北京大学

PEKING UNIVERSITY

上机报告

学　　院：数学科学学院

班　　级：硕士一班

姓　　名：谢地

学　　号：1801210098

2019~2020 第一学期

使用 LaTeX 撰写于 2019 年 11 月 30 日

目录

作业一 Algorithms for l_1 minimization

1.1 上机内容

考虑如下优化问题：

$$\min_x \quad \frac{1}{2}\|Ax-b\|_2^2+\mu\|x\|_1 \tag{1.1}$$

其中 $A\in\mathbb{R}^{m\times n}, b\in\mathbb{R}^m, \mu>0$ 是给定的.

1. 对光滑处理的问题(1.1)使用梯度法
2. 对光滑处理的问题(1.1)使用快速梯度法
3. 用近端梯度法求解问题(1.1)
4. 用快速近端度方法求解问题(1.1)

1.2 问题一

对原问题(1.1)进行光滑处理后使用梯度法 (Smoothed gradient method). 因为问题(1.1)的目标函数是不可导的, 而不可导项是 $\|x\|_1$, 因此一个比较自然的想法就是用一个光滑可导的函数去近似 $\|x\|_1$, 然后对这个近似的函数作极小化, 把这个极小值去近似原问题的极小值. 因为 $\|x\|_1=\sum_{i=1}^n|x_i|$, 因此我们只需要用一个连续可导的函数近似 $|x|$ 即可. 在这里我们使用 Huber 惩罚函数近似去近似 $|x|$, 即

$$\phi_\mu(x)=\begin{cases}\frac{x^2}{2\mu} & |x|\le\mu\\ |x|-\frac{\mu}{2} & |x|\ge\mu\end{cases}$$

对 $\phi_\mu(x)$ 求导有

$$\partial\phi=\begin{cases}x/\mu & |x|\le\mu\\ 1 & x\ge\mu\\ -1 & x\le-\mu\end{cases}$$

这个函数的图像如下图所示,

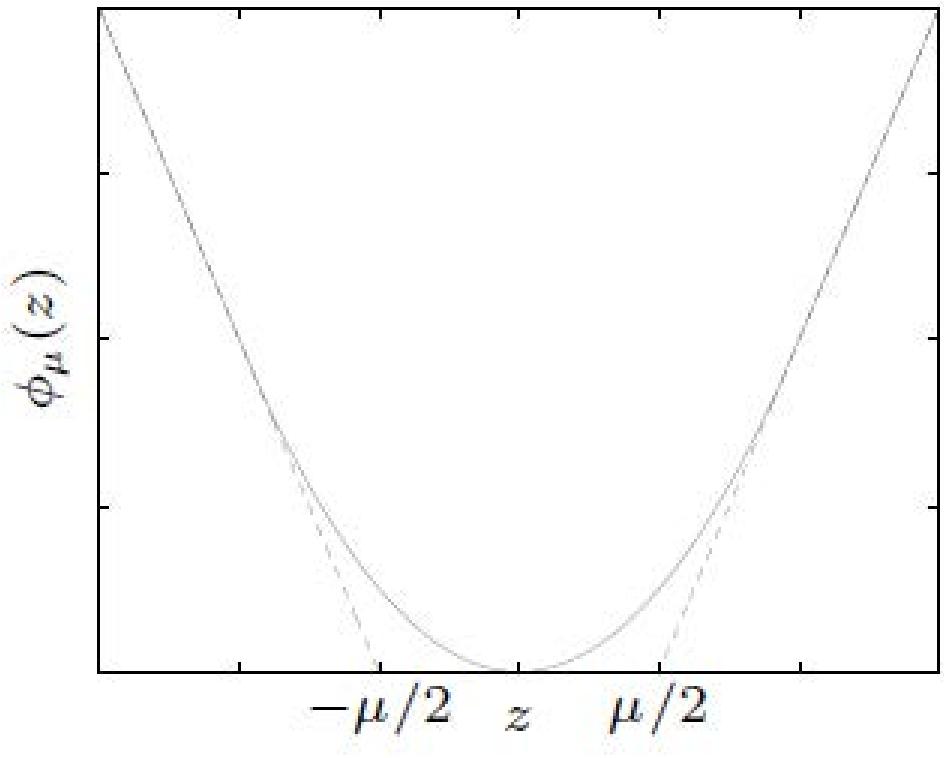

图 1.1: Huber 惩罚函数

可以看到这个函数是由两段连接起来的,μ 用来控制 $\phi_\mu(x)$ 的准确性和光滑性, 很显然 μ 越小则 $\phi_\mu(x)$ 与 $|x|$ 越接近, 但是在 $|x| \le \mu$ 内的光滑性会很差, 会导致梯度算法效果变差, 因此在实际使用中需要选取合适的 μ. 对问题(1.1), 我们取 $\mu = 1e-9$. 我们还可以发现,$\phi_\mu(x)$ 的导数与 $|x|$ 的次梯度很像, 当 μ 逐渐接近 0 时,$\phi_\mu(x)$ 的导数也与 $|x|$ 的次梯度越接近. 当我们选用 $\phi_\mu(x)$ 近似 $|x|$ 后, 则原问题(1.1)就可以用光滑的函数近似, 对这个近似函数用梯度法求解即可.

1.3 问题二

对原问题(1.1)使用光滑处理后的快速梯度法 (Fast smoothed gradient method). 实际上这个问题就是将加速算法运用到问题一中, 在这里我们使用 Nesterov 加速算法的最简单形式,Nesterov 加速是一个二步算法, 因此需要多储存一个迭代点. 具体地当有前两步迭代点 x^{k-1}, x^{k-2} 后, 取

$$y = x^{k-1} + \frac{k-2}{k-1}(x^{k-1} - x^{k-2})$$

作为迭代点, 得到的下一个迭代点记为 x^k, 即

$$x^k = y - t_k \nabla g(y)$$

其中 g 是光滑化后的函数的梯度. 这类加速算法可能不会单调收敛到极小, 但是收敛速度有很大的提高.

1.4 问题三

对原问题(1.1)使用近端梯度法 (Proximal gradient method). 考虑如下极小化问题

$$f(x) = g(x) + h(x)$$

其中 g 是凸的且可微的,h 是凸的且不可微的, 但是其 prox 算子是容易求解的, 近端梯度法的思想是在当前迭代点 $x^{(k)}$, 将 $g(x)$ 在 $x^{(k)}$ 出做线性近似并加一个二次惩罚项, 即

$$g(x) \approx g(x^{(k)}) + \nabla g(x^{(k)})^T (x - x^{(k)}) + \tfrac{1}{2t}\|x - x^{(k)}\|_2^2$$

其中 t 是惩罚因子, 然后将这个近似后的函数与 $h(x)$ 求和求极小点作为下一个迭代点, 即

$$\begin{aligned}
&= \arg\min_x \left(h(x) + g(x^{(k)}) + \nabla g(x^{(k)})^T (x - x^{(k)}) + \frac{1}{2t}\|x - x^{(k)}\|_2^2 \right) \\
&= \arg\min_x \left(h(x) + \frac{1}{2t}\|x - x^{(k)} + t\nabla g(x^{(k)})\|_2^2 \right) \\
&= \mathrm{prox}_{th}(x^{(k)} - t\nabla g(x^{(k)}))
\end{aligned}$$

其中

$$\mathrm{prox}_h(x) = \arg\min_u \left(h(u) + \tfrac{1}{2}\|u - x\|_2^2 \right)$$

称为凸函数 h 的 prox 算子. 可以看出这个迭代与最速下降法很像, 是在用最速下降法得到的迭代点再在 $th(x)$ 上作一个 prox 投影. 因此 t 也可以看作是迭代步长. 回到问题(1.1)上, 在这里我们有

$$g(x) = \frac{1}{2}\|Ax - b\|_2^2, \quad h(x) = \mu\|x\|_1$$

因此近端梯度法的迭代步骤为

$$x^{(k+1)} = \mathrm{prox}_{th}(x^{(k)} - t\nabla g(x^{(k)}))$$

而 $\nabla g(x) = A^T(Ax-b)$, 因此只需要知道 $th(x)$ 的 prox 算子, 问题就解决了. 由定义

$$\text{prox}_{th}(x) = \arg\min_u \left(th(u) + \tfrac{1}{2}\|u-x\|_2^2\right)$$

令 $l(u) = t\mu\|u\|_1 + \frac{1}{2}\|u-x\|_2^2$, 则有 $l(u)$ 的次梯度为

$$\partial l = t\mu\partial\|u\|_1 + u - x$$

u 为极小值的条件为 $0 \in \partial l(u)$. 若 $u_i > 0$, 则 $(\partial l(u))_i = \mu t + u_i - x_i = 0$, 所以有 $u_i = x_i - \mu t > 0$; 若 $u_i < 0$, 则 $(\partial l(u))_i = -\mu t + u_i - x_i = 0$, 所以有 $u_i = x_i + \mu t < 0$; 若 $u_i = 0$, 此时 x_i 可以为 $[-\mu t, \mu t]$ 之间的任意数. 因此有

$$\text{prox}_{th}(x)_i = \begin{cases} x_i - 1, & x_i > \mu t \\ 0, & -\mu t \le x_i \le \mu t \\ x_i + 1, & x_i < -\mu t \end{cases}$$

也称为软阈值 (soft-threshold) 算子. 得到 $\mu\|x\|_1$ 的 prox 算子后, 我们就可以用近端梯度法求解(1.1)了.

1.5 问题四

对原问题(1.1)使用快速近端度方法 (Fast proximal gradient method). 与问题二一样, 只需要将 Nesterov 加速算法用在近端梯度法上即可. 具体地, 当已知前两个迭代点 x^{k-1}, x^{k-2} 后 x^k 的计算方法为

$$y = x^{k-1} + \frac{k-2}{k+1}(x^{k-1} - x^(k-2))$$
$$x^k = \text{prox}_{t_k h}(y - t_k \nabla g(y))$$

其中

$$\text{prox}_h(x) = \arg\min_u \left(h(u) + \frac{1}{2}\|u-x\|_2^2\right)$$
$$g(x) = \frac{1}{2}\|Ax-b\|_2^2, \quad h(x) = \mu\|x\|_1$$

1.6 数值实验

在使用以上四种方法求解问题时, 与第一次上机实验一样, 我们对 μ 从比较高的值逐渐递减到 $\mu = 0.001$, 这样能让算法更快地收敛到极小值. 比如在第一次上机实验中, 取 μ 从 100 到 0.001, 每次变化缩小十倍, 也就是说一共改变六次.

为了能更好地比较这几种方法的优劣性, 将第一次上机实验中的投影梯度法和次梯度方法也放入到这次的实验中与 CVX 的 mosek 求解器进行比较, 因为在第一次上机实验中我们有 gurobi 求解器与 mosek 的解误差有 10^{-6} 的量级, 因此在这里我们对六种方法进行调参, 在达到相同量级精度下与 mosek 的运行时间的比较, 以及六种方法中迭代次数等一些其他值的比较.

需要调节的参数有初始 μ 值、固定迭代步长 t、算法终止的精度 eps 和每个 μ 值的最大迭代次数 Max. 当迭代次数达到设置的最大值或者两次迭代的函数值的差小于设置的精度 eps 时, 即迭代已经不能使得函数值有明显下降时, 将 μ 取向下一个值, 直到 $\mu = 0.001$. 具体地, 六种方法的 Max = 300, 其他三个参数值见下表

表 1.1: 六种方法的参数设置

Parameter	PGD	SGD	GD	FGD	ProxGD	FProxGD
μ	100	100	100	1	10	1
t	4.5e-4	4e-4	4e-4	4e-4	7e-4	3e-4
eps	1e-9	1e-10	1e-10	1e-10	1e-9	1e-9

调整好参数后, 我们对问题一到问题四, 以及第一次上机实验的六种方法比较它们的函数值、运行时间、精度、恢复效果、稀疏性以及迭代次数. 其中精度为各种方法的解与直接调用 CVX 的 mosek 求解器得到的解的误差, 恢复效果用 $\|Ax - b\|_2$ 衡量, 稀疏性用 $\|x\|_1$ 衡量. 得到的结果如下

表 1.2: 六种方法的比较

Solver	Fval	Time	Errfun	Iter	Sparisity	recovery
cvx_mosek	8.0877e-2	1.24	–	–	80.8764	5.2e-04
PGD	8.0876e-2	0.31	1.50e-06	1577	80.8762	5.6e-04
SGD	8.0877e-2	0.36	1.27e-06	1662	80.8764	6.5e-04
GD	8.0877e-2	0.31	6.92e-06	1623	80.8765	7.7e-04
FGD	8.0877e-2	0.17	2.91e-06	1012	80.8768	9.3e-04
ProxGD	8.0877e-2	0.18	1.83e-06	1059	80.8763	5.4e-04
FProxGD	8.0876e-2	0.12	2.62e-06	772	80.8763	4.7e-04

可以看到, 每种方法都可以收敛到极小值, 因此这六种方法是有效的. 下面从各个指标上对六种方法进行比较.

在最优值上, 显然函数值越小对问题(1.1)的求解效果越好. 在与 CVX 的 mosek 求解器的解的误差精度达到 10^{-6} 的条件下, 六种方法得到的最优值都很接近. 其中 PGD 和 FProxGD 方法的解相对好一些.

在运行时间上, 六种方法都要比直接使用求解器要快很多, 其中前三种方法大致在 0.3 秒多, 而后三种方法可以控制在 0.2 秒以内. 这在迭代次数上也能看出, 因为这几种方法每次迭代的运算量是差不多的, 因此迭代次数也可以间接决定运行的时间. 明显后三种方法的迭代次数要比前三种方法要少很多. 特别地, 我们比较使用快速算法与不使用快速算法的运行速度, 不论是 GD 与 FGD, 还是 ProxGD 与 FProxGD. 加速算法确实起到了加速的效果, 并且大致提高了 30% 的速度.

在稀疏度上,$\|x\|_1$ 越小说明越稀疏. 表1.2表明了六种方法的稀疏度基本一致, 如果将很接近 0 的 x_i, 例如 $|x_i| < 10^{-5}$ 认为是 0 的话, 则 x 的稀疏度, 即 $|x|_0$ 应该是相等的.

在恢复效果上,$\|Ax - b\|_2$ 越小说明恢复效果越好.ProxGD 和 FProxGD 的效果要比较好一些,GD 和 FGD 的效果要差一些, 可能是光滑化的的原因使得求解的问题与原问题(1.1)还是有一点差异.

总的来说, 不使用加速方法的话, 近端梯度法要比其他几种方法的效果要好, 具体快的原因是近端梯度法可以取比较大的步长 t, 而且相对较小的初始值 μ 也使得近端梯度法收敛的比较好的精度, 然后加速算法可以在其他方法的基础上再进行一个改进, 使得算法的效果更好. 虽然使用加速算法的近端梯度法的步长不能取的和之前一样大, 但是有依然有很快地收敛速度, 而且初始 μ 值可以更小. 最后我们观察一下 GD 与 FGD,ProxGD 与 FProxGD 的相对函数值随着迭代次数的变化曲线, 其中相对函数值定义为

$$y = \frac{f - f^*}{|f^*|}$$

其中 f 为算法的每次迭代的目标函数值,f^* 为使用 CVX 的 mosek 求解器得到的最优值. 因为 y 的值变化比较大, 所以在这里对其取对数, 即用 $\log_{10} y$ 来观察函数值的变化情况, 因为这里使用 CVX 的 mosek 求解器的最优值作为极小值 f^*, 而这个值显然不是精确的极小值, 所以到迭代的后期难免会有 f 比 f^* 小的情况, 这时取对数的话就会出错, 会出现负数, 因此 matlab 会出现警告, 在这里我们就按 matlab 中计算的结果, 即忽略虚部. 得到的结果如下

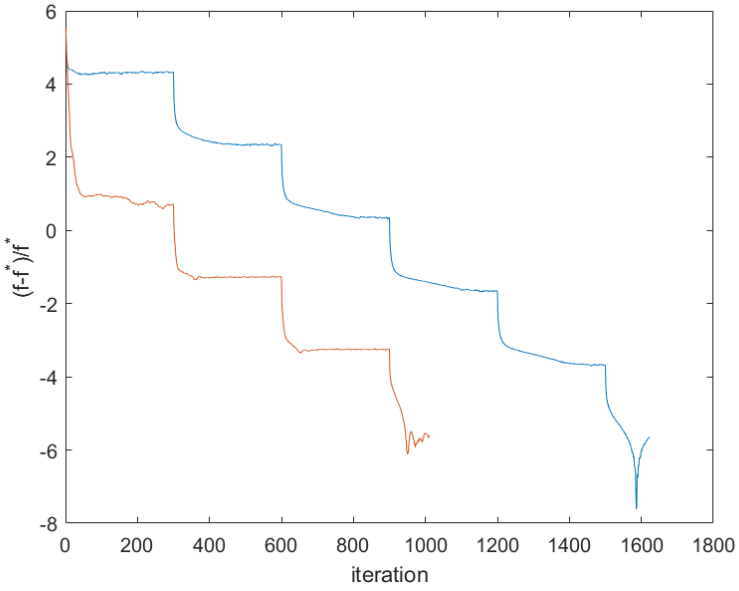

图 1.2: GD 与 FGD 的变化

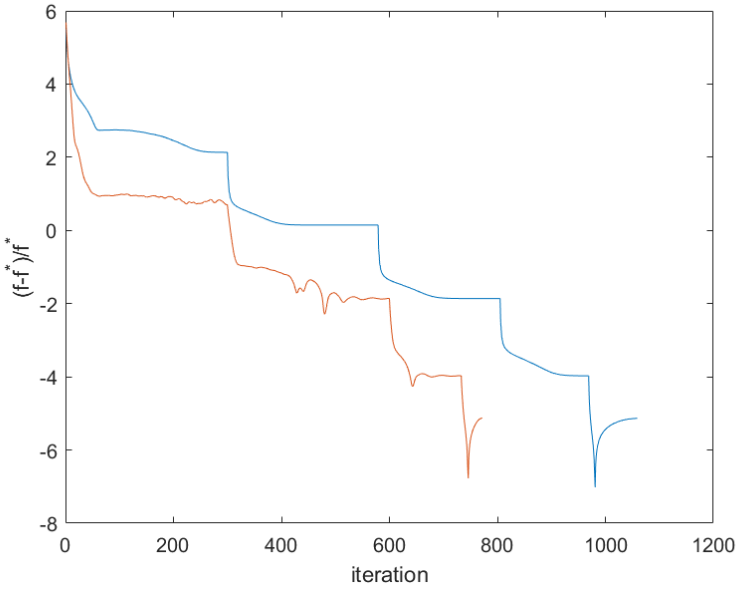

图 1.3: ProxGD 与 FProxGD 的变化

其中红色线表示使用加速算法, 蓝色线表示未使用加速算法. 每一个快速的下降的部分代表着 μ 的值发生改变了, 平坦的部分表示在当前 μ 值下下降情况已经不明显了. 可以看到加速算法在每个方法中确实起到了很大的作用.

实验总结

因为有了第一次上机作业的经验, 这次的上机作业相对来说难度小了很多. 因为所有方法的结构都是一样的,μ 的取值、最大迭代次数、算法结束的精度都是相似的, 不同的只是导数的计算、使不使用加速算法而已. 在算法编程上没有什么很困难的地方.

对于这次上机实验新使用的两类方法, 光滑化的处理方法思想比较简单, 就是用一个连续可导的函数去近似一个不可导的函数, 但是这种光滑化的函数的选取也要据问题而改变. 近端梯度法是我这次收获很大的地方, 近端梯度法将目标函数线性化并加入惩罚项从而引入 proximal 算子, 通过计算 proximal 算子来计算迭代点, 这种方法与其他的类似梯度的方法略有不同, 而且近端梯度法也是这几种方法中效果最好的. 而在这次上机中使用的 Nesterov 加速算法, 它在编程中其实是很容易实现的, 仅仅增加了一个迭代点的存储, 却使得算法的效果得到很大的提升, 是一个很好用的方法.

这次的上机实验用一个简单的优化问题对几种优化方法进行了一个简单的比较, 通过这次实验对这几种方法的效果有了初步的认识.