最优化大作业上机报告

沙柯岑 2200010611

2024 年 12 月 13 日

目录

1 问题描述

考虑以下分组 LASSO 最优化问题

$$\min_{x\in\mathbb{R}^{n\times l}} \frac{1}{2}\|Ax-b\|_F^2+\mu\|x\|_{1,2}, \tag{1}$$

这里 $A\in\mathbb{R}^{m\times n}$, $b\in\mathbb{R}^{n\times l}$, $\mu>0$ 是给定的常数, 并且

$$\|x\|_{1,2}=\sum_{i=1}^{n}\|x(i,1:l)\|_2.$$

1. 分别用 cvx 调用 gurobi 和 mosek 求解器求解上述最优化问题.
2. 首先写出（1）的等价模型，使得可以被 mosek 和 gurobi 直接求解，并实施相应的代码.
3. 分别按照以下方法求解最优化问题（1）:
   (a) 对原问题使用次梯度法;
   (b) 对原问题使用光滑化的梯度法;
   (c) 对原问题使用临近点算法;
   (d) 对原问题使用加速的临近点算法;
   (e) 对对偶问题使用增广拉格朗日函数法;
   (f) 对对偶问题使用交替方向乘子法;
   (g) 对原问题结合线性化技巧使用交替方向乘子法.

2 问题分析及相应结果

为保证结果可以复现，以下问题所有地结果均在一固定种子下生成，在代码开头将种子设定为“24847563”，即可得到相应的结果.

2.1 问题 1

直接将原问题的形式在 cvx 中等价地表达出来即可，以下是两个不同的求解器得到的结果，数值结果保留 5 位有效数字[1], 时间以秒为单位保留两位小数[2]. 从表 1可以看出两个求解器的效果相差不大.

solver	Fval	Time	Sparisity	Errfun_exact	Err-to-cvx-mosek	Err-to-cvx-gurobi
cvx-mosek	6.0715e-01	8.42	0.110	4.1480e-05	0	2.5553e-07
cvx-gurobi	6.0715e-01	9.96	0.110	4.1312e-05	2.5553e-07	0

表 1: 用 cvx 求解的结果

2.2 问题 2

对于 mosek 求解器，将原问题转化为二次锥规划问题如下：

$$\begin{aligned}\min_{x\in\mathbb{R}^{n\times l},t\in\mathbb{R},s\in\mathbb{R}^n}\quad & \frac{t}{2}+\mu\cdot\mathbf{1}^\top s,\\ \text{s.t.}\quad & \|x(i,1:l)\|_2\le s(i),\qquad 1\le i\le n\\ & \|Ax-b\|_F^2\le t.\end{aligned} \tag{2}$$

[1]后面的表格中的数据也按照这个标准.

[2]8，9 秒的时间是不必要的，后来进行优化减少了时间（见总表），事后发现是建立模型时不够简洁导致的

这里，$\mathbf{1} = (1, 1, \cdots, 1)^\top$, 在实际写代码时，还需将最后一个旋转二次锥约束转化为标准的二次锥约束. 根据 mosek 语法，我们创建了 $nl + n + 1$ 个变量分别表示 $x(i, j)$, $s(i)$ 以及 t. Mosek 要求所有二次锥约束都写成标准形式

$$Fx + g \in \mathcal{S},$$

其中 $\mathcal{S} = \{(u, v)^\top \in \mathbb{R}^{k+1} | \|u\|_2 \le v\}$, 对(2)中的 $n + 1$ 个锥约束写出相应的 F, g 再将它们组合在一起，即可使用 mosek 对原问题进行求解 (具体可参见相应代码文件).

对于 gurobi 求解器，需要将问题转化为带二次约束的二次规划的形式如下：

$$\begin{aligned} \min_{u \in \mathbb{R}^{m\times l}, x \in \mathbb{R}^{n\times l}, s \in \mathbb{R}^n} \quad & \frac{1}{2}\|u\|_F^2 + \mu \cdot \mathbf{1}^\top s, \\ \text{s.t.} \quad & x(i, 1:l)x(i, 1:l)^\top - s(i)^2 \le 0, \qquad 1 \le i \le n \\ & s(i) \ge 0, \\ & Ax - u = -b. \end{aligned} \tag{3}$$

这里，$\mathbf{1} = (1, 1, \cdots, 1)^\top$. 类似 mosek, 我们需要创建 $(m + n)l + n$ 个变量分别表示 u, x, s. 在实际写代码时，根据 gurobi 语法，需要分别写出一、二次项部分 F, $Q \in \mathbb{R}^{[(m+n)l+n]\times[(m+n)l+n]}$, 变量范围 lb,$ub \in \mathbb{R}^{(m+n)l+n}$（分别表示各变量的下界和上界）, 以及(3)中的 n 个二次约束，完成这些内容即可用 gurobi 对原问题进行求解 (具体可参见相应代码文件).

以下是建立上述模型后分别直接用 mosek 和 gurobi 求解器得到的结果. 从表 2可以看出我们建立的 gurobi 模型求解效果相比 cvx-mosek、cvx-gurobi、mosek 都要差一些[3]，尤其是稀疏性效果不佳，因此仍然有改进的空间. 当然，因为问题(3)可以写成其他的等价形式, 也可以用 gurobi 语法实现，比如：

$$\begin{aligned} \min_{x \in \mathbb{R}^{n\times l}, s \in \mathbb{R}^n, t \in \mathbb{R}} \quad & \frac{t}{2} + \mu \cdot \mathbf{1}^\top s, \\ \text{s.t.} \quad & x(i, 1:l)x(i, 1:l)^\top - s(i)^2 \le 0, \qquad 1 \le i \le n \\ & s(i) \ge 0, \\ & \|Ax - b\|_F^2 \le t. \end{aligned} \tag{4}$$

或者

$$\begin{aligned} \min_{x \in \mathbb{R}^{n\times l}, s \in \mathbb{R}^n} \quad & \frac{1}{2}\sum_{i=1}^{l}(x_i^\top A^\top A x_i - 2b_i^\top x_i + b_i^\top b_i) + \mu \cdot \mathbf{1}^\top s, \\ \text{s.t.} \quad & x(i, 1:l)x(i, 1:l)^\top - s(i)^2 \le 0, \qquad 1 \le i \le n \\ & s(i) \ge 0. \end{aligned} \tag{5}$$

这里 x_i 表示 x 的第 i 列，b_i 表示 b 的第 i 列. 这两个模型求解的效果比(3)更差，甚至不能收敛到最优解, 结果如表 3所示，因此在使用 gurobi 求解器解二次规划问题时，应尽量简化目标函数二次型的形式，（比较(3)和(4)），优先选择线性约束而非二次约束（比较(3)和(5)）.

solver	Fval	Iter	Time	Sparisity	Errfun_exact	Err-to-cvx-mosek	Err-to-cvx-gurobi
mosek	6.0715e-01	11	0.79	0.110	4.1413e-05	1.0231e-07	1.6911e-07
gurobi	6.0715e-01	11	0.83	0.121	4.3251e-05	2.2605e-06	2.4753e-06

表 2: 直接用 mosek 和 gurobi 求解的结果

2.3 问题 3.（a）

记目标函数为 $f(x, \mu)$，则 $f(x, \mu)$ 关于 x 的次梯度为 (按每一行表示)：

$$\partial f(x, \mu)(i, :) = (A^\top(Ax - b))(i, :) + \begin{cases} \mu x(i, 1:l)/\|x(i, 1:l)\|_2 & \text{if } x(i, 1:l) \ne 0, \\ \{\mu g \in \mathbb{R}^{1\times l} | \|g\|_2 \le 1\} & \text{else.} \end{cases}$$

[3]gurobi 的版本会影响结果，用 gurobi 11.0.3 得到的结果会略好一些，该实验采用的是 gurobi 12.0.0 版本.

solver	Fval	Iter	Time	Sparisity	Errfun_exact	Err-to-cvx-mosek	Err-to-cvx-gurobi
gurobi(3)	6.0715e-01	11	0.83	0.121	4.3251e-05	2.2605e-06	2.4753e-06
gurobi(4)	6.8843e-01	30	2.07	0.992	2.9837e-03	2.9617e-03	2.9618e-03
gurobi(5)	6.0759e-01	15	0.92	0.274	5.1274e-05	1.1977e-05	1.2164e-05

表 3: 用不同的 gurobi 模型求解的结果

我们采用次梯度法，并且结合 BB 步长、回退法线搜索、连续化策略完成对该问题的求解. 具体算法步骤如下：

Algorithm 1: 次梯度法求解最优化问题（1）

Input: 初始点 x_0, A, b, μ, 常数 L, 连续化参数 $\rho > 1$, $k \in \mathbb{N}^*$, $iter = 0$, $f^* = \inf$.

Result: 返回最优解 x_0，最小值 f^*, 迭代次数 $iter$.

$v = \mu \times \rho^k$; $x_1 = x_0 - \frac{1}{L}\partial f(x_0, v)$;
while $v \neq \mu \vee |f(x_1, \mu) - f(x_0, \mu)| \geq 10^{-10}$ **do**
  $s = x_1 - x_0$, $y = \partial f(x_1, v) - \partial f(x_0, v)$, $g = \partial f(x_1, v)$;
  $\alpha_{BB} = \|s\|_F^2 / \langle s, y\rangle$, $a = \max(\alpha_{BB}, L)$;
  while $a > \frac{1}{L} \wedge f(x_1, v) - f(x_1 - ag, v) < \frac{1}{2L}\|g\|_F^2$ **do**
    $a = a/2$;
  end
  while $f(x_1, v) - f(x_1 - ag, v) < 0$ **do**
    $a = a/2$;
    if $a < 10^{-15}$ **then**
      break
    end
  end
  $x_0 = x_1$, $x_1 = x_1 - ag$, $iter = iter + 1$, $f^* = \min(f^*, f(x_0, \mu))$;
  if $v > \mu \wedge \|f(x_0, v) - f(x_1, v)\| \leq 10^{-5} \cdot v$ **then**
    $v = v/\rho$;
  end
end

Output: x_0, $iter$, f^*

在该算法中，我们先将 μ 适当增大至 v, 对 $f(x, v)$ 求解最优化问题，当求解到一定程度时适当减小 v, 直至 $v = \mu$. 理论上不可微函数没有 Lipchitz 连续性，这里为了防止 BB 步长过大，设置的默认 $L = \|A^\top A\|_2$[4], 为了防止 BB 步长过小，设置了最小步长 10^{-15}.

此外，次梯度的选取也很重要，因为当 $\|x(i,:)\|_2$ 过小时，次梯度 $x(i, 1:l)/\|x(i, 1:l)\|_2$ 的误差会很大，因此需要设置一个阈值 ϵ, 当 $\|x(i,:)\|_2 \leq \epsilon$ 时，将其视为 0[5]，并且此时取对应的次梯度分量 $\partial f(x, \mu)(i,:) = (A^\top(Ax - b))(i,:) + x(i,:)/\epsilon$, 经过试验，取 $\epsilon = 10^{-6}$ 就能够得到较好的结果.

推荐的连续化参数为 $\rho = 5$, $k = 5$. 线搜索采用回退法，分为两个阶段，步长 $> L$ 时，采用弱化的 armijo 准则（$c_1 = \frac{1}{2}$，要求从 $\frac{a}{2}$ 降至 $\frac{1}{2L}$），步长 $< L$ 时，只需保证每一次单调下降即可.

下面是次梯度下降法得到的结果图像，作为比较，还试验了不加 BB 步长和不加连续化策略的结果. 在本例中，不加 BB 步长也能达到很好的效果，然而，在问题 3（b）我们将看到 BB 步长的作用. 此外，如果不用连续化技巧，算法需要约 20 万步才能收敛.

[4]L 的选取受梯度下降法的启发，当 $\|x\|$ 不太小时，可以将 f 看成梯度 $L - Lipchitz$ 连续的函数

[5]严格地说，应该是不用 $x(i, 1:l)/\|x(i, 1:l)\|_2$ 去计算，而是取一个 $\|g\|_2 \leq 1$，正如后文所示.

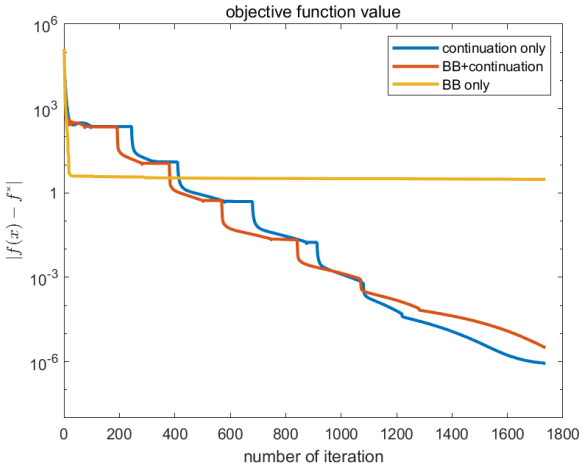


图 1: 次梯度法求解得到的结果

2.4 问题 3.（b）

采用光滑化策略对 SGD 算法进行加速，我们采用 huber 光滑函数对 $\|x(i,:)\|_2$ 光滑化, huber 函数为

$$h_u(x) = \begin{cases} \frac{x^2}{2u} & \textbf{if } 0 \le x \le u, \\ x - \frac{u}{2} & \textbf{if } x \ge u. \end{cases}$$

光滑化后的函数为

$$f_u(x,\mu) = \frac{1}{2}\|Ax-b\|_F^2 + \mu \sum_{i=1}^{n} h_u(\|x(i,:)\|_2),$$

并且

$$|f_u(x,\mu) - f(x,\mu)| \le \frac{\mu n u}{2}.$$

容易证明，$h_u(x)$ 是梯度 $\frac{1}{u}$-Lipchitz 连续的，进而 $f_u(x,\mu)$ 是梯度 $(\|A^\top A\|_2 + \frac{\mu\sqrt{n}}{u})$-Lipchitz 连续的[6]，因此可以用标准的梯度算法求解，也可以使用 BB 步长 + 线搜索的方法. 如 3（a）一样我们采用连续化策略，这里我们采取了一个小技巧，随着 u 的增大，光滑化函数的梯度 Lipchitz 常数也在增大，因此在降低 u 的同时可以同时降低 μ, 保证在迭代初期 Lipchitz 常数不会增大，有较快的收敛速度.

下面是光滑化梯度下降法得到的结果图像，我们一共做了 4 组试验. 由图 2可以看出，BB 步长在加速收敛起到关键作用，比连续化策略的作用更大，二者结合可以得到更快的收敛速度. 尽管光滑化技巧所需要的迭代次数略多，但能够达到更好的精度，并且非光滑的次梯度算法结果受参数的影响很大, 要想达到更高的精度需要相当多的迭代步数，表 4是光滑化与非光滑结果的比对.

solver	Fval	Iter	Time	Sparisity	Errfun_exact	Err-to-cvx-mosek	Err-to-cvx-gurobi
non-smoothed	6.0715e-01	1983	0.51	0.113	4.5045e-05	4.0763e-06	4.2597e-06
smoothed	6.0715e-01	3325	0.70	0.110	4.1520e-05	1.9296e-07	2.3676e-07

表 4: 3(a) 与 3(b) 结果对比

2.5 问题 3（c），3（d）

记 $f(x) = \frac{1}{2}\|Ax-b\|_F^2$, $h_\mu(x) = \mu\|x\|_{1,2}$, 则 $h_\mu(x)$ 的临近点算子表达式为 (按行分量分别表示)：

$$\text{prox}_{th_\mu}(y)(i,:) = \begin{cases} 0 & \text{if } \|y(i:)\|_2 \le t\mu, \\ y(i,:) - t\mu\frac{y(i,:)}{\|y(i,:)\|_2} & \text{else.} \end{cases}, 1 \le i \le n$$

[6]实际中取 $L = \|A^\top A\|_2 + \frac{\mu}{u}$ 就可以达到很好的效果

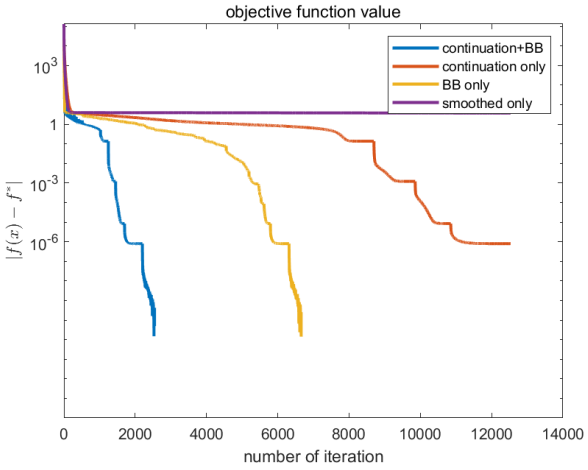


图 2: 光滑化梯度法求解得到的结果

记 $L = \|A^\top A\|_2$, 则 $f(x)$ 是梯度 $L-$Lipchitz 连续的，从而可用标准的近似点梯度法求解，迭代格式如下：

$$y_k = x_k - \frac{1}{L}\nabla f(x_k);$$
$$x_{k+1} = \text{prox}_{\frac{1}{L}h_\mu}(y_k).$$

我们仍然想尝试用 BB 步长观察是否有加速效果，以及连续化策略的加速效果，方式类似算法 1，问题 3（d）采用的是 FISTA 加速算法以及连续化策略，如算法 2所示：

Algorithm 2: 使用 FISTA 加速临近点算法求解最优化问题

Input: 矩阵 $A \in \mathbb{R}^{m\times n}$, 向量 $b \in \mathbb{R}^l$, 参数 μ, 初始点 x_0, 常数 L, 连续化参数 $\rho > 1$, $N \in \mathbb{N}^*$.

Result: 返回最优解 x_0，最小值 f^*, 迭代次数 $iter$.

$iter = 0$; $k = 1$; $v = \mu \times \rho^N$; $x_1 = \text{prox}_{\frac{1}{L}h_v}\left(x_0 - \frac{1}{L}A^T(Ax_0 - b), \frac{1}{L}\right)$;

while $v \geq \mu$ **do**

  $f^* = f(x_0)$; $iter = iter + 1$; $k = k + 1$;

  $y = x_1 + \frac{k-2}{k+1}(x_1 - x_0)$; $x_0 = x_1$; $x_1 = \text{prox}_{\frac{1}{L}h_v}\left(y - \frac{1}{L}A^T(Ay - b), \frac{1}{L}\right)$;

  if $\|x_0 - x_1\|_F < 10^{-5} \wedge v > \mu$ **then**

    $v = v/\rho$; $k = 1$;

  end

  if $v == \mu \wedge \|x_0 - x_1\|_F < 10^{-8}$ **then**

    break;

  end

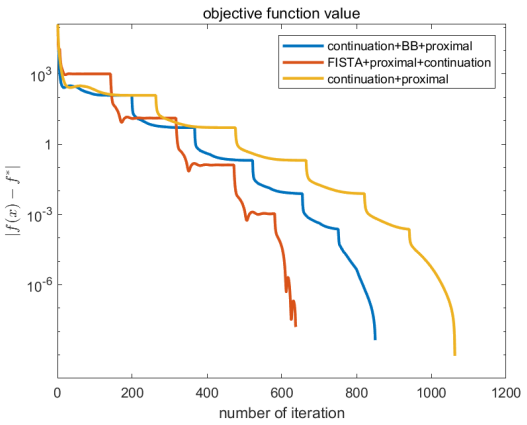
end

Output: x_0, $iter$, f^*

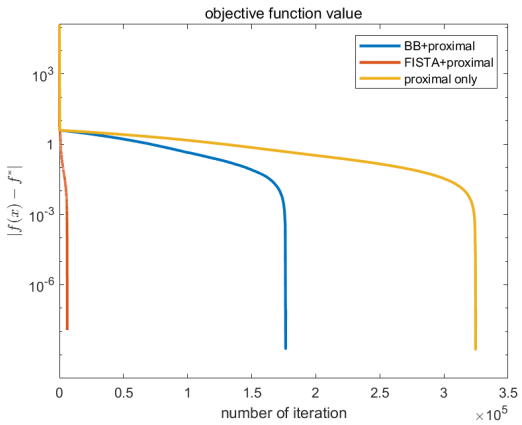
图 3是采用近似点梯度法得到的数值结果，分为两组，一组包含连续化策略而另一组不包含. 推荐的连续化参数为 $\rho = 10$, $N = 5$, 可以看到，与光滑化方法不同，连续化策略在临近点算法的效果比 BB 方法更显著. 相比于次梯度法和光滑化梯度法，临近点算法的收敛速度更快，精度也更高，表 5是用临近点算法得到的性能指标(均使用连续化策略).

solver	Fval	Iter	Time	Sparisity	Errfun_exact	Err-to-cvx-mosek	Err-to-cvx-gurobi
proximal+BB	6.0715e-01	947	0.13	0.110	4.1329e-05	2.4914e-07	2.6723e-07
proximal+FISTA	6.0715e-01	841	0.04	0.110	4.1328e-05	2.4999e-07	2.5531e-08

表 5: 3(d) 与 3(e) 的结果对比



(a) 带连续化技巧的临近点算法



(b) 不带连续化技巧的临近点算法

图 3: 近似点算法求解得到的结果

2.6 问题 3（e）

原问题的对偶问题是：

$$\begin{aligned} \min_{y\in\mathbb{R}^{m\times l}} \quad & \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle, \\ \text{s.t.} \quad & \|A^\top y\|_{1,2} \le \mu. \end{aligned}$$

为了让约束函数可微，我们改写为如下等价的优化问题：

$$\begin{aligned} \min_{y\in\mathbb{R}^{m\times l}, s\in\mathbb{R}^{n\times l}} \quad & \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle, \\ \text{s.t.} \quad & \|(A^\top y)_i\|^2 \le \mu^2. \qquad 1 \le i \le n \end{aligned}$$

其中 $(A^\top y)_i$ 表示矩阵 $A^\top y$ 的第 i 行. 约束是不等式约束，通过消元，可以得到如下增广拉格朗日函数：

$$L_\sigma(y,\lambda) = \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle + \frac{\sigma}{2}\sum_{i=1}^{n}(\max\{\frac{\lambda_i}{\sigma} + \|(A^\top y)_i\|_2^2 - \mu^2, 0\}^2 - \frac{\lambda_i^2}{\sigma^2}).$$

记乘子更新函数为：

$$U_i(\sigma, y, \lambda) = \max\{\lambda_i + \sigma(\|(A^\top y)_i\|_2^2 - \mu^2), 0\}, \quad 1 \le i \le n$$

$$\vec{U}(\sigma, y, \lambda) = (U_i(\sigma, y, \lambda))^\top.$$

增广拉格朗日函数关于 y 的梯度为：

$$\nabla_y L_\sigma(y,\lambda) = y - b + 2A \cdot diag(\vec{U}(\sigma, y, \lambda)) \cdot A^\top y.$$

乘子更新公式为：

$$\lambda_i^{k+1} = U_i(\sigma_k, y_{k+1}, \lambda^k) = \max\{\lambda_i^k + \sigma_k(\|(A^\top y_{k+1})_i\|_2^2 - \mu^2), 0\}. \quad 1 \le i \le n$$

约束违反度为：

$$V(\sigma_k, y_{k+1}, \lambda^k) = \frac{\|\vec{U}(\sigma_k, y_{k+1}, \lambda^k) - \lambda^k\|_2}{\sigma_k}.$$

采用一般约束问题增广拉格朗日函数法算法如算法 3所示：

Algorithm 3: 增广拉格朗日乘子法用于对偶问题

Input: 矩阵 A, 向量 b, 参数 μ, 罚因子增长率 ρ，初始罚因子 σ，罚因子上限 M, 目标精度 η, 目标约束违反度 ε, 初始精度 η_0，初始约束违反度 ε_0, 衰减指数 β

初始化 $\lambda^0 = \mathbf{0}_{n\times 1}$, $y_0 = \mathbf{0}_{m\times l}$，外循环次数 $k = 0$
while $V(\sigma_k, y_k, \lambda^k) > \varepsilon \vee \|\nabla L_{\sigma_k}(y_k, \lambda^k)\|_F > \eta$ **do**
  if $\sigma > M$ **then**
    break
  end
  以初始点 y_k 用 Nesterov 加速算法求解最优化子问题 $y_{k+1} = \operatorname{argmin}_z L_\sigma(z, \lambda^k)$ 使得 $\|\nabla_y L_\sigma(y_{k+1}, \lambda)\| \le \eta_k$
  if $V(\sigma_k, y_k, \lambda^k) \le \varepsilon_k$ **then**
    if $V(\sigma_k, y_k, \lambda^k) \le \varepsilon_0 \wedge \|\nabla_y L_{\sigma_k}(y_k, \lambda^k)\|_F \le \eta_0$ **then**
      break
    end
    $\lambda^{k+1} = \vec{U}(\sigma_k, y_{k+1}, \lambda^k)$
    $\varepsilon_{k+1} = \varepsilon_{k+1}/\rho$
    $\sigma_{k+1} = \sigma_k$
    $\eta_{k+1} = \eta_k$
  else
    $\sigma_{k+1} = \rho \cdot \sigma_k$
    $\eta_{k+1} = \eta_k/\rho^\beta$
    $\lambda^{k+1} = \lambda^k$
    $\varepsilon_{k+1} = \varepsilon_{k+1}$
  end
  $k = k + 1$
end

Result: 极小点 y^*, 乘子 λ_*, 外循环次数 k

算法中的 Nesterov 加速方法类似于算法 2的 FISTA，只不过没有取临近点. 推荐的参数是 $\sigma_0 = 10^5$, $\rho = 10$, $\beta = \lg 5$, $M = 10^7$, $\varepsilon = 10^{-7}$, $\eta = 10^{-5}$, $\eta_0 = \varepsilon_0 = 10^{-2}$.

除此以外，还设置了罚因子上限，经验表明，初始罚因子应取得比较大，相同的精度比约束违反度更难满足，因此精度上升的速度慢一些（由衰减指数 β 控制）得到的数值结果将在最后的总表中显示.

2.7 问题 3（f）

原问题的对偶问题还可以写成

$$\begin{aligned} \min_{y\in\mathbb{R}^{m\times l}, s\in\mathbb{R}^{n\times l}} \quad & \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle, \\ \text{s.t.} \quad & A^\top y - s = 0, \\ & \|s\|_{1,2} \le \mu. \end{aligned}$$

去掉不等式约束，令集合 $\mathbf{C} = \{s \in \mathbb{R}^{n\times l} \mid \|s\|_{1,2} \le \mu\}$ 对偶问题还可以写成

$$\begin{aligned} \min_{y\in\mathbb{R}^{m\times l}, s\in\mathbb{R}^{n\times l}} \quad & \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle + \mathbf{1}_{\mathbf{C}}(s), \\ \text{s.t.} \quad & A^\top y - s = 0. \end{aligned}$$

问题化为这种形式即可使用交替方向乘子法. 首先写出该问题的增广拉格朗日函数

$$L_\sigma(y, s, \lambda) = \frac{1}{2}\|y\|_F^2 - \langle y, b\rangle + \mathbf{1}_{\mathbf{C}}(s) + \langle \lambda, A^\top y - s\rangle + \frac{\sigma}{2}\|A^\top y - s\|_F^2.$$

首先固定 s, 对 L_σ 关于 y 求极小值，这是关于 y 的二次函数，极小值有显式表达：

$$y^* = (I + \sigma AA^\top)^{-1}(b + A\lambda + \sigma As). \tag{6}$$

然后固定所求的 y, 再对 L_σ 关于 s 求极小值, 经过简单计算可知 s 也有显示表达[7]：

$$s_i = \left((A^\top y)_i + \frac{\lambda_i}{\sigma}\right) \min\{\frac{\mu}{\|(A^\top y)_i + \frac{\lambda_i}{\sigma}\|}, 1\}, \quad 1 \le i \le n \tag{7}$$

其中 s_i 表示矩阵变量 s 的第 i 行. 在内循环中求解(6)时，先求出 $I+\sigma AA^\top$ 的 Cholesky 分解再求出相应的 y^*，得到如下 ADMM 算法，并且根据原始可行性条件和对偶可行性条件违反度的相对情况动态调节罚因子：

Algorithm 4: 交替方向乘子法用于对偶问题

Input: 矩阵 A, 向量 b, 参数 μ, 罚因子变化率 ρ，初始罚因子 σ，协调率 η, 停机阈值 ε

初始化 $\lambda^0 = \mathbf{0}_{n\times 1}$, $y_0 = \mathbf{0}_{m\times l}$, $s^1 = \mathbf{1}_{n\times l}$, $s^0 = \mathbf{0}_{n\times l}$, 迭代次数 $k = 0$, 初始 Cholesky 分解 $I + \sigma AA^\top = R^\top R$

while $\|A(s^1 - s^0)\|_F > \varepsilon$ **do**

  $s^0 = s^1$

  $y_0 = R^{-1}(R^{-\top}(b + A\lambda^k + \sigma As^1))$

  $s_i^1 = \left((A^\top y_0)_i + \frac{\lambda_i^k}{\sigma}\right) \min\left\{\frac{\mu}{\|(A^\top y_0)_i + \frac{\lambda_i^k}{\sigma}\|}, 1\right\}, \quad 1 \le i \le n$

  $\lambda^{k+1} = \lambda^k + \sigma(A^\top y_0 - s^1)$

  if $\|s^1 - A^\top y_0\|_F > \eta\|A(s^1 - s^0)\|_F$ **then** 增大罚因子

    $\sigma = \rho\sigma$, $R = \mathrm{chol}(I + \sigma AA^\top)$

  else if $\|A(s^1 - s^0)\|_F > \eta\|s^1 - A^\top y_0\|_F$ **then** 减小罚因子

    $\sigma = \sigma/\rho$, $R = \mathrm{chol}(I + \sigma AA^\top)$

  end

  $k = k + 1$

end

Result: 极小点 y_0, 乘子 λ^k, 迭代次数 k

推荐的 $\rho = 2$, $\sigma = 1$, $\eta = 10$, $\varepsilon = 10^{-11}$. 当然，由于该算法每一个子问题都求的是精确解，因此参数的选择非常灵活.

2.8 问题 3（g）

将原问题进行如下改写：

$$\begin{aligned} \min_{s\in\mathbb{R}^{m\times l}, x\in\mathbb{R}^{n\times l}} \quad & \frac{1}{2}\|s\|_F^2 + \mu\|x\|_{1,2} \\ \text{s.t.} \quad & Ax - s - b = 0. \end{aligned}$$

它的增广拉格朗日函数为

$$L_\sigma(s, x, \lambda) = \frac{1}{2}\|s\|_F^2 + \mu\|x\|_{1,2} + \langle\lambda, Ax - s - b\rangle + \frac{\sigma}{2}\|Ax - s - b\|_F^2.$$

首先固定 x, 对 $L_\sigma(s, x, \lambda)$ 关于 s 求极小值，此时是二次函数，容易求得

$$s^* = \frac{\sigma(Ax - b) + \lambda}{1 + \sigma}.$$

接着固定 s，采用线性化技巧，原本应求

$$x^* = \arg\min_y \mu\|y\|_{1,2} + \frac{\sigma}{2}\|Ay - s - b + \frac{\lambda}{\sigma}\|_F^2,$$

[7]在实际写代码时不需要将每个分量写出来，这里只是为了表达方便和直观.

该问题没有显式表达，假设目前的变量取值为 (s_1, x_1, λ_1)，则改为求其在当前点的二次近似最小值：

$$x^* = \arg\min_y \mu\|y\|_{1,2} + \sigma y^\top A^\top (Ax_1 - s_1 - b + \frac{\lambda_1}{\sigma}) + \frac{1}{2\eta}\|y - x_1\|_F^2,$$

其中 η 为步长参数，这等价于采用近似点梯度步：

$$x^* = \text{prox}_{\eta\mu\|\cdot\|_{1,2}}\left(x_1 - \eta\sigma A^\top (Ax_1 - s_1 - b + \frac{\lambda_1}{\sigma})\right). \tag{8}$$

步长参数的选取采用 BB 步长 (还需 armoji 准则 + 回退法线搜索)，即 $\eta = \frac{\|x_1 - x_0\|_F^2}{\sigma\|A(x_1 - x_0)\|_F^2}$, 其中 x_0 是上一步迭代得到解. 最后得到的算法如下所示：

Algorithm 5: 交替方向乘子法结合线性化技巧用于原问题

Input: 矩阵 A, 向量 b, 参数 μ, 罚因子变化率 ρ，初始罚因子 σ，协调率 u, 停机阈值 ε

初始化 $\lambda = \mathbf{0}_{n\times 1}$, $s_0 = \mathbf{0}_{m\times l}$, $x_1 = \mathbf{1}_{n\times l}$, $x_0 = \mathbf{0}_{n\times l}$, 迭代次数 $k = 0$

while $\|A(x_0 - x_1)\|_F > \varepsilon$ **do**

  $r = \frac{\|x_0 - x_1\|_F^2}{\sigma\|A(x_0 - x_1)\|_F^2}$

  $x_0 = x_1, k = k + 1$

  $s_0 = (\sigma(Ax_1 - b) + \lambda)/(1 + \sigma)$

  $r = \text{linesearch}(r)$

  $g = x_1 - r\sigma(A^\top(Ax_1 - b - s_0 + \lambda/\sigma))$

  $x_1 = \text{prox}_{\eta\mu\|\cdot\|_{1,2}}\left(x_1 - \eta\sigma A^\top(Ax_1 - s_0 - b + \frac{\lambda_1}{\sigma})\right)$, $\lambda = \lambda + \sigma(Ax_1 - b - s_0)$

  if $u\|A(x_0 - x_1)\|_F < \|Ax_1 - b - s_0\|_F$ **then**

    $\sigma = \sigma \cdot \rho$

  end

  else if $u\|Ax_1 - b - s_0\|_F < \|A(x_0 - x_1)\|_F$ **then**

    $\sigma = \sigma/\rho$

  end

end

Result: 极小点 x_1, 乘子 λ, 迭代次数 k

为了算法简洁明了，算法 5中省略了线搜索的具体过程，详细步骤可参见相应代码. 推荐的 $\rho = 10$, $u = 10$, $\varepsilon = 10^{-10}$, $\sigma = 10^{-4}$. 注意 σ 的取值最好不超过 1, 由(8)可知 σ 越大，η 越小，临近点算子的效果越弱，因此在保证精度的情况下应使罚因子尽可能小.

3 结果汇总

3.1 总表

下面给出所有问题的求解性能指标[8]，对每个问题，我们调整适当的精度，使得得到的解和 cvx-mosek,cvx-gurobi 的误差不超过 2.5×10^{-6}（SGD 除外，达不到这样的精度). 由于最后对 cvx-mosek, cvx-gurobi 作了一些修正[9]，因此结果与前面并不完全相同.

3.2 各结果的可视化分析

[8]最优值都是 0.607146, −1 表示无法获得该方法的迭代次数，ALM Dual 统计的是内层循环的总迭代次数.

[9]简化了模型的表达式，减少了 cvx 建立模型的时间

方法	CPU (s)	迭代次数	稀疏性	误差至精确解	误差至 CVX-Mosek	误差至 CVX-Gurobi
CVX-Mosek	1.33	-1	0.110	4.15×10^{-5}	0.00	2.71×10^{-7}
CVX-Gurobi	1.10	-1	0.110	4.17×10^{-5}	2.71×10^{-7}	0.00
Mosek	0.48	11	0.110	4.14×10^{-5}	1.02×10^{-7}	2.92×10^{-7}
Gurobi	0.65	11	0.121	4.33×10^{-5}	2.26×10^{-6}	2.21×10^{-6}
SGD Primal	0.54	1983	0.113	4.50×10^{-5}	4.08×10^{-6}	3.96×10^{-6}
GD Primal	0.81	3325	0.110	4.15×10^{-5}	1.93×10^{-7}	1.86×10^{-7}
FGD Primal	0.56	6897	0.110	4.13×10^{-5}	2.52×10^{-7}	3.42×10^{-7}
ProxGD Primal	0.15	947	0.110	4.13×10^{-5}	2.49×10^{-7}	3.39×10^{-7}
FProxGD Primal	0.05	810	0.110	4.13×10^{-5}	2.50×10^{-7}	3.41×10^{-7}
ALM Dual	0.83	7608	0.110	4.14×10^{-5}	8.74×10^{-7}	8.63×10^{-7}
ADMM Dual	0.11	497	0.110	4.13×10^{-5}	2.57×10^{-7}	3.49×10^{-7}
ADMM Primal	0.20	539	0.110	4.13×10^{-5}	2.57×10^{-7}	3.49×10^{-7}

表 6: 不同优化算法的性能对比

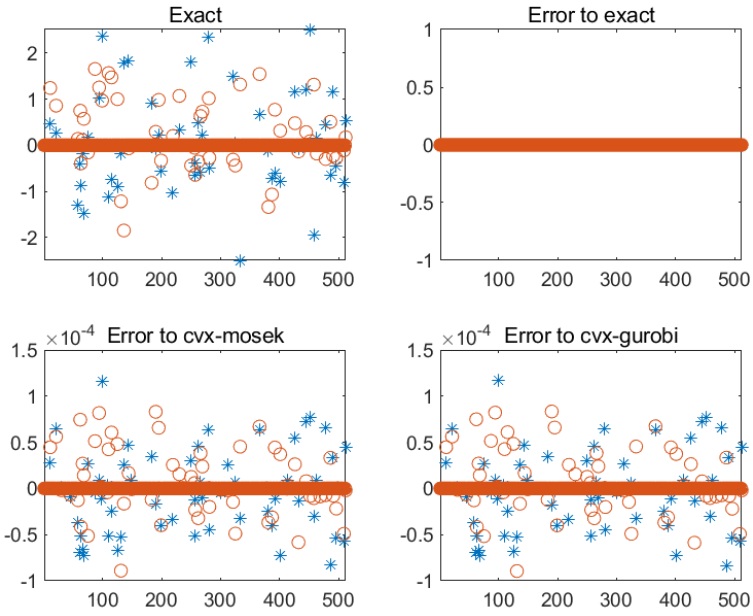


图 4: Exact solution and error analysis.

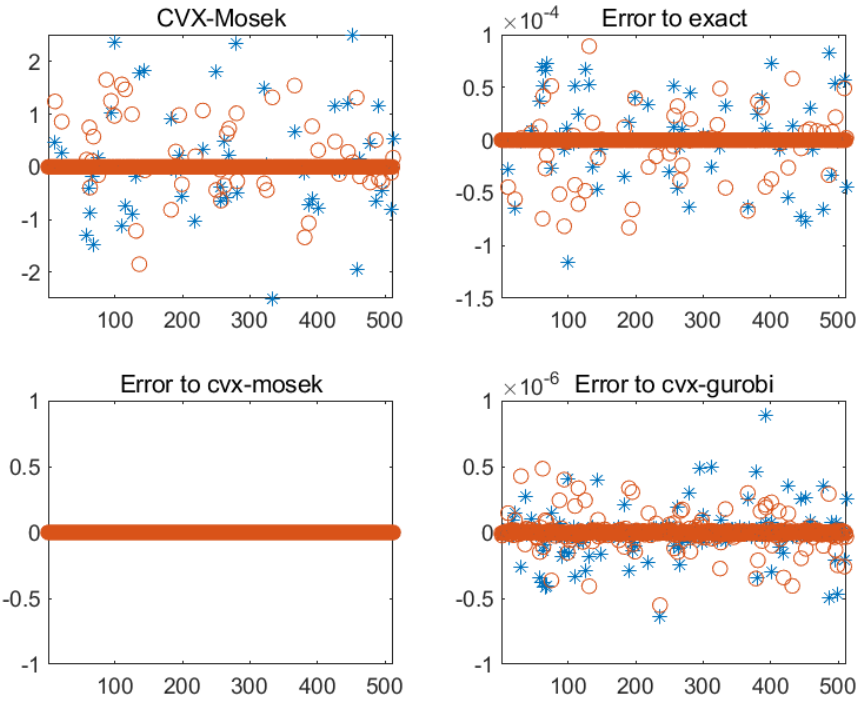


图 5: Error analysis for CVX-Mosek.

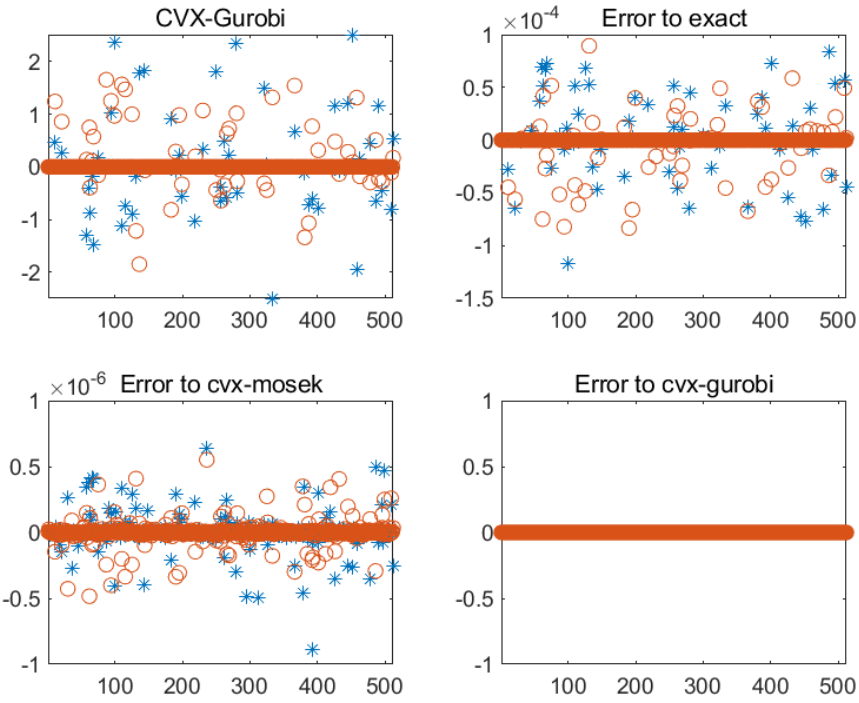


图 6: Error analysis for CVX-Gurobi.

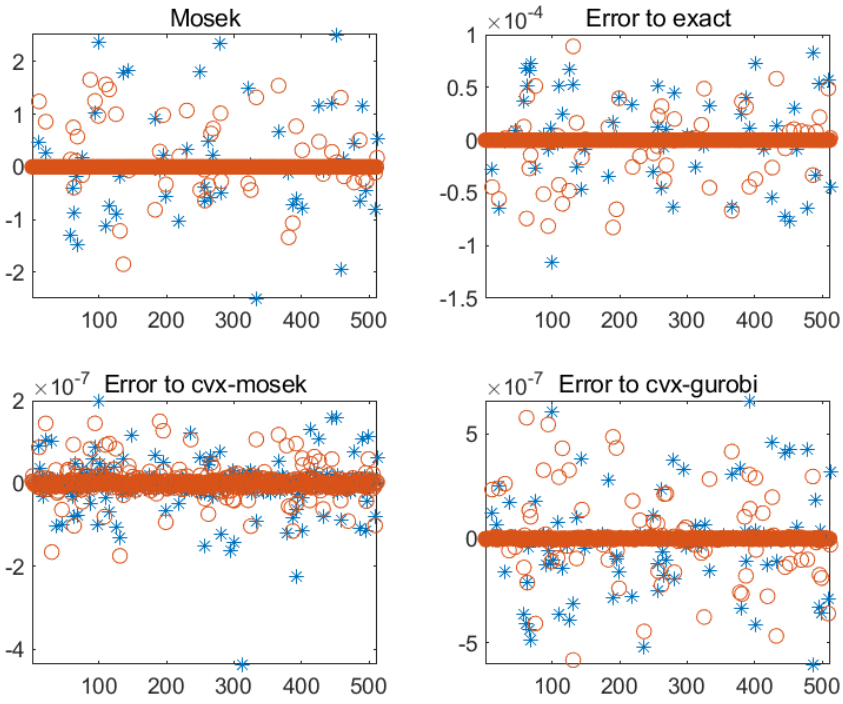


图 7: Error analysis for Mosek.

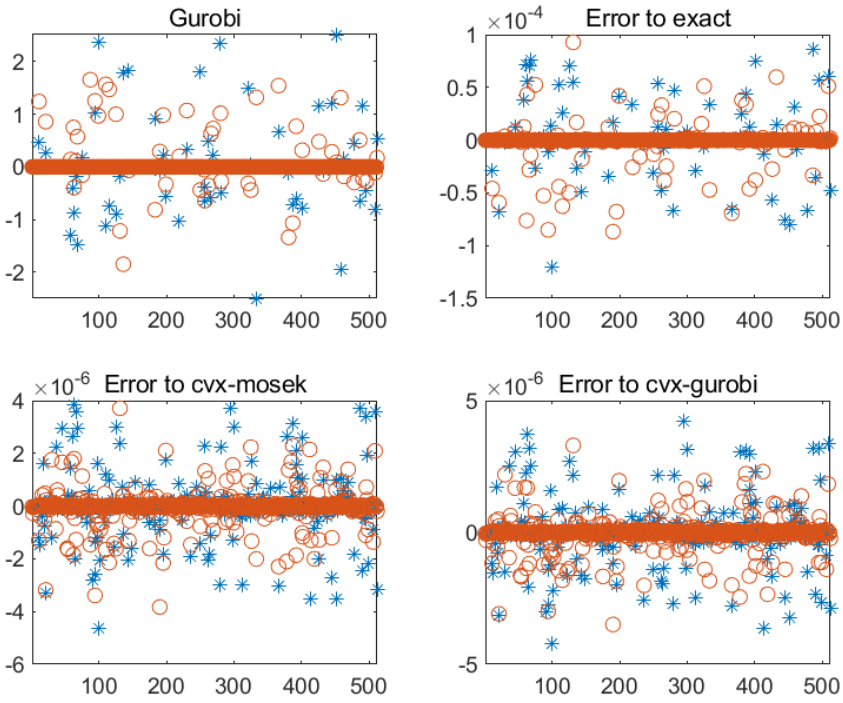


图 8: Error analysis for Gurobi.

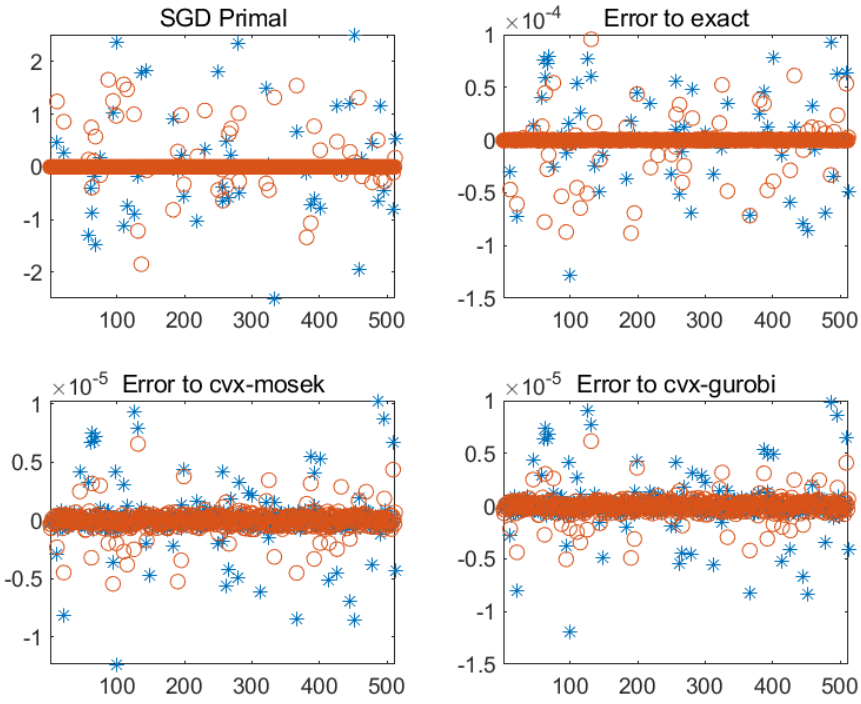


图 9: Error analysis for SGD Primal.

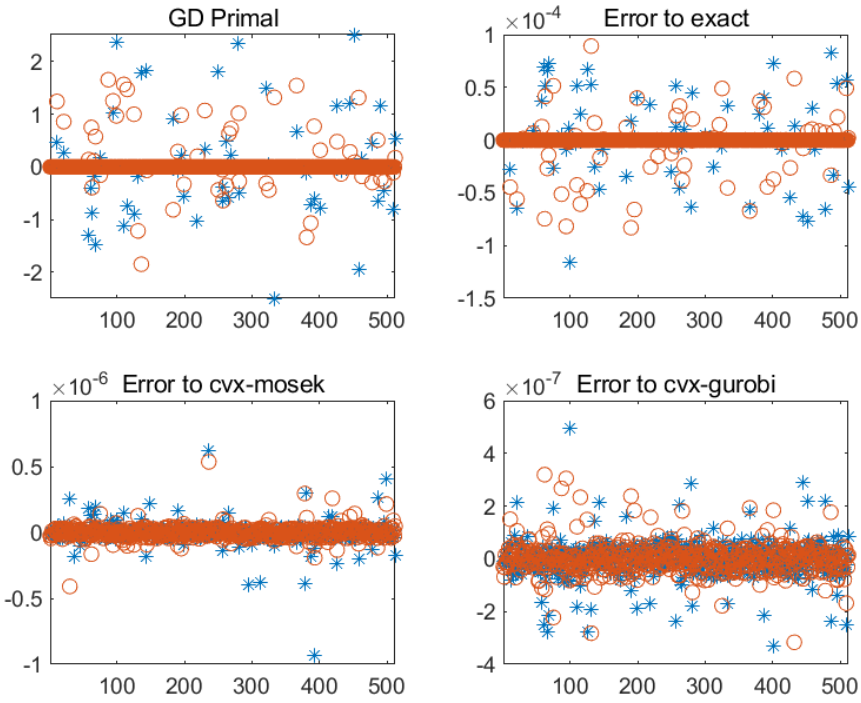


图 10: Error analysis for GD Primal.

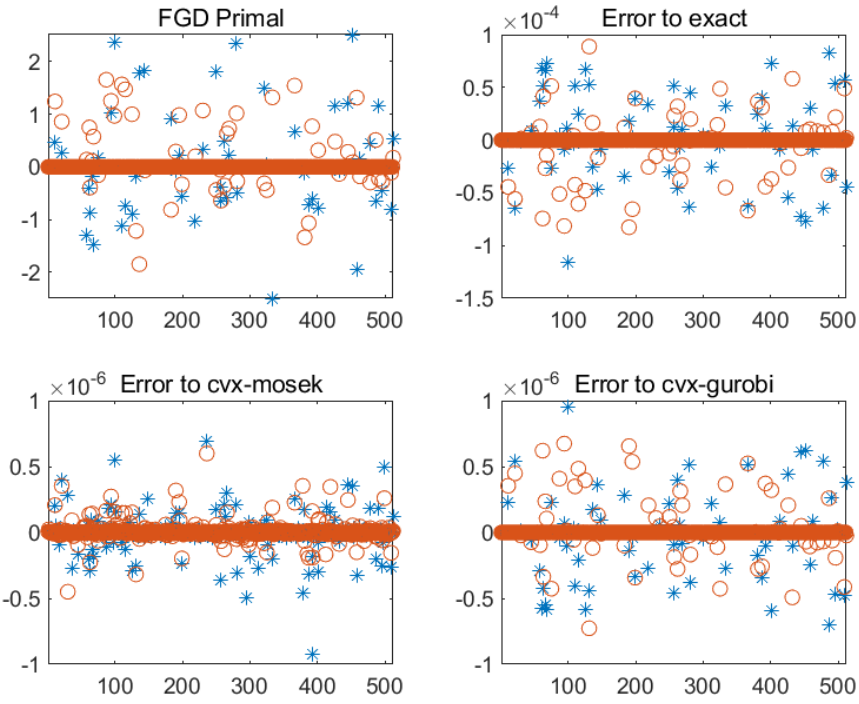


图 11: Error analysis for FGD Primal.

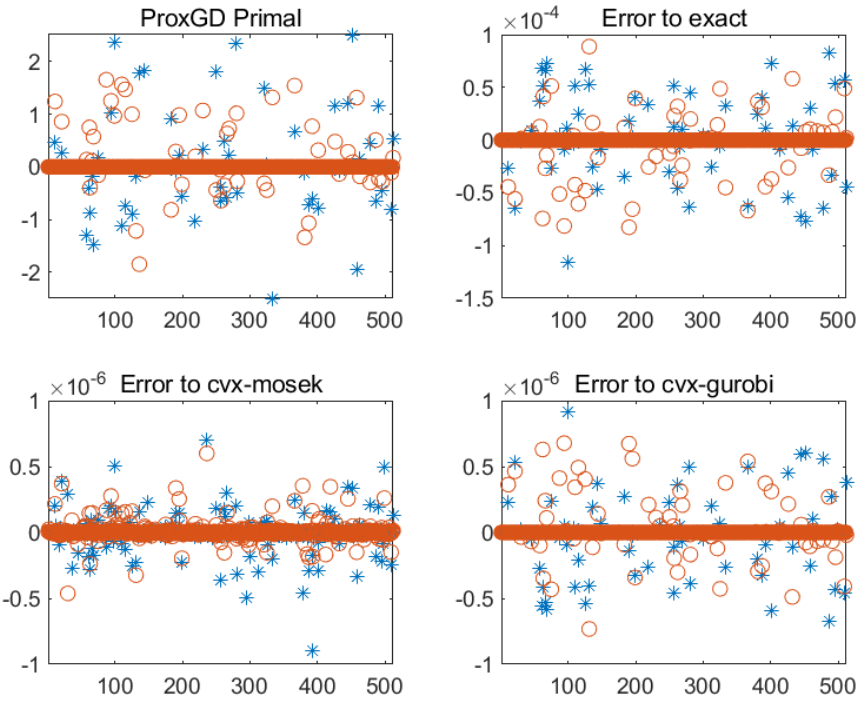


图 12: Error analysis for ProxGD Primal.

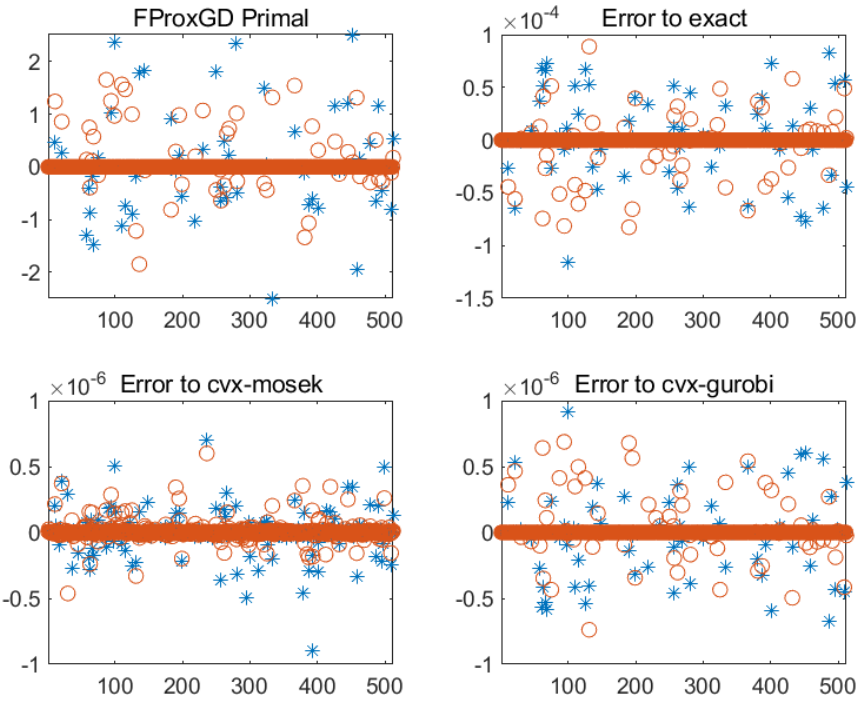


图 13: Error analysis for FProxGD Primal.

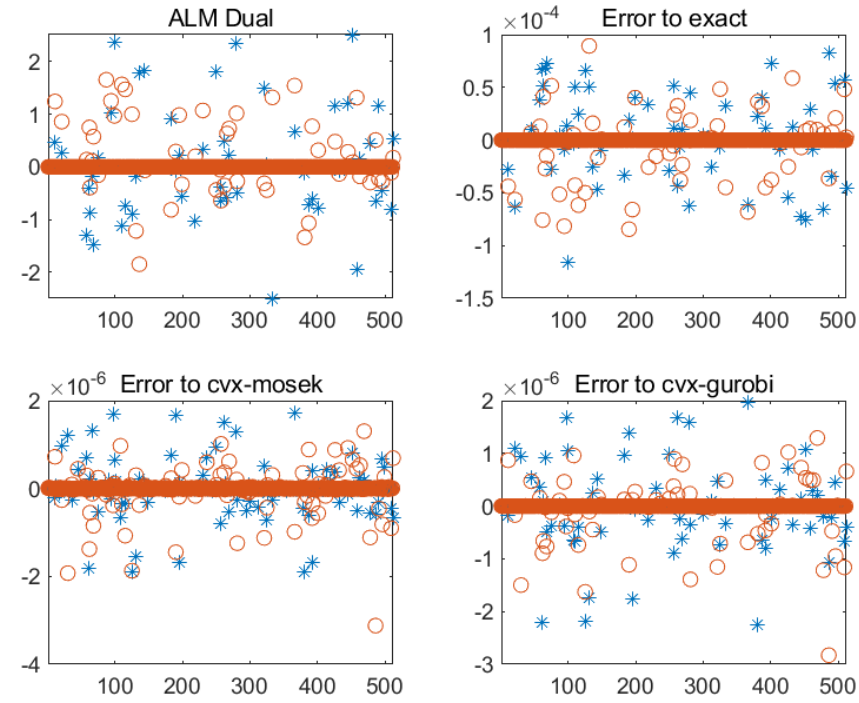


图 14: Error analysis for ALM Dual.

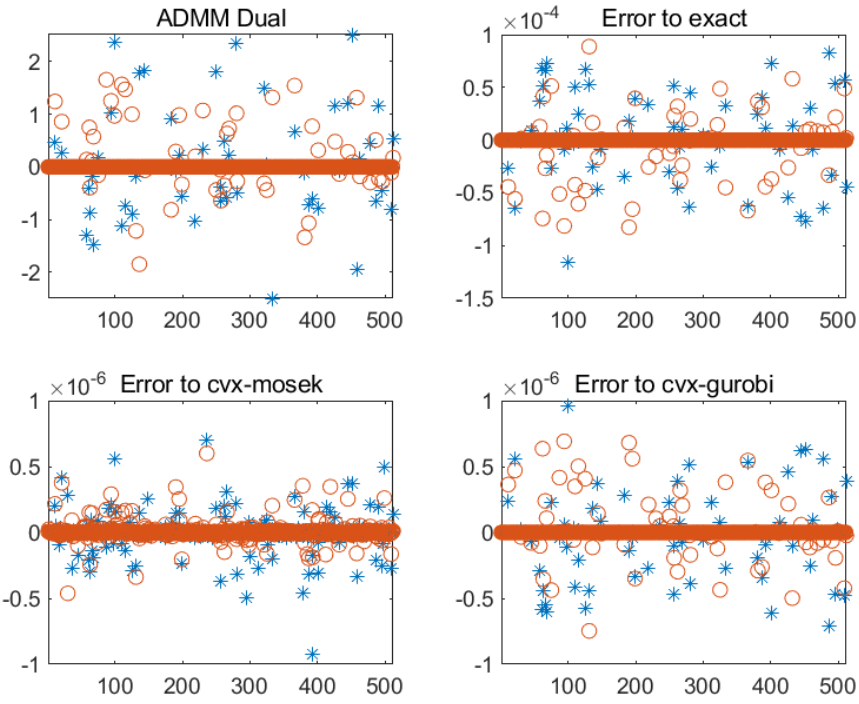


图 15: Error analysis for ADMM Dual.

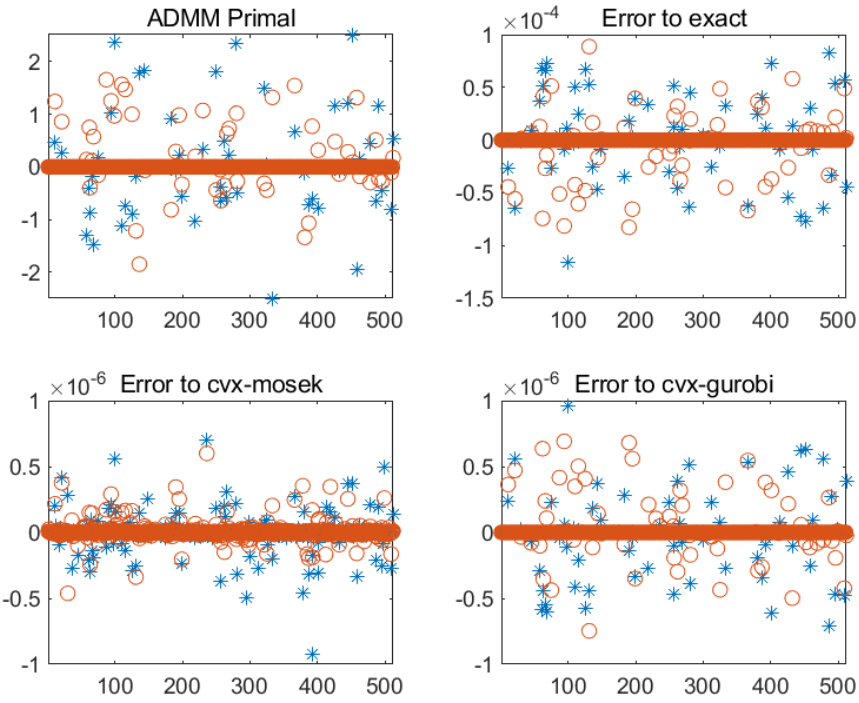


图 16: Error analysis for ADMM Primal.