choicelist = [grad_sum + regu_lambda, grad_sum + np.random.uniform(- regu_lambda, regu_
        g_x = np.select(condlist, choicelist)

        if i > 0:
            step_size_k = step_size / (i ** 0.5)
        else:
            step_size_k = step_size

        x -= step_size_k * g_x
        iter_list.append(np.copy(x))

    return x, iter_list
```

数值实验、结果讨论

先暂时固定正则化系数为1，比较3种优化算法。x_{opt} 由交替方向乘子法得出。

1. 临近点梯度法

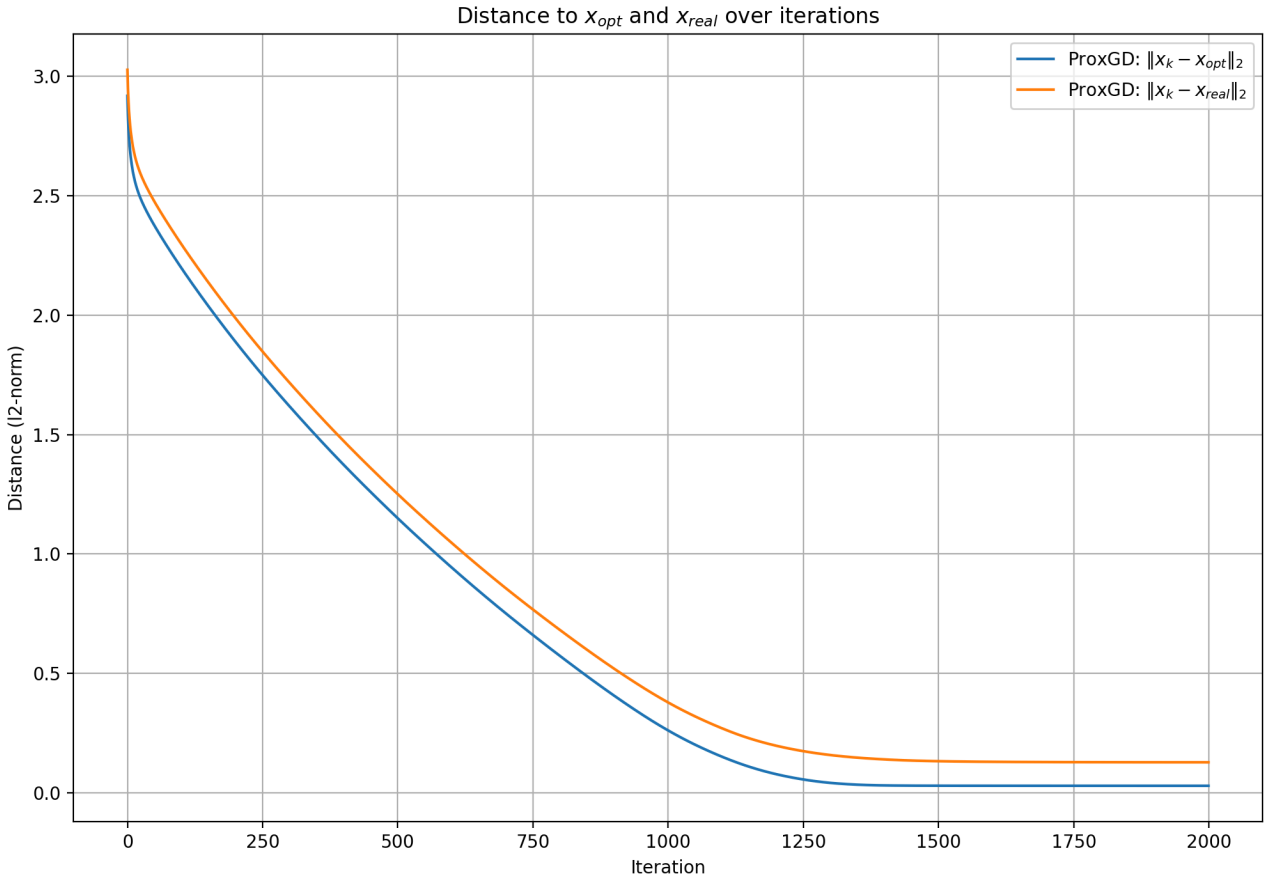
- 此时的 `step_size` 不能选太大，否则会让 x 离真值的距离越来越远，也就是步长过大导致发散了，没有收敛。 具体实验如下：
- 选择 `step_size=1e-2` ：



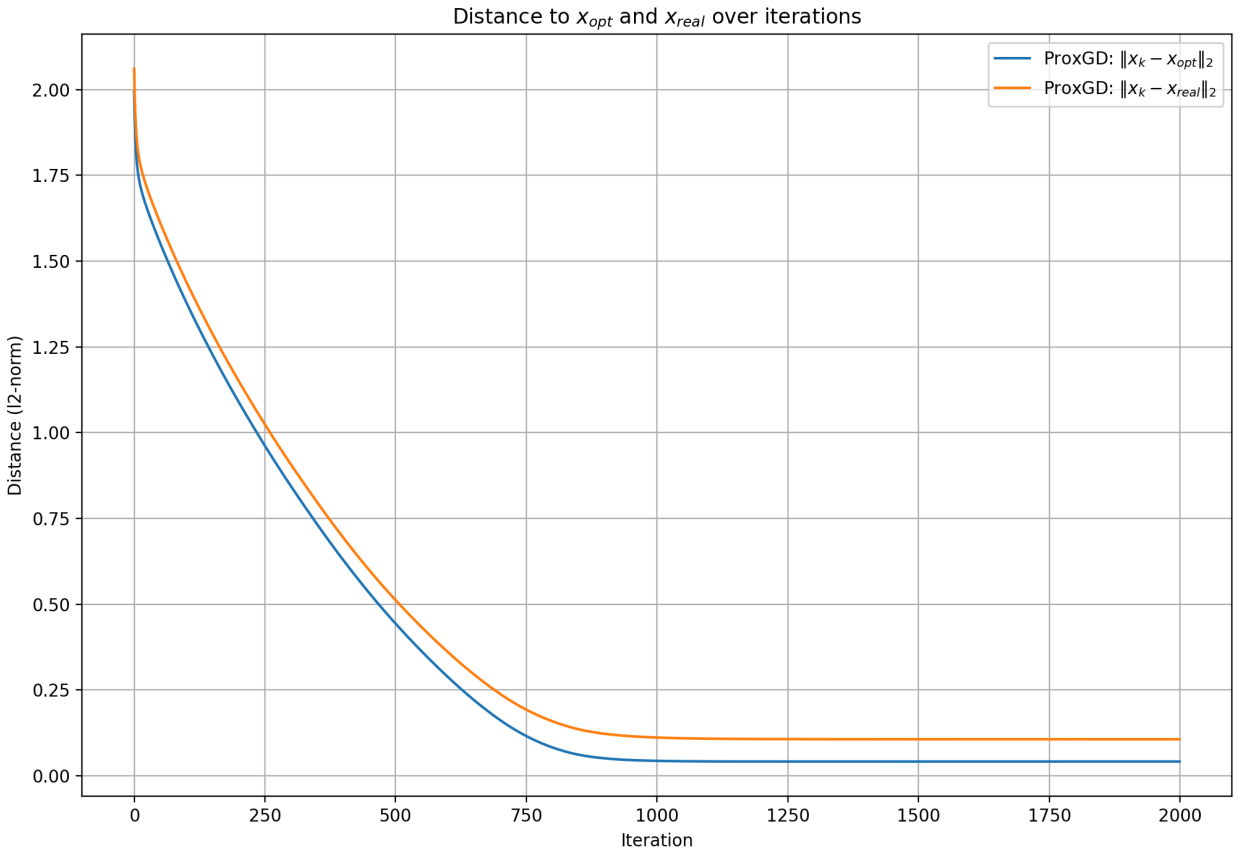
x_k 随着迭代，到最优解的距离变成了极大，算法d而迭代结果发散了，没有收敛。

- 选择 `step_size=1e-3` 的几次实验结果：

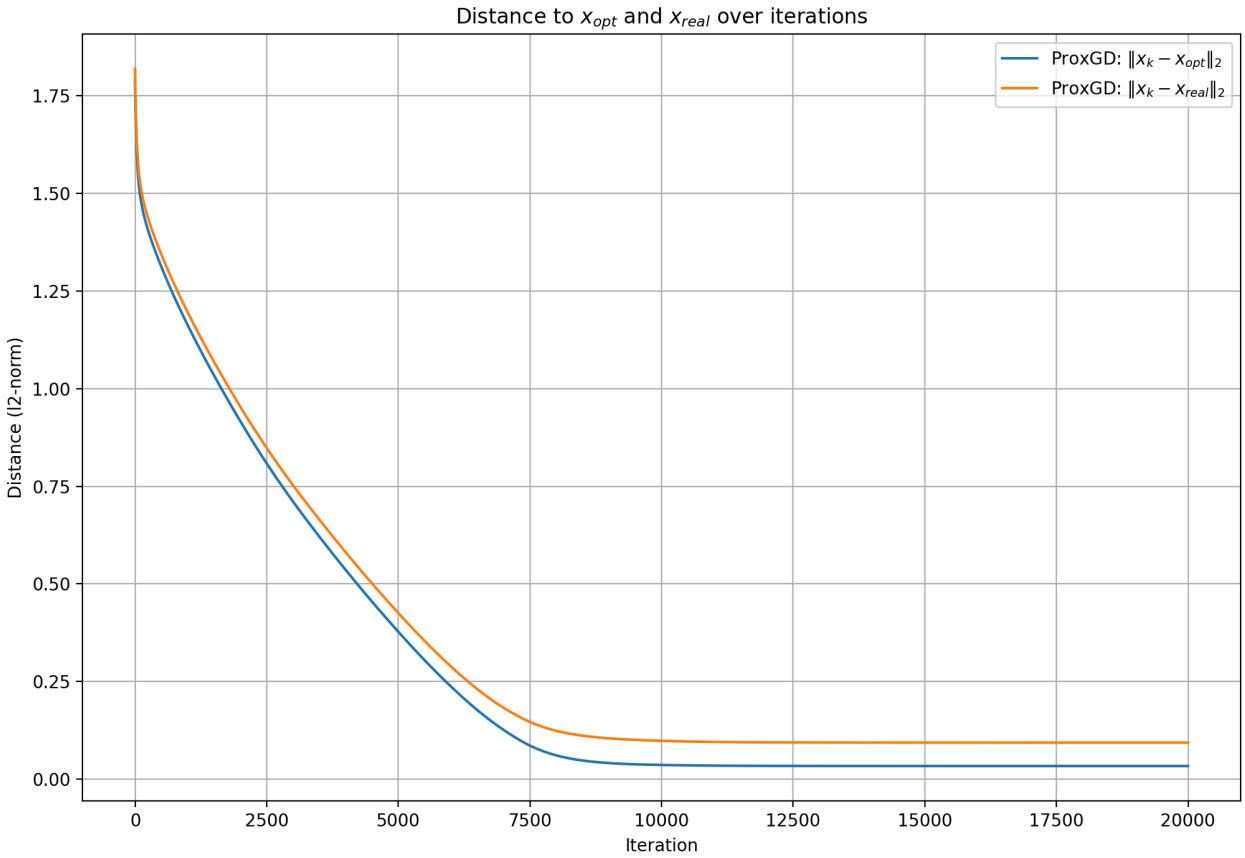
  1）大概1200轮迭代后收敛：



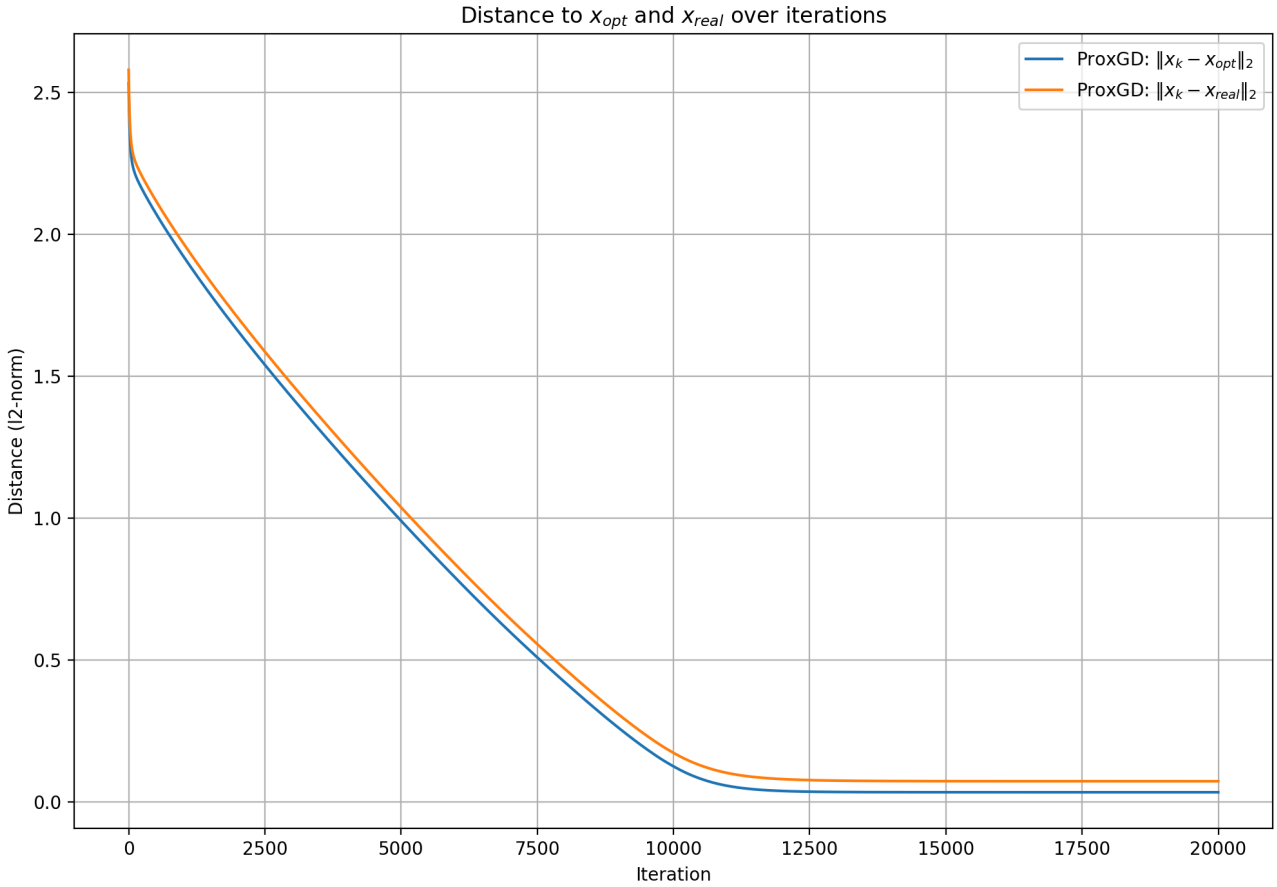
2）大概900轮迭代后收敛：



- 选择 `step_size=1e-4` 的几次实验结果：
  可以看到此时算法最终也能收敛，但由于步长选择过小，收敛较慢，需要更多的迭代轮数：
  1)



  2)

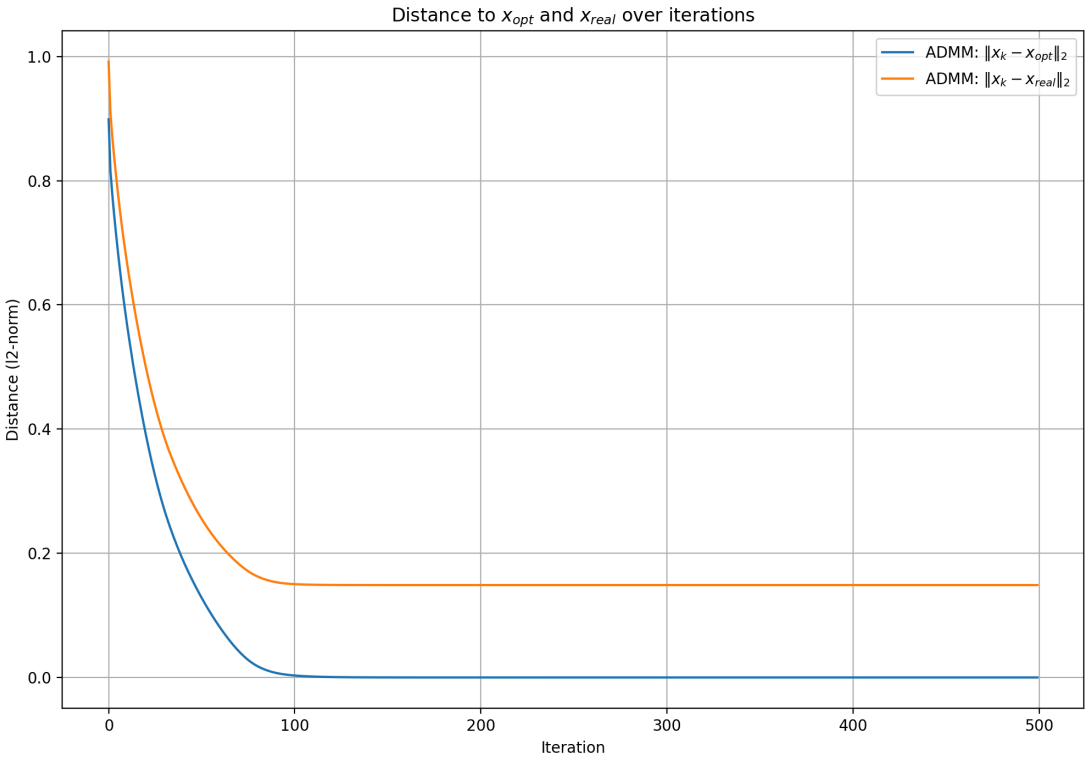


- 于是，在我们的实验中，对临近点梯度法，选择 `step_size=1e-3` 比较合适。

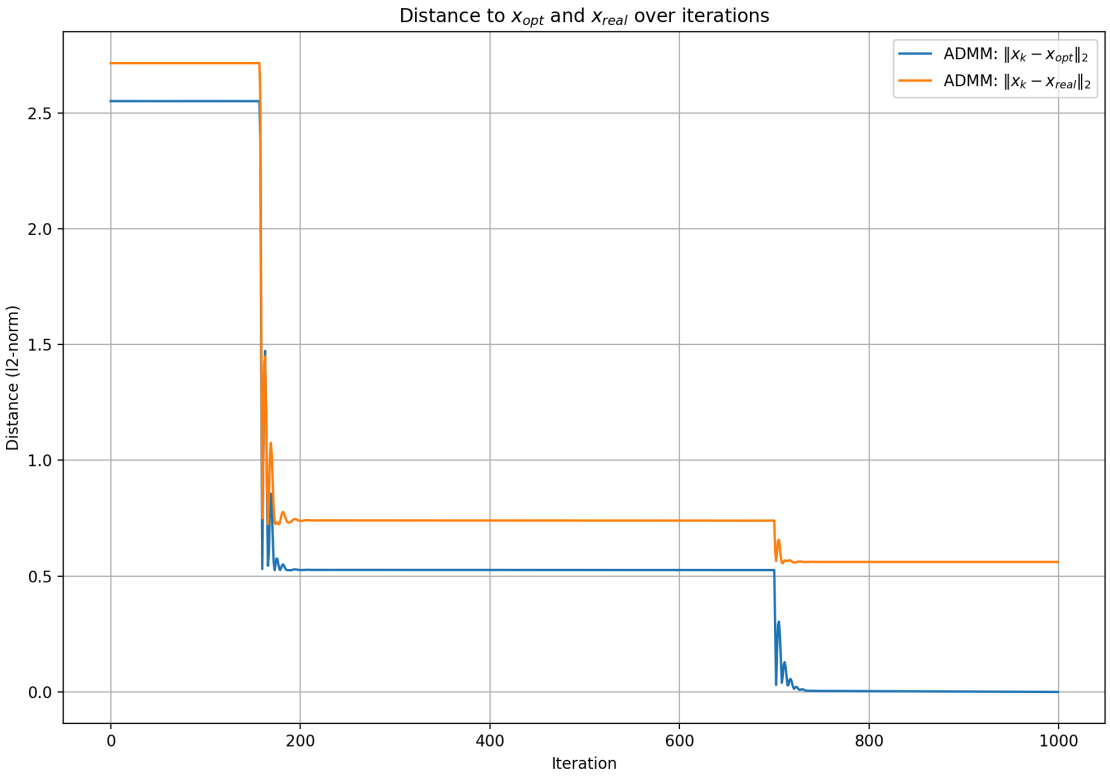
2. 交替方向乘子法（ADMM）

发现交替方向乘子法是比较稳定的，适合更大的步长。

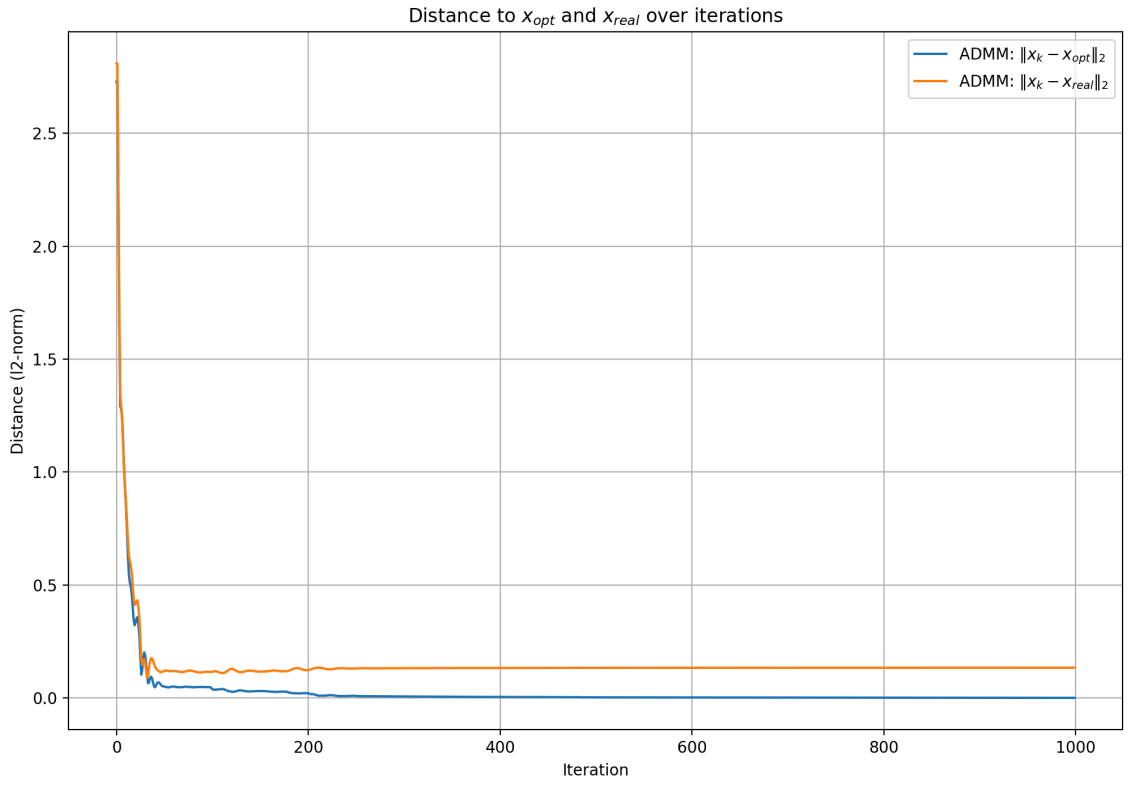
- 即使步长选很大（如下， `step_size=100` ，最后也收敛了）



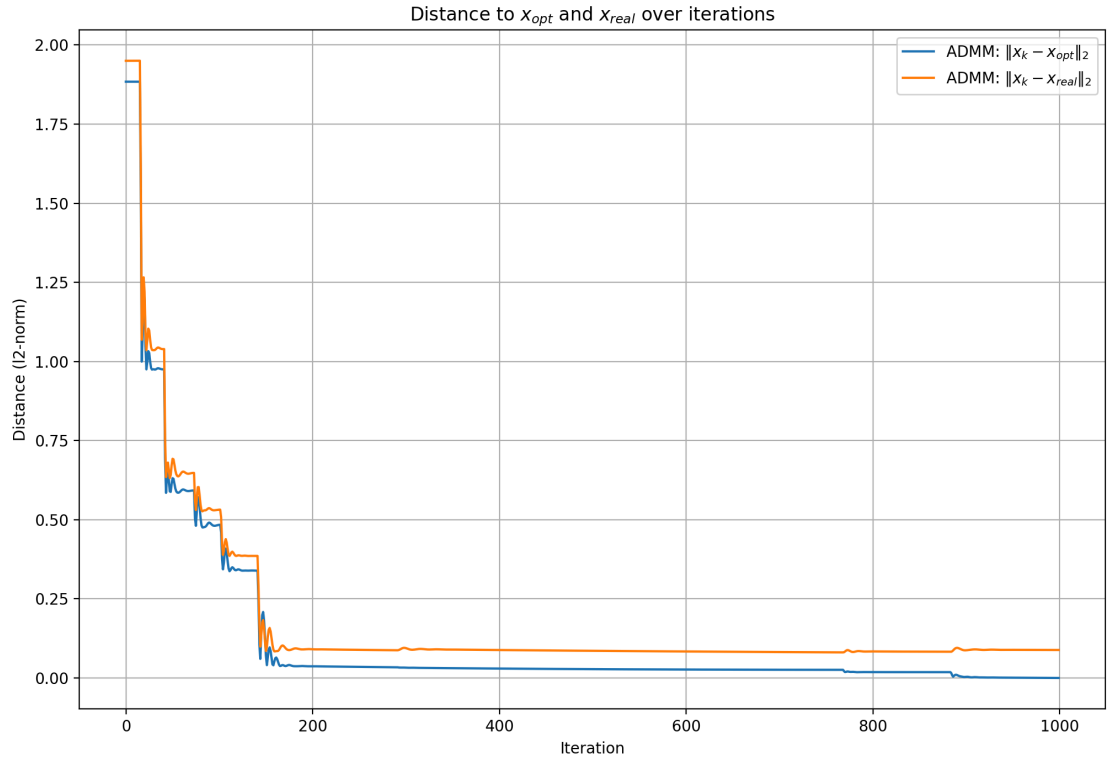
- 选择 `step_size=1e-2` 这样的步长又会使得收敛太慢：



- `step_size=1`：



- `step_size=1e-1`：



- 于是，在我们的实验中，对交替方向乘子法，选择 `step_size=1e-1` 或者 `step_size=1` 比较合适。

3. 次梯度法

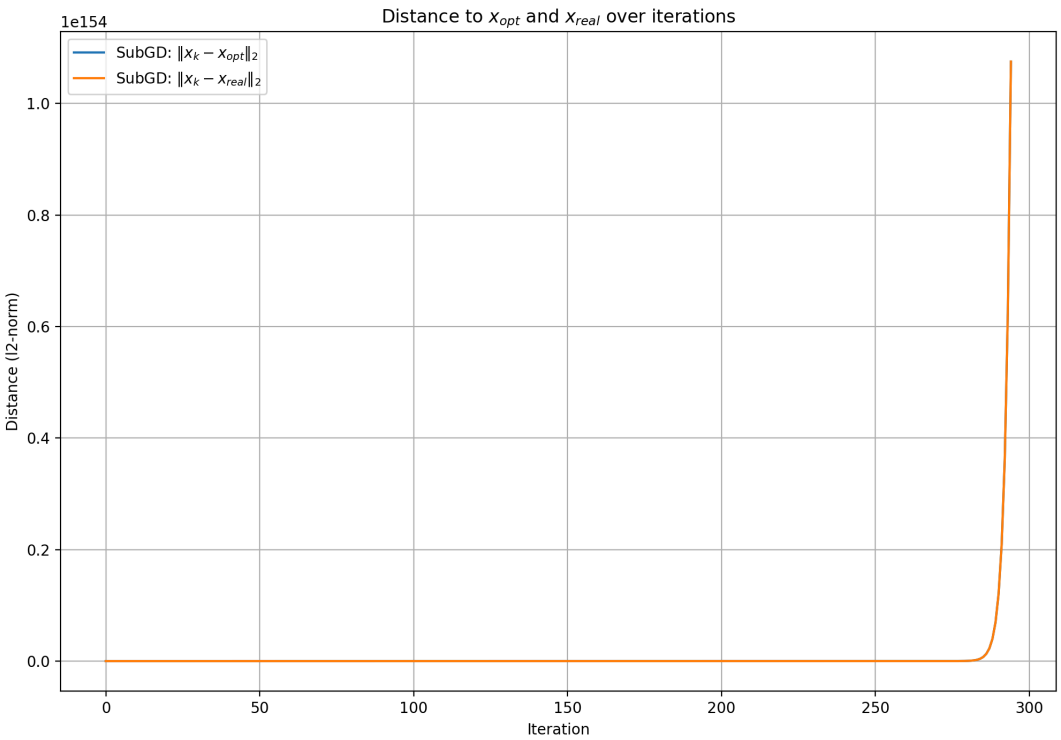
次梯度法要使用递减步长。实验中按照：

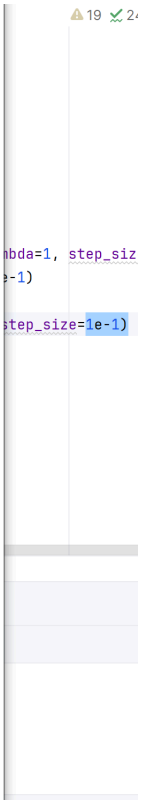
$$\alpha_k = \frac{c}{\sqrt{k}}$$

其中 c 是初始步长，k 是迭代轮数。

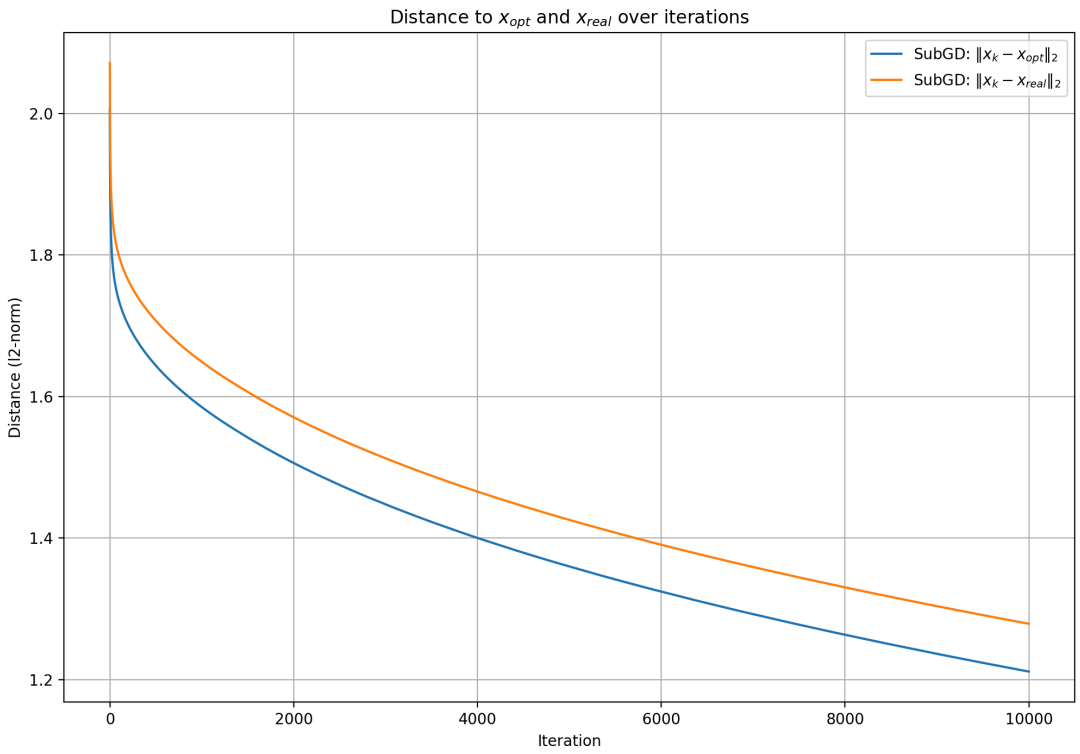
```
if i > 0:
    step_size_k = step_size / (i ** 0.5)
else:
    step_size_k = step_size
```

- 次梯度法的初始步长也不能选太大， `step_size=1e-1` 时，发散了：

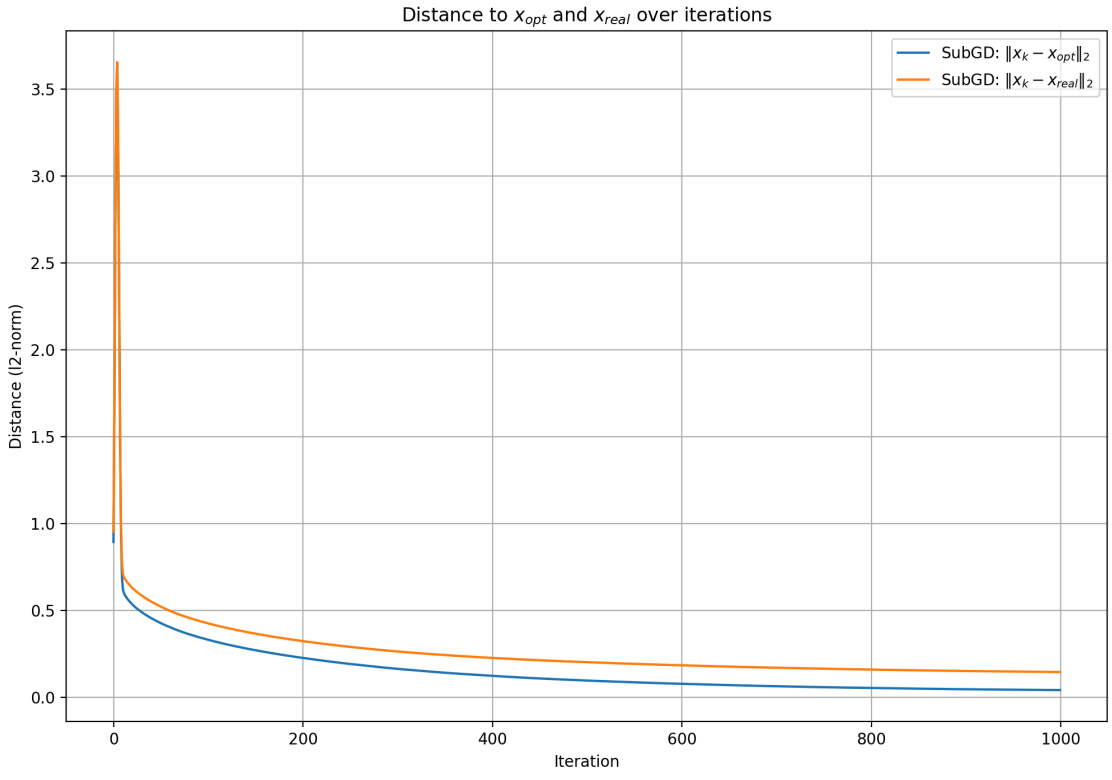




- `step_size=1e-3` ，收敛较慢：

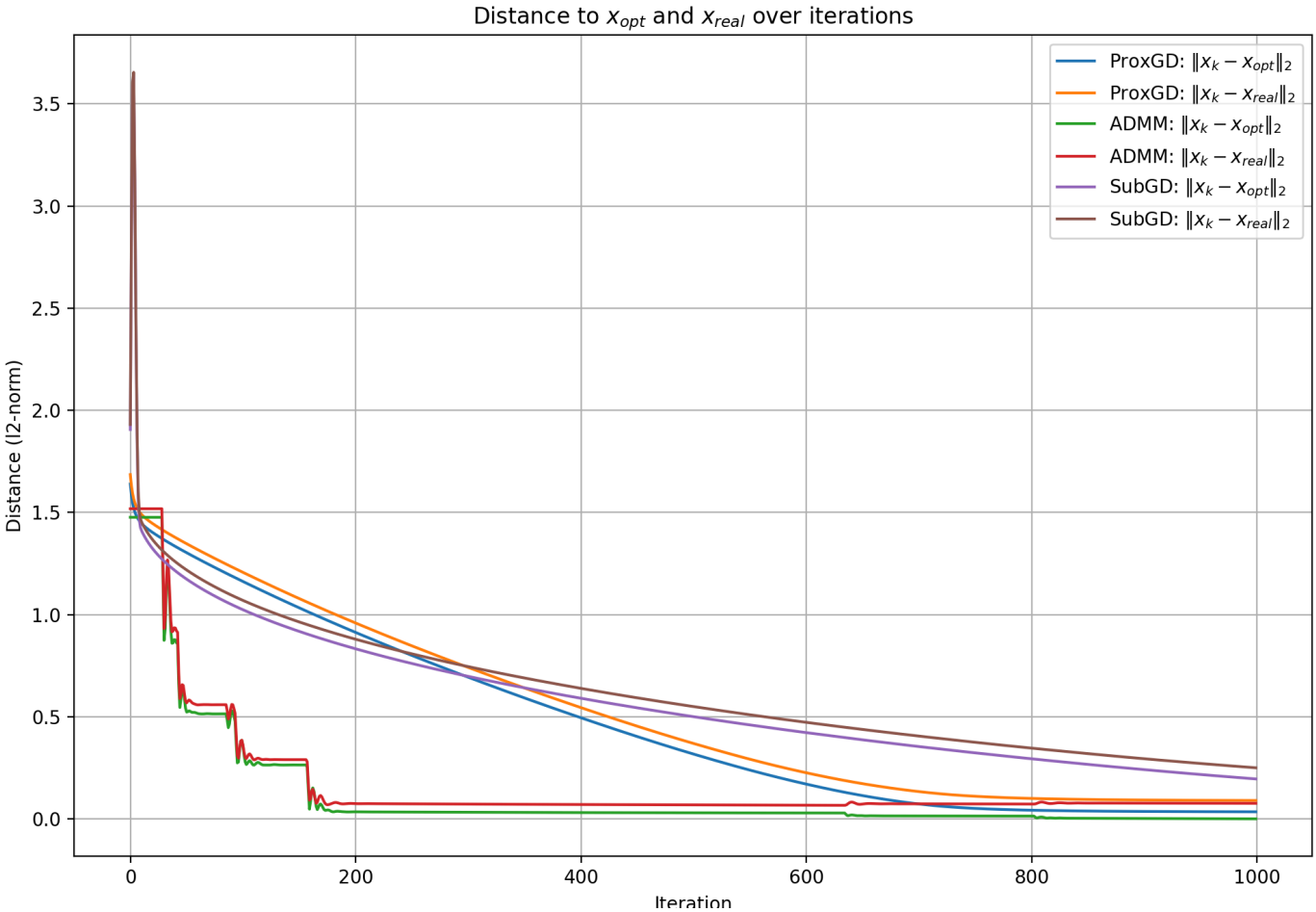


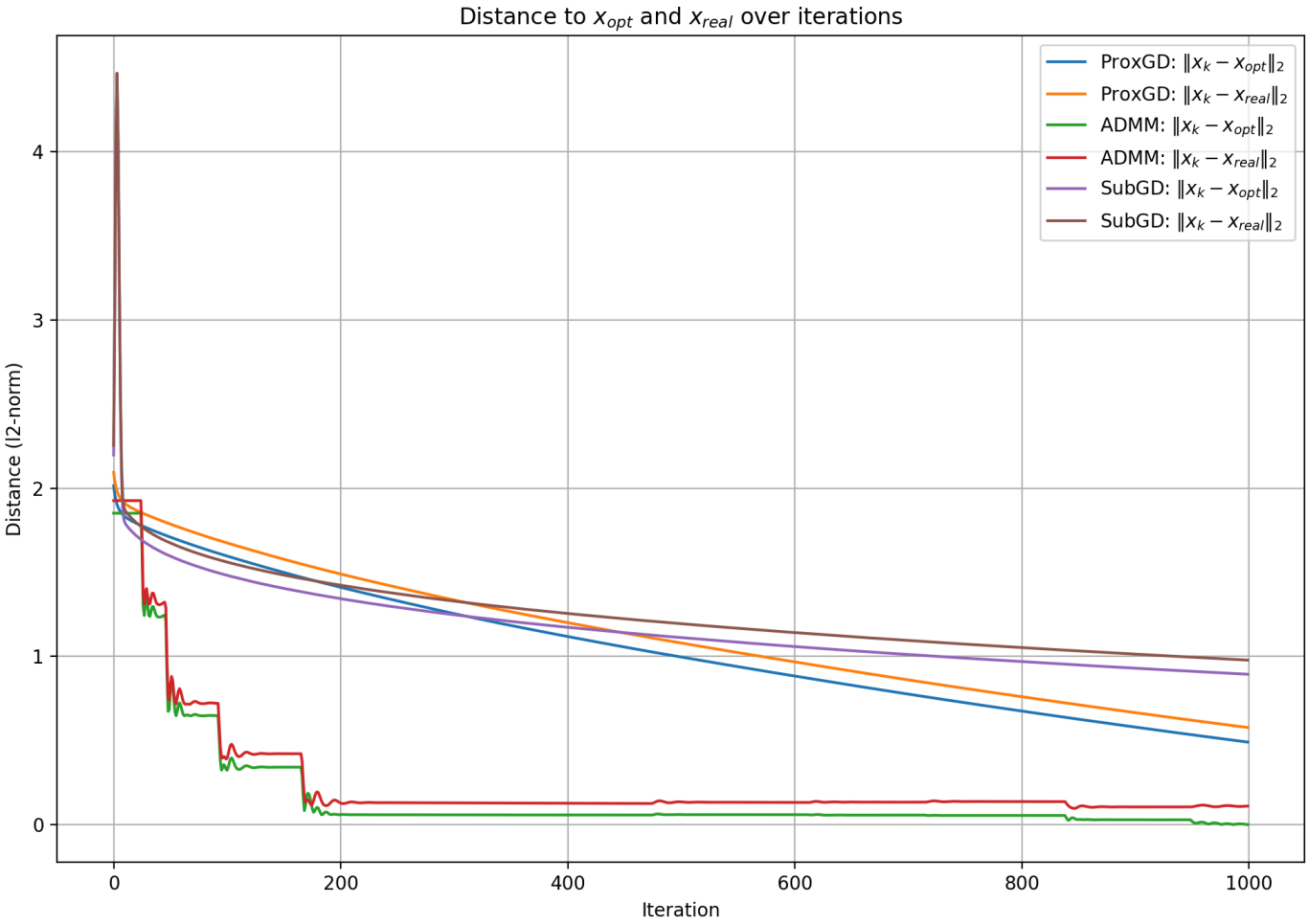
- 对次梯度法， `step_size=1e-2` 是较合适的选择，但也会出现不稳定的现象（如下曲线中，一开始算法迭代解 x_k 离最优解 x_{opt} 的距离会突然增大）

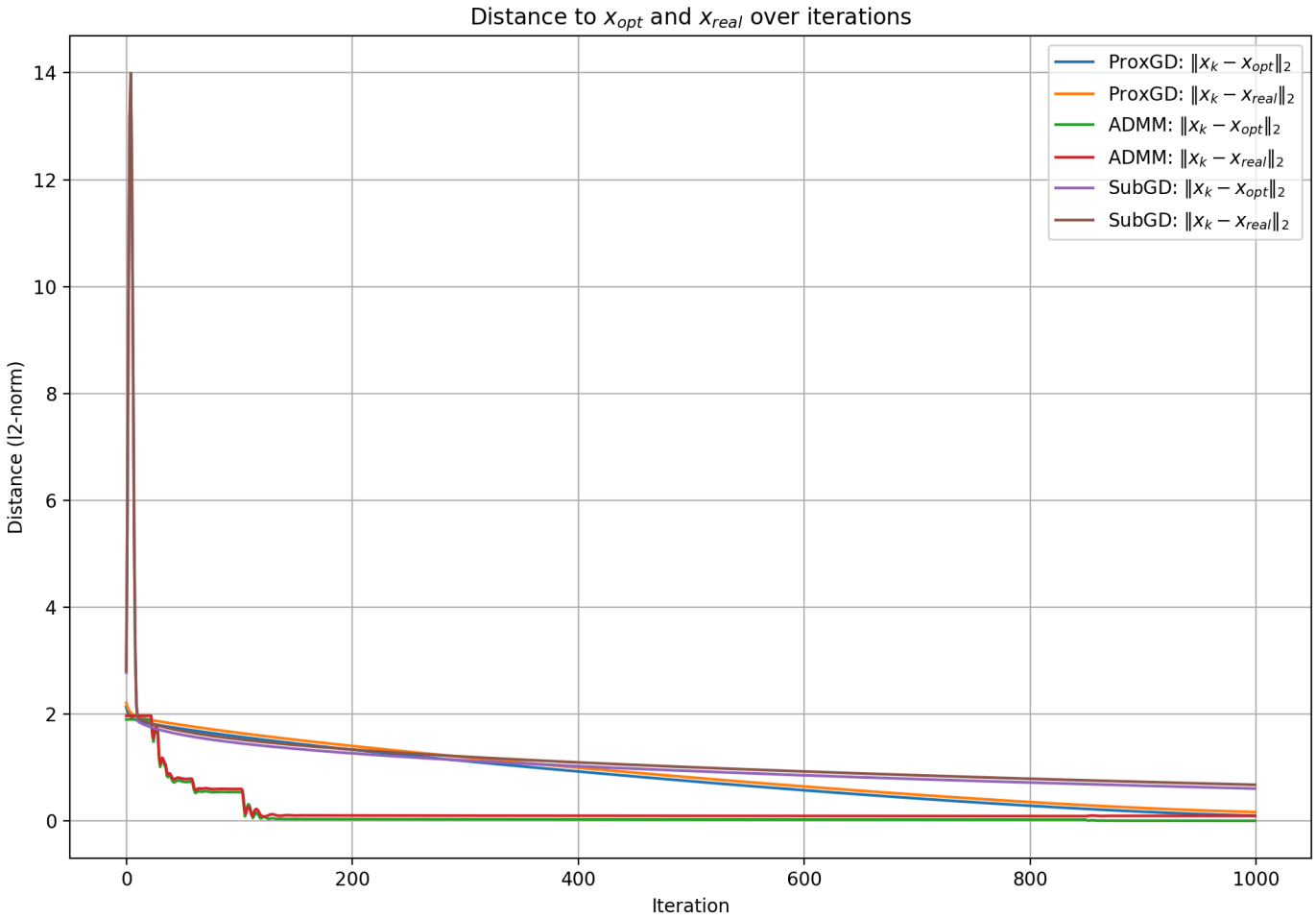


3种优化算法的比较

每个算法分别选择自己较适合的步长。正则化系数仍1。


Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
200
400
600
800
1000

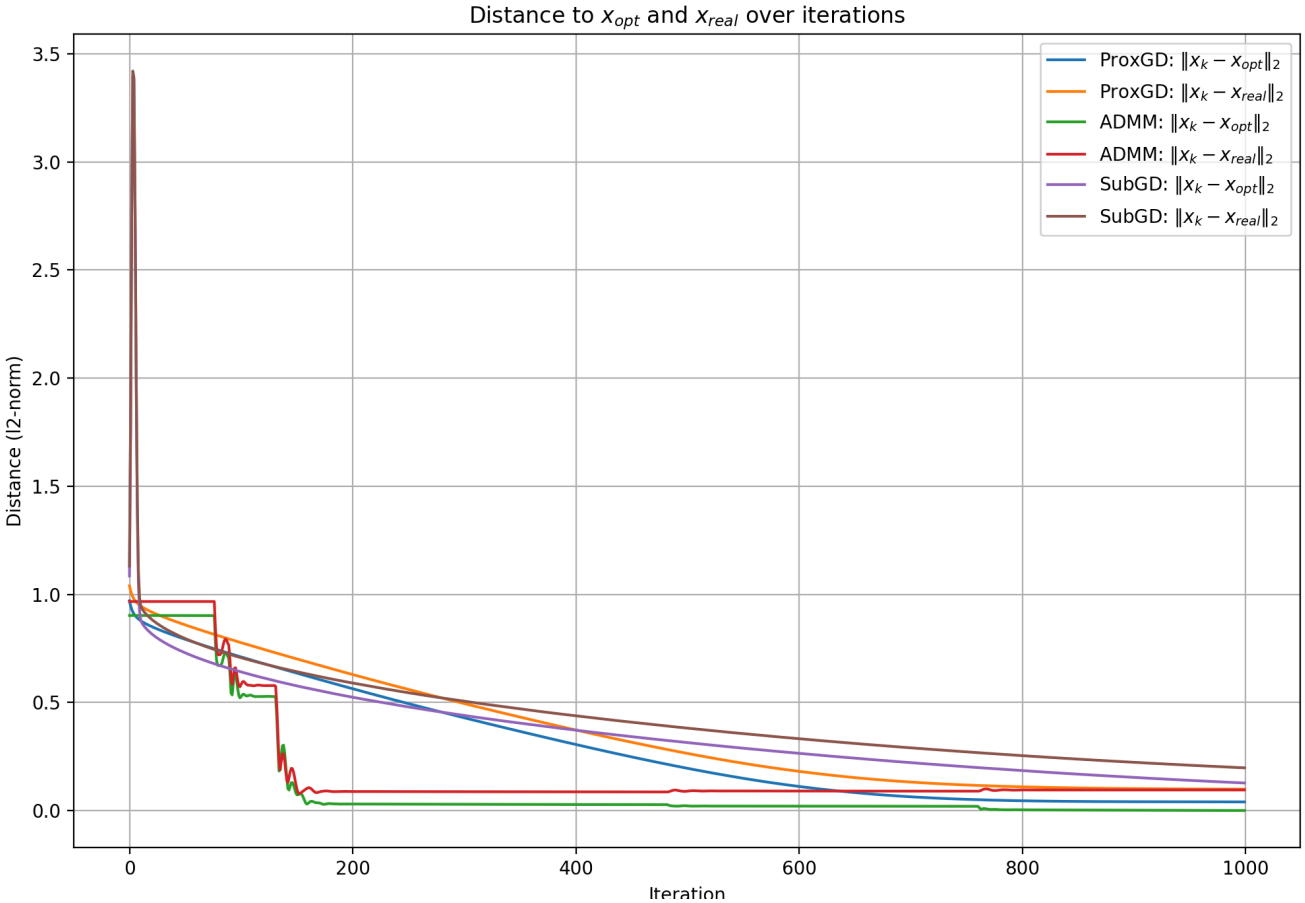

Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0
1
2
3
4
0
200
400
600
800
1000



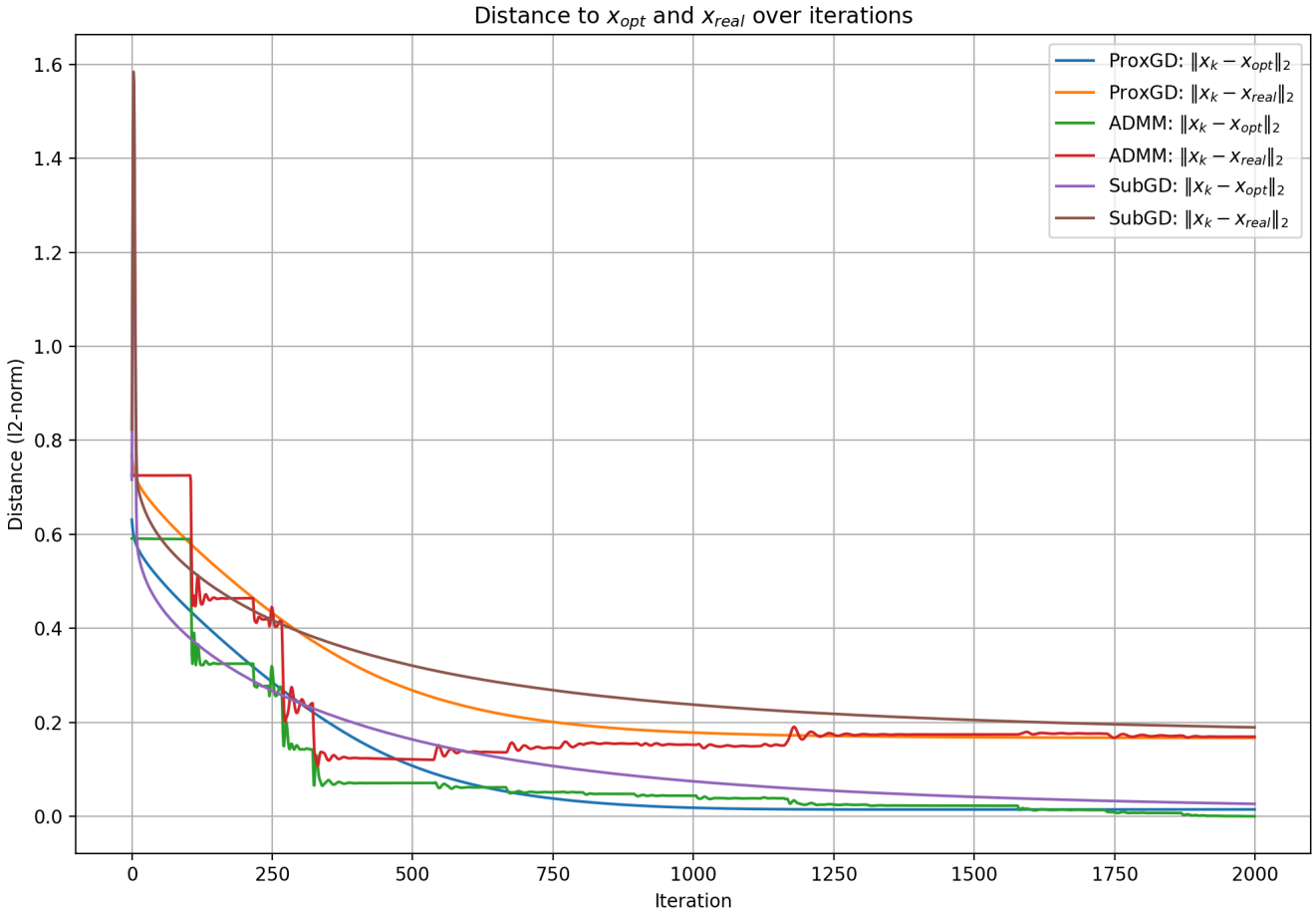
- 可以观察到，交替方向乘子法是性能最好的，并且收敛速度远远快于另2种方法，可以用更少的迭代轮数，收敛到离真值 x_{real} 最近的解。
- 临近点梯度法的性能次之，收敛速度比交替方向乘子法慢，但最后也能得到一个离真值 x_{real} 很近的解。
- 次梯度法收敛最慢，并且收敛到的解离真值的距离是2种方法中最远的。而且次梯度法不稳定。

正则化系数 λ 对收敛的影响

- 选择 $\lambda = 1$:
  1000轮迭代:

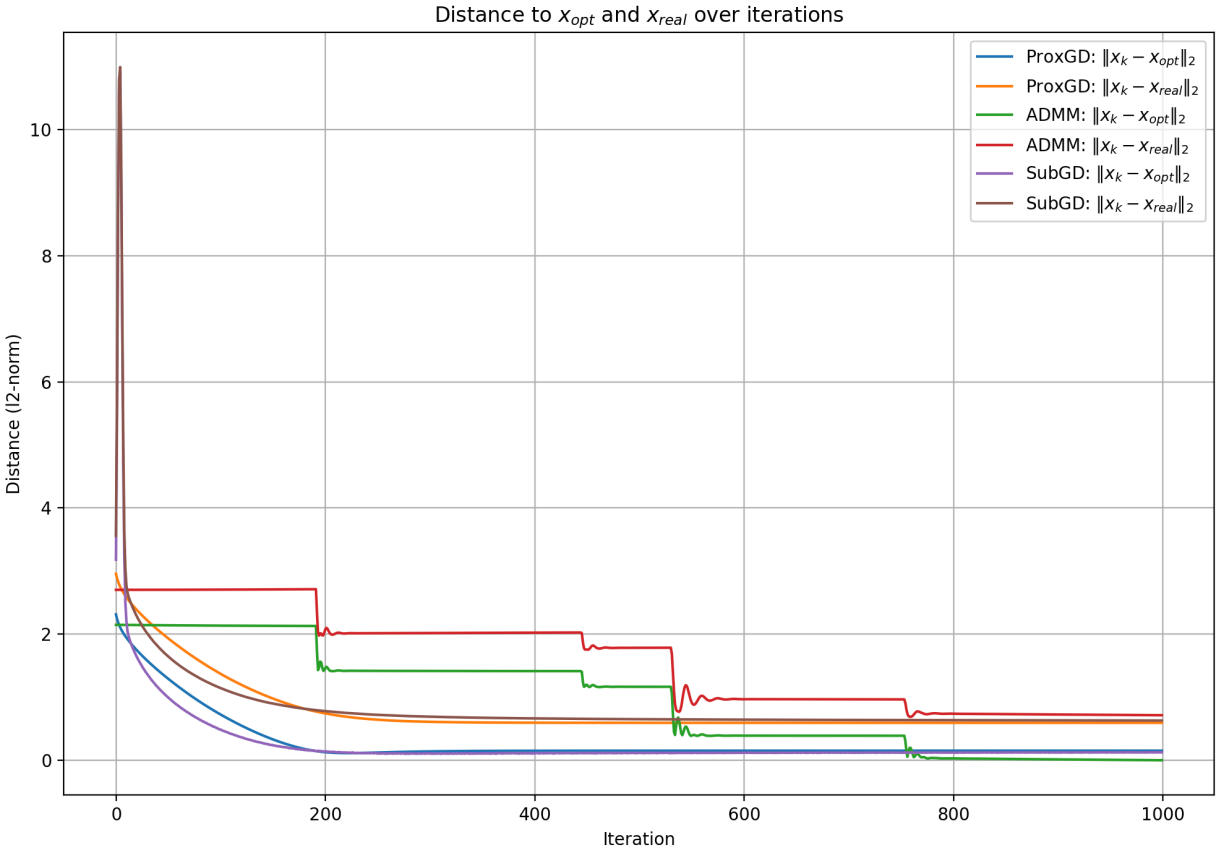

Distance to x_{opt} and x_{real} over iterations
Distance (l2-norm)
Iteration
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
200
400
600
800
1000

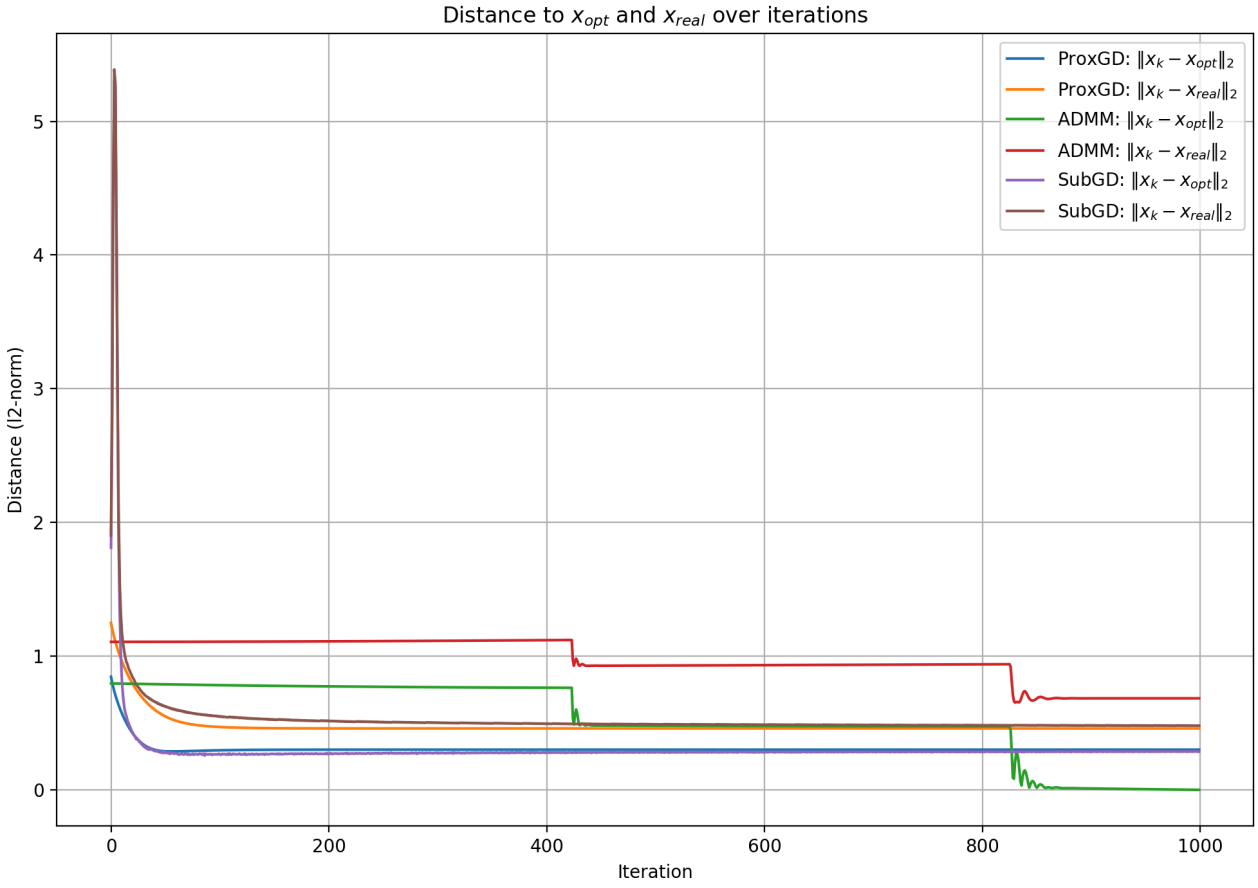
2000轮迭代：



$\|x_k - x_{real}\|_2$ 的结果大概能在 0.25 以下。

- 选择 $\lambda = 10$:

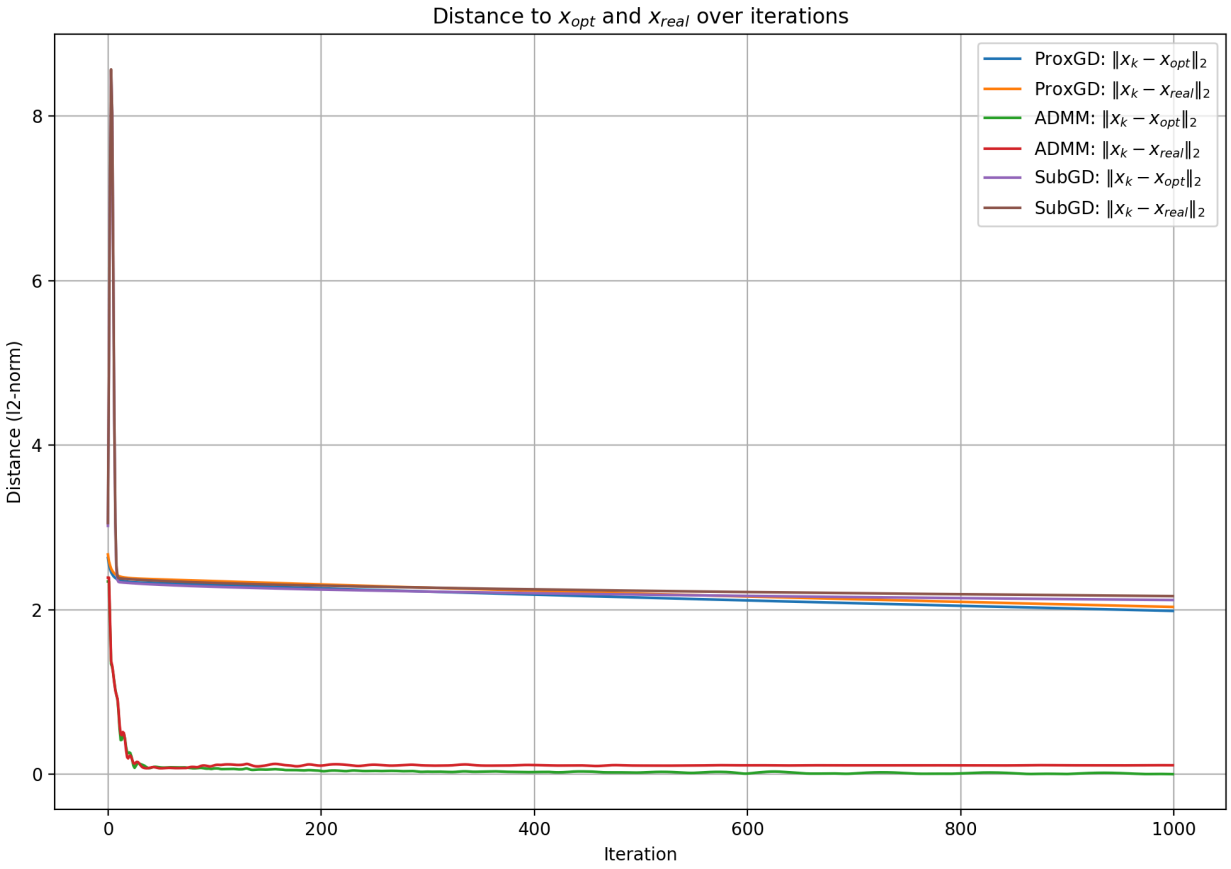

Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0
2
4
6
8
10
0
200
400
600
800
1000

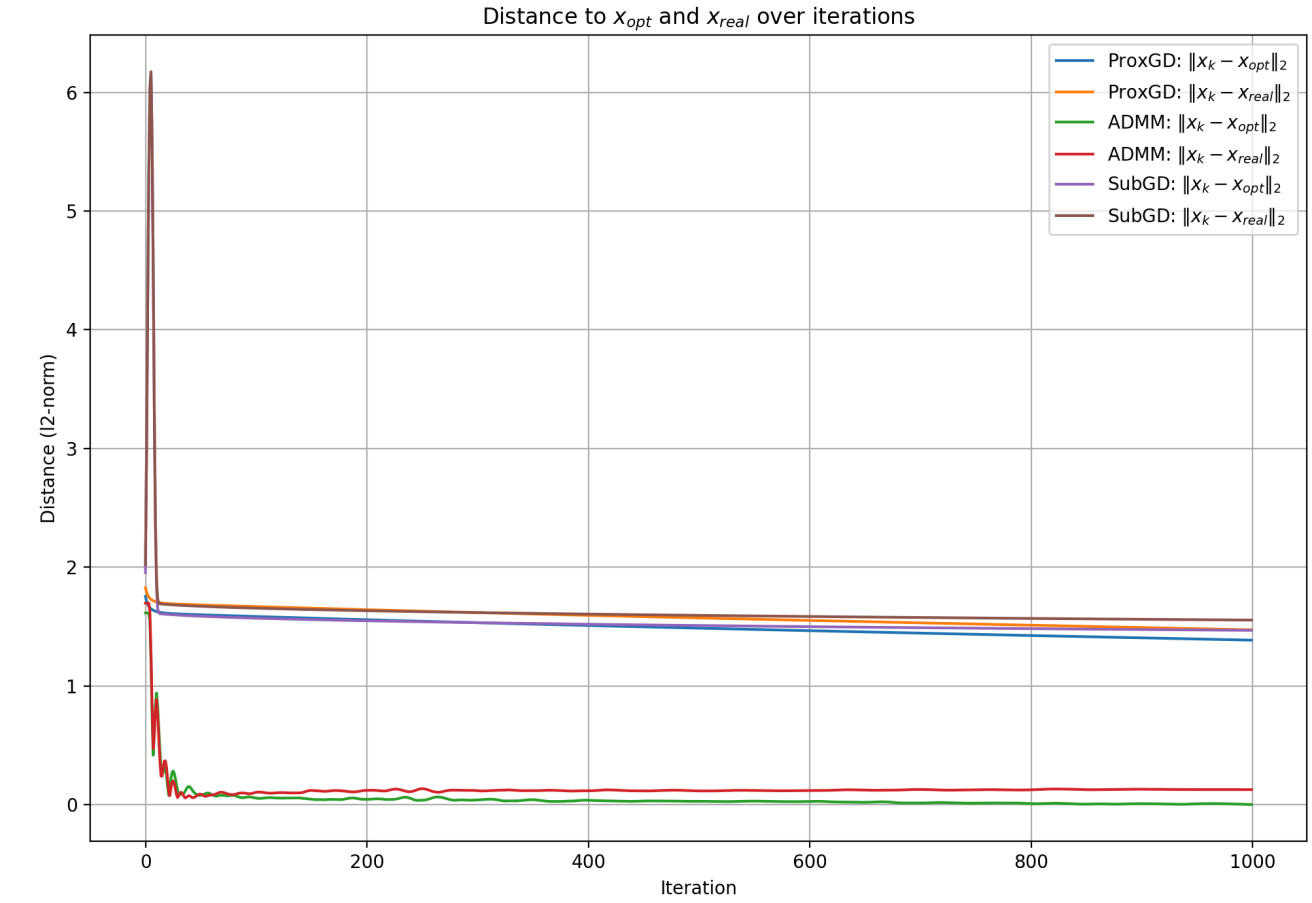

Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0
1
2
3
4
5
0
200
400
600
800
1000

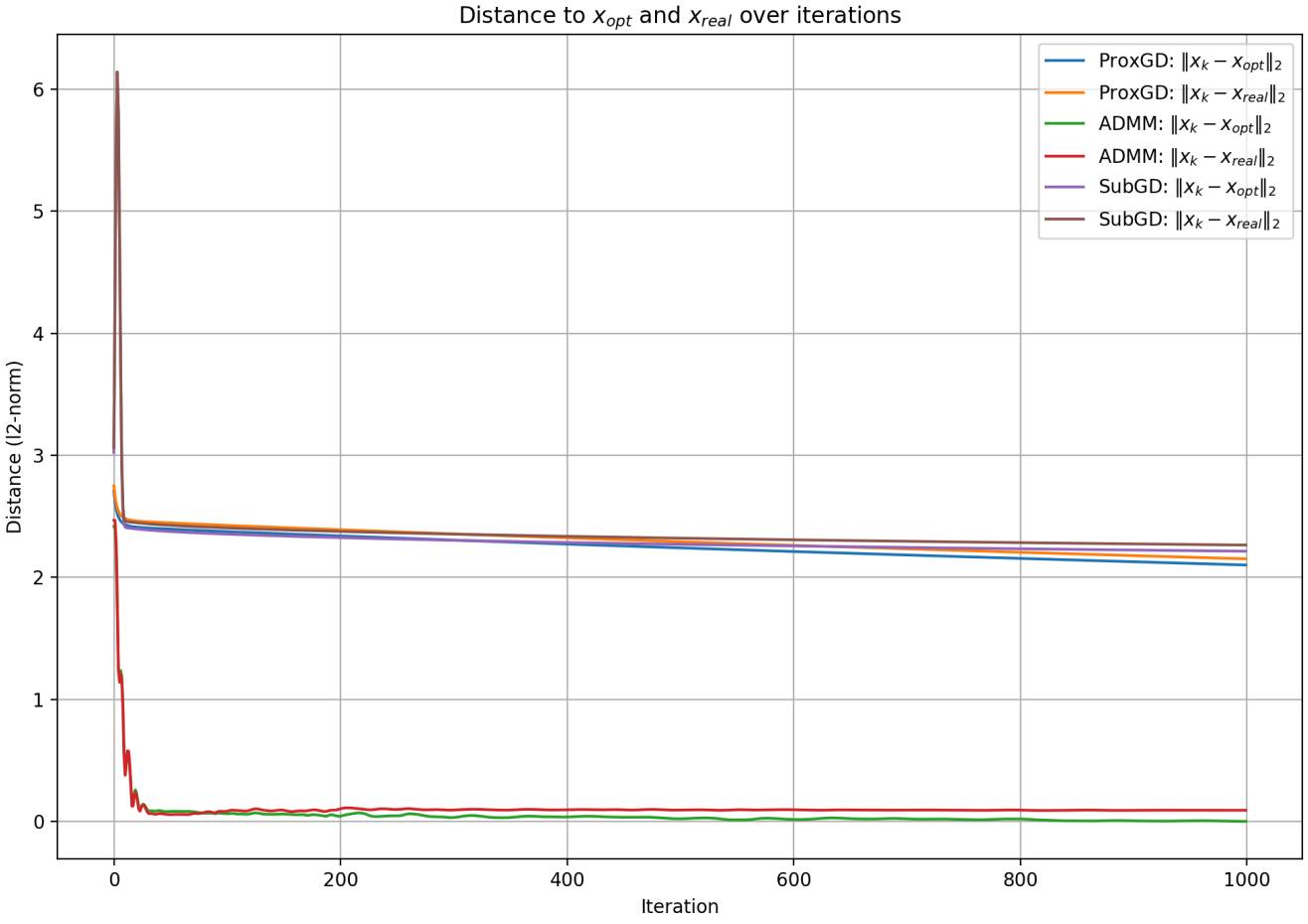
此时会大大加快收敛速度。

但最后收敛到的解 x 到 x_{real} 的距离，尤其是对于交替方向乘子法和临近点梯度法，是变大了的。$\lambda = 1$ 时，这个二范数结果最后能很接近 0，但上图中 $\lambda = 10$ 时，$\|x_k - x_{real}\|_2$ 最后离 0 有一定的一段稳定距离，大概在 $(0.5, 1)$ 之间。

- 选择 $\lambda = 0.1$:




Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0
1
2
3
4
5
6
0
200
400
600
800
1000


Distance to x_{opt} and x_{real} over iterations
ProxGD: $\|x_k - x_{opt}\|_2$
ProxGD: $\|x_k - x_{real}\|_2$
ADMM: $\|x_k - x_{opt}\|_2$
ADMM: $\|x_k - x_{real}\|_2$
SubGD: $\|x_k - x_{opt}\|_2$
SubGD: $\|x_k - x_{real}\|_2$
Distance (l2-norm)
Iteration
0
1
2
3
4
5
6
0
200
400
600
800
1000

- 发现对交替方向乘子法ADMM，效果和 $\lambda = 1$ 相似，最后得到的解能和真值很近，即 $\|x_k - x_{real}\|_2$ 接近 0；
- 但对临近点梯度法和次梯度法发现，此时虽然更快收敛了，但对真值 x_{real} 的拟合效果明显变差了，$\|x_k - x_{real}\|_2$ 的最终结果较不稳定，跑几次实验发现大概在 2 左右。
- 说明对于这个针对稀疏向量 x 的恢复问题，在临近点梯度法和次梯度法中，正则化系数较小（$\lambda = 0.1$），会导致算法因关注不到向量的稀疏特征，而使得对真值的恢复效果变差，最后得到一个并不稀疏、也距离真值更远的解。
- ADMM虽然最后的解距离真值较近（相差的值的二范数结果较小），但其实也可能并没有很好地拟合稀疏向量，见下面的讨论。

正则化系数 λ 对收敛到的解的稀疏程度的影响

- 为了探究收敛到的解的稀疏程度，又发现算法最后收敛到的用来拟合真值 x_{real} 的解 x_k 会形如下面这样：

```
 -1.45160920e-03 -4.32659068e-03  3.22841322e-03  3.12078711e-03
 -7.82078861e-04 -2.40383667e-03  1.66909345e-03  1.61349669e-01
 -2.19494828e-03  1.94100067e-04 -2.19474972e-04  1.98448663e-03
 -1.71973708e-04  8.67982895e-05 -1.16759446e-03  1.84803970e-03
 -1.84544965e-04  4.33037804e-04  5.80154664e-04  3.48959242e-03
  3.77275971e-04  1.58576239e-03  2.96940325e-04 -1.48978447e-03
  3.59124824e-04 -2.45073015e-03  2.49739470e-03  1.63985155e-03
 -3.31662732e-03 -2.97306821e-03 -1.88191288e-03 -2.19224599e-04
  6.16598154e-05  1.96376395e-03 -1.09130601e-01 -1.15371174e-03
  3.56513430e-01 -1.90290924e-03 -2.28892383e-03 -3.33733268e-04
  2.41124948e-03 -1.88609298e-02 -9.23980225e-04  3.03393654e-03
 -3.42084622e-03  1.70136706e-03 -1.65207236e-03  8.74768500e-04
 -1.55954667e-03  1.26321476e-03 -2.33310834e-03 -6.81214001e-04
 -1.68615546e-03 -8.75490095e-04  1.35271650e-03 -4.28851641e-03
  5.53282328e-04  2.20924612e-03  2.73787414e-03  8.09693088e-04]
[ 2.65915613e-04 -2.32349855e-04 -5.50378187e-05  9.93177688e-05
  4.78149463e-04  2.23201596e-04 -3.92587950e-05  2.43484625e-04
```

- 于是进行近似的分析，在如下的 `approximate_sparsity()` 方法中，如果向量中的一个元素 `<threshold` 就，视作 `0`：

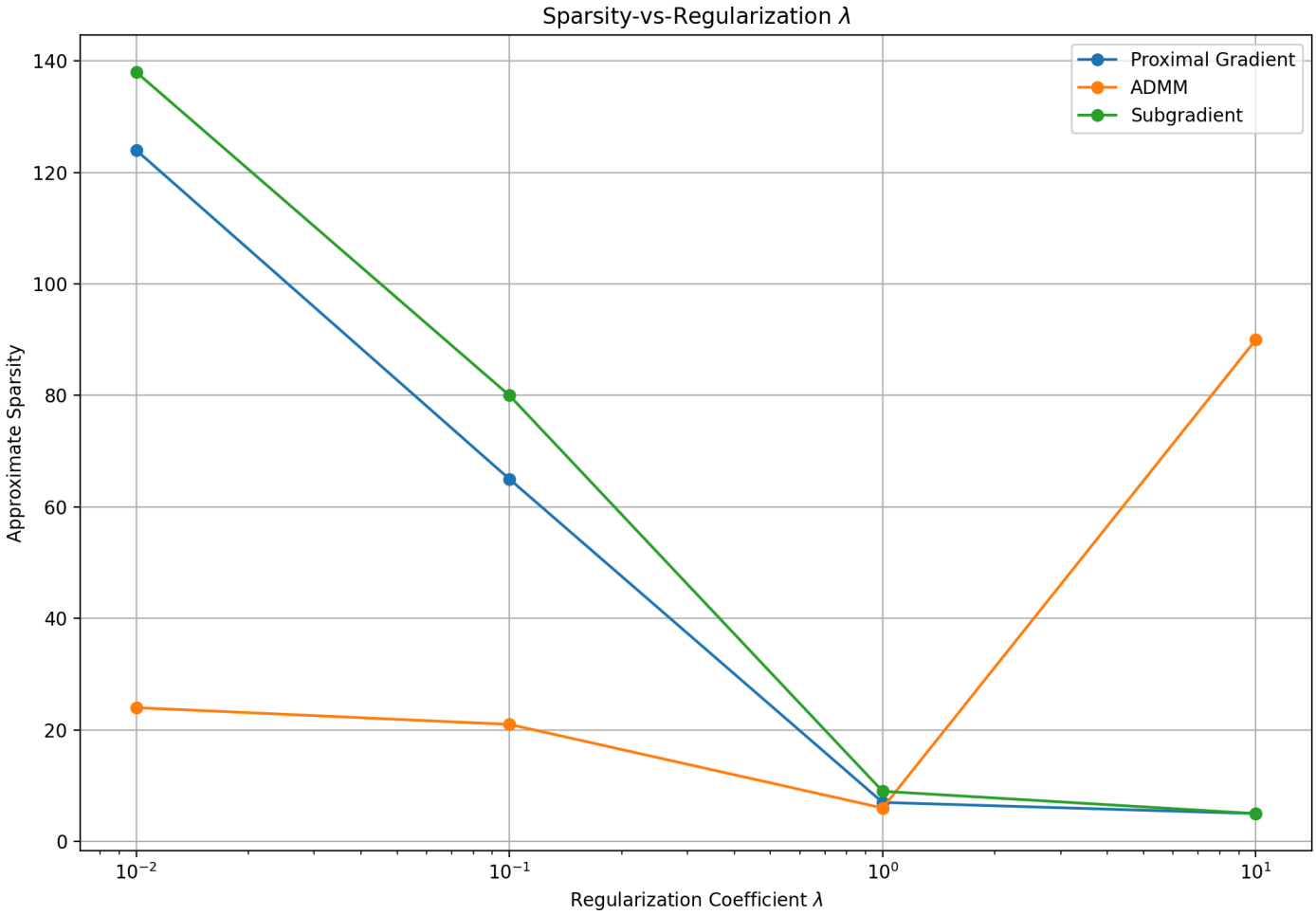
```
def approximate_sparsity(x, threshold):
    """
    compute the approximate sparsity of x
    :param x:
    :param threshold: if x_i < threshold, we think it is zero
    :return: sparsity
    """
    non_zero_count = np.sum(np.abs(x) > threshold)
    return non_zero_count
```

针对这些不同的正则化系数值，得到解的大概稀疏程度，画图：

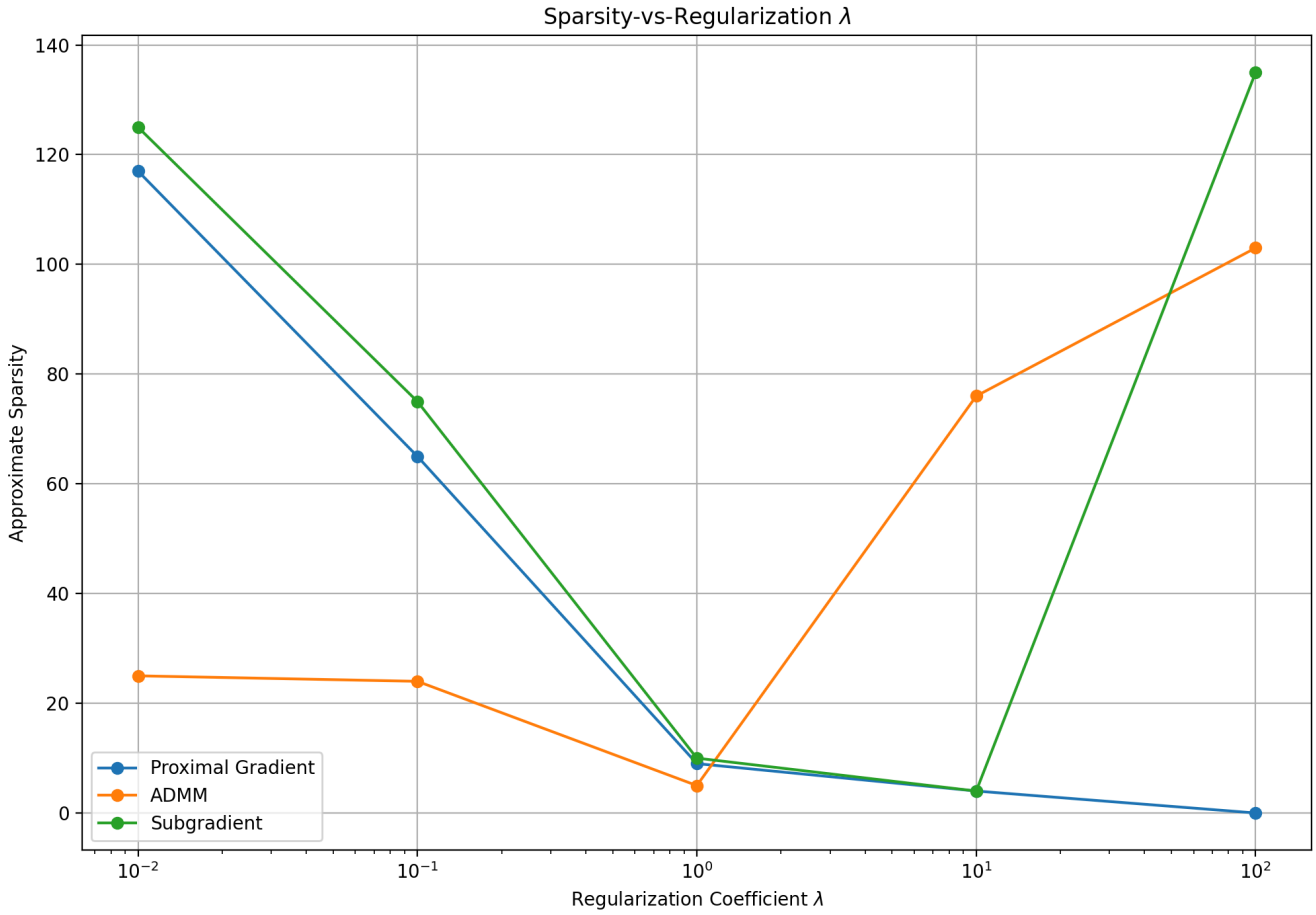
```
    regu_lambdas = [1e-2, 1e-1, 1, 10, 100]

    prox_gd_sparsities = []
    admm_sparsities = []
    sub_gd_sparsities = []

    for regu_lambda in regu_lambdas:
        # Proximal Gradient Method
        prox_gd_x_final, _ = proximal_gradient(x_0, measure_matrix, measure_result, iters=2000,
        prox_gd_sparsities.append(approximate_sparsity(prox_gd_x_final, threshold=1e-2))

        # ADMM
        admm_x_final, _ = ADMM(x_0, measure_matrix, measure_result, iters=2000, regu_lambda=regu
        admm_sparsities.append(approximate_sparsity(admm_x_final, threshold=1e-2))

        # Subgradient Method
        sub_gd_x_final, _ = subgradient(x_0, measure_matrix, measure_result, iters=2000, regu_la
        sub_gd_sparsities.append(approximate_sparsity(sub_gd_x_final, threshold=1e-2))

    plt.figure(figsize=(12, 8))
    plt.plot(regu_lambdas, prox_gd_sparsities, label="Proximal Gradient", marker='o')
    plt.plot(regu_lambdas, admm_sparsities, label="ADMM", marker='o')
    plt.plot(regu_lambdas, sub_gd_sparsities, label="Subgradient", marker='o')
```

首先，如果都只1000轮迭代（根据上面的实验结果，1000轮迭代之后$\|x_k - x_{real}\|_2$与$\|x_k - x_{opt}\|_2$都变化不大了），发现：

得到如下图象：



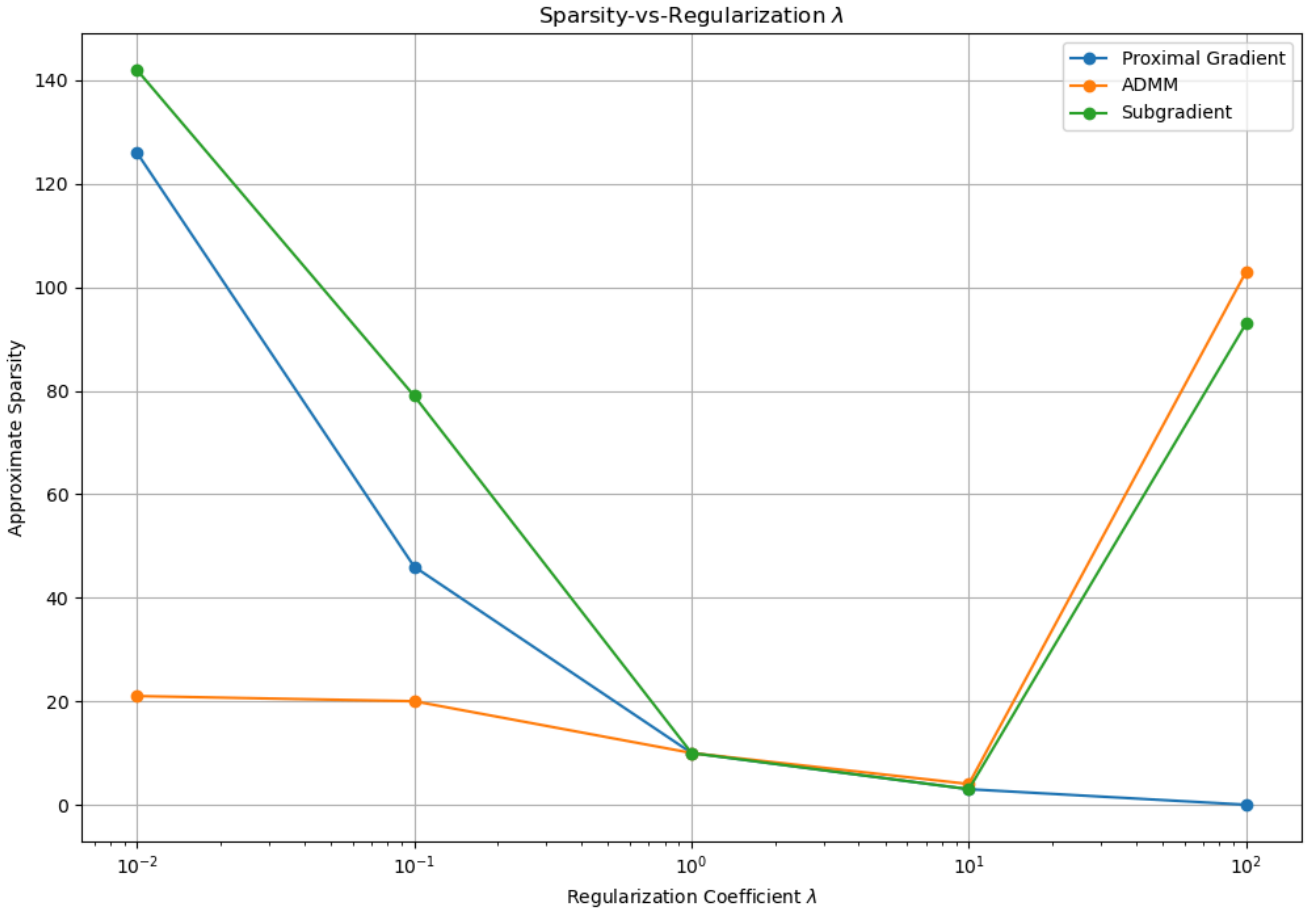
- 可以看出来，在一定合适范围内，一范数正则化项系数越大，算法得到的解越稀疏；
- 迭代轮数不够时，交替方向乘子法适合较小的正则化项系数（$\lambda \leq 1$）（增大迭代轮数会改变这一点，后面的实验结果可以看到）。
- 如果只进行1000轮迭代，根据上图和下图的实验结果，发现正则化项系数也不能过大，否则对于交替方向乘子法和次梯度法，非0元的个数反而增加，如下：



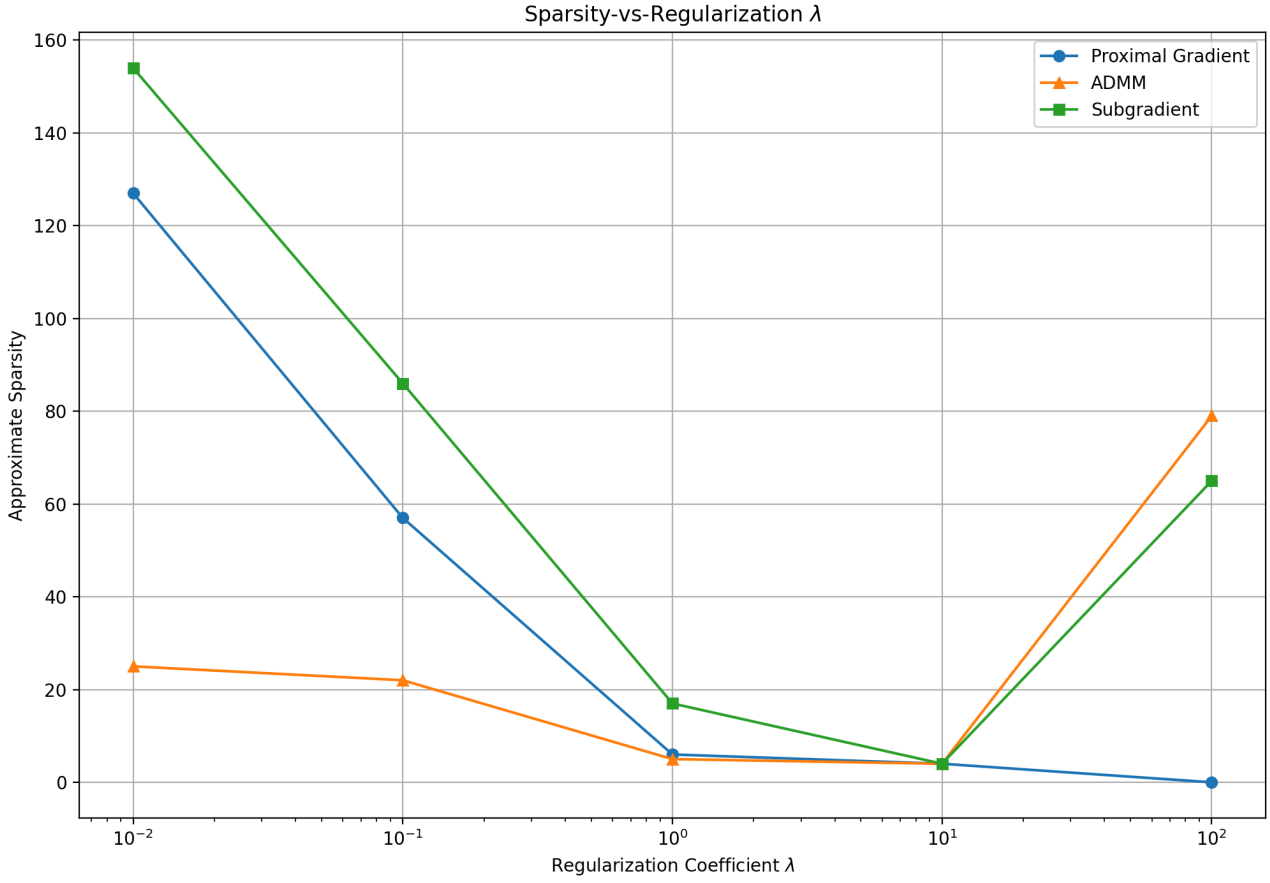
另外，$\lambda = 100$ 时，对于临近点梯度法，得到的解就可能太过稀疏了，每个元素的值都很小。

尝试增加迭代轮数进行试验，如下：

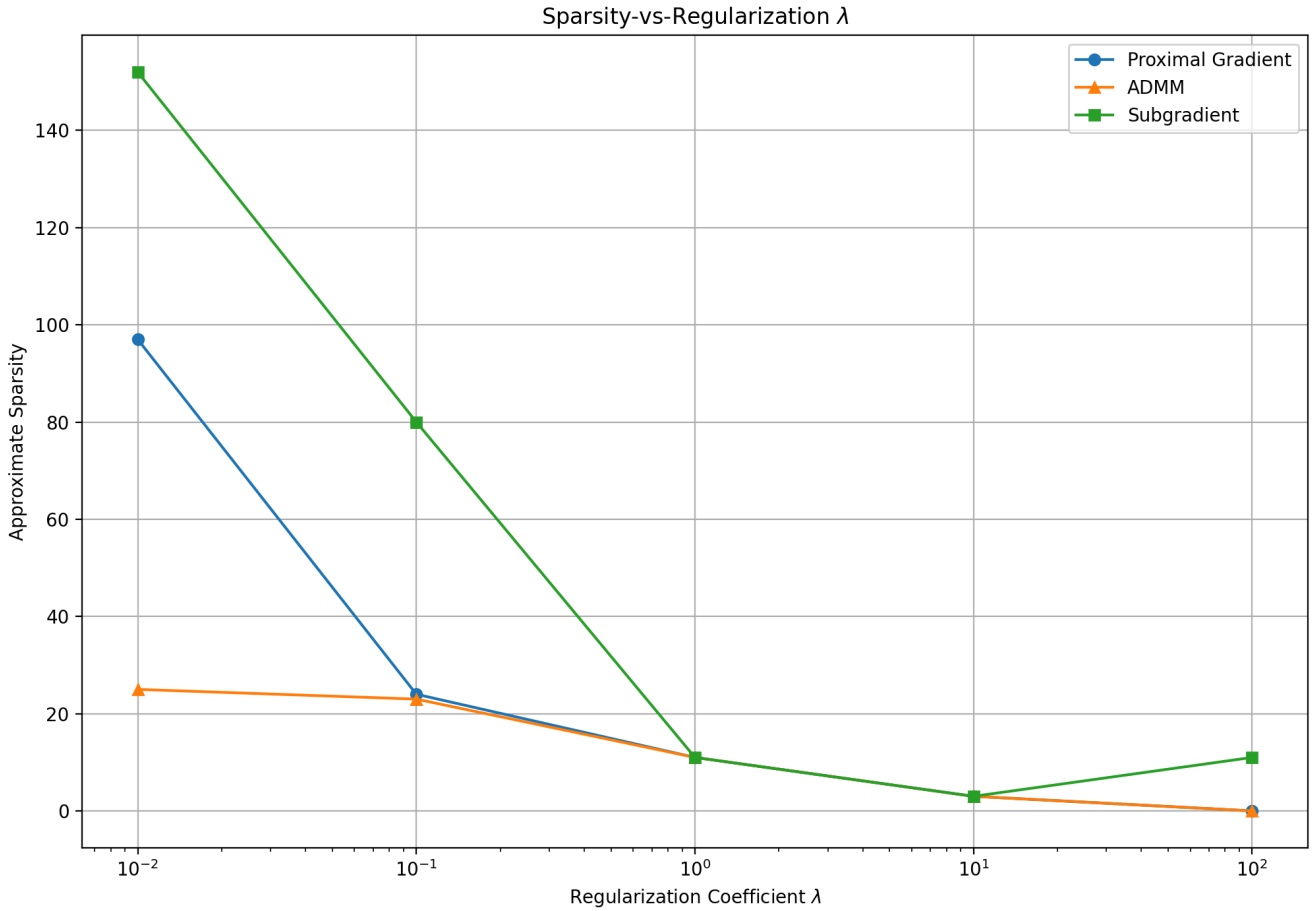
增加到2000轮迭代：


Sparsity-vs-Regularization λ
Proximal Gradient
ADMM
Subgradient
Approximate Sparsity
Regularization Coefficient λ
0
20
40
60
80
100
120
140
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

增加到5000轮迭代:


Sparsity-vs-Regularization λ
Proximal Gradient
ADMM
Subgradient
Approximate Sparsity
Regularization Coefficient λ
0
20
40
60
80
100
120
140
160
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

增加到10000轮迭代：



- 增加迭代轮数，能让ADMM在正则化项系数较大时（$\lambda = 10$），收敛到的解更稀疏；
- 但可以看到无论怎样改变迭代次数，太大的正则化项系数（$\lambda = 100$）得到的结果总是表现不佳的，要么非0元素的个数比更小的正则化系数实验结果更多了，要么就是过于稀疏，元素全部都变成了0
- 因为过大的正则化项系数会过于强调$\lambda\|\boldsymbol{x}\|_1$ 项的影响，而使得优化目标$\min_{\boldsymbol{x}} \frac{1}{2}\|A_1\boldsymbol{x} - \boldsymbol{b}_1\|_2^2 + \cdots + \frac{1}{2}\|A_{10}\boldsymbol{x} - \boldsymbol{b}_{10}\|_2^2$ 没有被关注到。
- 总之，1000的迭代轮数对于交替方向乘子法而言，$\|x_k - x_{real}\|_2$与$\|x_k - x_{opt}\|_2$ 已足够收敛了，但可能收敛到的解的稀疏程度并不是最优的；增大迭代轮数并选择稍微大一点的正则化项系数，可以收敛到一个稀疏性质更好的解。
- 对于这个问题，一范数正则化项系数 $\lambda \in [1, 10]$ 比较合适。

结果分析、总结

根据上面的理论计算分析与数值实验结果讨论，得到如下结果：

1. **交替方向乘子法**收敛速度最快；
2. **次梯度法**不稳定，同时也是收敛最慢的；
3. **临近点梯度法**和**交替方向乘子法**最后都能收敛到一个离真值的差距的二范数较小的解

4. 在一定的合理范围内（$\lambda \in [1e-2, 10]$），**一范数正则化项系数 λ 越大，算法得到的解向量越稀疏。**
5. 这样的实验结果确实是理论上合理的，因为：

- **交替方向乘子法**选用增广拉格朗日函数进行优化，更稳定，并且分解为易求解的子问题，能降低每一步的计算复杂度，同时交替更新不同的子问题变量，收敛更快；
- **次梯度法**是在近似梯度，对于不光滑的点（一范数正则化项导致的），算法的更新就容易不稳定；
- **临近点梯度法**针对问题的特殊结构，使用软门限操作，所以虽然比ADMM要慢，但也能能逐渐逼近一个稀疏且较优的解；
- **一范数正则化项系数 λ 越大**，说明对优化目标中非0元素的惩罚值$\lambda||\boldsymbol{x}||_1$越大，于是在最小化优化目标的情况下，就会趋于减少非0元素的个数，也就是让解更加稀疏。