# Chapter 12 Beyond the Boyd and Vandenberghe Book

Shixiao Willing Jiang

Last update on 2025-04-30 20:15:15+08:00

# Table of contents

Conjugate Gradient and Krylov subspace

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

see Conjugate Gradient Krylov subspace

see dqq note

# Three classes of methods for linear equations

methods to solve linear system $Ax = b$, $A \in \mathbb{R}^{n \times n}$

- dense direct (factor-solve methods)
  - runtime depends only on size; independent of data, structure, or sparsity
  - work well for $n$ up to a few thousand
- sparse direct (factor-solve methods)
  - runtime depends on size, sparsity pattern; (almost) independent of data
  - can work well for $n$ up to $10^4$ or $10^5$ (or more)
  - requires good heuristic for ordering
- indirect (iterative methods)
  - runtime depends on data, size, sparsity, required accuracy
  - requires tuning, preconditioning, ...
  - good choice in many cases; only choice for $n = 10^6$ or larger

# Symmetric positive definite linear systems

SPD system of equations

$$Ax = b, \qquad A \in \mathbb{R}^{n \times n}, \qquad A = A^T \succ 0$$

- Newton/interior-point search direction: $\nabla^2 \phi(x) \Delta x = -\nabla \phi(x)$
- least-squares normal equations: $(A^T A)x = A^T b$
- regularized least-squares: $(A^T A + \mu I)x = A^T b$
- minimization of convex quadratic function $(1/2)x^T Ax - b^T x$
- solving (discretized) elliptic PDE (e.g., Poisson equation)

# Gradient descent and steepest gradient descent

general gradient descent

- let
$$g_\ell = \nabla f(x_\ell) = Ax_\ell - b$$
be the gradient ascent direction, then we search the descent direction along the negative $g_\ell$
- let $\alpha_\ell$ be the step size, then the GD method is
$$\begin{cases} g_\ell = Ax_\ell - b \\ x_{\ell+1} = x_\ell - \alpha_\ell g_\ell \end{cases}$$

steepest descent method

- optimize the step size such that
$$\hat{\alpha}_\ell = \text{argmin}_{\alpha_\ell}\ h = \text{argmin}_{\alpha_\ell}\ f(x_\ell - \alpha_\ell g_\ell)$$
- take the derivative
$$\frac{\partial h}{\partial \alpha_\ell} = \alpha_\ell g_\ell^T A g_\ell - g_\ell^T (Ax_\ell - b)$$

- the step size becomes

$$\alpha_\ell = \frac{g_\ell^T (Ax_\ell - b)}{g_\ell^T A g_\ell} = \frac{g_\ell^T g_\ell}{g_\ell^T A g_\ell}$$

- the steepest descent method is

$$\begin{cases} g_\ell = Ax_\ell - b \\ \alpha_\ell = \frac{g_\ell^T g_\ell}{g_\ell^T A g_\ell} \\ x_{\ell+1} = x_\ell - \alpha_\ell g_\ell \end{cases}$$

- moreover, since $\partial h / \partial \alpha_\ell = 0$

$$\partial h / \partial \alpha_\ell = -g_\ell^T (Ax_{\ell+1} - b) = -g_\ell^T g_{\ell+1} = 0$$

then $g_\ell$ and $g_{\ell+1}$ are orthogonal

# CG overview

- proposed by Hestenes and Stiefel in 1952 (as direct method)
- conjugate gradient, also called conjugate gradient descent, is a classical iteration optimization approach for specific unconstrained optimization problem
- solves SPD system $Ax = b$
  - in theory (i.e., exact arithmetic) in $n$ iterations
  - each iteration requires a few inner products in $\mathbb{R}^n$, and one matrix-vector multiply $z \to Az$
- for $A$ dense, matrix-vector multiply $z \to Az$ costs $n^2$, so total cost is $n^3$, same as direct methods
- get advantage over dense if matrix-vector multiply is cheaper than $n^2$
- with roundoff error, CG can work poorly (or not at all)
- but for some $A$ (and $b$), can get good approximate solution in $\ll n$ iterations
- the advantage of CG is (1) easy for implementation not hard for tuning parameters (2) in general faster than general gradient descent

# Solution, error and residual

- $x^* = A^{-1}b$ is solution
- $x^*$ minimizes (convex function) $f(x) = (1/2)x^T Ax - b^T x$
- $\nabla f(x) = Ax - b$ is (steepest) descent direction of $f$ (in $\ell^2$-norm) or general gradient descent direction in arbitrary norm
- with $f^* = f(x^*)$, we have

$$\begin{aligned} f(x) - f^* &= (1/2)x^T Ax - b^T x - (1/2)x^{*T} Ax^* + b^T x^* \\ &= (1/2)(x - x^*)^T A(x - x^*) \\ &= (1/2)\|x - x^*\|_A^2 \end{aligned}$$

  i.e., $f(x) - f^*$ is half of squared $A$-norm of error $x - x^*$
- $r = b - Ax$ is called the residual at $x$ with $r = -\nabla f(x) = A(x^* - x)$

# Krylov subspace

(a.k.a. controllability subspace)

$$\mathcal{K}_k = \text{span}\{b, Ab, ..., A^{k-1}b\} = \{p(A)b \mid p \text{ polynomial}, \deg p < k\}$$

we define the Krylov sequence $x^{(1)}, x^{(2)}, ...$ as

$$x^{(k)} = \underset{x \in \mathcal{K}_k}{\text{argmin}}\, f(x) = \underset{x \in \mathcal{K}_k}{\text{argmin}}\, \|x - x^*\|_A^2$$

the CG algorithm (among others) generates the Krylov sequence

# Properties of Krylov sequence

- $f(x^{(k+1)}) \leq f(x^{(k)})$ (but $\|r\|$ can increase)
- $x^{(n)} = x^*$ (i.e., $x^* \in \mathcal{K}_n$ even when $\mathcal{K}_n \neq \mathbb{R}^n$)
- $x^{(k)} = p_k(A)b$, where $p_k$ is a polynomial with deg $p_k < k$
- this is a sequence of nested subsequence $\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq ... \subseteq \mathcal{K}_k \subseteq ... \subseteq \mathbb{R}^n$
- less obvious: there is a two-term recurrence

$$x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)} + \beta_k (x^{(k)} - x^{(k-1)})$$

  for some $\alpha_k, \beta_k$ (basis of CG algorithm)
- key property. If $\mathcal{K}_{k+1} = \mathcal{K}_k$, then $\mathcal{K}_s = \mathcal{K}_k$ for all $s \geq k$

# Cayley-Hamilton theorem

we claim that $x^* = A^{-1}b \in \mathcal{K}_n$ even $\mathcal{K}_n \neq \mathbb{R}^n$

characteristic polynomial of $A$:

$$\chi(s) = \det(sI - A) = s^n + \alpha_1 s^{n-1} + \cdots + \alpha_n$$

by Cayley-Hamilton theorem

$$\chi(A) = A^n + \alpha_1 A^{n-1} + \cdots + \alpha_n I = 0$$

and so

$$A^{-1} = -(1/\alpha_n)A^{n-1} - (\alpha_1/\alpha_n)A^{n-2} - \cdots - (\alpha_{n-1}/\alpha_n)I$$

we see that $x^* = A^{-1}b \in \mathcal{K}_n$

- a set of vectors $\{p_0, ..., p_{k-1}\}$ is conjugate with respect to matrix $A \in \mathbb{S}^n_{++}$ if $p_i^T A p_j = 0$ for all $i \neq j \in \{0, ..., k-1\}$
- if let $\overline{p}_i = A^{1/2} p_i$, then $\overline{p}_i$ is perpendicular to $\overline{p}_j$
- if $\{p_i\}$ conjugate then $\{p_i\}$ are linear independent
- define the residual $r_\ell = b - Ax_\ell$ and we hope that the residuals are orthogonal, $r_i^T r_j = 0, \ i \neq j$
- let $d_i$ be the search direction for $x_i$,

$$x_{\ell+1} = x_\ell + \alpha_\ell d_\ell$$

  and let $d_0 = r_0 = b - Ax_0$ be the initial search direction.
- We hope that the search direction $\{d_i\}$ are conjugate and $\{d_i\}$ can be updated by

$$d_{\ell+1} = r_{\ell+1} + \beta_\ell d_\ell$$

- then the residual $r_{\ell+1}$ can be updated by

$$b - Ax_{\ell+1} = b - Ax_\ell - \alpha_\ell A d_\ell \quad \Longrightarrow \quad r_{\ell+1} = r_\ell - \alpha_\ell A d_\ell$$

- the conjugate gradient method is roughly

$$\begin{cases} x_{\ell+1} = x_\ell + \alpha_\ell d_\ell \\ r_{\ell+1} = r_\ell - \alpha_\ell A d_\ell \\ d_{\ell+1} = r_{\ell+1} + \beta_\ell d_\ell \end{cases}$$

- since $r_i^T r_j = 0$ for $i \neq j$, we can compute $\alpha_\ell$,

$$\begin{aligned} r_{\ell+1}^T r_\ell &= (r_\ell - \alpha_\ell A d_\ell)^T r_\ell = 0 \\ \alpha_\ell &= \frac{r_\ell^T r_\ell}{r_\ell^T A d_\ell} = \frac{r_\ell^T r_\ell}{(d_\ell - \beta_{\ell-1} d_{\ell-1})^T A d_\ell} \\ &= \frac{r_\ell^T r_\ell}{d_\ell^T A d_\ell} \end{aligned}$$

- for conjugate $\{d_\ell\}$, we compute $\beta_\ell$

$$\begin{aligned} d_{\ell+1}^T A d_\ell &= (r_{\ell+1} + \beta_\ell d_\ell)^T A d_\ell = 0 \\ \beta_\ell &= -\frac{r_{\ell+1}^T A d_\ell}{d_\ell^T A d_\ell} = -\frac{r_{\ell+1}^T (r_\ell - r_{\ell+1})}{\alpha_\ell d_\ell^T A d_\ell} \\ &= \frac{r_{\ell+1}^T r_{\ell+1}}{r_\ell^T r_\ell} \end{aligned}$$

# CG algorithm

(follows C. T. Kelley)

$x := 0, \quad r := b, \quad \rho_0 := \|r\|_2^2$

for $k = 1, ..., N_{\mathbf{max}}$

quit if $\sqrt{\rho_{k-1}} \leq \epsilon \|b\|_2$

if $k = 1$

then $d := r$;

else

$d := r + (\rho_{k-1}/\rho_{k-2})d$

$w := Ad$

$\alpha := \rho_{k-1}/d^T w$

$x := x + \alpha d$

$r := r - \alpha w$

$\rho_k := \|r\|_2^2$ or $\beta_k = \|r_k\|_2^2/\|r_{k-1}\|_2^2$

# Efficient matrix-vector multiply

- sparse $A$
- structured (e.g., sparse plus low rank)
- products of easy-to-multiply matrices
- fast transforms (FFT, wavelet, ... )
- inverses of lower/upper triangular (by forward/backward substitution)
- fast Gauss transform, for $A_{ij} = \exp(-\|v_i - v_j\|^2/\sigma^2)$ (via multipole)

# Shifting

- suppose we have guess $\hat{x}$ of solution $x^*$
- we can solve $Az = b - A\hat{x}$ using CG, then get $x^* = \hat{x} + z$
- in this case $x^{(k)} = \hat{x} + z^{(k)} = \underset{x \in \hat{x} + \mathcal{K}_k}{\text{argmin}}\, f(x)$, (where $\hat{x} + \mathcal{K}_k$ is called shifted Krylov subspace)
- same as initializing CG algorithm with $x := \hat{x},\ r := b - Ax$
- good for 'warm start', i.e., solving $Ax = b$ starting from a good initial guess (e.g., the solution of another system $\tilde{A}x = \tilde{b}$, with $A \approx \tilde{A},\ b \approx \tilde{b}$)

# Preconditioned conjugate gradient algorithm

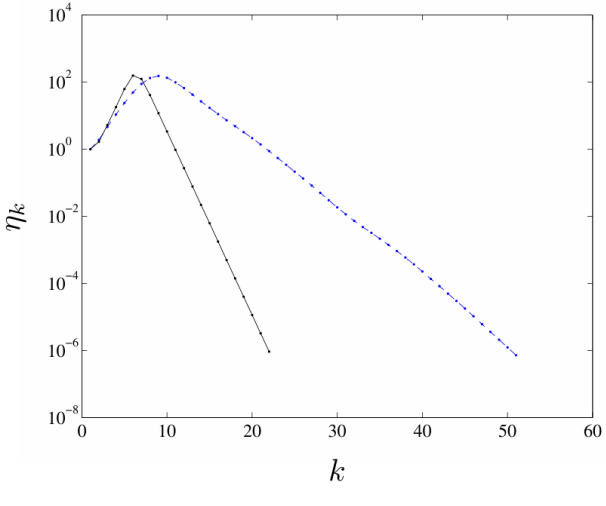
- idea: apply CG after linear change of coordinates $x = Ty$, $\det T \neq 0$
- use CG to solve $T^T A T y = T^T b$; then set $x^* = T^{-1} y^*$
- $T$ or $M = TT^T$ is called preconditioner
- in naive implementation, each iteration requires multiplies by $T$ and $T^T$ (and $A$); also need to compute $x^* = T^{-1} y^*$ at end
- can re-arrange computation so each iteration requires one multiply by $M$ (and $A$), and no final solve $x^* = T^{-1} y^*$
- called preconditioned conjugate gradient (PCG) algorithm

# Choice of preconditioner

- if spectrum of $T^T AT$ (which is the same as the spectrum of $MA$) is clustered, PCG converges fast
- extreme case: $M = A^{-1}$
- trade-off between enhanced convergence, and extra cost of multiplication by $M$ at each step
- goal is to find $M$ that is cheap to multiply, and approximate inverse of $A$ (or at least has a more clustered spectrum than $A$)

# Larger example

residual convergence with and without diagonal preconditioning



# CG summary

- in theory (with exact arithmetic) converges to solution in $n$ steps
  - the bad news: due to numerical round-off errors, can take more than $n$ steps (or fail to converge)
  - the good news: with luck (i.e., good spectrum of $A$), can get good approximate solution in $\ll n$ steps
- each step requires $z \to Az$ multiplication
  - can exploit a variety of structure in $A$
  - in many cases, never form or store the matrix $A$
- compared to direct (factor-solve) methods, CG is less reliable, data dependent; often requires good (problem-dependent) preconditioner
- but, when it works, can solve extremely large systems

# Krylov subspace method

- Krylov subspace method can be classified into two groups:
  - Krylov Arnoldi iteration is for general, nonsymmetric, without structured matrices
  - Krylov Lanczos is for conjugate symmetric matrices which is almost the same with the Arnoldi iteration but fully use the symmetric structure to reduce the computational cost
- solving linear equations
  - GMRES (Generalized Minimal Residual) is for large-scale, sparse linear equations based on Krylov Arnoldi iteration. It is designed for nonsymmetric and without structured problems. (The residual $r_j$ has minimum norm for $x_j$ in $\mathcal{K}_j$)
  - MINRES (Minimum Residual) (The residual $r_j$ has minimum norm for $x_j$ in $\mathcal{K}_j$) and SymmLQ (Symmetric LQ Method) (The error $e_j$ has minimum norm) are for large-scale, sparse, and symmetric linear equations based on Krylov Lanczos iteration.
  - CG (Conjugate Gradient) is for large-scale, sparse, and symmetric positive-definite linear equations also based on Krylov Lanczos iteration. (The residual $r_j = b - Ax_j$ is orthogonal to $\mathcal{K}_j$)
  - BiCG (Biconjugate Gradient Method) is for large-scale, sparse linear equations which is a generalization of CG but can be applied to nonsymmetric problems. ($r_j$ is orthogonal to a different space $\mathcal{K}_j(A^T)$)
    1. BiCGSTAB (Biconjugate Gradient Stabilized Method)
    2. QMR (Quasi-Minimal Residual)

Conjugate Gradient and Krylov subspace

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

# definition

Recall that for convex and differentiable $f$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \quad \text{for all } x, y$$

This is first-order condition. Linear approximation always underestimates $f$

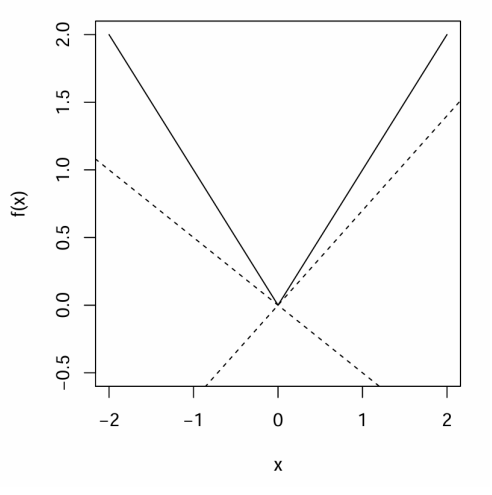
A subgradient of a convex function $f$ at $x$ is any $g \in \mathbb{R}^n$ such that

$$f(y) \geq f(x) + g^T (y - x) \quad \text{for all } y$$

- Always exists
- If $f$ differentiable at $x$, then $g = \nabla f(x)$ uniquely
- Same definition works for nonconvex $f$ (however, subgradients need not exist)

# Examples of subgradients

Consider $f : \mathbb{R} \to \mathbb{R},\ f(x) = |x|$




- For $x \neq 0$, unique subgradient $g = \text{sign}(x)$
- For $x = 0$, subgradient $g$ is any element of $[-1, 1]$

# Subdifferential

Set of all subgradients of convex $f$ is called the subdifferential:

$$\partial f(x) = \{g \in \mathbb{R}^n : g \text{ is a subgradient of } f \text{ at } x\}$$

- Nonempty (only for convex $f$)
- $\partial f(x)$ is closed and convex (even for nonconvex f)
- If $f$ is differentiable at $x$, then $\partial f(x) = \{\nabla f(x)\}$
- If $\partial f(x) = \{g\}$ , then $f$ is differentiable at $x$ and $\nabla f(x) = g$

# Optimality condition

For any $f$ (convex or not),

$$f(x^*) = \min_x f(x) \iff 0 \in \partial f(x^*)$$

That is, $x^*$ is a minimizer if and only if 0 is a subgradient of $f$ at $x^*$. This is called the subgradient optimality condition

Why? Easy: $g = 0$ being a subgradient means that for all $y$

$$f(y) \geq f(x^*) + 0^T(y - x^*) = f(x^*)$$

Note the implication for a convex and differentiable function $f$, with $\partial f(x) = \{\nabla f(x)\}$

# Subgradient method

Now consider $f$ convex, having $\mathbf{dom}(f) = \mathbb{R}^n$, but not necessarily differentiable

Subgradient method: like gradient descent, but replacing gradients with subgradients. Initialize $x^{(0)}$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot g^{(k-1)}, \qquad k = 1, 2, 3, ...$$

where $g^{(k-1)} \in \partial f(x^{(k-1)})$, any subgradient of $f$ at $x^{(k-1)}$

Subgradient method is not necessarily a descent method, thus we keep track of best iterate $x^{(k)}_{\text{best}}$ among $x^{(0)}, ..., x^{(k)}$ so far, i.e.,

$$f(x^{(k)}_{\text{best}}) = \min_{i=0,...,k} f(x^{(i)})$$

Conjugate Gradient and Krylov subspace

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

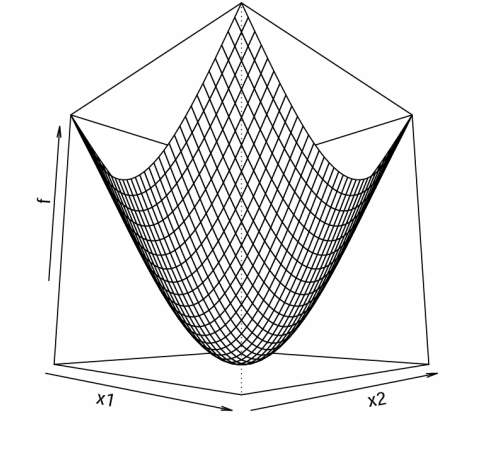
# Coordinatewise optimality

We now focus on a very simple technique that can be surprisingly efficient, scalable: coordinate descent, or more appropriately called coordinatewise minimization

Q: Given convex, differentiable $f : \mathbb{R}^n \to \mathbb{R}$, if we are at a point $x$ such that $f(x)$ is minimized along each coordinate axis, then have we found a global minimizer?

That is, does $f(x + \delta e_i) \geq f(x)$ for all $\delta$, $i \implies f(x) = \min_z f(z)$?

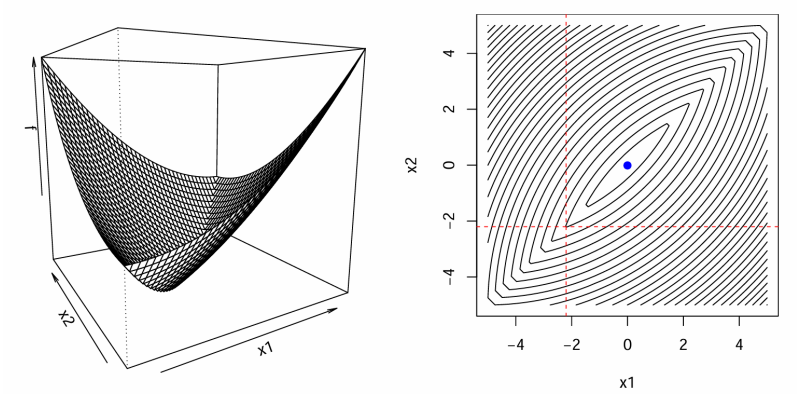
(Here $e_i = (0, ..., 1, ..., 0) \in \mathbb{R}^n$ is the $i$th standard basis vector)



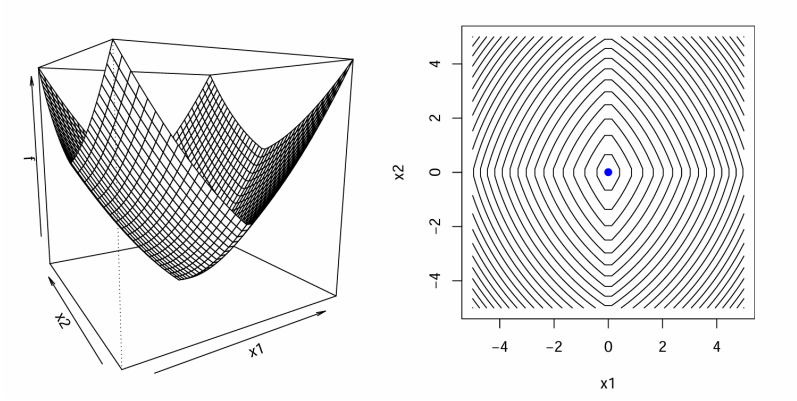

- A: Yes! Proof:

$$0 = \nabla f(x) = \Big( \frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_n}(x) \Big)$$

- Q: Same question, but now for $f$ convex, and not differentiable?




- A: No! Look at the above counterexample
- Q: Same, now $f(x) = g(x) + \sum_{i=1}^n h_i(x_i)$, with $g$ convex, smooth, and each $h_i$ convex? (Here the nonsmooth part is called separable)




- A: Yes! Proof: using convexity of $g$ and subgradient optimality

$$
\begin{aligned}
f(y) - f(x) &= g(y) - g(x) + \sum_{i=1}^{n} [h_i(y_i) - h_i(x_i)] \\
&\geq \sum_{i=1}^{n} \underbrace{\left[\partial_{x_i} g(x)(y_i - x_i) + h_i(y_i) - h_i(x_i)\right]}_{\geq 0} \geq 0
\end{aligned}
$$

# Coordinate descent

This suggests that for the problem

$$\min_x f(x)$$

where $f(x) = g(x) + \sum_{i=1}^n h_i(x_i)$, with $g$ convex and differentiable and each $h_i$ convex, we can use coordinate descent: let $x^{(0)} \in \mathbb{R}^n$, and repeat

$$x_i^{(k)} = \underset{x_i}{\text{argmin}}\, f\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big), \qquad i = 1, ..., n$$

for $k = 1, 2, 3, ...$

Important note: we always use most recent information possible

Tseng (2001) showed that for such $f$ (provided $f$ is continuous on compact set $\{x : f(x) \leq f(x^{(0)})\}$ and $f$ attains its minimum), any limit point of $x^{(k)},\ k = 1, 2, 3, ...$ is a minimizer of $f$

Notes:

- Order of cycle through coordinates is arbitrary, can use any permutation of $\{1, 2, ..., n\}$
- Can everywhere replace individual coordinates with blocks of coordinates
- "One-at-a-time" update scheme is critical, and "all-at-once" scheme does not necessarily converge
- The analogy for solving linear systems: Gauss-Seidel versus Jacobi method

# Example: linear regression

Given $y \in \mathbb{R}^n$, and $X \in \mathbb{R}^{n \times p}$ with columns $X_1, ..., X_p$, consider the linear regression problem:

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

Minimizing over $\beta_i$, with all $\beta_j, j \neq i$ fixed:

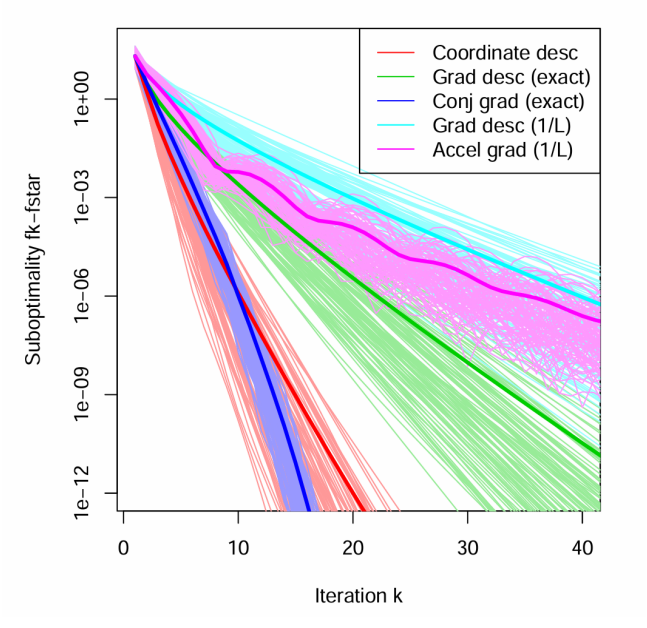
$$0 = \partial_{\beta_i} f(\beta) = X_i^T(X\beta - y) = X_i^T(X_i\beta_i + X_{-i}\beta_{-i} - y)$$

i.e., we take

$$\beta_i = \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}$$

Coordinate descent repeats this update for $i = 1, 2, ..., p, 1, 2, ....$ Note that this is exactly Gauss-Seidl for the system $X^T X\beta = X^T y$

Coordinate descent vs gradient descent for linear regression: 100 random instances with $n = 100$, $p = 20$



Is it fair to compare 1 cycle of coordinate descent to 1 iteration of gradient descent? Yes, if were clever

- Gradient descent: $\beta \leftarrow \beta + tX^T(y - X\beta)$, costs $O(np)$ flops
- Coordinate descent, one coordinate update:

$$\beta_i \leftarrow \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i} = \frac{X_i^T r}{\|X_i\|_2^2} + \beta_i$$

  where $r = y - X\beta$
- Each coordinate costs $O(n)$ flops: $O(n)$ to update $r$, $O(n)$ to compute $X_i^T r$
- One cycle of coordinate descent costs $O(np)$ operations, same as gradient descent

# Example: lasso regression

Consider the lasso problem:

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

Note that nonsmooth part here is separable: $\|\beta\|_1 = \sum_{i=1}^p |\beta_i|$. Minimizing over $\beta_i$, with $\beta_j$, $j \neq i$ fixed:

$$0 = X_i^T X_i \beta_i + X_i^T (X_{-i}\beta_{-i} - y) + \lambda s_i$$

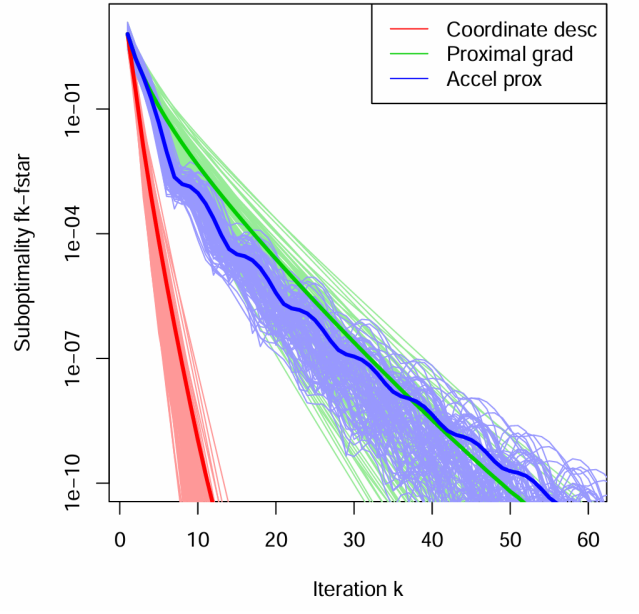
where $s_i \in \partial|\beta_i|$. Solution is simply given by soft-thresholding

$$\beta_i = S_{\lambda/\|X_i\|_2^2}\left(\frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}\right)$$

Repeat this for $i = 1, 2, ..., p, 1, 2, ....$ Here, the soft-thresholding operator $S_t$ is defined as

$$S_t(x_j) = \text{sign}(x_j)\,\mathbf{max}\{|x_j| - t, 0\} = \begin{cases} x_j - t, & x_j > t \\ 0, & -t \leq x_j \leq t, \\ x_j + t & x_j < -t \end{cases} \qquad j = 1, ..., p$$

Coordinate descent vs proximal gradient for lasso regression: 100 random instances with $n = 200,\ p = 50$ (all methods cost $O(np)$ per iter)



# Coordinate descent in statistics and ML

History in statistics/ML:

- Idea appeared in Fu (1998), and then again in Daubechies et al. (2004), but was inexplicably ignored
- Later, three papers in 2007, especially Friedman et al. (2007), really sparked interest in statistics and ML communities

Why is it used?

- Very simple and easy to implement
- Careful implementations can achieve state-of-the-art
- Scalable, e.g., don't need to keep full data in memory

Examples: lasso regression, lasso GLMs (under proximal Newton), SVMs, group lasso, graphical lasso (applied to the dual), additive modeling, matrix completion, regression with nonconvex penalties

# Coordinate gradient descent

For a smooth function $f$, the iterations

$$x_i^{(k)} = x_i^{(k-1)} - t_{ki} \cdot \partial_{x_i} f\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big), \qquad i = 1, ..., n$$

for $k = 1, 2, 3, ...$ are called coordinate gradient descent, and when $f = g + h$, with $g$ smooth and $h = \sum_{i=1}^n h_i$, the iterations

$$x_i^{(k)} = \mathrm{prox}_{h_i, t_{ki}}\Big(x_i^{(k-1)} - t_{ki} \cdot \partial_{x_i} g\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big)\Big), \quad i = 1, ..., n$$

for $k = 1, 2, 3, ...$ are called coordinate proximal gradient descent

When $g$ is quadratic, (proximal) coordinate gradient descent is the same as coordinate descent under proper step sizes

Conjugate Gradient and Krylov subspace

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

# Stochastic gradient descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x)$$

Gradient descent would repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, ...$$

In comparison, stochastic gradient descent or SGD (or incremental gradient descent) repeats:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, ...$$

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration $k$

Two rules for choosing index $i_k$ at iteration $k$:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, ..., m, 1, 2, ..., m, ...$

Randomized rule is more common in practice. For randomized rule, note that

$$E[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as using an unbiased estimate of the gradient at each step

Main appeal of SGD:

- Iteration cost is independent of $m$ (number of functions)
- Can also be a big savings in terms of memory useage

# Example: stochastic logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, ..., n$, recall logistic regression:

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta))\right)}_{f_i(\beta)}$$

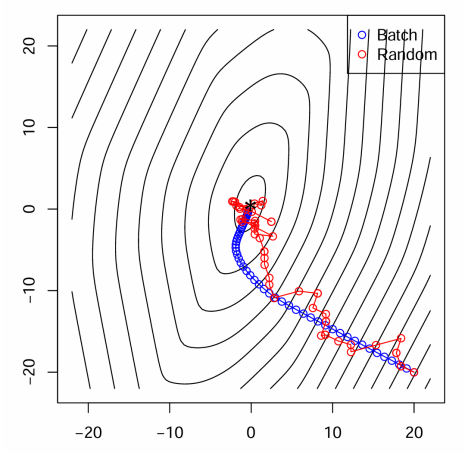
Gradient computation $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - p_i(\beta)\big) x_i$ is doable when $n$ is moderate, but not when $n$ is huge

Full gradient (also called batch) versus stochastic gradient:

- One batch update costs $O(np)$
- One stochastic update costs $O(p)$

Clearly, e.g., 10K stochastic steps are much more affordable

Small example with $n = 10, p = 2$ to show the classic picture for batch versus stochastic methods:




Blue: batch steps, $O(np)$ Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

# Mini-batches

See more about step sizes and convergence rates

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, ..., m\}$, $|I_k| = b \ll m$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \qquad k = 1, 2, 3, ...$$

Again, we are approximating full gradient by an unbiased estimate:

$$\mathbb{E}\Big[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\Big] = \nabla f(x)$$

Using mini-batches reduces variance by a factor $1/b$, but is also $b$ times more expensive. Theory is not convincing: under Lipschitz gradient, rate goes from $O(1/\sqrt{k})$ to $O(1/\sqrt{bk} + 1/k)$[3]

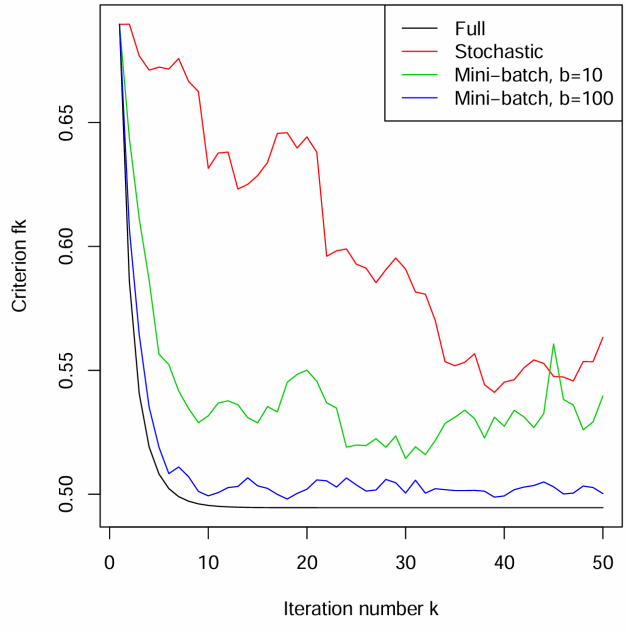
Back to logistic regression, lets now consider a regularized version:

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left( -y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta)) \right) + \frac{\lambda}{2} \|\beta\|_2^2$$

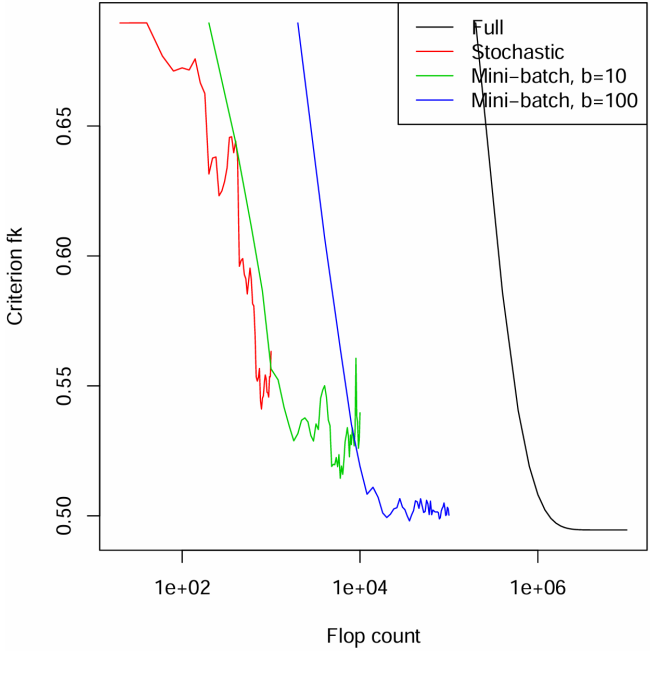
Comparison between methods:

- One batch update costs $O(np)$
- One mini-batch update costs $O(bp)$
- One stochastic update costs $O(p)$

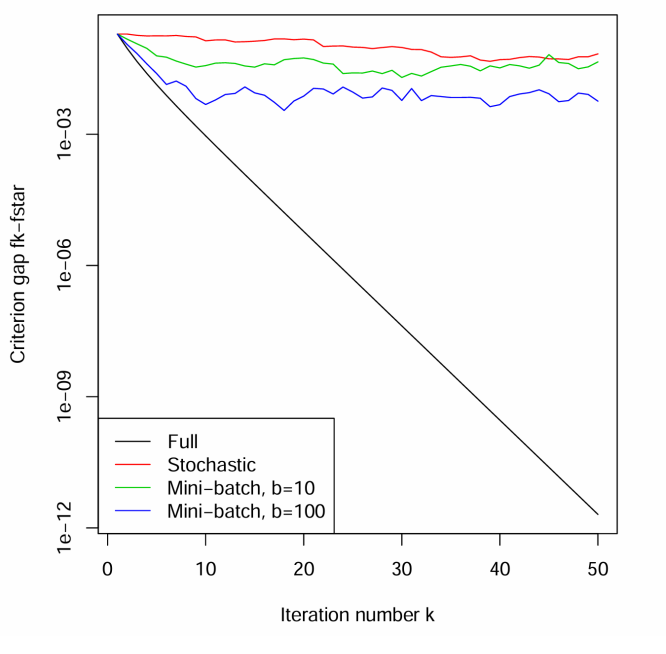
Example with $n = 10000, p = 20$, all methods use fixed step sizes:



Whats happening? Now lets parametrize by flops:



Finally, looking at suboptimality gap (on log scale):



# End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory
- But SGD is slow to converge, can't adapt to strong convexity
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice)

# SGD in large-scale ML

SGD has really taken off in large-scale machine learning

- In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance
- Thus (in contrast to what classic theory says) fixed step sizes are commonly used in ML applications
- One trick is to experiment with step sizes using small fraction of training before running SGD on full data set
- Momentum/acceleration, averaging, adaptive step sizes are all popular variants in practice
- SGD is especially popular in large-scale, continuous, nonconvex optimization, but it is still not particular well-understood there (a big open issue is that of implicit regularization)

Conjugate Gradient and Krylov subspace

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

# Framework of ADMM

- solve the following problem using ADMM

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c \end{aligned}$$

where $x \in \mathbb{R}^p, z \in \mathbb{R}^q, A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{m \times q}, c \in \mathbb{R}^k$ and

$$f : \mathbb{R}^p \to \mathbb{R}, \qquad g : \mathbb{R}^q \to \mathbb{R}$$

- the objective function is separable and constraint only contains equalities
- augmented Lagrangian method modifies the augmented objective function

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & Q_\rho(x, z) = f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \\ \text{subject to} \quad & Ax + Bz = c \end{aligned}$$

- augmented Lagrangian

$$\begin{aligned} L_\rho(x,z,\nu) &= Q_\rho(x,z) + \nu^T(Ax+Bz-c) \\ &= f(x)+g(z)+\frac{\rho}{2}\|Ax+Bz-c\|_2^2+\nu^T(Ax+Bz-c) \end{aligned}$$

- algorithm for the $k$th iteration:
  1. update $x$: $x^{(k)} = \mathsf{argmin}_x L_\rho(x, z^{(k-1)}, \nu^{(k-1)})$
  2. update $z$: $z^{(k)} = \mathsf{argmin}_z L_\rho(x^{(k)}, z, \nu^{(k-1)})$
  3. update $\nu$: $\nu^{(k)} = \nu^{(k-1)} + \rho(Ax^{(k)} + Bz^{(k)} - c)$

# conjugate function

- the basic idea is to use the connection between primal problem and dual problem. We solve the dual problem using gradient descent method so that we obtain the optimal of primal problem at the same time
- recall that the conjugate function is

$$f^*(y) = \max_x \ x^T y - f(x) = \max_x \ L(x, y) = L(x^*, y)$$

- even $f(x)$ is not convex, its conjugate $f^*(y)$ is always convex
- according to envelop theorem,

$$\frac{\partial f^*(y)}{\partial y} = \frac{\partial L(x^*, y)}{\partial y} = x^* = \operatorname*{argmin}_x \ x^T y - f(x)$$

# Dual gradient ascent

- the primal problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & Ax = c \end{array}$$

- the Lagrangian is

$$L(x, \nu) = f(x) + \nu^T (Ax - c)$$

- dual function is

$$g(\nu) = \min_x \; L(x, \nu) = -f^*(-A^T \nu) - c^T \nu$$

# Dual gradient ascent

- to solve the unconstrained dual problem, we need to compute the gradient of the objective

$$\nabla g(\nu) = -\frac{\partial f^*(-A^T\nu)}{\partial \nu} - c = A\frac{\partial f^*(-A^T\nu)}{\partial(-A^T\nu)} - c = Ax^* - c$$

- the iteration with the learning rate $\alpha_k$ is

$$\nu^{(k)} = \nu^{(k-1)} + \alpha_k \nabla g(\nu^{(k-1)})$$

- the algorithm of dual gradient ascent is:
  1. update $x$: $x^{(k)} = \text{argmin}_x \; L(x, \nu^{(k-1)})$
  2. update $\nu$: $\nu^{(k)} = \nu^{(k-1)} + \alpha_k[Ax^{(k)} - c]$
- dual decomposition can be used as a trick in dual gradient ascent method when the objective $f(x)$ is separable

# Augmented Lagrangian method

- consider the augmented form of the primal problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) + (\rho/2)\|Ax - c\|_2^2 \\ \text{subject to} & Ax = c \end{array}$$

- $\rho$ is a penalty parameter and the whole penalty term is to "increase" the convexity of the problem
- the augmented Lagrangian is

$$L_\rho(x, \nu) = f(x) + \frac{\rho}{2}\|Ax - c\|_2^2 + \nu^T(Ax - c)$$

- the algorithm of the augmented problem using dual gradient ascent is:
  1. update $x$: $x^{(k)} = \text{argmin}_x \; L_\rho(x, \nu^{(k-1)})$
  2. update $\nu$: $\nu^{(k)} = \nu^{(k-1)} + \rho[Ax^{(k)} - c]$
- the learning rate is taken to be $\rho$ which speeds up the convergence

# Scaled form

- augmented Lagrangian

$$L_\rho(x, z, \nu) \quad = \quad f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 + \nu^T(Ax + Bz - c)$$

- Scaled form: let $u = \nu/\rho$, then augmented Lagrangian becomes

$$L_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c + u\|_2^2 + C,$$

  where $C$ is independent of $x, z$
- algorithm for the $k$th iteration of ADMM updates:
  1. update $x$: $x^{(k)} = \mathsf{argmin}_x L_\rho(x, z^{(k-1)}, u^{(k-1)})$
  2. update $z$: $z^{(k)} = \mathsf{argmin}_z L_\rho(x^{(k)}, z, u^{(k-1)})$
  3. update $\nu$: $u^{(k)} = u^{(k-1)} + (Ax^{(k)} + Bz^{(k)} - c)$

# Example: lasso regression

- Lasso = loss + $\ell^1$ penalty
- gradient descent or Newton method are not applicable
- proximal gradient descent, coordinate descent, ADMM can be considered
- reference
  1. https://zhuanlan.zhihu.com/p/448289351
  2. Boyd ADMM paper
  3. CMU convex optimization course

# Example: lasso regression

Given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, recall the lasso problem:

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

We can rewrite this as:

$$\begin{aligned} \underset{\beta,\alpha}{\text{minimize}} \quad & \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\alpha\|_1 \\ \text{subject to} \quad & \beta = \alpha \end{aligned}$$

augmented Lagrangian

$$L_\rho(\beta, \alpha, w) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\alpha\|_1 + \frac{\rho}{2}\|\beta - \alpha + w\|_2^2$$

Scaled form ADMM steps:

1. update $\beta$:

$$\beta^{(k)} = \mathsf{argmin}_\beta \frac{1}{2}\|y - X\beta\|_2^2 + \frac{\rho}{2}\|\beta - \alpha^{(k-1)} + w^{(k-1)}\|_2^2$$

2. update $\alpha$:

$$\alpha^{(k)} = \mathsf{argmin}_\alpha \lambda\|\alpha\|_1 + \frac{\rho}{2}\|\beta^{(k)} - \alpha + w^{(k-1)}\|_2^2 = S_{\lambda/\rho}(\beta^{(k)} + w^{(k-1)})$$

3. update $w$:

$$w^{(k)} = w^{(k-1)} + (\beta^{(k)} - \alpha^{(k)})$$

Scaled form ADMM steps:

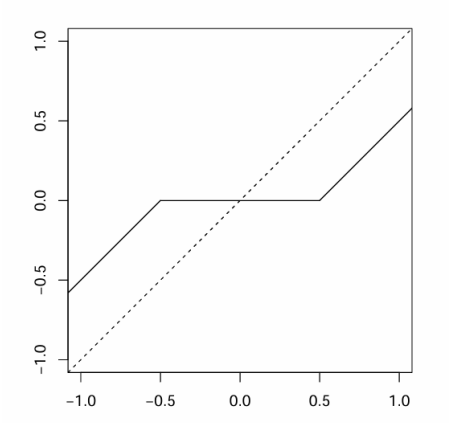
$$\begin{aligned}
\beta^{(k)} &= (X^T X + \rho I)^{-1}\big(X^T y + \rho(\alpha^{(k-1)} - w^{(k-1)})\big) \\
\alpha^{(k)} &= S_{\lambda/\rho}(\beta^{(k)} + w^{(k-1)}) \\
w^{(k)} &= w^{(k-1)} + \beta^{(k)} - \alpha^{(k)}
\end{aligned}$$

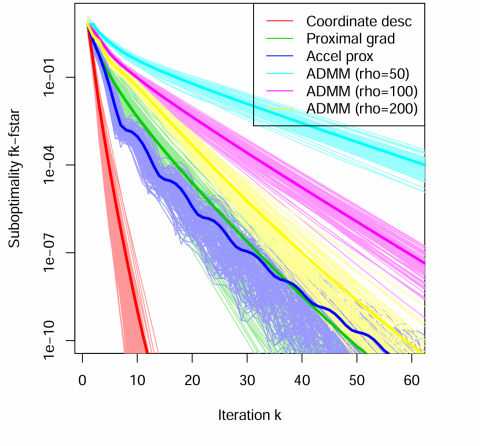
Notes:

- The matrix $X^T X + \rho I$ is always invertible, regardless of $X$
- If we compute a factorization (say Cholesky) in $O(p^3)$ flops, then each $\beta$ update takes $O(p^2)$ flops
- The $\alpha$ update applies the soft-thresholding operator $S_t$, which recall is defined as

$$[S_t(x)]_j = \text{sign}(x_j) \max\{|x_j| - t, 0\} = \begin{cases} x_j - t, & x_j > t \\ 0, & -t \leq x_j \leq t \\ x_j + t & x_j < -t \end{cases} \qquad j = 1, ..., p$$

- ADMM steps are almost like repeated soft-thresholding of ridge regression coefficients




- Soft-thresholding in one variable
- Comparison of various algorithms for lasso regression: 100 random instances with $n = 200, p = 50$