The Revival of Coordinate Descent Methods

Stephen Wright

University of Wisconsin-Madison

Johns Hopkins, November 2014

1. Coordinate Descent Methods
2. Asynchronous Random Kaczmarz (AsyRK)
3. Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm with Inconsistent Read (AsySPCD)
4. Extensions and Applications

Collaborators: **Ji Liu** (U. Rochester), Victor Bittorf (Cloudera), Yijun Huang, Chris Ré (Stanford), Krishna Sridhar (GraphLab).

Coordinate Descent

Simplest setting: unconstrained minimization.

$$\min_{x \in \mathbb{R}^n} f(x)$$

where f is a smooth continuous function.

Often OK for applications — and necessary for analysis — to make additional assumptions on f, such as

- smooth, e.g. Lipschitz continuously differentiable
- convex
- strongly convex

We consider later extensions to structured nonsmooth objectives.

Basic Coordinate Descent (CD) Framework

Set $j \leftarrow 0$ and choose $x^0 \in \mathbb{R}^n$;
repeat
  Choose index $i(j) \in \{1, 2, \ldots, n\}$;
  $x^{j+1} \leftarrow x^j - \alpha_j [\nabla f(x^j)]_{i(j)} e_{i(j)}$;
  $j \leftarrow j + 1$;
until termination test satisfied;

$e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T$: the ith coordinate vector.

$[\nabla f(x)]_i = i$th component of the gradient $\nabla f(x)$.

There are many variants within this framework.

Variants

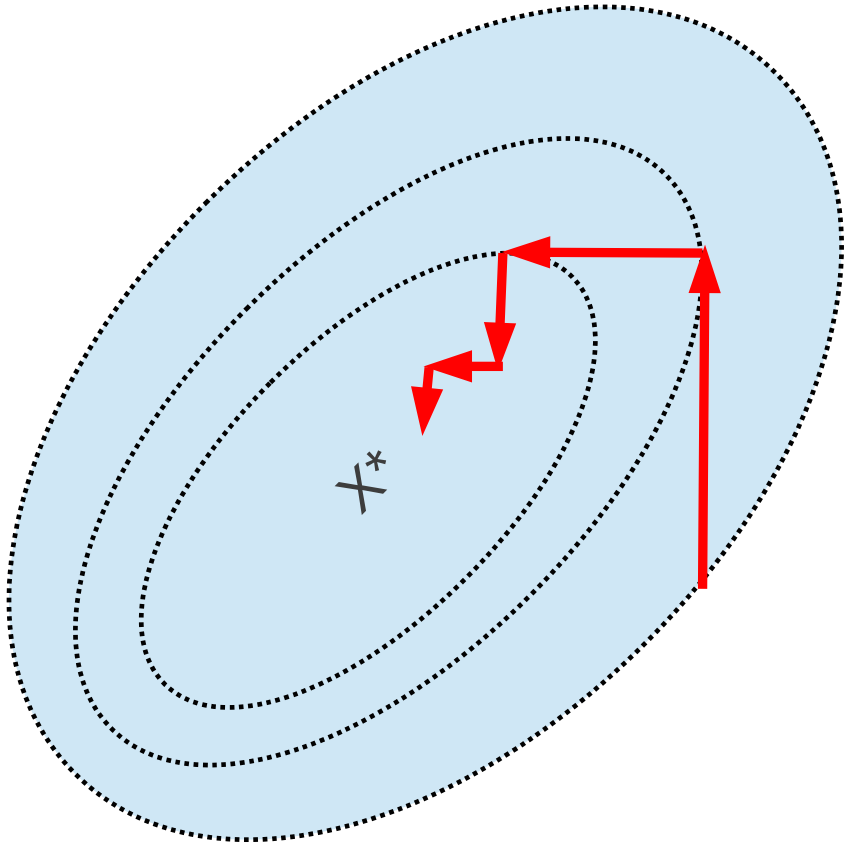
Selection of coordinate $i(j)$:

- Cycle through the coordinates: $i(j+1) = (i(j)+1) \mod n+1$;
- Essentially cyclic: touch each coordinate i at least once in each stretch of T iterations;
- Randomized: Select $i(j)$ at random and independently at each iteration.

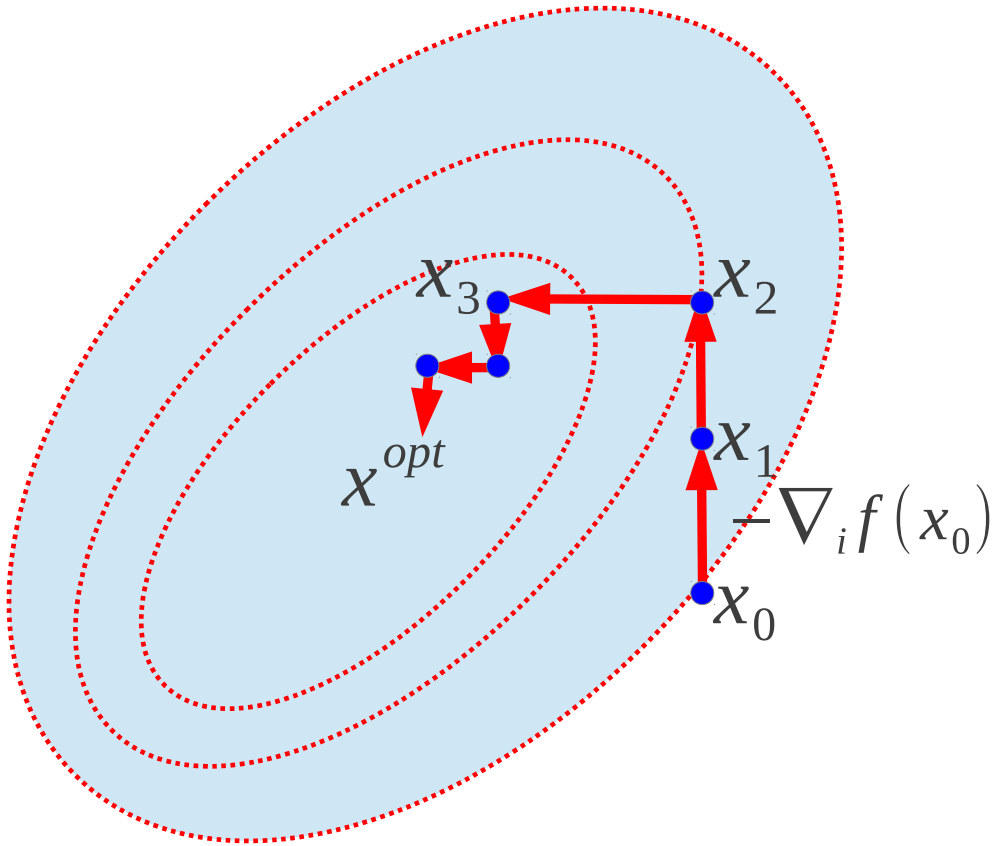
Line Search α_j:

- Short step: α_j prescribed by global knowledge about properties of f
- Line search: Choose α_j to approximately minimize f along the coordinate direction $i(j)$;
- Exact: Choose α_j to exactly minimize f along $i(j)$ coordinate.

Cyclic Coordinate Descent



Stochastic Coordinate Descent



Choose components $i(j)$ randomly, independently at each iteration.

Coordinate Descent (CD): Background

CD methods have been popular with practitioners over many years:

- Intuitive: Solve an optimization problem by solving a sequence of easier problems — in this case, one-dimensional optimization.
- Often easy to implement.
- Handle bounds and regularization well.

There was little interest among optimization researchers until recently (with some very notable exceptions: Bertsekas, Tseng, Luo,....)

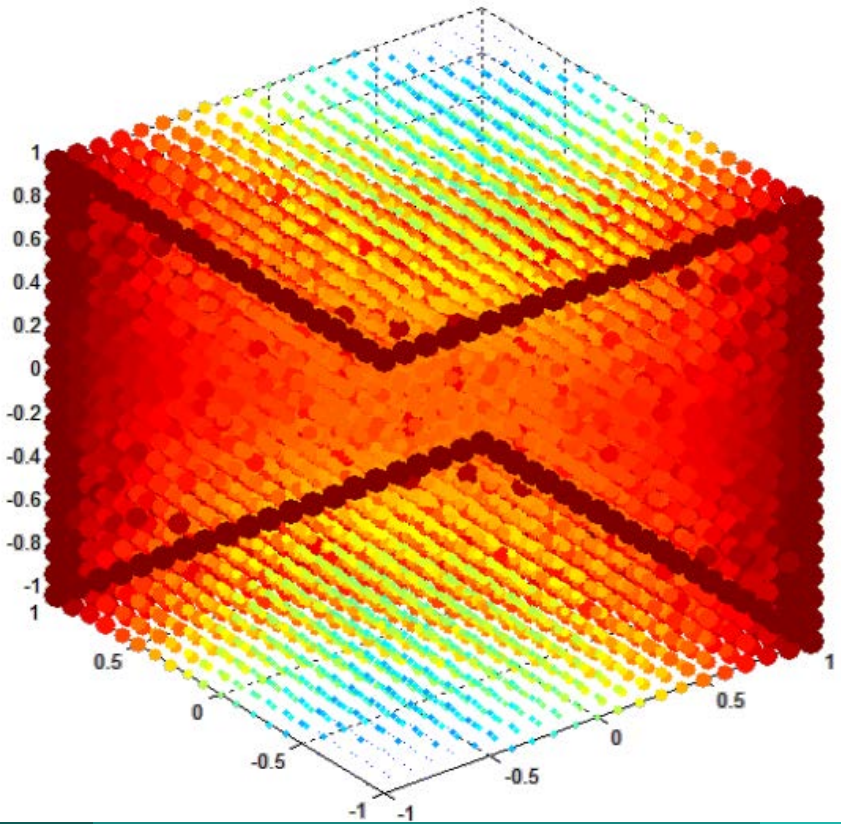
- Convergence analysis of popular variants was not obvious!
- According to certain assumptions on the cost of evaluating derivatives, it's not clear if CD is competitive in theory.

But interest among optimization specialists has grown since 2009, and new applications appear steadily.

Convergence? Not Always: Powell (1973)

$$f(x_1, x_2, x_3) = -(x_1x_2 + x_2x_3 + x_1x_3) + \sum_{i=1}^{3}(|x_i| - 1)_+^2.$$

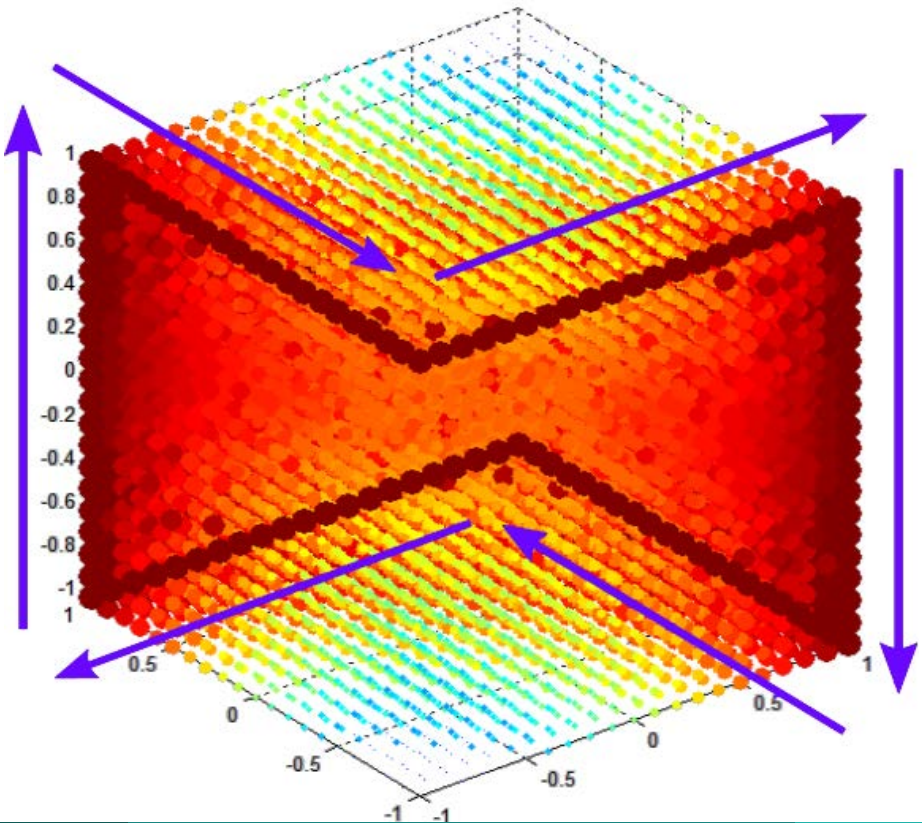
Nonconvex. Minimizers at $(1, 1, 1)^T$ and $(-1, -1, -1)^T$. CD with cyclic ordering, exact minimization cycles around the six nonoptimal vertices.



Convergence? Not Always: Powell (1973)

$$f(x_1, x_2, x_3) = -(x_1x_2 + x_2x_3 + x_1x_3) + \sum_{i=1}^{3}(|x_i| - 1)_+^2.$$

Nonconvex. Minimizers at $(1, 1, 1)^T$ and $(-1, -1, -1)^T$. CD with cyclic ordering, exact minimization cycles around the six nonoptimal vertices.



Convergence: Convex, Randomized, Short-Step

Convergence can be proved for the case of smooth, convex f.

[Nesterov, 2009] showed that randomized, short-step methods can be analyzed in a similar fashion to the corresponding full-gradient methods.

RCD (Randomized Coordinate Descent)
Set $j \leftarrow 0$ and choose $x^0 \in \mathbb{R}^n$;
repeat
  Choose index $i(j) \in \{1, 2, \ldots, n\}$ at random with equal probability;
  Set $\alpha_j \equiv 1/L_{\max}$;
  $x^{j+1} \leftarrow x^j - \alpha_j [\nabla f(x^j)]_{i(j)} e_{i(j)}$;
  $j \leftarrow j + 1$;
until termination test satisfied;

Here $L_{\max}$ is a "componentwise Lipschitz constant" for ∇f:

$$|[\nabla f(x + te_i)]_i - [\nabla f(x)]_i| \leq L_{\max}|t|, \quad \text{for all } x,\ t,\ i.$$

Full-Gradient Convergence

The classical short-step full-gradient descent method for minimizing f takes steps:

$$x^{j+1} = x^j - \alpha_j \nabla f(x^j), \quad \text{where } \alpha_j \equiv 1/L,$$

where L is the Lipschitz constant for ∇f, that is,

$$\|\nabla f(x+d) - \nabla f(x)\| \leq L\|d\|, \quad \text{for all } x,\ d.$$

Analysis uses Taylor's theorem:

$$\begin{aligned} f(x^{j+1}) &\leq f(x^j) - \alpha_j \nabla f(x^j)^T \nabla f(x^j - t_j \alpha_j \nabla f(x^j)) \quad (\text{some } t_j \in (0,1)) \\ &\leq f(x^j) - \alpha_j \left(1 - \frac{\alpha_j}{2} L\right) \|\nabla f(x^j)\|^2 \\ &= f(x^j) - \frac{1}{2L} \|\nabla f(x^j)\|^2, \quad \text{since } \alpha_j \equiv 1/L. \end{aligned}$$

All iterates x^j lie in the level set $\{x \mid f(x) \leq f(x^0)\}$. Suppose there is R_0 such that $\|x - x^*\| \leq R_0$ for all x in this set. Then

$$f(x^j) - f(x^*) \leq \nabla f(x^j)^T (x^j - x^*) \leq R_0 \|\nabla f(x^j)\|.$$

By substituting this bound for $\|\nabla f(x^j)\|$, and subtracting $f(x^*)$ from both sides, we get

$$[f(x^{j+1}) - f(x^*)] \leq [f(x^j) - f(x^*)] - \frac{1}{2LR_0^2}[f(x^j) - f(x^*)]^2.$$

So defining $\phi_j := f(x^j) - f(x^*)$, we get

$$\phi_{j+1} \leq \phi_j - \frac{1}{2LR_0^2}\phi_j^2.$$

Some clever manipulation involving sequence ϕ_j yields

$$[f(x^j) - f(x^*)] \leq \frac{2nLR_0^2}{j}.$$

Sublinear convergence with rate $1/j$.

Q: How does this analysis change for randomized CD?

A: Not much different, when we take expectations over the random variables $i(j)$.

Initial Taylor-series step yields:

$$\begin{aligned} f(x^{j+1}) &\leq f(x^j) - \alpha_j [\nabla f(x^j)]_{i(j)} [\nabla f(x^j - t_j \alpha_j [\nabla f(x^j)]_{i(j)})]_{i(j)} \\ &\leq f(x^j) - \alpha_j \left(1 - \frac{\alpha_j}{2} L_{\max}\right) [\nabla f(x^j)]^2_{i(j)} \\ &= f(x^j) - \frac{1}{2L_{\max}} [\nabla f(x^j)]^2_{i(j)}. \end{aligned}$$

Expectation of the last term w.r.t $i(j)$ is simply its average over all possible values $i(j) = 1, 2, \ldots, n$:

$$E_{i(j)} \left([\nabla f(x^j)]^2_{i(j)} \right) = \frac{1}{n} \sum_{i=1}^{n} [\nabla f(x^j)]^2_i = \frac{1}{n} \|\nabla f(x^j)\|^2.$$

Thus taking expectation over *all* random variables $i(0), i(1), \ldots, i(j)$ encountered so far, and redefining $\phi_j = E[f(x^j) - f(x^*)]$, we obtain

$$\phi_{j+1} \leq \phi_j - \frac{1}{2nL_{\max}} E(\|\nabla f(x^j)\|^2).$$

The rest of the proof follows the full-gradient case (plus Jensen's inequality). We end with the sublinear rate:

$$E[f(x^j) - f(x^*)] \leq \frac{2nL_{\max}R_0^2}{j}.$$

Differs from the full-gradient result in that

- extra factor of n — not surprising, as we are using only "one-nth" of the full gradient;
- $L_{\max}$ replaces L (possibly $L_{\max} < L$).

Strongly Convex Case

When f is strongly convex with modulus σ, we have

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{\sigma}{2}\|y-x\|^2$$

By minimizing both sides of this expression w.r.t y, we get a different bound on $\|\nabla f(x^j)\|^2$:

$$\|\nabla f(x^j)\|^2 \geq 2\sigma[f(x^j) - f(x^*)].$$

Proceed as before to get a linear rate, in expectation:

$$E[f(x^j) - f(x^*)] \leq \left(1 - \frac{\sigma}{nL_{\max}}\right)^j [f(x^0) - f(x^*)].$$

Again, similar to the full-gradient rate in this case, except for the factor of n and the different Lipschitz constant.

Convergence: Convex, Cyclic

The "cyclic" choice of indices $1, 2, \ldots, n, 1, 2, \ldots, n, 1, 2, \ldots$ is perhaps the most intuitive. Early results reported in [Bertsekas, 1999], [Luo and Tseng, 1992] (and in [Tseng, 2001] for an "essentially cyclic" case).

Recent work [Beck and Tetruashvili, 2013] derives rates similar to the Randomized CD method. For convex smooth f with steps $\alpha_k \equiv L$ (where L is the usual Lipschitz constant) have sublinear

$$f(x^j) - f(x^*) \leq \frac{4L(n+1)R_0^2}{j + 8/n}.$$

For strongly convex case, have a linear rate:

$$f(x^j) - f(x^*) \leq \left(1 - \frac{\sigma}{2(n+1)L}\right)^k [f(x^0) - f(x^*)].$$

Sublinear vs Linear Convergence

Sublinear convergence (e.g. error decreases like C/j or C/j^2 in iteration number j, for some constant C) is slow.

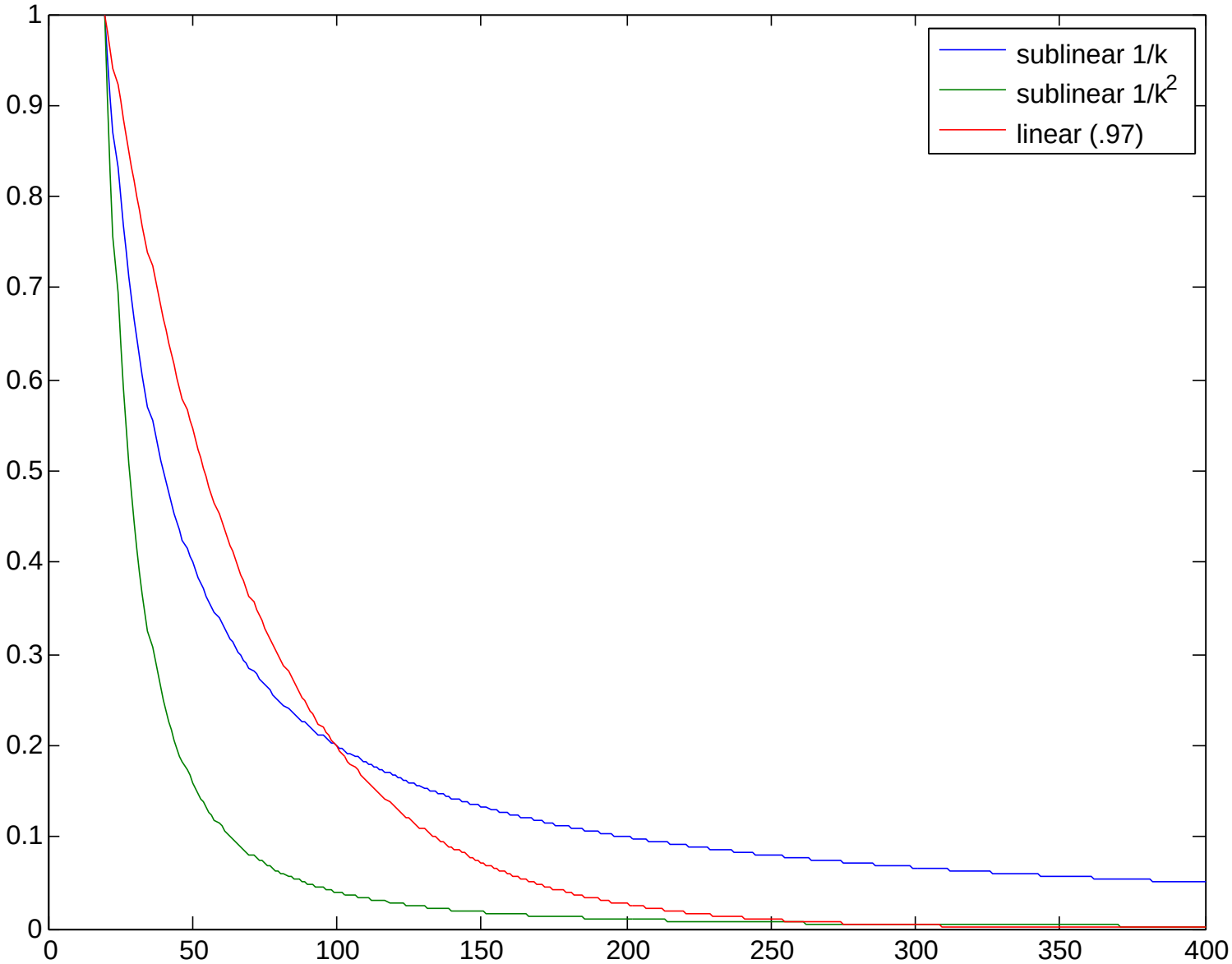
To achieve a specified accuracy ϵ, need $O(C/\epsilon)$ iterations for a $1/k$ process, and $O(\sqrt{C/\epsilon})$ for a $1/k^2$ process.

Linear convergence is asymptotically faster. If error decreases like $(1-\delta)^k$ for some small positive δ, need $O(|\log \epsilon|/\delta)$ iterates to reach accuracy ϵ.

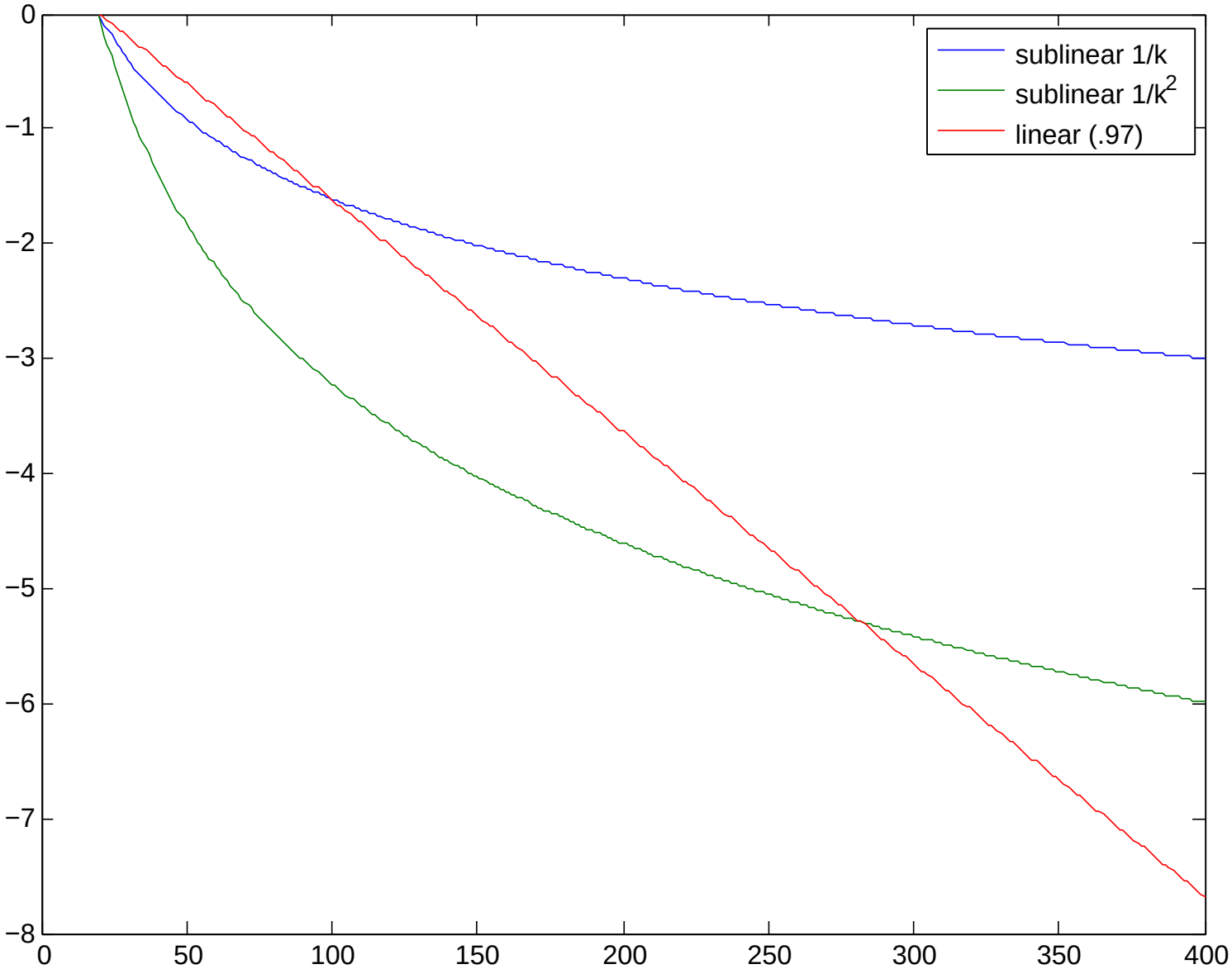
Sublinear rates were often viewed almost as uninteresting. But some modern applications need only a low-accuracy solution, and a sublinear algorithm may suffice.

(The constants in the $O()$ terms and the cost per iteration are significant.)

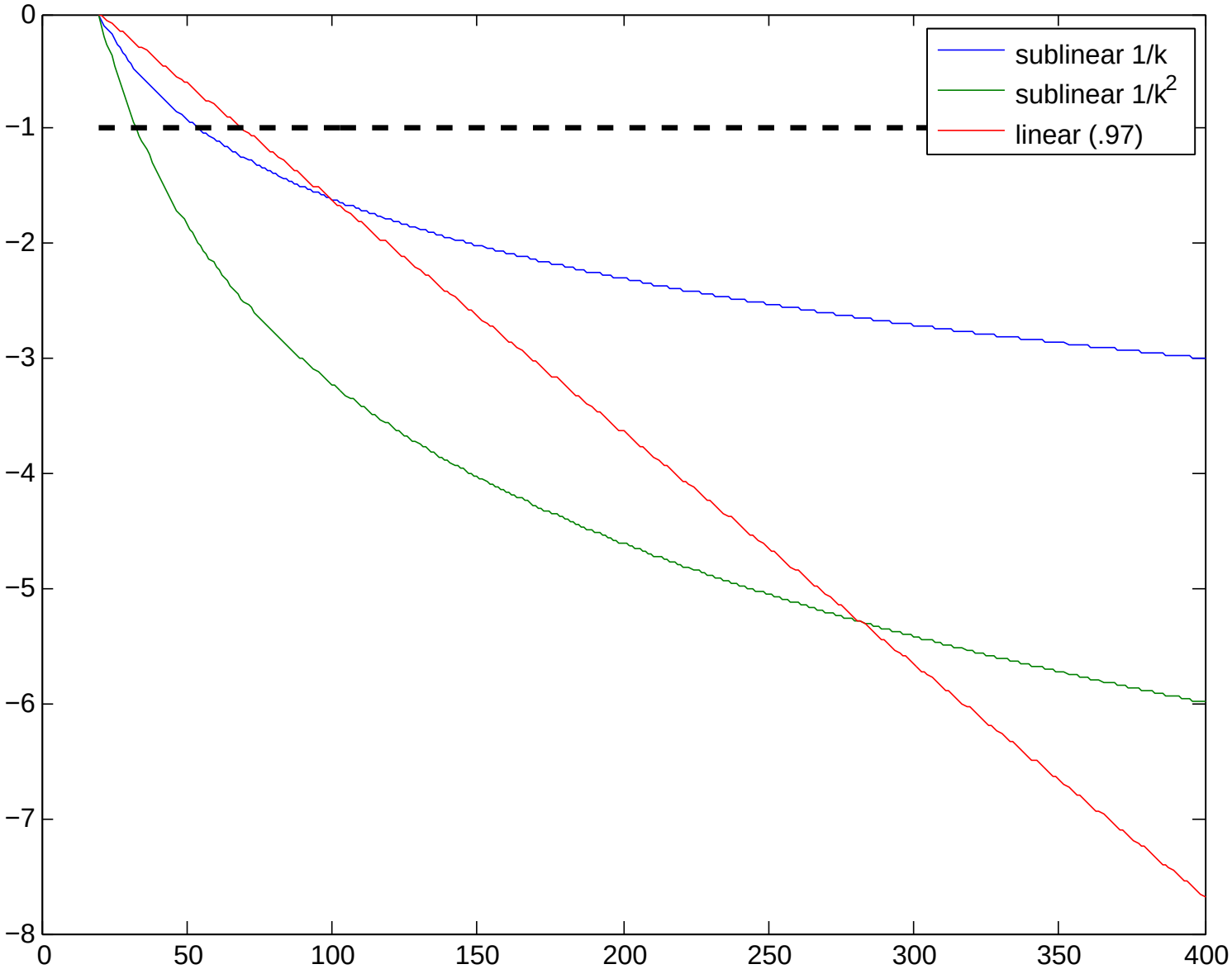
Sublinear vs Linear Convergence



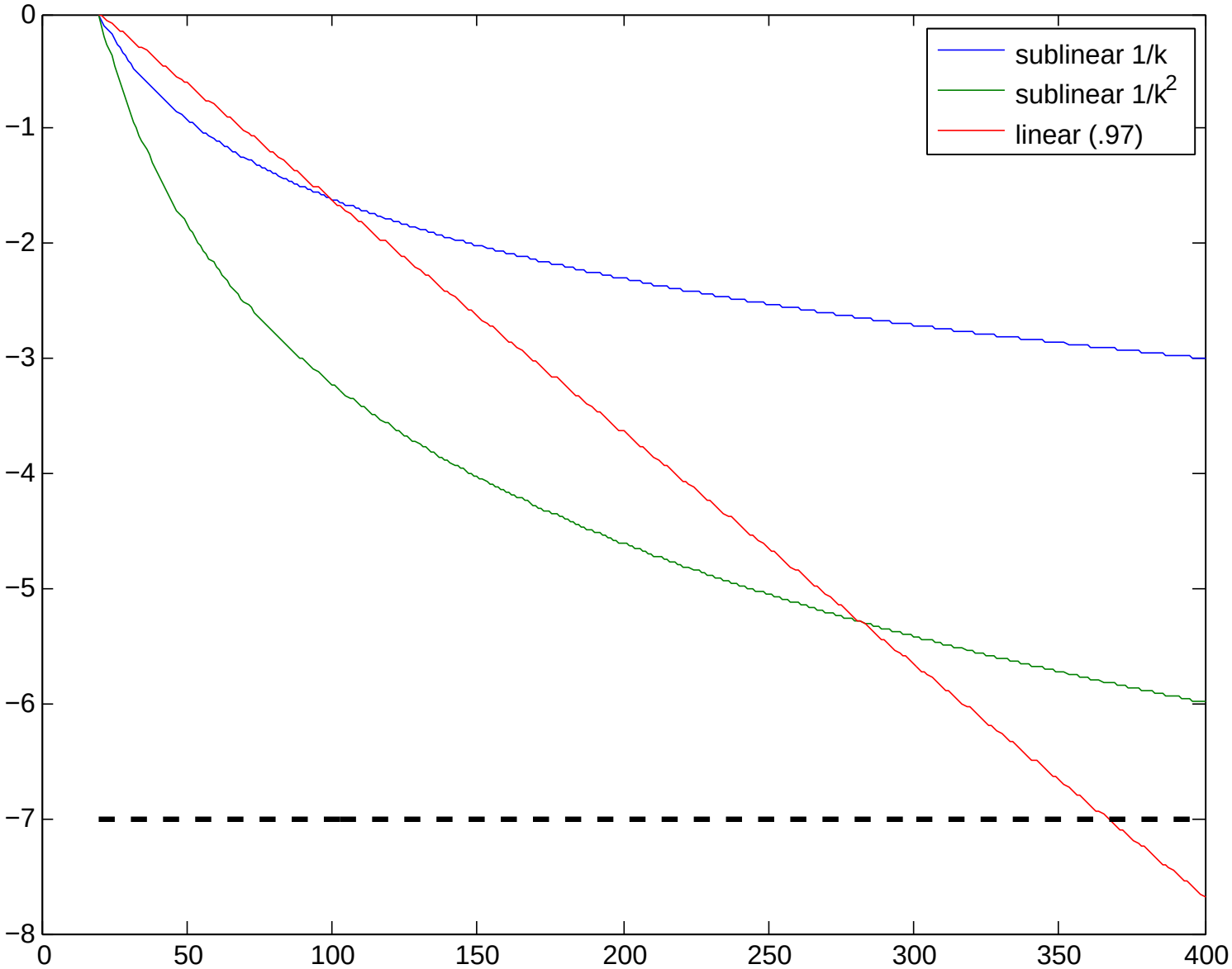
Sublinear vs Linear Convergence



Sublinear vs Linear Convergence



Sublinear vs Linear Convergence



Kaczmarz Algorithm for $Ax = b$.

Consider linear equations $Ax = b$, where the equations are consistent and matrix A is $m \times n$, not necessarily square or full rank. Write

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix}, \quad \text{where } \|A_i\|_2 = 1, \quad \forall_i \quad \text{(rows } A_i \text{ are normalized)}.$$

Iteration j of Randomized Kaczmarz:

- Select row index $i(j) \in \{1, 2, \ldots, m\}$ randomly with equal probability.
- Set

$$x^{j+1} \leftarrow x^j - (A_{i(j)} x^j - b_{i(j)}) A_{i(j)}^T.$$

Project x onto the plane of equation $i(j)$, so that $A_{i(j)} x^{j+1} = b_{i(j)}$.

Kaczmarz and CD

Kaczmarz is actually coordinate descent applied to a dual formulation!

Seek a *least-norm* solution of the system $Ax = b$:

$$\min_x \frac{1}{2}\|x\|^2 \quad \text{s.t. } Ax = b.$$

Dual of this problem is

$$\min_z \frac{1}{2}\|A^T z\|^2 - b^T z,$$

with optimal primal and dual solutions related through $x^* = A^T z^*$.

The CD step on the dual at iteration j, with step $\alpha_j = 1$, is

$$z^{j+1} \leftarrow z^k - (A_{i(j)} A^T z^j - b_{i(j)}) e_{i(j)},$$

where $i(j) \in \{1, 2, \ldots, m\}$.

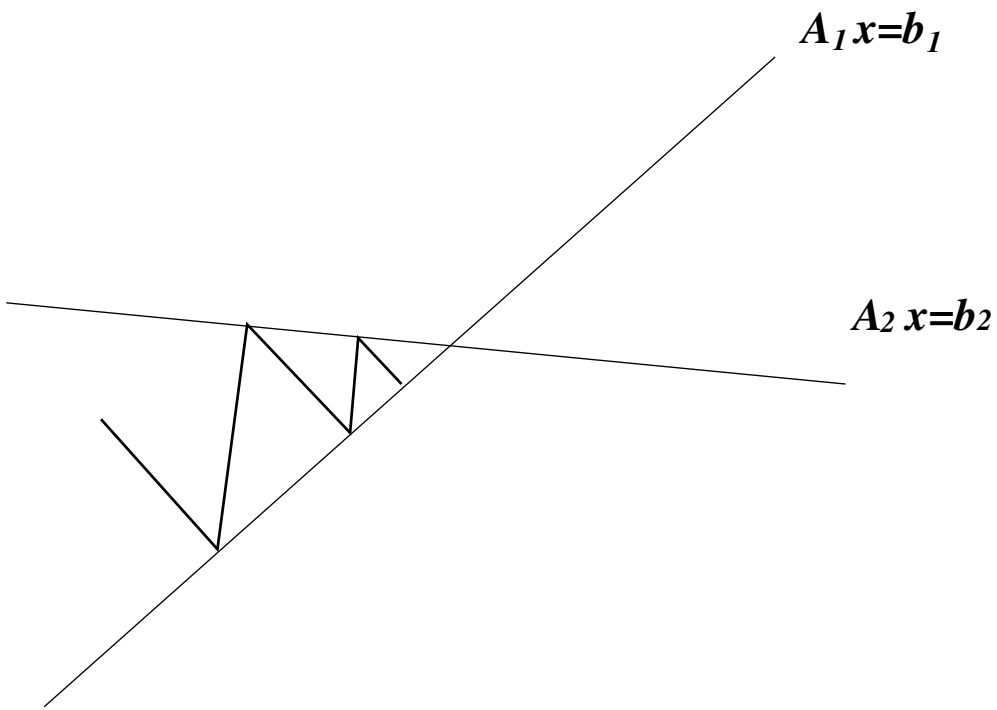
If instead of generating the sequence z^j, $j = 0, 1, 2 \ldots$ we maintain the transformed sequence $x^j := A^T z^j$, $j = 0, 1, 2, \ldots$.

Multiplying both sides by A^T, CD update formula becomes

$$A^T z^{j+1} = A^T z^j - (A_{i(j)} A^T z^j - b_{i(j)}) A_{i(j)}^T,$$

which is equivalent to the Kaczmarz update:

$$x^{j+1} = x^j - (A_{i(j)} x^j - b_{i(j)}) A_{i(j)}^T.$$



Randomized Kaczmarz: Simplified Convergence Analysis

Recall that $\|A_i\| = 1$ for all i. $\lambda_{\min,\mathrm{nz}}$ denotes minimum nonzero eigenvalue of $A^T A$. $P(\cdot)$ is projection onto solution set.

$$\begin{aligned}\frac{1}{2}\|x^{j+1} - P(x^{j+1})\|^2 &\leq \frac{1}{2}\|x^j - A_{i(j)}^T(A_{i(j)}x^j - b_{i(j)}) - P(x^j)\|^2 \\ &= \frac{1}{2}\|x^j - P(x^j)\|^2 - \frac{1}{2}(A_{i(j)}x^j - b_{i(j)})^2.\end{aligned}$$

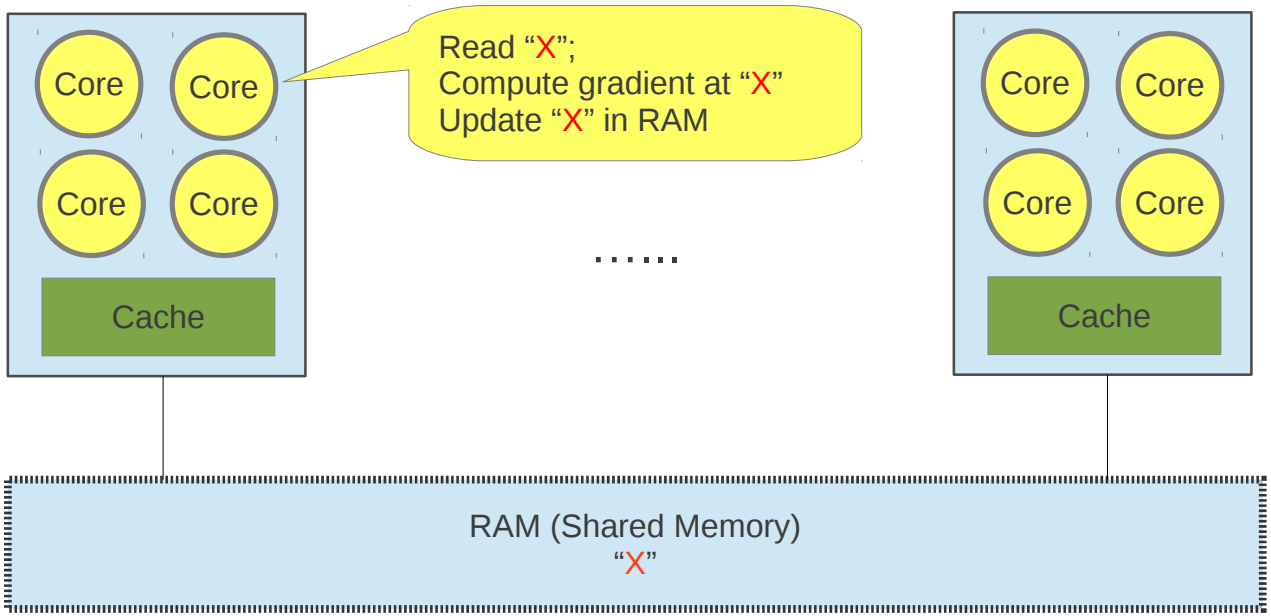
Taking expectations:

$$\begin{aligned}E\left[\frac{1}{2}\|x^{j+1} - P(x^{j+1})\|^2 \,|\, x^j\right] &\leq \frac{1}{2}\|x^j - P(x^j)\|^2 - \frac{1}{2}E\left[(A_{i(j)}x^j - b_{i(j)})^2\right] \\ &= \frac{1}{2}\|x^j - P(x^j)\|^2 - \frac{1}{2m}\|Ax^j - b\|^2 \\ &\leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m}\right)\frac{1}{2}\|x^j - P(x^j)\|^2.\end{aligned}$$

[Strohmer and Vershynin, 2009]. Linear rate, as for general case, but doesn't require strong convexity.

Asynchronous Parallel Optimization

Figure: *Asynchronous* parallel setup used in HOGWILD! [Niu, Recht, Ré, and Wright, 2011]



- All cores share the same memory, containing the variable x;
- All cores run the same optimization algorithm independently;
- All cores update the coordinates of x concurrently *without* any software locking.

We use the same model of computation for our asynchronous algorithms.

Asynchronous Random Kaczmarz (Liu, Wright, 2014)

Assumes that x is stored in shared memory, accessible to all cores.

Each core runs a simple process, repeating indefinitely:

- Choose index $i \in \{1, 2, \ldots, m\}$ uniformly at random;
- Choose component $t \in \text{supp}(A_i)$ uniformly at random;
- Read the $\text{supp}(A_i)$-components of x (from shared memory), needed to evaluate $A_i x$;
- Update the t component of x:

$$x_t \leftarrow x_t - \gamma \|A_i\|_0 (A_i)_t (A_i x - b_i)$$

  for some step size γ (a unitary operation);

Note that x can be updated by other cores between the time it is read and the time that the update is performed.

Differs from Randomized Kaczmarz in that each update is using outdated information and we update just a single component of x (in theory).

AsyRK: Global View

From a "central" viewpoint, aggregating the actions of the individual cores, we have the following: At each iteration j:

- Select $i(j)$ from $\{1, 2, \ldots, m\}$ with equal probability;
- Select $t(j)$ from the support of $A_{i(j)}$ with equal probability;
- Update component $t(j)$:

$$x^{j+1} = x^j - \gamma \|A_{i(j)}\|_0 (A_{i(j)} x^{k(j)} - b_{i(j)}) A_{i(j),t(j)} e_{t(j)},$$

where $k(j)$ is some iterate prior to j but no more than τ cycles old:

$$j - k(j) \leq \tau.$$

If all computational cores are roughly the same speed, we can think of the delay τ as being similar to the number of cores.

Consistent Reading

Assumes consistent reading, that is, the $x^{k(j)}$ used to evaluate the residual is an x that actually existed at some point in the shared memory.

(This condition may be violated if two or more updates happen to the $\text{supp}(A_{i(j)})$-components of x while they are being read.)

When the vectors A_i are sparse, inconsistency is not too frequent.

More on this later!

AsyRK Analysis: A Key Element

Key parameters:

- $\mu := \max_{i=1,2,\ldots,m} \|A_i\|_0$ (maximum nonzero row count);
- $\alpha := \max_{i,t} \|A_i\|_0 \|(Ae_t)A_{i,t}\| \leq \mu\|A\|$;
- $\lambda_{\max} =$ max eigenvalue of $A^T A$.

Idea of analysis: Choose some $\rho > 1$ and choose steplength γ small enough that

$$\rho^{-1}\mathbb{E}(\|Ax^j - b\|^2) \leq \mathbb{E}(\|Ax^{j+1} - b\|^2) \leq \rho\mathbb{E}(\|Ax^j - b\|^2).$$

Not too much change to the residual at each iteration. Hence, don't pay too much of a price for using outdated information.

But don't want γ to be too tiny, otherwise overall progress is too slow.

Strike a balance!

Main Theorem

Theorem

Choose any $\rho > 1$ and define γ via the following:

$$\psi = \mu + \frac{2\lambda_{\max}\tau\rho^{\tau}}{m}$$

$$\gamma \leq \min\left\{\frac{1}{\psi},\ \frac{m(\rho-1)}{2\lambda_{\max}\rho^{\tau+1}},\ m\sqrt{\frac{\rho-1}{\rho^{\tau}(m\alpha^2+\lambda_{\max}^2\tau\rho^{\tau})}}\right\}$$

Then have

$$\rho^{-1}\mathbb{E}(\|Ax^j-b\|^2) \leq \mathbb{E}(\|Ax^{j+1}-b\|^2) \leq \rho\mathbb{E}(\|Ax^j-b\|^2)$$

$$\mathbb{E}(\|x^{j+1}-P(x^{j+1})\|^2) \leq \left(1-\frac{\lambda_{\min,\mathrm{nz}}\gamma}{m\mu}(2-\gamma\psi)\right)\mathbb{E}(\|x^j-P(x^j)\|^2),$$

A particular choice of ρ leads to simplified results, in a reasonable regime.

A Particular Choice

Corollary

Assume

$$2e\lambda_{\max}(\tau+1) \leq m$$

and set $\rho = 1 + 2e\lambda_{\max}/m$. *Can show that* $\gamma = 1/\psi$ *for this case, so expected convergence is*

$$\mathbb{E}(\|x^{j+1} - P(x^{j+1})\|^2) \leq \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m(\mu+1)}\right) \mathbb{E}(\|x^j - P(x^j)\|^2).$$

In the regime $2e\lambda_{\max}(\tau+1) \leq m$ considered here the delay τ doesn't really interfere with convergence rate. In this regime, speedup is linear in the number of cores!

Rate is consistent with serial randomized Kaczmarz

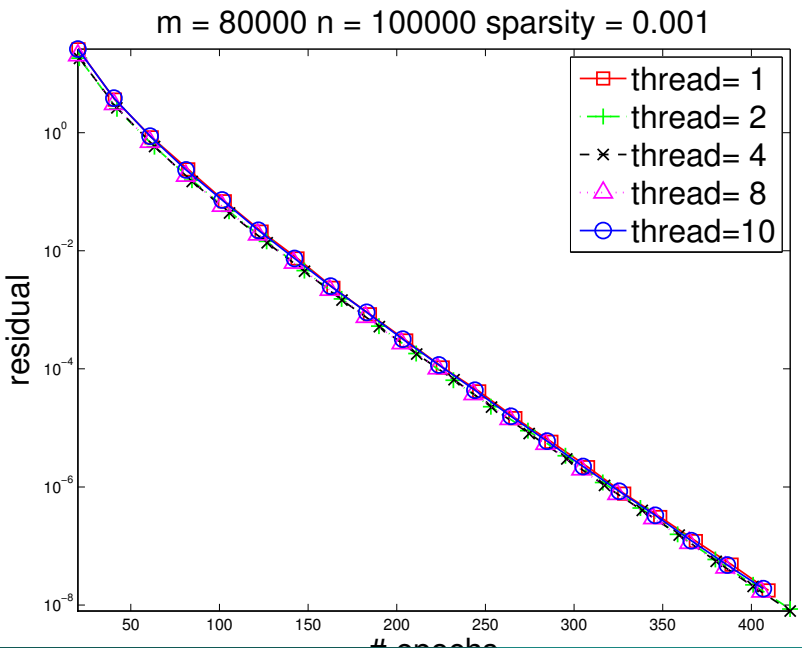
AsyRK: Near-Linear Speedup

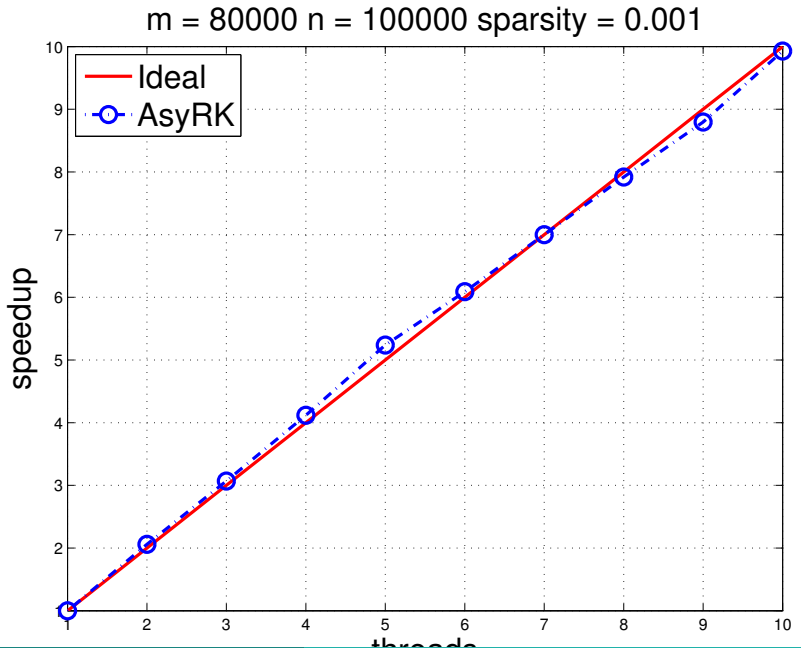
Run on an Intel Xeon 40-core machine. Used one socket — 10 cores).

Diverges a bit from the analysis:

- We update *all* components of x for $A_{i(j)}^T$ (not just component t);
- Choose $i(j)$ randomly, but use sampling without replacement to work through the rows of A, reordering after each "epoch"

Sparse Gaussian random matrix $A \in \mathbb{R}^{m \times n}$ with $m = 100000$ and $n = 80000$, sparsity $\delta = .001$. See linear speedup.

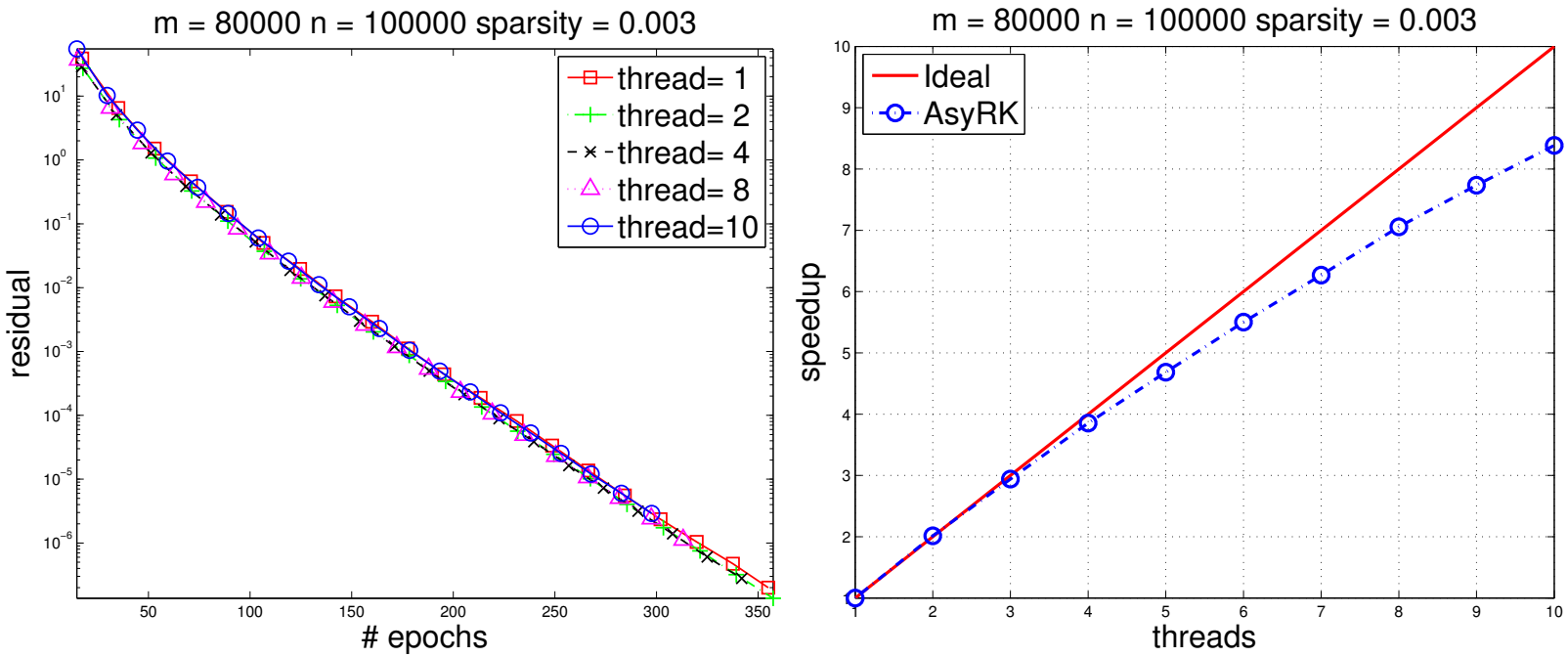




AsyRK: Near-Linear Speedup

Sparse Gaussian random matrix $A \in \mathbb{R}^{m \times n}$ with $m = 100000$ and $n = 80000$, sparsity $\delta = .003$.

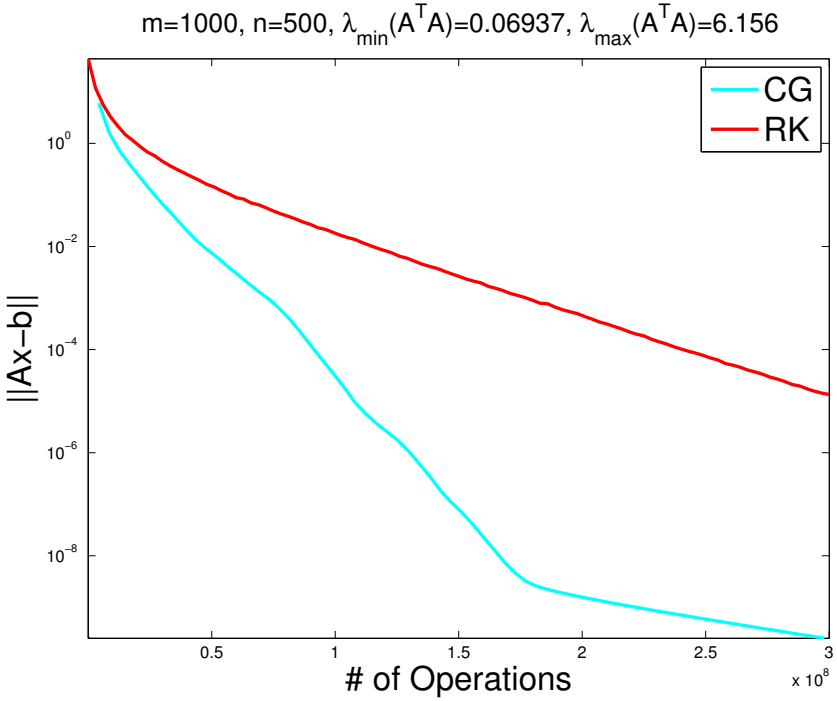
See slight dropoff from linear speedup for this slightly less-sparse problem.

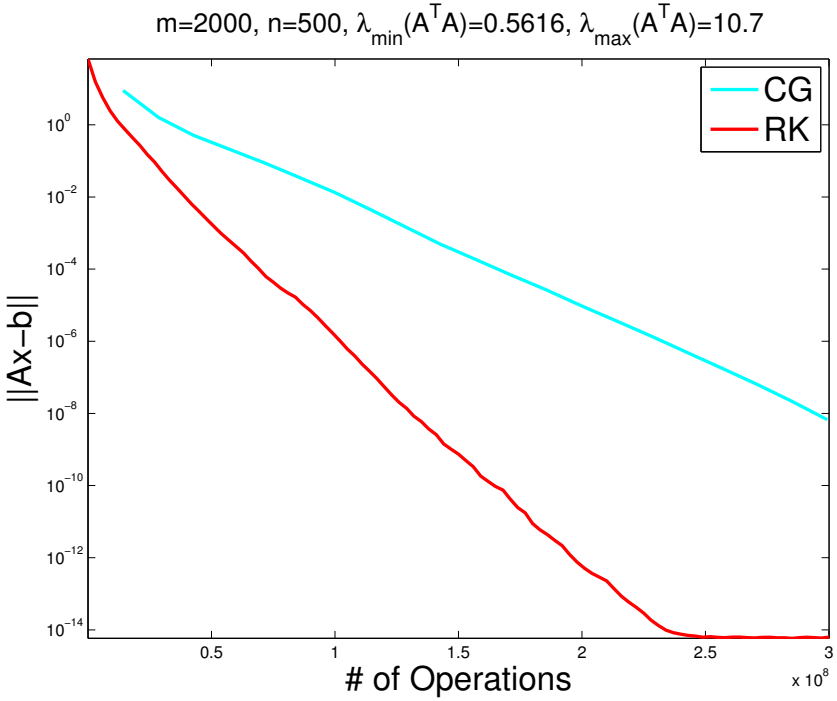


(Runtime: 18.4 seconds on 10 cores.)

RK vs Conjugate Gradient

We compare serial implementations of RK and CG. (The benefits of multicore implementation are similar for both.) Random A, $\delta = .1$.





CG does better in the more ill-conditioned case, probably due to nice distribution of dominant eigenvalues of $A^T A$. (Note slower convergence in later stages.) RK is competitive in the well-conditioned case.

Regularized Objectives

Many modern applications of optimization seek *approximate* minimizers of f with *desirable structure*. One way to achieve this is to add a *regularizer* $\Omega(x)$ to the objective:

$$\min_x \ f(x) + \lambda \Omega(x).$$

- $\Omega(x)$ is chosen to impose the desired structure on x;
- $\lambda \geq 0$ is a *regularization parameter* that controls the amount of regularization.

$$\lambda \text{ large} \Rightarrow \text{more emphasis on structure;}$$
$$\lambda \text{ small} \Rightarrow \text{more emphasis on optimizing } f.$$

Define $g(x) = \lambda \Omega(x)$ in the discussion below.

Regularized Formulation

$$\min_x : \ F(x) := f(x) + g(x) \tag{1}$$

- $f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$ is convex and differentiable;
- $g(\cdot) : \mathbb{R}^n \mapsto \mathbb{R} \cup \{+\infty\}$ is a proper closed convex real value extended function;
- $g(x)$ is separable: $g(x) = \sum_{i=1}^n g_i(x_i)$, $g_i(\cdot) : \mathbb{R} \mapsto \mathbb{R} \cup \{+\infty\}$.

Instances of $g(x)$:

- Unconstrained: $g(x) =$ constant.
- Box constrained: $g(x) = \sum_{i=1}^n \mathbf{1}_{[a_i,b_i]}(x_i)$ where $\mathbf{1}_{[a_i,b_i]}$ is an indicator function for $[a_i, b_i]$;
- ℓ_p norm regularization: $g(x) = \|x\|_p^p$ where $p \geq 1$.

Instances

Problems that fit this framework include the following:

- least squares: $\min_x \quad \frac{1}{2}\|Ax - b\|^2$;
- LASSO: $\min_x \quad \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$;
- support vector machine (SVM) with squared hinge loss:

$$\min_w \quad C\sum_i \max\{y_i(x_i^T w - b), 0\}^2 + \frac{1}{2}\|w\|^2$$

- support vector machine: dual form with bias term:

$$\min_{0\leq\alpha\leq C\mathbf{1}} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i.$$

Instances (continued)

- logistic regression with ℓ_p norm regularization ($p = 1, 2$):

$$\min_x \quad \frac{1}{n} \sum_i \log(1 + \exp(-y_i x_i^T w)) + \lambda \|w\|_p^p$$

- semi-supervised learning (Tikhonov Regularization)

$$\min_f \quad \sum_{i \in \{\text{labeled data}\}} (f_i - y_i)^2 + \lambda f^T L f$$

  where L is the Laplacian matrix.

- relaxed linear program:

$$\min_{x \geq 0} \quad c^T x \quad s.t. \quad Ax = b \quad \Rightarrow \quad \min_{x \geq 0} \quad c^T x + \lambda \|Ax - b\|^2$$

Stochastic Proximal Coordinate Descent SPCD

Define prox-operator $\mathcal{P}_h$ for a convex function h:

$$\mathcal{P}_h(y) = \arg\min_x \frac{1}{2}\|x - y\|^2 + h(x).$$

(It's nonexpansive: $\|\mathcal{P}_h(y) - \mathcal{P}_h(z)\| \leq \|y - z\|$.)

Basic Step: Select a coordinate i and compute the coordinate gradient $\nabla_i f(x)$; take a step along this direction and "shrink" to account for g_i.

$$x_i \leftarrow \mathcal{P}_{\alpha g_i}(\underbrace{x_i - \alpha[\nabla f(x)]_i}_{\text{coordinate gradient}}),$$

for some step length α.

This is equivalent to solving an approximate version of the coordinate-i problem in which f is replaced by a simple quadratic:

$$\min_{z_i} \nabla_i f(x)^T[z_i - x_i] + \frac{1}{2\alpha}[z_i - x_i]^2 + g_i(z_i).$$

Prox-Operator Examples

Prox-operators can be executed efficiently in many cases.

- $g_i(t) = |t|$: soft thresholding operation

$$\mathcal{P}_{\lambda g_i}(t) = \text{sgn}(t)\max\{|t| - \lambda, 0\}.$$

- $g_i(t) = \mathbf{1}_{[a,b]}$: projection operation

$$\mathcal{P}_{\lambda g_i}(t) = \arg\min_{s\in[a,b]} \frac{1}{2}\|s - t\|^2 = \text{mid}(a, b, t).$$

- g_i null (no regularization on component i): Then $\mathcal{P}_{\lambda g_i}(t) = t$. The basic CD step is then

$$x_i \leftarrow x_i - \alpha[\nabla f(x)]_i,$$

so it reduces to the non-regularized case that we discussed above.

Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm (AsySPCD)

All processors run a stochastic coordinate descent process concurrently and without synchronization:

- Select a coordinate $i \in \{1, 2, \ldots, n\}$ uniformly at random;
- Read "x" from the shared memory and compute the i gradient component using "x":

$$d_i \leftarrow [\nabla f(x)]_i;$$

- Update "x" in the shared memory by the proximal operation, performed atomically:

$$x_i \leftarrow \mathcal{P}_{(\gamma/L_{\max})g_i}\left(x_i - \frac{\gamma}{L_{\max}} d_i\right), \quad \text{for some } \gamma > 0.$$

Global View of AsySPCD

Global counter j incremented when one of the cores makes an update:

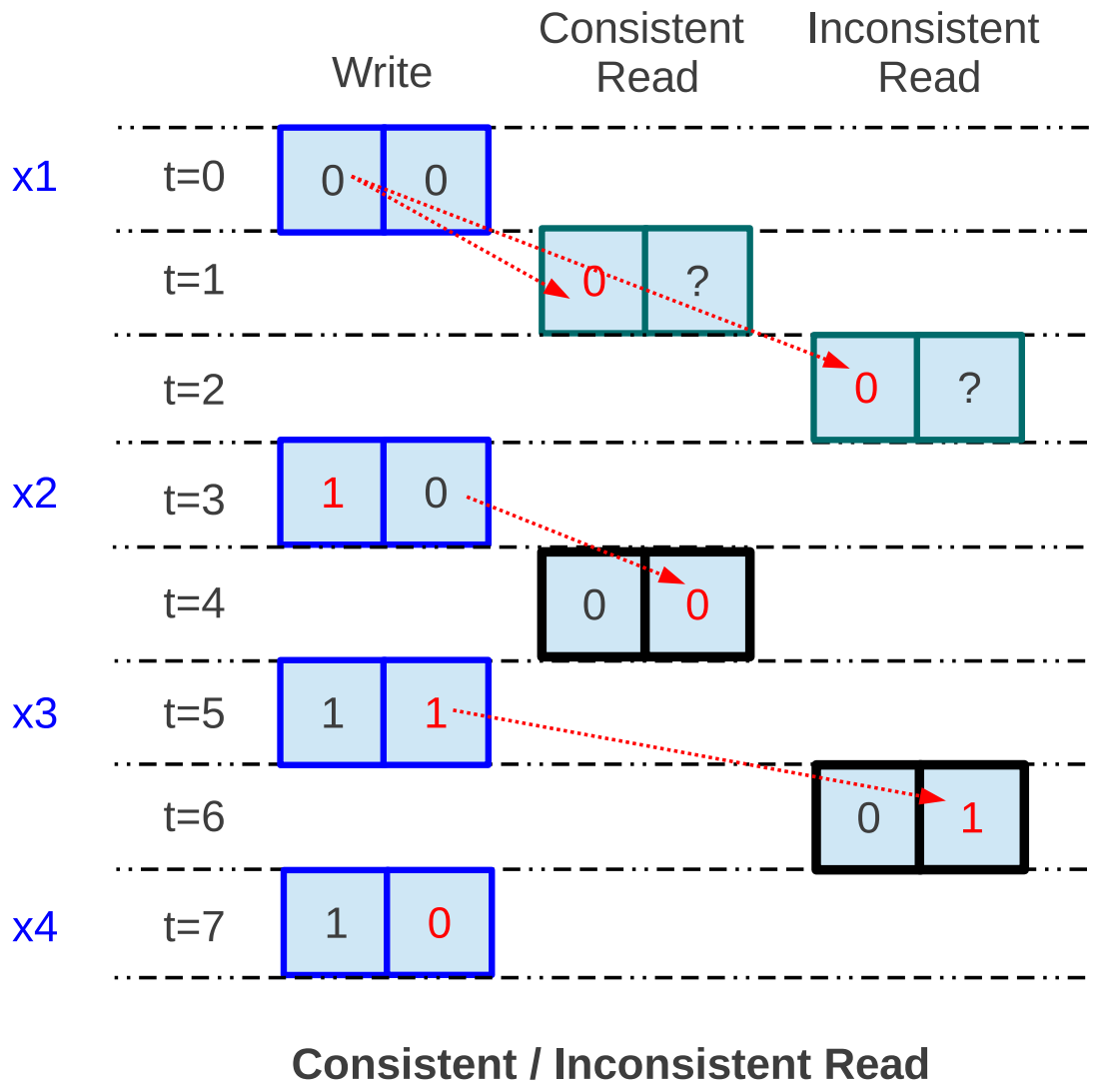
- Choose $i(j) \in \{1, 2, \cdots, n\}$ uniformly at random;
- Read components of x from shared memory needed to compute $[\nabla f(x)]_{i(j)}$, denoting the local version of x by $\hat{x}^j$;
- Update compoment $i(j)$ of x (atomically):

$$x_{i(j)}^{j+1} \leftarrow \mathcal{P}_{(\gamma/L_{\max})g_{i(j)}}\left(x_{i(j)}^{j} - \frac{\gamma}{L_{\max}}[\nabla f(\hat{x}^j)]_{i(j)}\right).$$

Note that $\hat{x}^j$ may never appear in shared memory at any point in time. The elements of x may have been updated repeatedly during reading of $\hat{x}^j$, which means that the components of $\hat{x}^j$ may have different "ages."

We call this phenomenon **inconsistent read**.

Consistent Read vs. Inconsistent Read



Expressing Read-Inconsistency

Difference between $\hat{x}^j$ and x^j is expressed in terms of "missed updates:"

$$x^j = \hat{x}^j + \sum_{t \in K(j)} (x^{t+1} - x^t)$$

where $K(j)$ defines the iterate set of updates missed in reading $\hat{x}^j$.

We assume τ to be the upper bound of ages of all elements in $K(j)$:

$$\tau \geq j - \min\{t \mid t \in K(j)\}.$$

Example: our assumptions would be satisfied with $\tau = 10$ when

$$x^{100} = \hat{x}^{100} + \sum_{t \in \{91,95,98,99\}} (x^{t+1} - x^t)$$

τ is related strongly to the number of cores / processors that can be used in the computation. The number of updates we would expect to miss between reading and updating x is about equal to the number of cores.

Notation

Recall that

- $L_{\max}$: component Lipschitz constant ("max diagonal of Hessian")

$$\|[\nabla f(x + te_i)]_i - [\nabla f(x)]_i\| \leq L_{\max}|t| \quad \forall x, t, i;$$

- L_{res}: restricted Lipschitz constant ("max row-norm of Hessian")

$$\|\nabla f(x + te_i) - \nabla f(x)\|_2 \leq L_{\text{res}}|t| \quad \forall x, t, i;$$

- $\Lambda := L_{\text{res}}/L_{\max}$ measures the degree of diagonal dominance.
  - 1 for separable f,
  - 2 for convex quadratic f with diagonally dominant Hessian,
  - $\sqrt{n}$ for general quadratic.
- S: the solution set of (1);

Key to Analysis

Recall iteration:

$$x_{i(j)}^{j+1} = \mathcal{P}_{(\gamma/L_{\max})g_{i(j)}}\left(x_{i(j)}^{j} - \frac{\gamma}{L_{\max}}[\nabla f(\hat{x}^j)]_{i(j)}\right).$$

Choose some $\rho > 1$ and choose γ so that

$$\mathbb{E}(\|x^j - x^{j-1}\|^2) \leq \rho\mathbb{E}(\|x^{j+1} - x^j\|^2) \quad \text{"}\rho\text{-condition"}.$$

Not too much change in the step at each iteration
$\Rightarrow$ not too much change in the gradient
$\Rightarrow$ not too much price to pay for using outdated information.

Want to choose γ small enough to satisfy this property but large enough to get steady convergence.

Strike a balance, as in asynchronous randomized Kaczmarz.

Main Assumption: Optimal Strong Convexity (OSC)

Optimal strong convexity parameter $\sigma > 0$

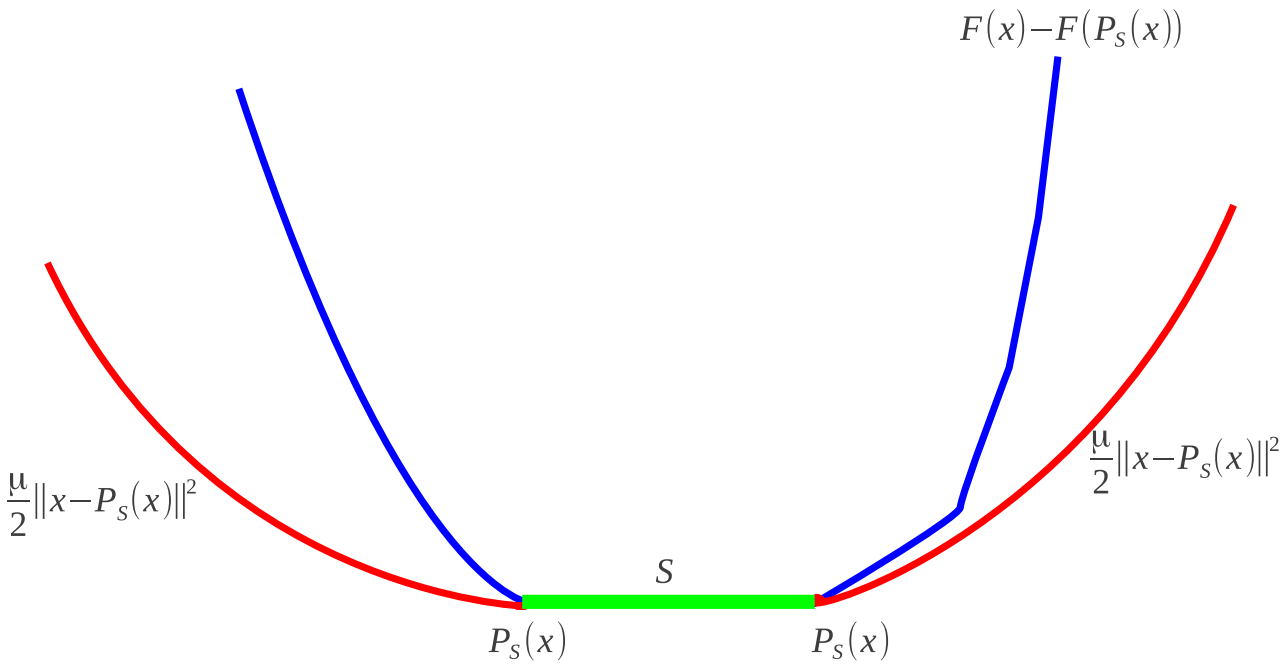
$$F(x) - F(\mathcal{P}_S(x)) \geq \frac{\sigma}{2}\|x - \mathcal{P}_S(x)\|^2$$

for all $x \in \mathrm{dom} F$.

Weaker than usual strong convexity — allows nonunique solutions.

- $F(x) = f(Ax)$ with strongly convex f.
- Squared hinge loss: $F(x) = \sum_k \max(a_k^T x - b_k,\ 0)^2$;

An OSC (but not strongly convex) function:



Main Theorem: OSC yields a Linear Rate

Theorem

For any $\rho > 1 + 4/\sqrt{n}$, define

$$\theta := \frac{\rho^{(\tau+1)/2} - \rho^{1/2}}{\rho^{1/2} - 1} \quad \theta' := \frac{\rho^{(\tau+1)} - \rho}{\rho - 1} \quad \psi := 1 + \frac{\tau\theta'}{n} + \frac{\Lambda\theta}{\sqrt{n}}.$$

and choose

$$\gamma \leq \min\left\{\frac{1}{\psi}, \frac{\sqrt{n}(1-\rho^{-1}) - 4}{4(1+\theta)\Lambda}\right\}.$$

Then the "ρ-condition" is satisfied at all j, and we have

$$\mathbb{E}\|x^j - \mathcal{P}_S(x^j)\|^2 + 2\gamma(\mathbb{E}F(x^j) - F^*)$$
$$\leq \left(1 - \frac{\sigma}{n(I + \gamma^{-1})}\right)^j \left(\|x^0 - \mathcal{P}_S(x^0)\|^2 + 2\gamma(F(x^0) - F^*)\right).$$

Notes on the Result

Rate depends intuitively on the various quantities involved:

- Smaller $\gamma \Rightarrow$ slower rate;
- Smaller $\sigma \Rightarrow$ slower rate;
- Larger $\Lambda = L_{\text{res}}/L_{\max}$ implies smaller γ and thus slower rate.
- Larger delay $\tau \Rightarrow$ slower rate.

Dependence on ρ is a bit more complicated, but best to choose ρ near its lower bound.

Special Case

Corollary

Consider the regime in which τ satisfies

$$4e\Lambda(\tau+1)^2 \leq \sqrt{n},$$

and define

$$\rho = \left(1 + \frac{4e\Lambda(\tau+1)}{\sqrt{n}}\right)^2.$$

Thus we can choose $\gamma = \frac{1}{2}$, and the rate simplifies to:

$$\mathbb{E}(F(x^j) - F^*) \leq \left(1 - \frac{\sigma}{n(l+2L_{\max})}\right)^j (L_{\max}\|x^0 - \mathcal{P}_S(x^0)\|^2 + F(x^0) - F^*).$$

If the diagonal dominance properties are good ($\Lambda \sim 1$) we have $\tau \sim n^{1/4}$.

In earlier work, with consistent read and no regularization, get $\tau \sim n^{1/2}$.

General Convex (without OSC): Sublinear Rate

Theorem

Define ψ and γ as in the main theorem, have

$$\mathbb{E}(F(x^j) - F^*) \leq \frac{n(L_{\max}\gamma^{-1}\|x^0 - \mathcal{P}_S(x^0)\|^2 + 2(F(x^0) - F^*))}{2(j+n)}.$$

Roughly "$1/j$" behavior (sublinear rate)

Corollary

Assuming $4e\Lambda(\tau+1)^2 \leq \sqrt{n}$ and setting ρ and $\gamma = 1/2$ as above, we have

$$\mathbb{E}(F(x^j) - F^*) \leq \frac{n(L_{\max}\|x^0 - \mathcal{P}_S(x^0)\|^2 + F(x^0) - F^*)}{j+n}.$$

Computational Experiments

Implemented on a 40-core Intel Xeon, containing 4 sockets $\times$ 10 cores.

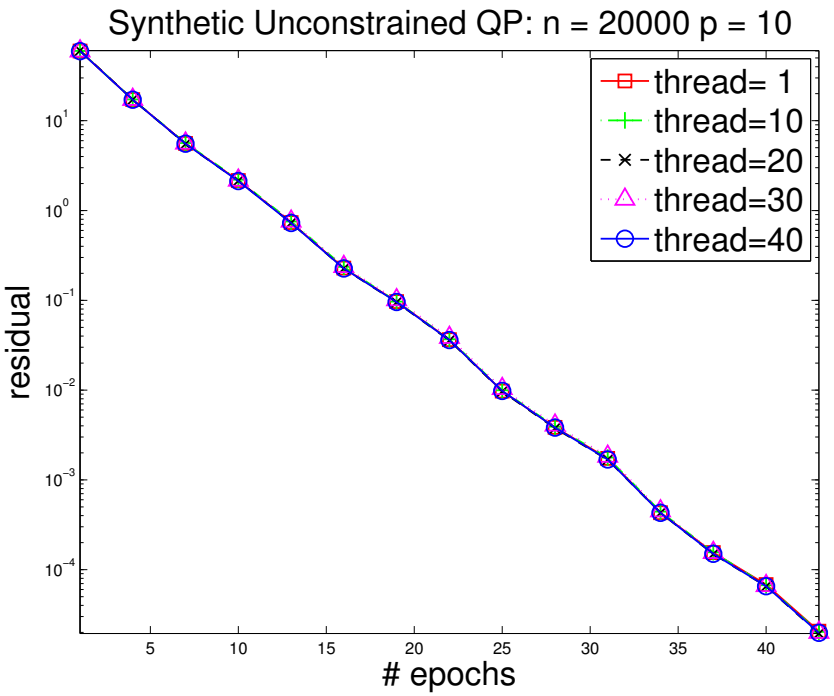
We don't do "sampling with replacement" as in the algorithm described above. Rather, each thread/core is assign a subset of gradient components, and sweeps through these in order: "sampling without replacement."

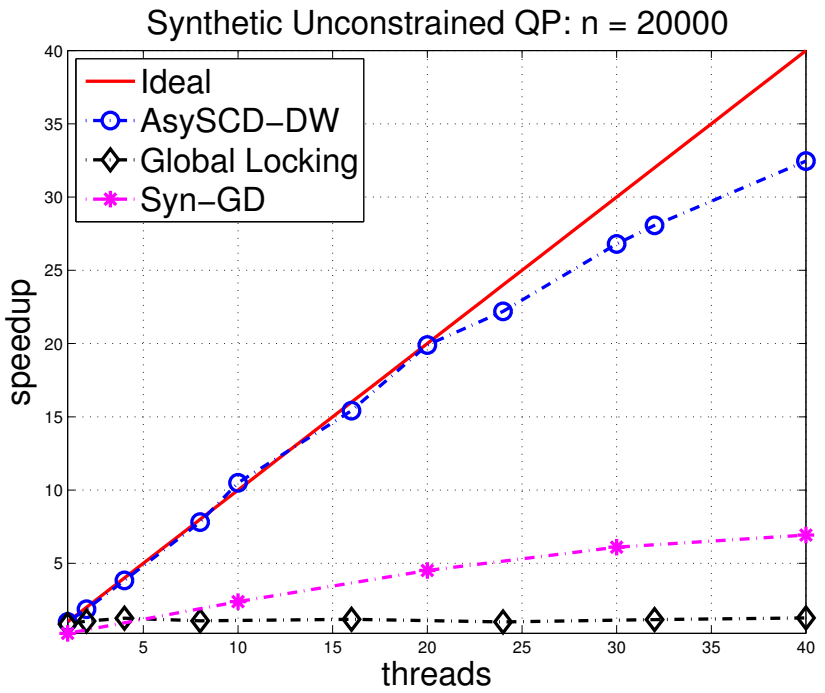
The order of indices is shuffled periodically - either between every pass, or less frequently.

Unconstrained: 4-socket, 40-core Intel Xeon

$$\min_x \quad \|Ax - b\|^2 + 0.5\|x\|^2$$

where $A \in \mathbb{R}^{m \times n}$ is a Gaussian random matrix ($m = 6000$, $n = 20000$, data size≈3GB, columns are normalized to 1). $\Lambda \approx 2.2$. Choose $\gamma = 1$. 3-4 seconds to achieve the accuracy 10^{-5} on 40 cores.

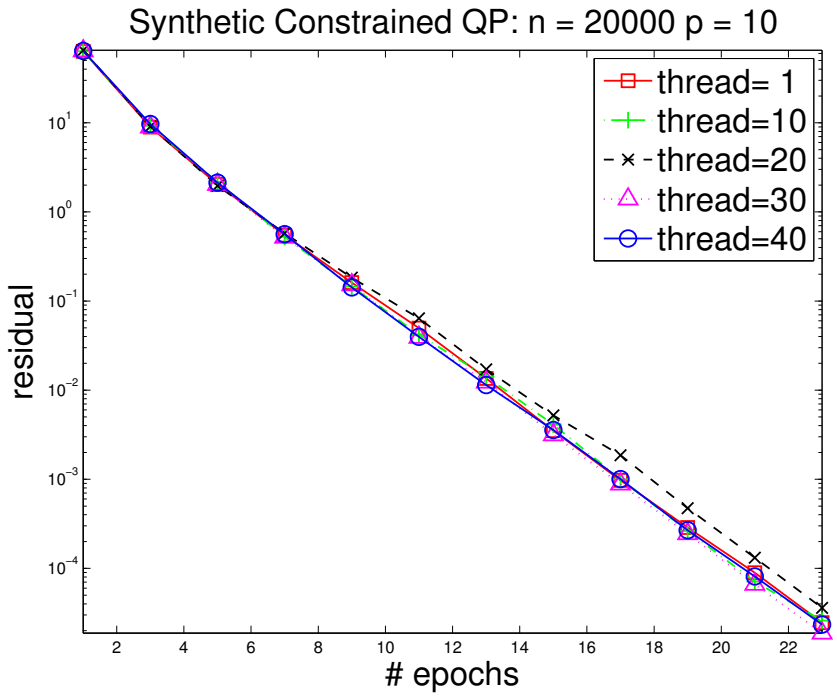


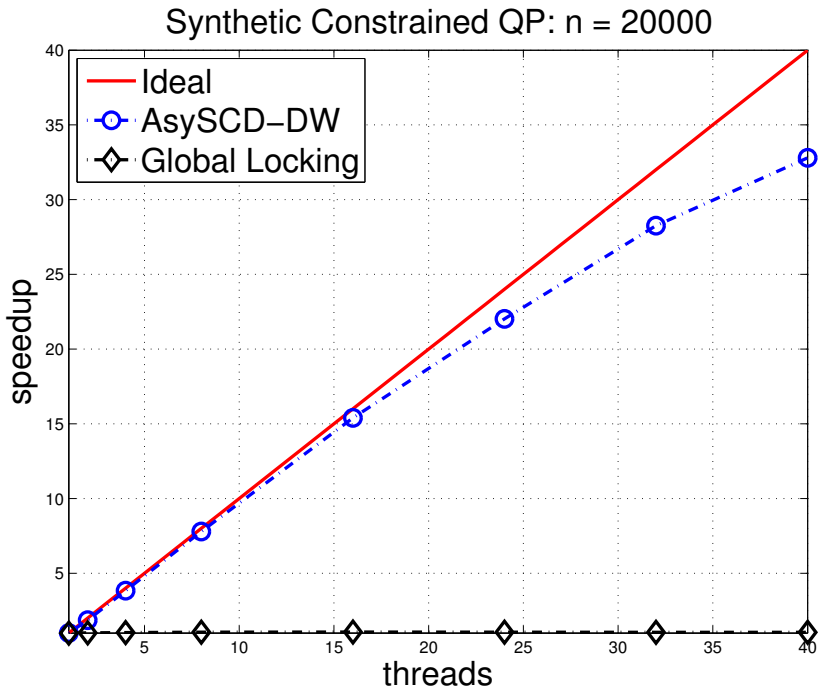


Constrained: 4-socket, 40-core Intel Xeon

$$\min_{x \geq 0} \quad (x - z)^T (A^T A + 0.5I)(x - z) \quad ,$$

where $A \in \mathbb{R}^{m \times n}$ is a Gaussian random matrix ($m = 6000$, $n = 20000$, columns are normalized to 1) and z is a Gaussian random vector. $L_{\text{res}}/L_{\max} \approx 2.2$. Choose $\gamma = 1$.

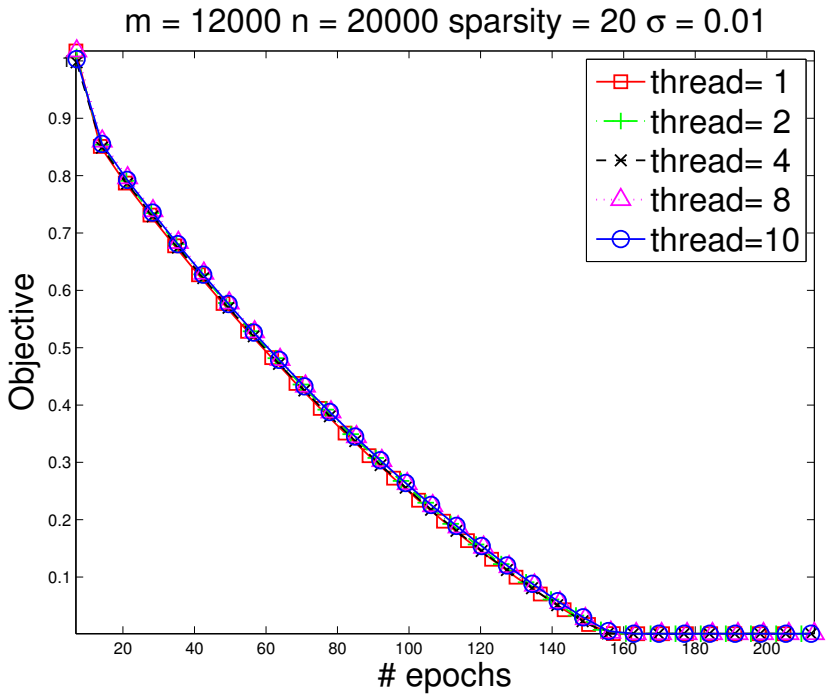


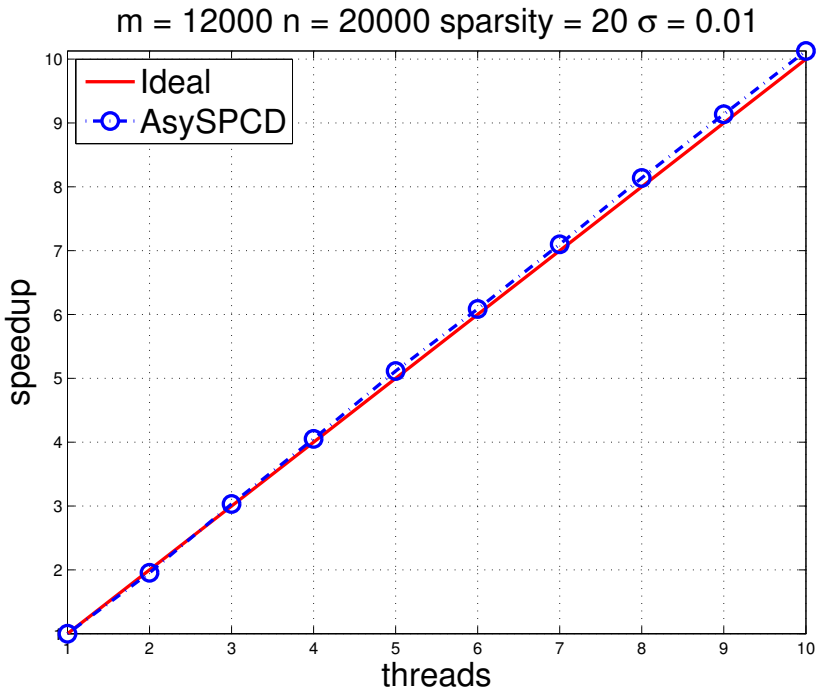


Experiments: 1-socket, 10-core Intel Xeon

$$\min_x \quad \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1,$$

where $A \in \mathbb{R}^{m\times n}$ is a Gaussian random matrix ($m = 12000$, $n = 20000$, data size≈3GB), $b = A * \text{sprandn}(n, 1, 20) + 0.01 * \text{randn}(n, 1)$, and $\lambda = 0.2\sqrt{m \log(n)}$. $L_{\text{res}}/L_{\max} \approx 2.2$. Choose $\gamma = 1$.





Block Coordinate Descent (BCD)

A common and useful extension is to update blocks of coordinates, rather than single coordinates.

At iteration j, select a subset $\mathcal{I}(j) \subset \{1, 2, \ldots, n\}$ and update just the components in $\mathcal{I}(j)$.

As for basic coordinate descent, there are different techniques for choosing $\mathcal{I}(j)$ (random, cyclic, essentially cyclic), and different strategies for updating the $\mathcal{I}(j)$ components of x.

Can do regularized block coordinate descent provided that the regularizer $\Omega(x)$ is separable with respect to $\mathcal{I}(j)$, that is,

$$\Omega(x) = \Omega_{\mathcal{I}(j)}(x_{\mathcal{I}(j)}) + \Omega_{\mathcal{I}(j)^c}(x_{\mathcal{I}(j)^c}).$$

Fairly straightforward to generalize the analysis of basic CD.

BCD also opens a path to synchronous parallel implementation, e.g. the components within $\mathcal{I}(j)$ can be updated simultaneously, in parallel.

Accelerated Variants

Accelerated first-order methods [Nesterov, 1983] have also become highly popular in recent years; the revival traces to [Nesterov, 2004] and [Beck and Teboulle, 2009].

They maintain multiple sequences of vectors $\{x^j\}$, $\{v^j\}$, $\{y^j\}$. Steps are combinations of all gradients encountered so far, not just the latest one. Convergence rates are faster e.g. $O(1/k^2)$ instead of $O(1/k)$.

[Nesterov, 2009] describes an accelerated methods that uses just a subvector of the gradient $\nabla f(x^j)$. But is has the apparent drawback of having to manipulate dense vectors, so at least $O(n)$ operations per iteration are required.

Recent work has shown that this approach can be implemented efficiently in some cases! An early idea for accelerated Kaczmarz is [Liu et al., 2013b], but [Lee and Sidford, 2013] have a much nicer idea. Uses a clever change of variables.

Applications

Mentioned several machine learning / statistics applications above. Also the quadratic penalty function for linear programming. Others:

- Positron emission tomography, optical diffusion tomography.
- Protein structure - adjusting dihedral angles in a protein chain so that the end of the chain is in a specified position.
- Gene expression studies (via logistic regression).
- Recovering origin-destination matrices from traffic observations.
- Functional MRI image analysis.
- Generalized linear models in statistics.
- Transceiver design via tensor optimization.
- Phase retrieval in X-ray crystallography.
- Self-calibrating sensing models: $y = A(\theta)x$.

Conclusions

- There is a renewed interest in coordinate descent methods, because of modern applications, particularly in machine learning.
- CD methods are a good match to parallel (multicore) computer architectures.
- We can analyze asynchronous parallel algorithms, with a computing model that approximates reality pretty well.

[Liu et al., 2013a], [Liu and Wright, 2014a], [Liu and Wright, 2014b].

References I


Beck, A. and Teboulle, M. (2009).
A fast iterative shrinkage-threshold algorithm for linear inverse problems.
SIAM Journal on Imaging Sciences, 2(1):183–202.


Beck, A. and Tetruashvili, L. (2013).
On the convergence of block coordinate descent methods.
SIAM Journal on Optimization, 23(4):2037–2060.


Bertsekas, D. P. (1999).
Nonlinear Programming.
Athena Scientific, second edition.


Lee, Y. T. and Sidford, A. (2013).
Efficient accelerated coordinate descent methods and faster algorihtms for solving linear systems.
In *54th Annual Symposium on Foundations of Computer Science*, pages 147–156.


Liu, J. and Wright, S. J. (2014a).
An asynchronous parallel randomized kaczmarz algorithm.
Technical report, Computer Sciences Department, University of Wisconsin-Madison.

References II

Liu, J. and Wright, S. J. (2014b).
Asynchronous stochastic coordinate descent: Parallelism and convergence properties.
Technical report, University of Wisconsin, Madison.
arXiv:1403.3862.

Liu, J., Wright, S. J., Ré, C., Bittorf, V., and Sridhar, S. (2013a).
An asynchronous parallel stochastic coordinate descent algorithm.
Technical Report arXiv:1311.1873, Computer Sciences Department, University of Wisconsin-Madison.
To appear in *Journal of Machine Learning Research*.

Liu, J., Wright, S. J., and Sridhar, S. (2013b).
An accelerated randomized Kaczmarz algorithm.
Technical Report arXiv 1310.2887, Computer Sciences Department, University of Wisconsin-Madison.
To appear in *Mathematics of Computation*.

Luo, Z. Q. and Tseng, P. (1992).
On the convergence of the coordinate descent method for convex differentiable minimization.
Journal of Optimization Theory and Applications, 72(1):7–35.

References III

Nesterov, Y. (1983).
A method for unconstrained convex problem with the rate of convergence $o(1/k^2)$.
Doklady AN SSSR, 269:543–547.

Nesterov, Y. (2004).
Introductory Lectures on Convex Optimization: A Basic Course.
Kluwer Academic Publishers.

Nesterov, Y. (2009).
Primal-dual subgradient methods for convex programs.
Mathematical Programming, Series B, 120:221–259.

Strohmer, T. and Vershynin, R. (2009).
A randomized Kaczmarz algorithm with exponential convergence.
Journal of Fourier Analysis and Applications, 15:262–278.

Tseng, P. (2001).
Convergence of a block coordinate descent method for nondifferentiable minimization.
Journal of Optimization Theory and Applications, 109(3):475–494.