Lasso 优化算法的探讨和比较

作者：组长：李宗翰，组员：蒋佩禧，袁浩然

Introduction

本文主要讨论 lasso 相关问题，着重介绍了近端梯度算法，在计算方面先是调用 CVX 程序包计算出结果，之后分别近端梯度算法（Proximal Gradient Method），加速近端梯度算法，ADMM（交替方向乘子算法）计算出结果，并对三种运算方式进行比较.

第一章 Lasso

1.1 引言

对于线性逆问题 $Y=Ax+w$，其中 $A\in mn$,$Y\in m$，且已知，w 是未知噪声，那么它可以用最小二乘法求解：$\hat{x}=\operatorname{argmin}_x\|Ax-y\|_2^2$ 且当 m=n，且 A 非奇异时，解为 $\hat{x}=\mathrm{A}^{-1}*y$，但是在大多数情况下，A 是病态的，所以用最小二乘法求解时，微小的差别会给结果带来巨大的差异

为了计算这种情况下的解，1996 年 Robert Tibshirani 提出了 lasso 回归，lasso 回归通过构造一个 l1 正则化的惩罚项，使得某些系数变小，适合参数的缩减与选择，且对异常值不敏感，在解决上述问题时有着较大的优势，如下

$$\hat{x}=\operatorname{argmin}_x\|Ax-y\|_2^2+\gamma\|\ x|_1$$

在本文中不讨论解 x ，而主要讨论 $\min\frac{1}{2}||Ax-y||_2^2+\gamma||x|_1$，对于这个问题本文采用 matlab 的 cvx 程序包计算出结果，之后采用近端梯度算法，加速近端梯度算法，ADMM 交替方向乘子算法）分别计算出结果，并比较它们之间的差异

1.2 算法

1.2.1 cvx 包

```
cvx_begin quiet
cvx_precision low
variable x(n)
minimize(0.5*sum_square(A*x - b) + gamma*norm(x,1))
cvx_end
```

1.2.2 近端梯度算法:

$$x^{k+1}:=\operatorname{prox}_{\lambda^k\gamma\|\cdot\|_1}\left(x^k-\lambda^kA^T\left(Ax^k-b\right)\right)$$

我们进行拆分：

$$f(x)=(1/2)\|Ax-b\|_2^2,\quad g(x)=\gamma\|x\|_1$$

引入近端和梯度的运算符：

$$\nabla f(x)=A^T(Ax-b),\quad \operatorname{prox}_{\gamma g}(x)=S_\gamma(x)$$

$$f(z)\approx f\left(x^{(k)}\right)+\nabla f\left(x^{(k)}\right)^T\left(z-x^{(k)}\right)+\frac{1}{2\lambda}||z-x^{(k)}||_2^2$$

$$
\begin{aligned}
x^{(k+1)} &= \operatorname{argmin}_z (f(z) + g(z)) \\
&= \operatorname{argmin}_z \left(f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)^T \left(z - x^{(k)}\right) + \frac{1}{2\lambda} ||z - x^{(k)}||_2^2 + g(z) \right) \\
&= \operatorname{argmin}_z \left(\frac{1}{2\lambda} ||z - \left(x - \lambda \nabla f\left(x^{(k)}\right)\right)||_2^2 + g(z) \right) \\
&= \operatorname{prox}_{g,y}\left(x^{(k)} - \lambda \nabla f\left(x^{(k)}\right)\right) \\
&= \operatorname{prox}_{g,y}\left(x^{(k)} - \lambda A^{\tau}\left(Ax^{(k)} - b\right)\right)
\end{aligned}
$$

其中前端函数 prox $_{g.\gamma}(x) = S_\gamma(x)$, 可以由软阈值算法求出因为 $f\left(x^{(k+1)}\right) \leq f\left(x^{(k)}\right)$, 所以求出的 $x^{(k+1)}$ 向最小值走了一步，依次迭代下去最终 $f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right)$ 符合精度要求时输出结果

Algorithm 1 Proximal gradient method

Input: x^k, λ^{k-1},

1: parameter $\beta \in (0,1)$
2: Let $\lambda := \lambda^{k-1}$
3: **repeat**
4:     Let $z := \operatorname{prox}_{\lambda g}\left(x^k - \lambda \nabla f\left(x^k\right)\right)$;
5:     break if $f(z) \leq \hat{f}_\lambda\left(z, x^k\right)$;
6:     Update $\lambda := \beta\lambda$
7: **return** $\lambda^k := \lambda, x^{k+1} := z$

```
for k = 1:MAX_ITER
while 1
mal_gra = AtA*x - Atb;
z = l1(x - lam*mal_gra, lam*gamma);
if f(z) <= f(x) + mal_gra'*(z - x) + (1/(2*lam))*sum_square(z - x)
break;
end
lam = Be*lam;
end
xpr = x;
x = z;
h.opt(k) = obj(A, b, gamma, x, x);
if k > 1 && abs(h.opt(k) - h.opt(k-1)) < ABS
break;
end
end
```

1.2.3 加速近端梯度算法

在近端梯度算法的基础上，更新策略如下：method:

$$
\begin{aligned}
y^{(k+1)} &= x^{(k)} + \frac{k}{k+3}\left(x^{(k)} - x^{(k-1)}\right) \\
x^{k+1} &= \operatorname{prox}_{g,y}\left(y^{(k+1)} - \lambda \nabla f\left(y^{(k+1)}\right)\right)
\end{aligned}
$$

works for $\omega^k = k/(k+3)$ and similar line search as before - faster $O\left(1/k^2\right)$ convergence rate, originated with Nesterov (1983)

```
y = x + (k/(k+3))*(x - xpr);
```

Algorithm 2 加速近端梯度算法

Input: y^k, λ^{k-1},
1: parameter $\beta \in (0,1)$
2: Let $\lambda := \lambda^{k-1}$
3: **repeat**
4: Let $z := \mathrm{prox}_{\lambda g}\left(y^k - \lambda \nabla f\left(y^k\right)\right)$;
5: break if $f(z) \leq \hat{f}_\lambda\left(z, y^k\right)$;
6: Update $\lambda := \beta\lambda$
7: **return** $\lambda^k := \lambda, x^{k+1} := z$

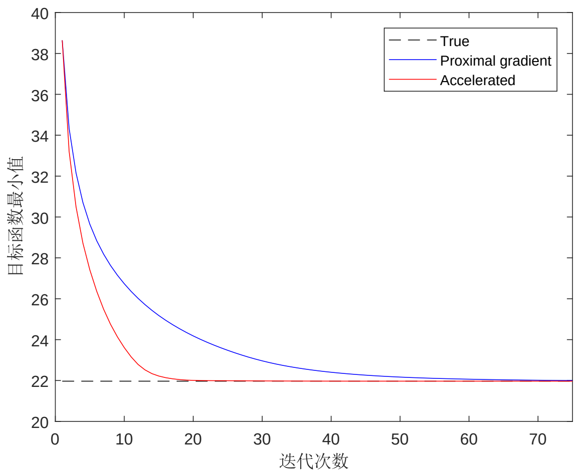

图 1.1: 近端梯度算法 vs 加速近端梯度算法

1.2.4 ADMM:

$$
\begin{aligned}
x^{k+1} &:= \left(I + \lambda A^T A\right)^{-1}\left(z^k - u^k - \lambda A^T b\right) \\
z^{k+1} &:= \mathrm{prox}_{\lambda\gamma\|\cdot\|_1}\left(x^{k+1} + u^k\right) \\
u^{k+1} &:= u^k + x^{k+1} - z^{k+1}
\end{aligned}
$$

1.3 数值实验

计算平台：IntelCore i5-8265 CPU @1.60GHz 8.00GB RAM Windows10

例 1.1 参数设置：A=500*2500,b=A*x0+v,

其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机

生成的数组

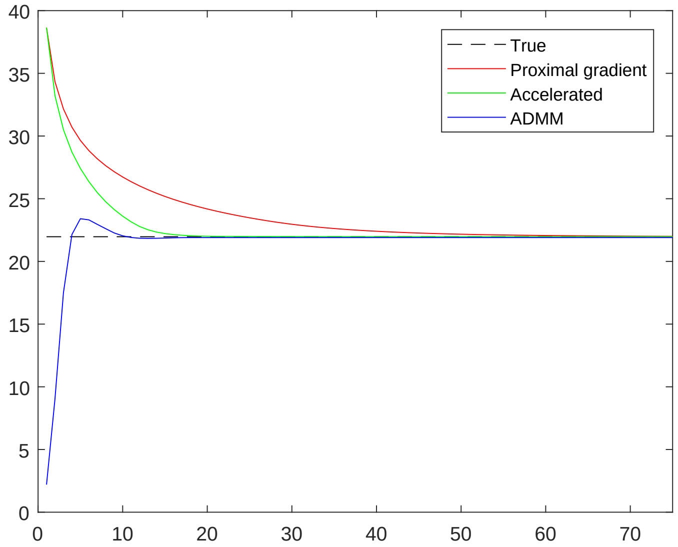

图 **1.2:** Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^{\star}$	Error (abs)	Error (rel)
CVX	15	26.53	16.5822	–	–
Proximal gradient	127	0.72	16.5835	0.09	0.01
Accelerated	23	0.15	16.6006	0.26	1.89
ADMM	20	0.07	16.6011	0.18	1.90

例 1.2 参数设置：A=1000*3500,b=A*x0+v,
其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机生成的数组

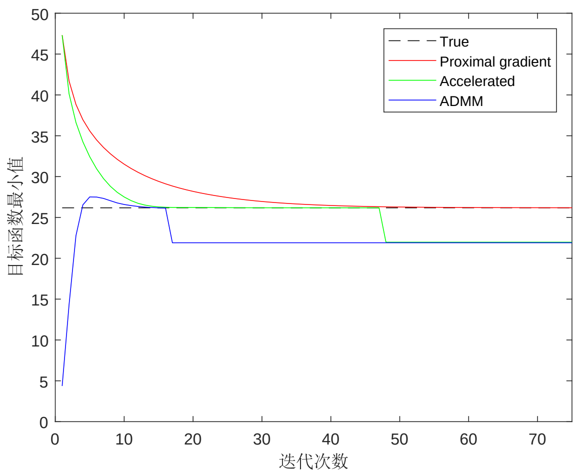

图 **1.3:** Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^\star$	Error (abs)	Error (rel)
CVX	15	144.0169	23.1788	–	–
Proximal gradient	70	0.9906	23.5835	0.05	0.01
Accelerated	50	0.15	26.1053	3.16	1.54
ADMM	27	0.07	26.5721	.115	1.53

在这个计算中,近端梯度算法迭代较慢,但迭代足够多的次数后,准确性较高,ADMM和加速近端梯度算法虽然迭代次数较少，但得出的低于真实目标函数的值

例 1.3 参数设置：A=600*2000,b=A*x0+v,
其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机生成的数组

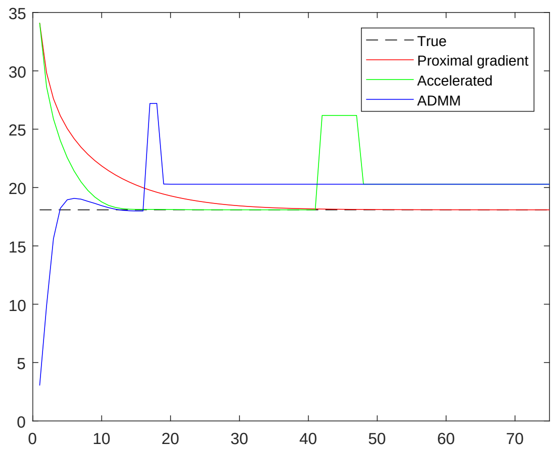

图 1.4: Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^\star$	Error (abs)	Error (rel)
CVX	15	27.5314	18.5822	–	–
Proximal gradient	100	0.2774	18.5835	0.07	0.01
Accelerated	59	0.1559	20.6006	2.26	1.5
ADMM	49	0.0589	20.6011	2.18	1.43

在这个计算中,近端梯度算法迭代较慢,但迭代足够多的次数后,准确性较高,ADMM和加速近端梯度算法虽然迭代次数较少，但得出的值高于真实目标函数的值

1.4 总结

本文通过三种方式计算 lasso 问题，并对这三种算法进行介绍，且着重介绍了近端梯度算法及其加速，我们利用数值实验计算来比较三个算法的优缺点。
近端梯度算法迭代较慢，但迭代足够多的次数后，准确性较高，ADMM 和加速近端梯度算法虽然迭代次数较少，但显示出了较大的误差
将三种算法集为一体并进行比较是本文的一大优点，但是由于自身知识面所限，难免会

有所错漏，有待改进。