近似点梯度法(续)

王 尧

西安交通大学智能决策与机器学习中心
(Email: yao.s.wang@gmail.com)

2022. 5

岭(Ridge)回归

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda\|\beta\|_2^2$$

- $\lambda \geq 0$ 为调控参数，控制L2正则项的作用：
  - $\lambda = 0$ 给出最小二乘估计
  - $\lambda = \infty$ 给出 $\hat{\beta}^{\text{ridge}} = 0$
  - 中间大小的λ有收缩系数的作用，通常交叉验证确定

直接的模型选择策略

- 随机将所有数据样本分为三份，比例通常取为为50%、25%与25%
- 50%的样本作为训练集用来训练模型，25%的样本作为验证集用来选择“好”的正则参数(验证误差最小)
- 在最后的25%的样本测试得到的模型的性能
- 不适合处理样本个数过少的情形(如Prostate Cancer数据集只有97个样本)

K倍交叉验证

1	2	3	4	5
Train	Train	Validation	Train	Train

- 这里将数据大致5等分，即$K=5$
- 给定一个正则参数，我们把标号为1，2，4，5的样本集做为训练集，标号为3的样本集做为验证集
- 如此轮换下去，直到5个标签的样本集合均被用作为验证集
- 更新一个正则参数，继续上述过程

K倍交叉验证

- Divide the set $\{1, \ldots n\}$ into K subsets (i.e., folds) of roughly equal size, $F_1, \ldots F_K$
- For $k = 1, \ldots K$:
  - Consider training on (x_i, y_i), $i \notin F_k$, and validating on (x_i, y_i), $i \in F_k$
  - For each value of the tuning parameter $\theta \in \{\theta_1, \ldots \theta_m\}$, compute the estimate $\hat{f}_\theta^{-k}$ on the training set, and record the total error on the validation set:

$$e_k(\theta) = \sum_{i \in F_k} \left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2$$

- For each tuning parameter value θ, compute the average error over all folds,

$$\mathrm{CV}(\theta) = \frac{1}{n} \sum_{k=1}^{K} e_k(\theta) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in F_k} \left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2$$

此页Slide来自于Ryan Tibshirani的notes

回顾：近似点梯度下降

Proximal gradient descent: choose initialize $x^{(0)}$, repeat:

$$x^{(k)} = \mathrm{prox}_{h,t_k}\big(x^{(k-1)} - t_k \nabla g(x^{(k-1)})\big), \quad k = 1, 2, 3, \ldots$$

To make this update step look familiar, can rewrite it as

$$x^{(k)} = x^{(k-1)} - t_k \cdot G_{t_k}(x^{(k-1)})$$

where G_t is the generalized gradient of f,

$$G_t(x) = \frac{x - \mathrm{prox}_{h,t}\big(x - t\nabla g(x)\big)}{t}$$

If we can't evaluate prox

Theory for proximal gradient, with $f = g + h$, assumes that prox function can be evaluated, i.e., assumes the minimization

$$\text{prox}_t(x) = \underset{z}{\text{argmin}} \ \frac{1}{2t}\|x - z\|_2^2 + h(z)$$

can be done exactly. In general, not clear what happens if we just minimize this approximately

But, if you can precisely control the errors in approximating the prox operator, then you can recover the original convergence rates

(近似点)梯度法收敛速率

Iteration complexities of (proximal) gradient methods

- strongly convex and smooth problems

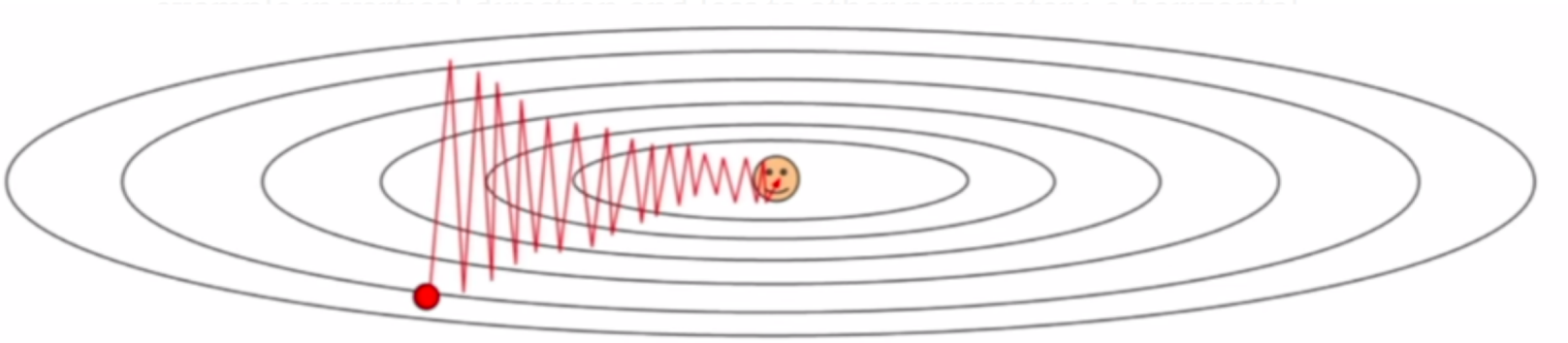
$$O(\log \tfrac{1}{\varepsilon})$$

- convex and smooth problems

$$O\left(\frac{1}{\varepsilon}\right)$$

Can one still hope to further accelerate convergence?

Zigzagging现象



Lost function is very sensitive to changes in one of the parameter for example in vertical direction and less to other parameter i.e horizontal direction.

以梯度法为例进行说明

引入动量项

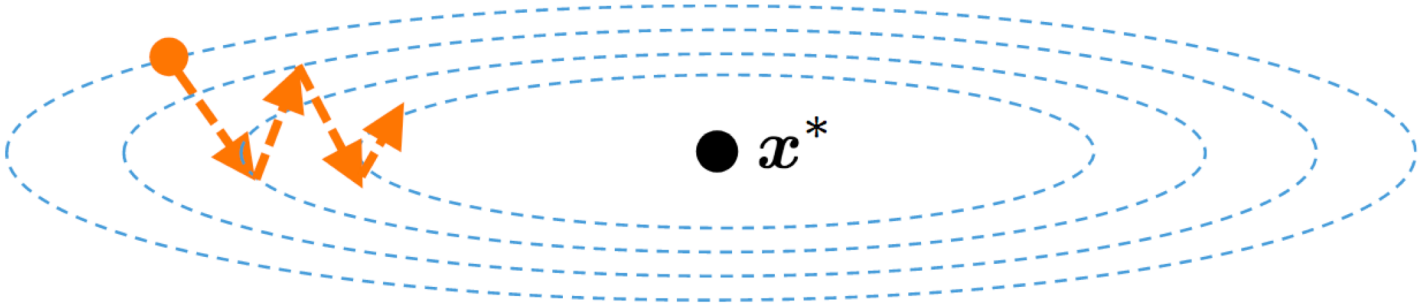
B. Polyak

$$\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x})$$

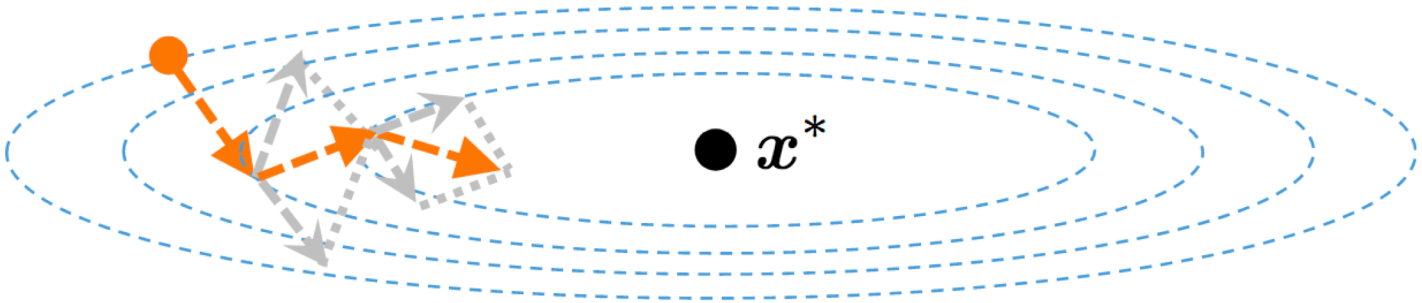
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta_t \nabla f(\boldsymbol{x}^t) + \underbrace{\theta_t(\boldsymbol{x}^t - \boldsymbol{x}^{t-1})}_{\text{momentum term}}$$

exploit information from the history (i.e. past iterates)

引入动量项



gradient descent



动量梯度下降

Nestrov加速策略

As before, consider:

$$\min_x \; g(x) + h(x)$$

where g convex, differentiable, and h convex. Accelerated proximal gradient method: choose initial point $x^{(0)} = x^{(-1)} \in \mathbb{R}^n$, repeat:

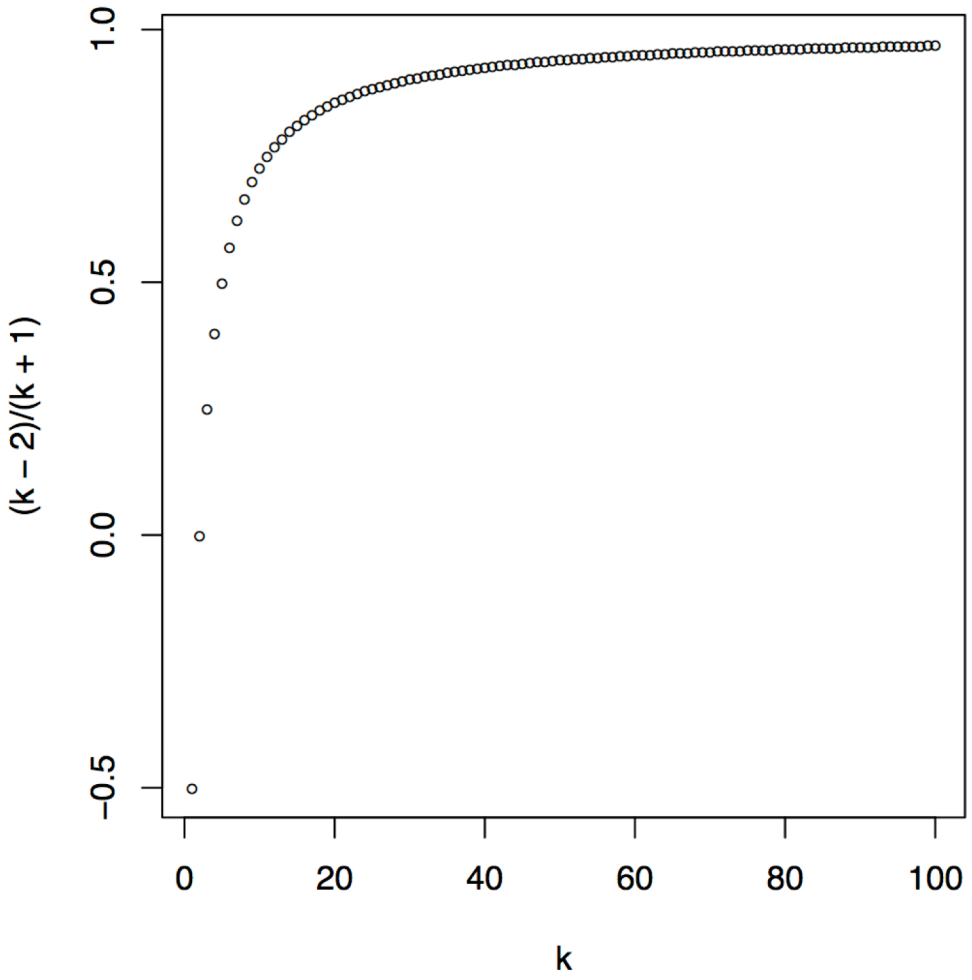
$$v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$$
$$x^{(k)} = \text{prox}_{t_k}\big(v - t_k \nabla g(v)\big)$$

for $k = 1, 2, 3, \ldots$

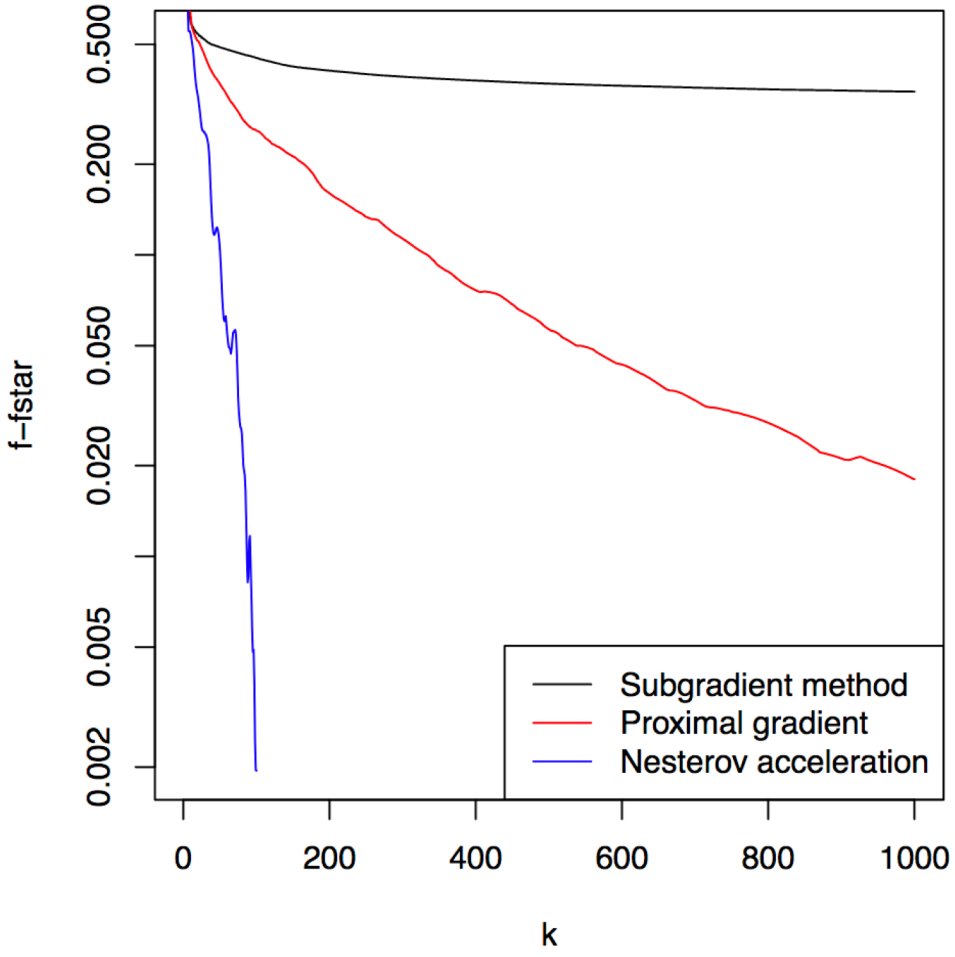
- First step $k = 1$ is just usual proximal gradient update
- After that, $v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$ carries some "momentum" from previous iterations
- When $h = 0$ we get accelerated gradient method

第一步沿着前两步的计算方向计算一个新点，第二步在该新点处做一步(近似点)梯度迭代

Momentum weights:



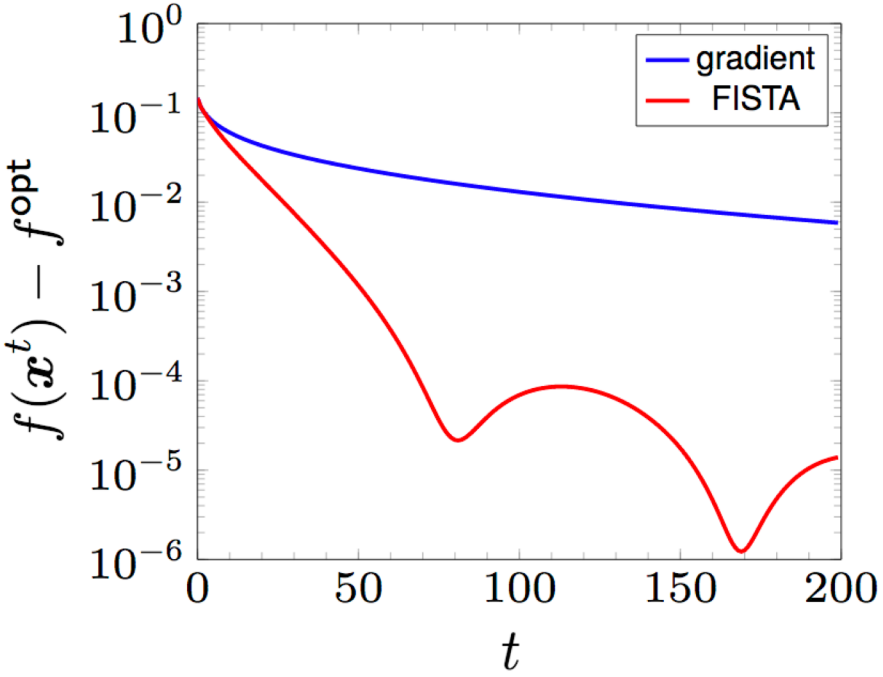
Back to lasso example: acceleration can really help!



taken from UCLA EE236C

$$\text{minimize}_{\boldsymbol{x}} \quad \log\left(\sum_{i=1}^{m}\exp(\boldsymbol{a}_i^\top \boldsymbol{x}+b_i)\right)$$

with randomly generated problems and $m = 2000$, $n = 1000$



注意：accelerated proximal gradient is not a descent method

收敛性

For criterion $f(x) = g(x) + h(x)$, we assume as before:

- g is convex, differentiable, $\mathrm{dom}(g) = \mathbb{R}^n$, and ∇g is Lipschitz continuous with constant $L > 0$
- h is convex, $\mathrm{prox}_t(x) = \mathrm{argmin}_z \{\|x - z\|_2^2/(2t) + h(z)\}$ can be evaluated

Theorem: Accelerated proximal gradient method with fixed step size $t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^\star \leq \frac{2\|x^{(0)} - x^\star\|_2^2}{t(k+1)^2}$$

and same result holds for backtracking, with t replaced by β/L

Achieves optimal rate $O(1/k^2)$ or $O(1/\sqrt{\epsilon})$ for first-order methods

FISTA

Back to lasso problem:

$$\min_{\beta} \ \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Recall ISTA (Iterative Soft-thresholding Algorithm):

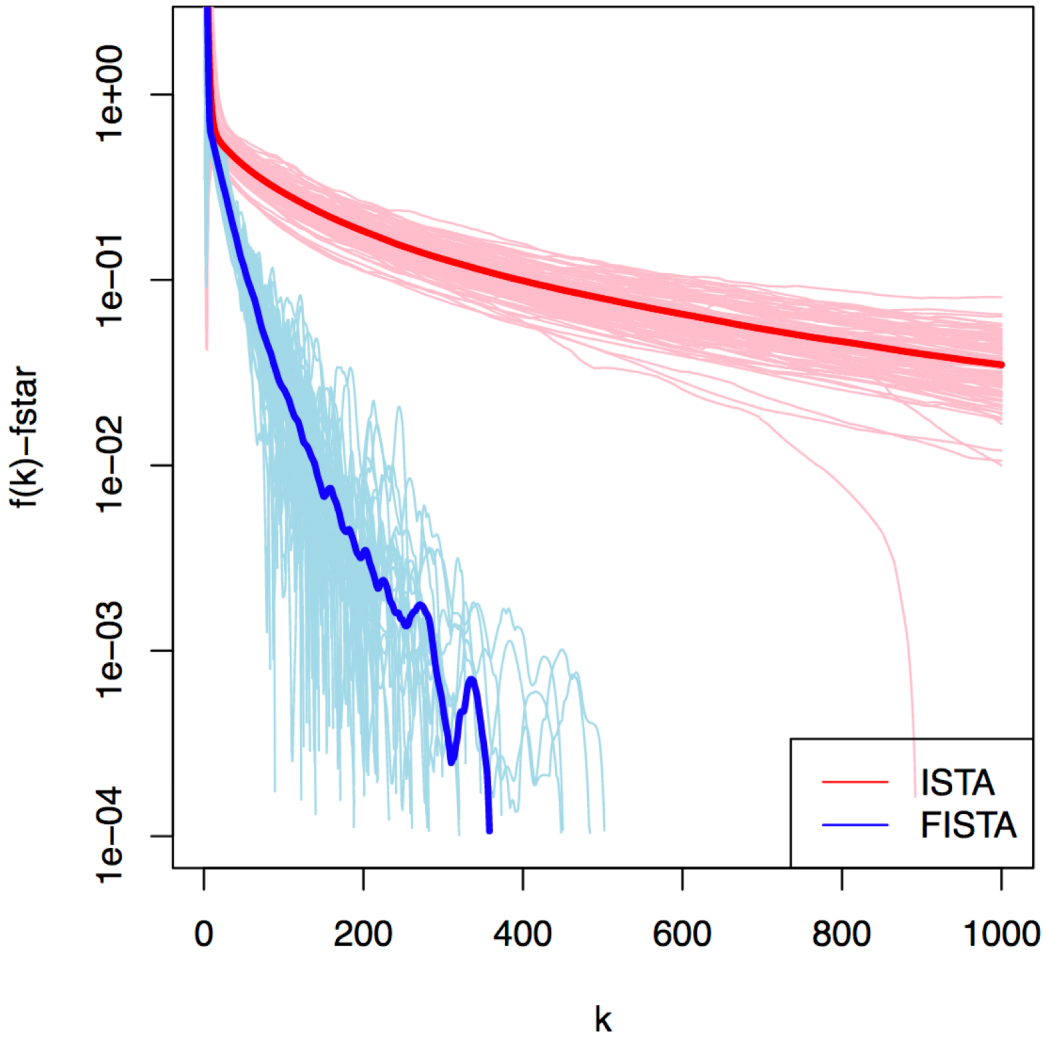
$$\beta^{(k)} = S_{\lambda t_k}(\beta^{(k-1)} + t_k X^T(y - X\beta^{(k-1)})), \quad k = 1, 2, 3, \ldots$$

$S_\lambda(\cdot)$ being vector soft-thresholding. Applying acceleration gives us FISTA (F is for Fast):

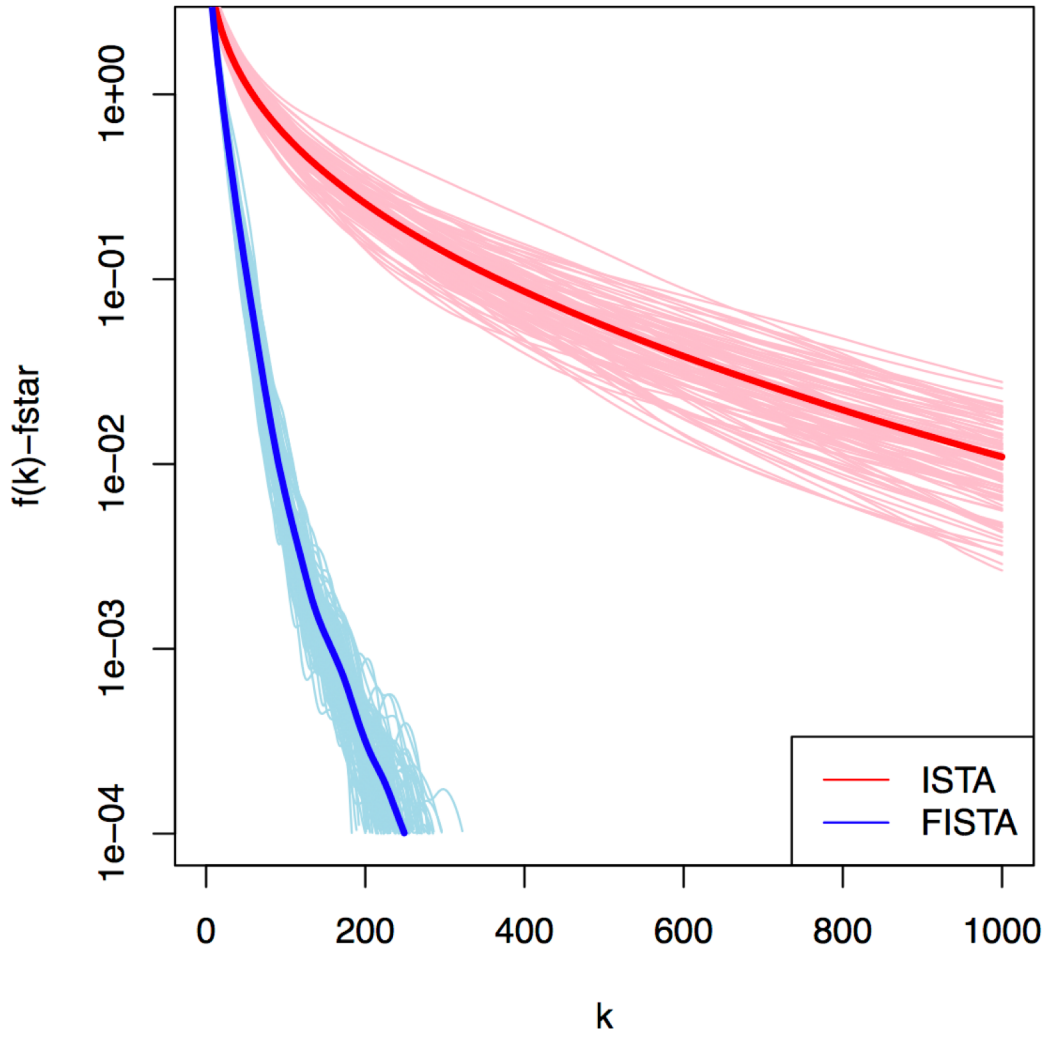
$$v = \beta^{(k-1)} + \frac{k-2}{k+1}(\beta^{(k-1)} - \beta^{(k-2)})$$
$$\beta^{(k)} = S_{\lambda t_k}\big(v + t_k X^T(y - Xv)\big), \quad k = 1, 2, 3, \ldots$$

推荐阅读：A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems，SIAM J. IMAGING SCIENCES，2009

Lasso regression: 100 instances (with $n = 100$, $p = 500$):



Lasso logistic regression: 100 instances ($n = 100$, $p = 500$):



应用：LASSO 问题求解

LASSO 问题是在多个算法中出现的应用，现将其整理如下：

- 使用连续化策略的 LASSO 问题求解
  - 实例：利用梯度法解 LASSO 问题
  - 实例：连续化次梯度法解 LASSO 问题
  - 实例：近似点梯度法及其 Nesterov 加速算法求解 LASSO 问题
  - 实例：近似点法解 LASSO 问题
  - 实例：分块坐标下降法求解 LASSO 问题
  - 实例：原始-对偶混合梯度算法求解 LASSO 问题
- 不使用连续化策略的 LASSO 问题求解
  - 实例：次梯度法解 LASSO 问题
  - 实例：交替方向乘子法解 LASSO 问题

https://bicmr.pku.edu.cn/~wenzw/optbook/pages/contents/contents.html#5

Is acceleration always useful?

Acceleration can be a very effective speedup tool ... but should it always be used?

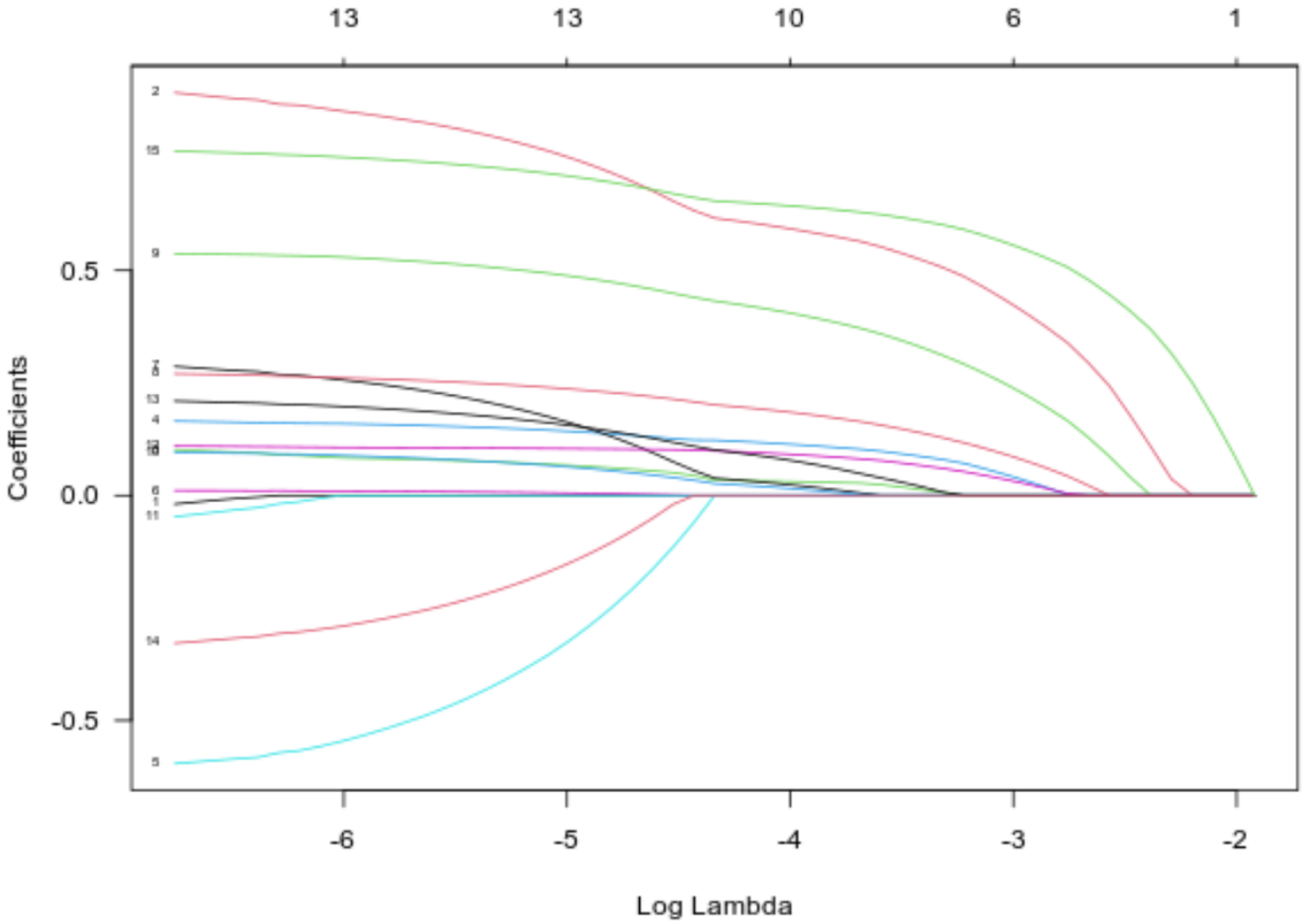
In practice the speedup of using acceleration is diminished in the presence of warm starts. For example, suppose want to solve lasso problem for tuning parameters values

$$\lambda_1 > \lambda_2 > \cdots > \lambda_r$$

- When solving for λ_1, initialize $x^{(0)} = 0$, record solution $\hat{x}(\lambda_1)$
- When solving for λ_j, initialize $x^{(0)} = \hat{x}(\lambda_{j-1})$, the recorded solution for λ_{j-1}

Over a fine enough grid of λ values, proximal gradient descent can often perform just as well without acceleration

Lasso的解径



sklearn.linear_model.Lasso

class `sklearn.linear_model.` **`Lasso`**(*alpha=1.0*, ***, *fit_intercept=True*, *normalize=False*, *precompute=False*, *copy_X=True*, *max_iter=1000*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*) [source]

Linear Model trained with L1 prior as regularizer (aka the Lasso)

The optimization objective for Lasso is:

```
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
```

Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=1.0` (no L2 penalty).

Read more in the User Guide.

Parameters:

alpha : *float, default=1.0*
Constant that multiplies the L1 term. Defaults to 1.0. `alpha = 0` is equivalent to an ordinary least square, solved by the **`LinearRegression`** object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Given this, you should use the **`LinearRegression`** object.

fit_intercept : *bool, default=True*
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use **`sklearn.preprocessing.StandardScaler`** before calling `fit` on an estimator with `normalize=False`.

precompute : *'auto', bool or array-like of shape (n_features, n_features), default=False*
Whether to use a precomputed Gram matrix to speed up calculations. If set to `'auto'` let us decide. The Gram matrix can also be passed as argument. For sparse input this option is always `True` to preserve sparsity.

Thank you for your attentions !