Fast edge-preserving image denoising via group coordinate descent on the GPU

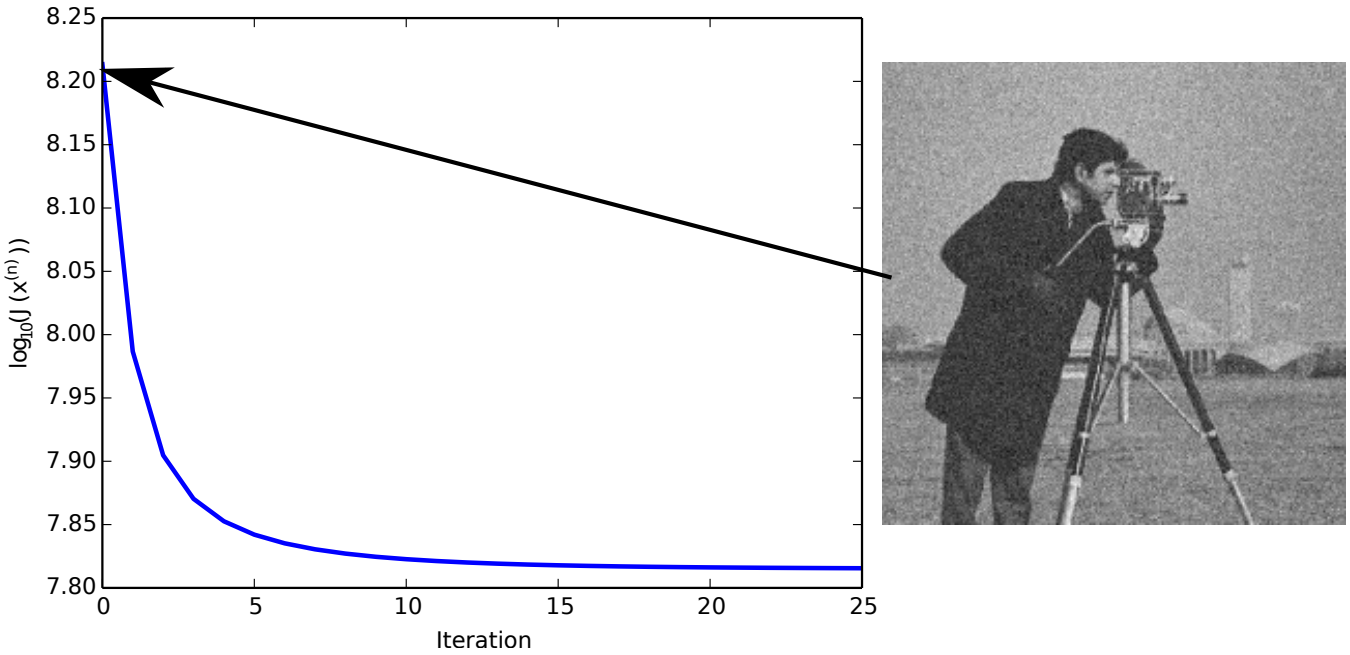
Madison G. McGaffin Jeffrey A. Fessler
University of Michigan

SPIE Computational Imaging XII
February 7th, 2014

Edge-preserving image denoising

Cost function: $$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

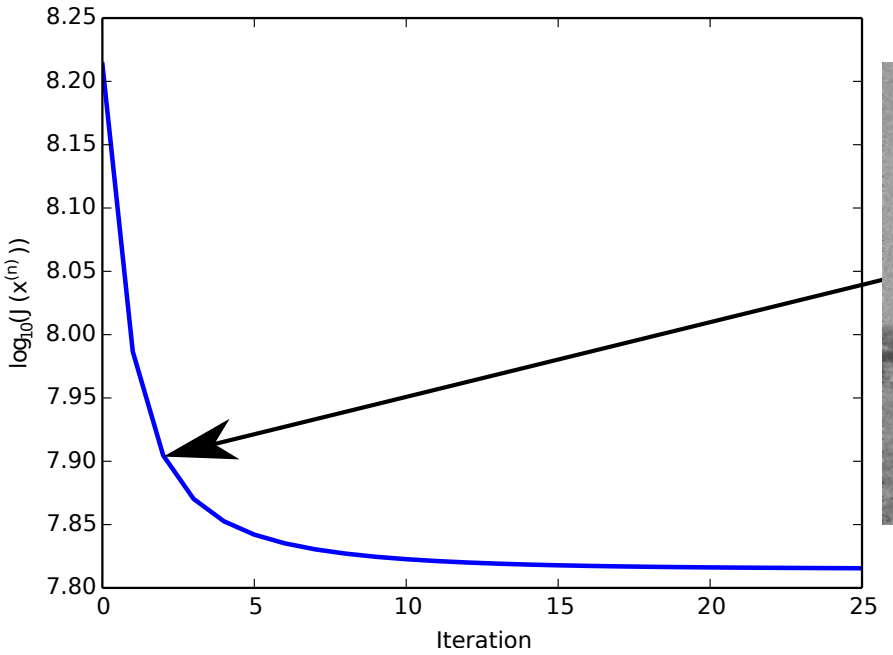
Regularizer: $$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$$



Edge-preserving image denoising

Cost function: $$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

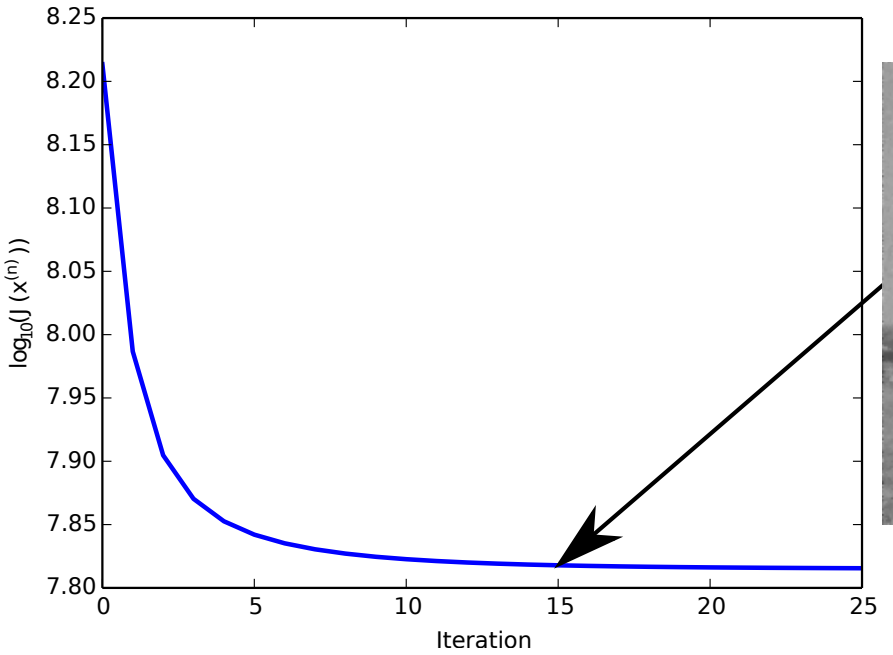
Regularizer: $$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$$



Edge-preserving image denoising

Cost function: $$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

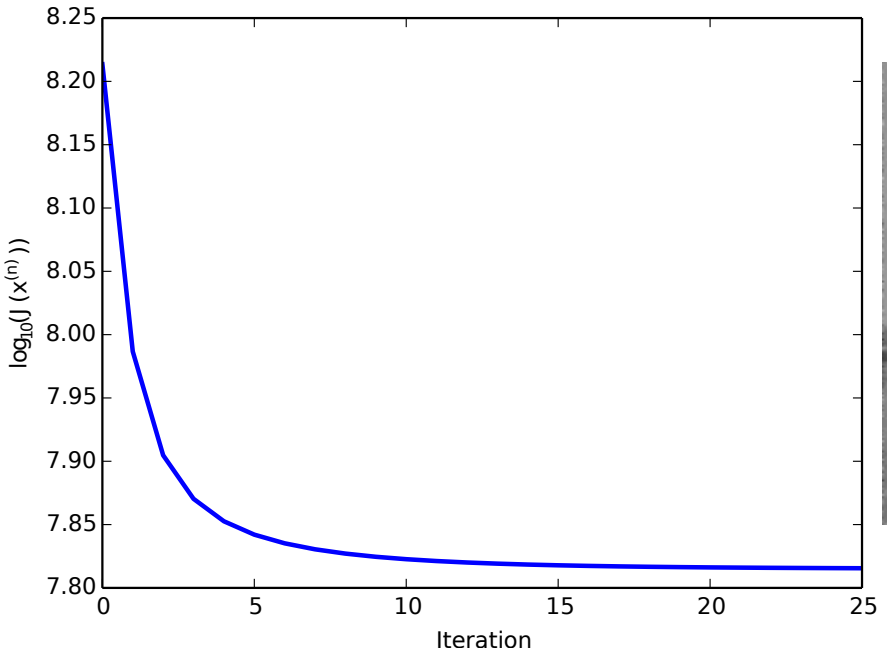
Regularizer: $$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$$



Edge-preserving image denoising

Cost function: $$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

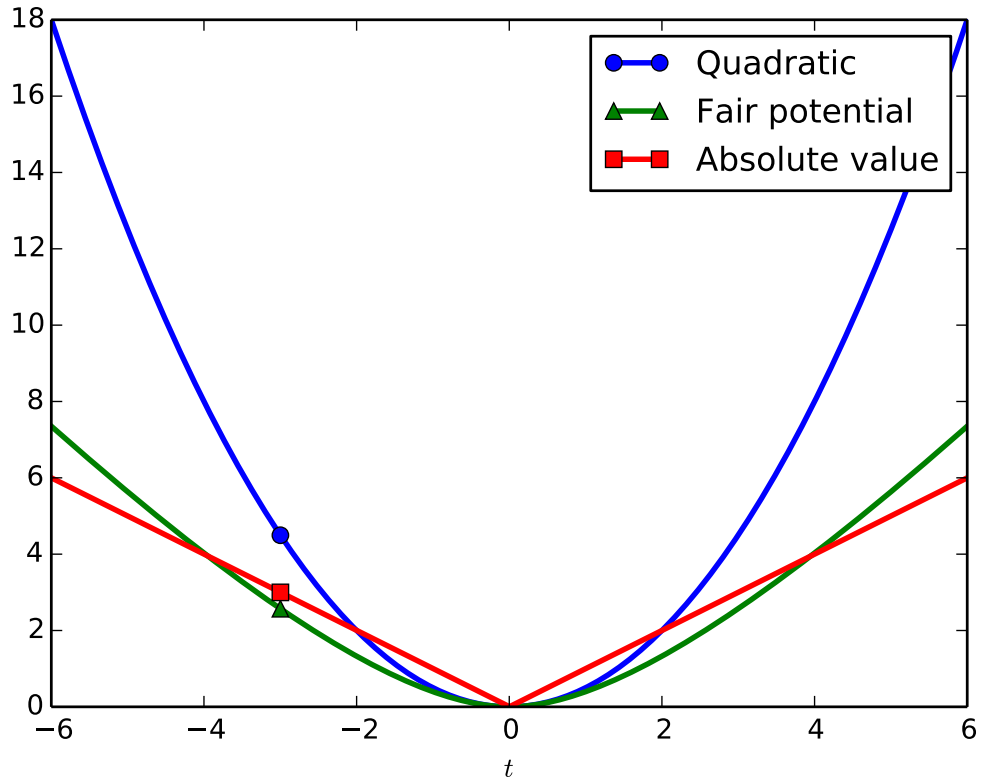
Regularizer: $$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi\left(x_j - x_k\right)$$



Edge-preserving image denoising

$$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi\left(x_j - x_k\right)$$

Different potential functions yield different solutions.



Edge-preserving image denoising

$$R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^{N} \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi\left(x_j - x_k\right)$$

Different potential functions yield different solutions.

: Quadratic

: Fair potential

: Absolute value

Algorithm design goals

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2} ||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

Iterative optimization algorithm design goals:

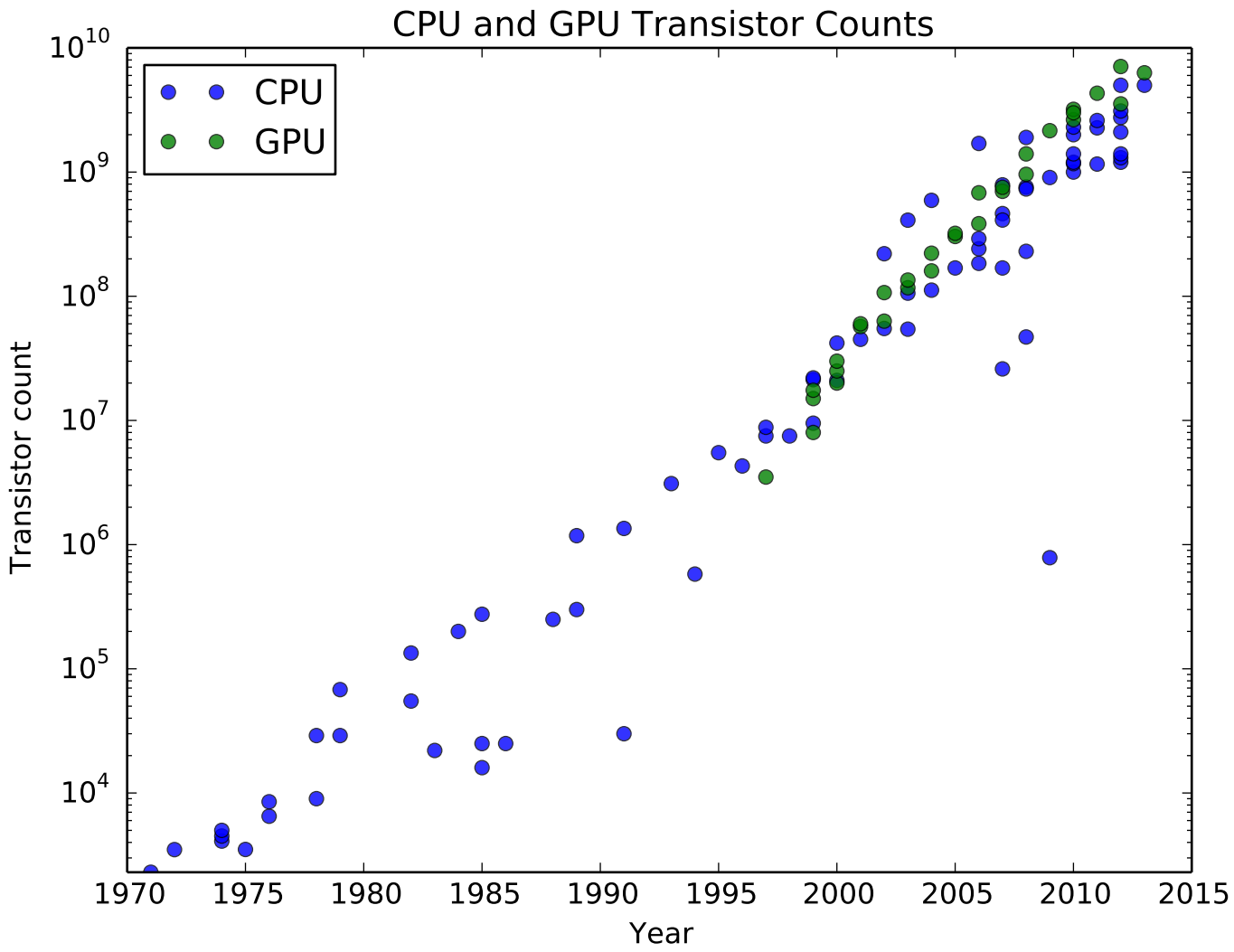
Algorithm design goals

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2} ||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

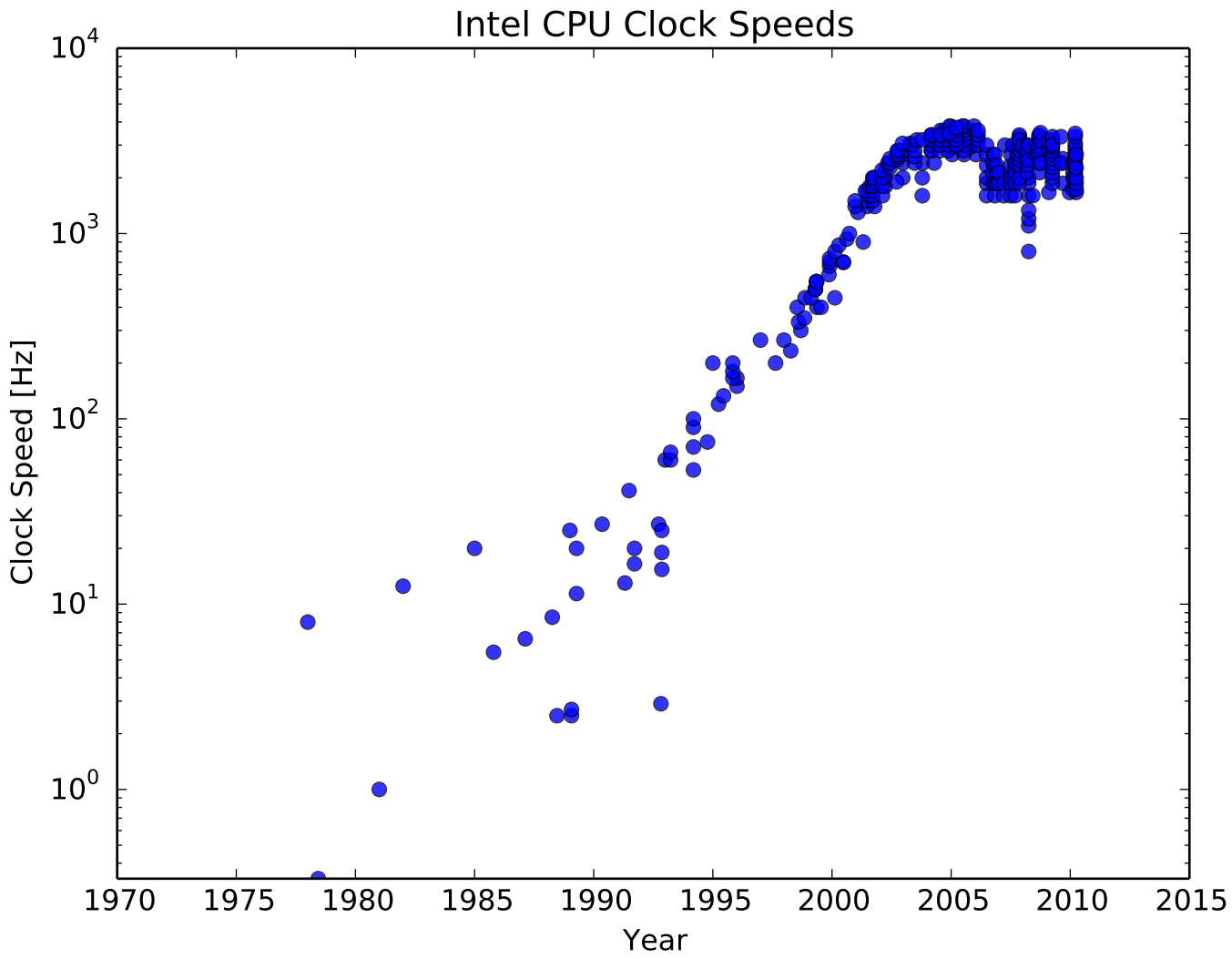
Iterative optimization algorithm design goals:

1. use a computer

Algorithm design goals



Algorithm design goals



Algorithm design goals

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

Iterative optimization algorithm design goals:

1. use a computer
2. be as parallelizable as possible:
   - many independent, identical operations
   - few sequential parts of the algorithm (Amdahl's Law)
   - no global operations (inner products!)
   - GPU: local, non-random memory accesses
   - GPU: modest memory requirements
   - ...

Group coordinate descent

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \succeq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + R(\mathbf{x}) \right\}$$

Divide the coordinates of $\mathbf{x}$ into K groups:

$$\mathbf{x} = [\mathbf{x}_1, \ \ldots, \ \mathbf{x}_K].$$

Group Coordinate Descent

In each of N_{iter} outer iterations, loop over group index k and solve:

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k}{\operatorname{argmin}} \ J\left(\mathbf{x}_1^{(n+1)}, \ldots, \mathbf{x}_{k-1}^{(n+1)}, \ \mathbf{x}_k, \ \mathbf{x}_{k+1}^{(n)}, \ldots, \ \mathbf{x}_K^{(n)}\right)$$

Group coordinate descent

1: **procedure** GCD-OUTER
2: **for** $n = 1$ up to N_{iter} **do**
3: **for** $k = 1$ up to K **do**
4:

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k}{\text{argmin}} \; J\left(\mathbf{x}_1^{(n+1)}, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}^{(n)}, \ldots, \mathbf{x}_K^{(n)}\right)$$

5: **end for**
6: **end for**
7: **end procedure**

Figure: Group coordinate descent outline.

Group coordinate descent

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k}{\operatorname{argmin}}\ J\left(\mathbf{x}_1^{(n+1)}, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}^{(n)}, \ldots, \mathbf{x}_K^{(n)}\right).$$

$$J(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 + \mathrm{R}(\mathbf{x}); \qquad \mathrm{R}(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{N}\sum_{k\in\mathcal{N}_j} \kappa_{kj}\beta_{kj}\phi\left(x_j - x_k\right),$$

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m\in S_k} \left[\frac{1}{2}(x_m - y_m)^2 + \sum_{j\in\mathcal{N}_m} \frac{\beta_{jm}\kappa_{jm}}{2}\phi\left(x_m - x_j\right).\right]$$

Key trick: If $\mathcal{N}_m \cap S_k = \emptyset$ for all $m \in S_k$, then this is $|S_k|$ independent 1D problems!

Group coordinate descent

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \left[\frac{1}{2}(x_m - y_m)^2 + \sum_{j \in \mathcal{N}_m} \frac{\beta_{jm}\kappa_{jm}}{2} \phi\big(x_m - x_j\big). \right]$$

1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4

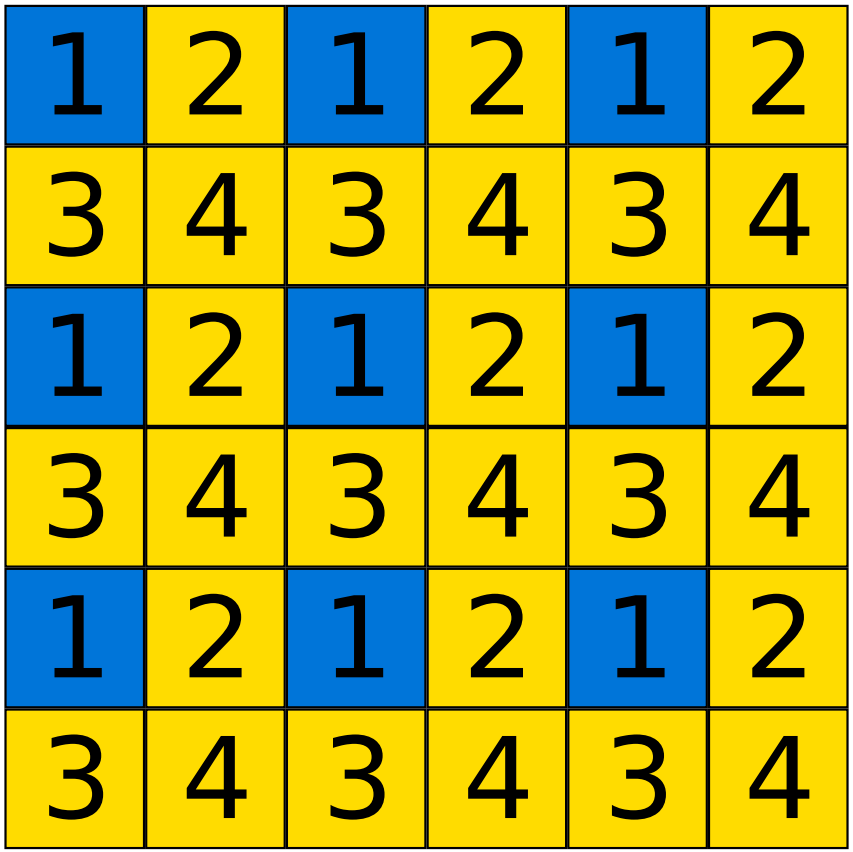
Group coordinate descent

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \left[\frac{1}{2}(x_m - y_m)^2 + \sum_{j \in \mathcal{N}_m} \frac{\beta_{jm}\kappa_{jm}}{2} \phi\big(x_m - x_j\big). \right]$$

1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4
1	2	1	2	1	2
3	4	3	4	3	4

Group coordinate descent

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \left[\frac{1}{2}(x_m - y_m)^2 + \sum_{j \in \mathcal{N}_m} \frac{\beta_{jm}\kappa_{jm}}{2} \phi(x_m - x_j). \right]$$



Group coordinate descent

1: **procedure** GCD-OUTER
2: **for** $n = 1$ up to N_{iter} **do**
3: **for** $k = 1$ up to K **do**
4: **Parfor** $j \in S_k$
5: Run inner algorithm to solve jth 1D problem
6: **EndParfor**
7: **end for**
8: **end for**
9: **end procedure**

Figure: Outer algorithm structure. The **Parfor** loop contains independent optimization operations which may implemented in parallel.

The 1D subproblem

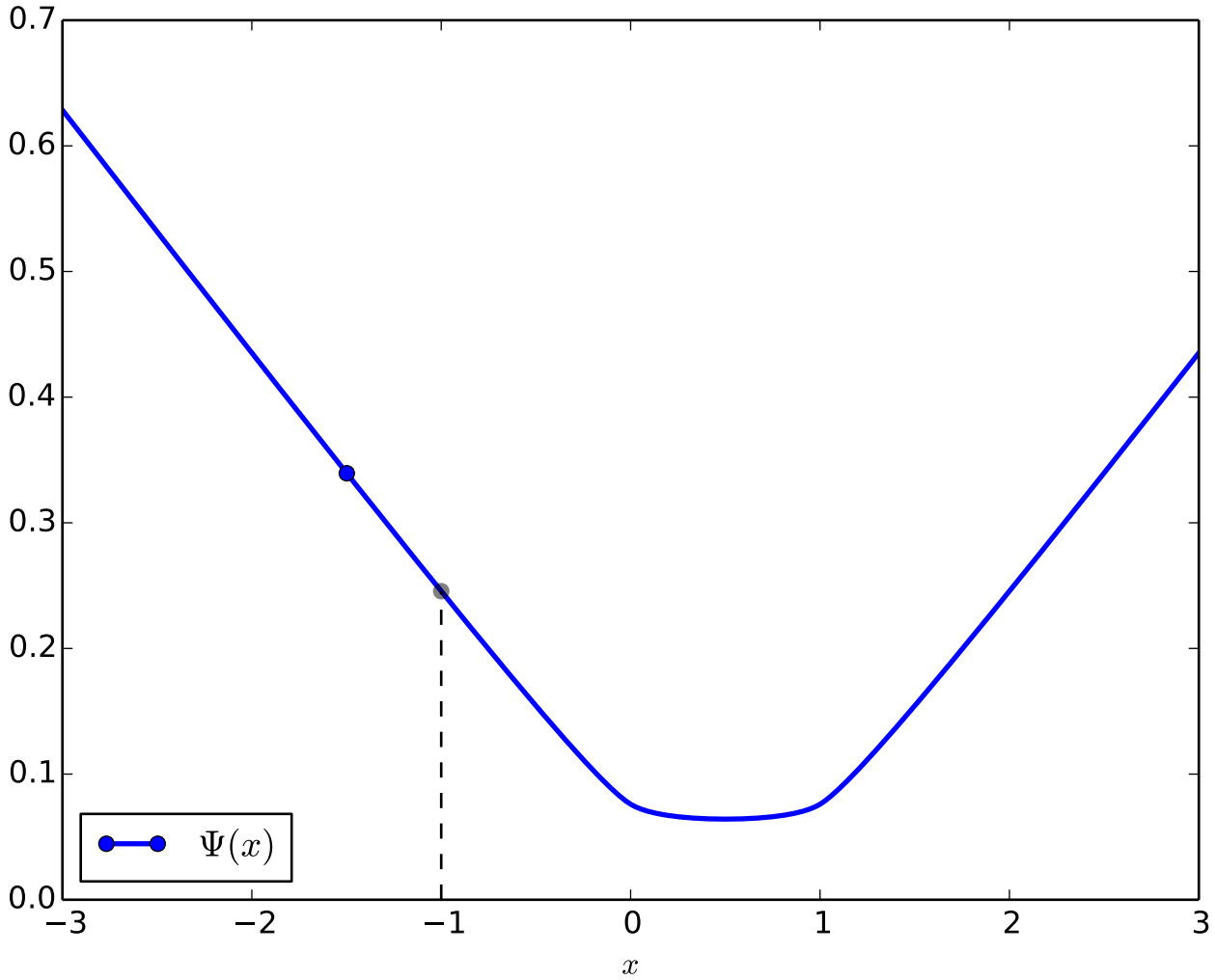
$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \psi_m^{(n)}(x_m)$$

$$\psi_m^{(n)}(x) = \frac{w_m}{2}(x - y_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi(x - x_j)$$

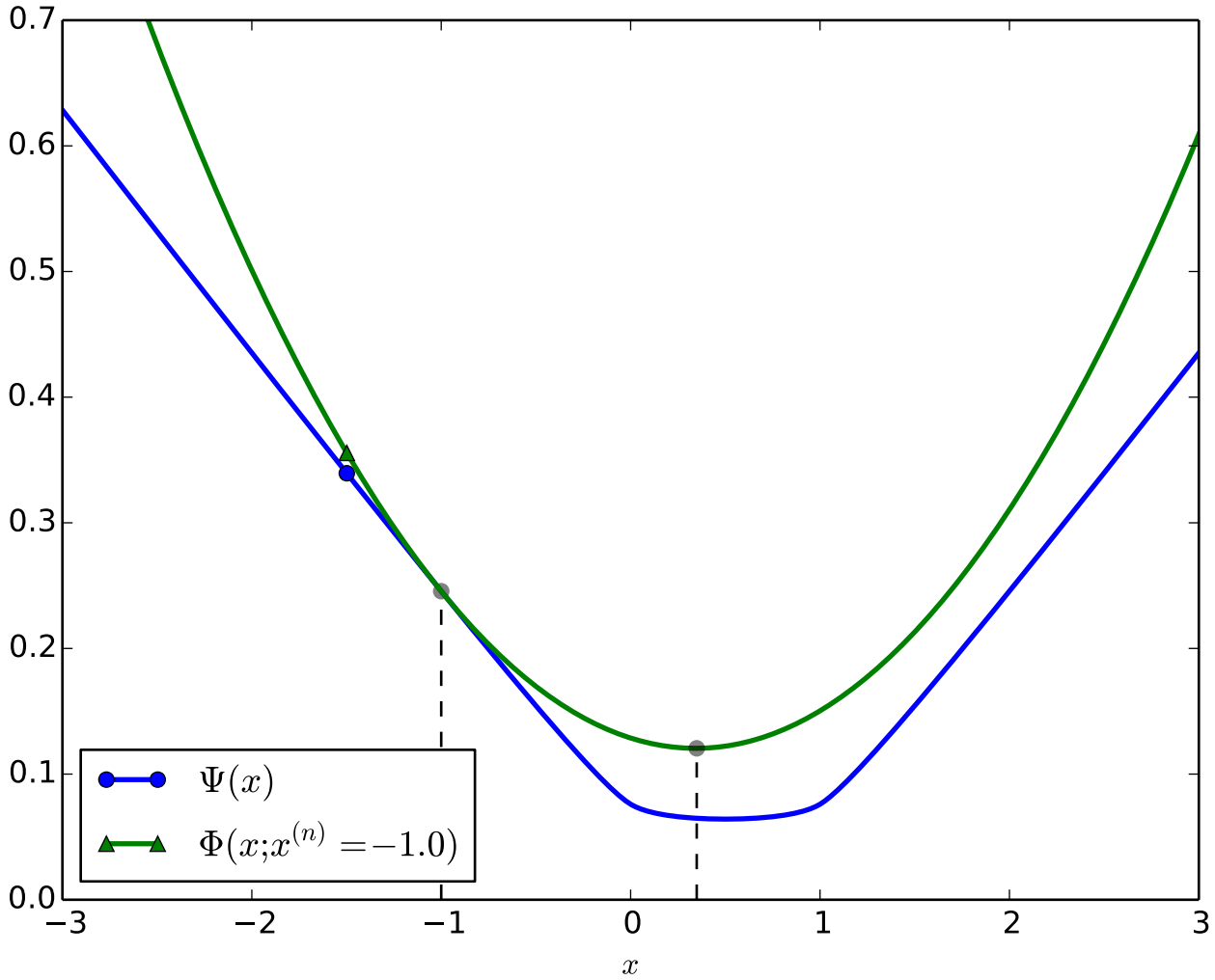
Is ϕ differentiable?

- we can use a simple majorize-minimize algorithm to minimize Ψ.

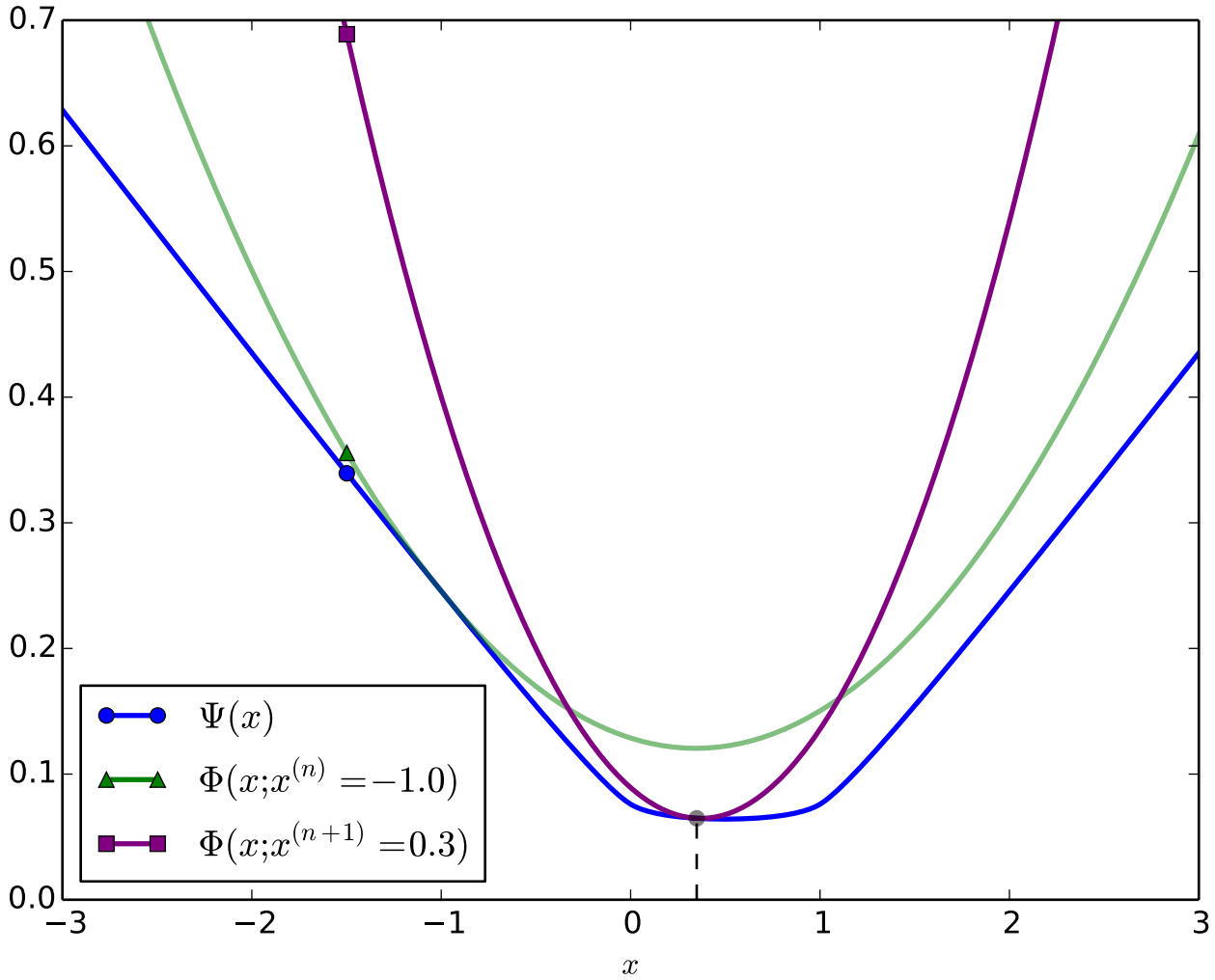
The 1D subproblem



The 1D subproblem



The 1D subproblem



The 1D subproblem

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \psi_m^{(n)}(x_m)$$

$$\psi_m^{(n)}(x) = \frac{w_m}{2}(x - y_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi\left(x - x_j\right)$$

Is ϕ differentiable?

- we can use a simple majorize-minimize algorithm to minimize Ψ. Huber's curvatures:

$$g_j = \phi'\left(x^{(n)} - x_j\right), \qquad c_j^{(m)} = \omega\left(x^{(n)} - x_j\right) = \frac{g_j}{\left|x^{(n)} - x_j\right|}$$

$$x^{(n+1)} = \max\left\{0, \frac{w \cdot y + \sum_{j \in \mathcal{N}} \beta_j \left(c_j x^{(n)} - g_j\right)}{w + \sum_{j \in \mathcal{N}} \beta_j c_j}\right\}$$

Experimental information

All algorithms run on an NVIDIA Tesla C2050 GPU:

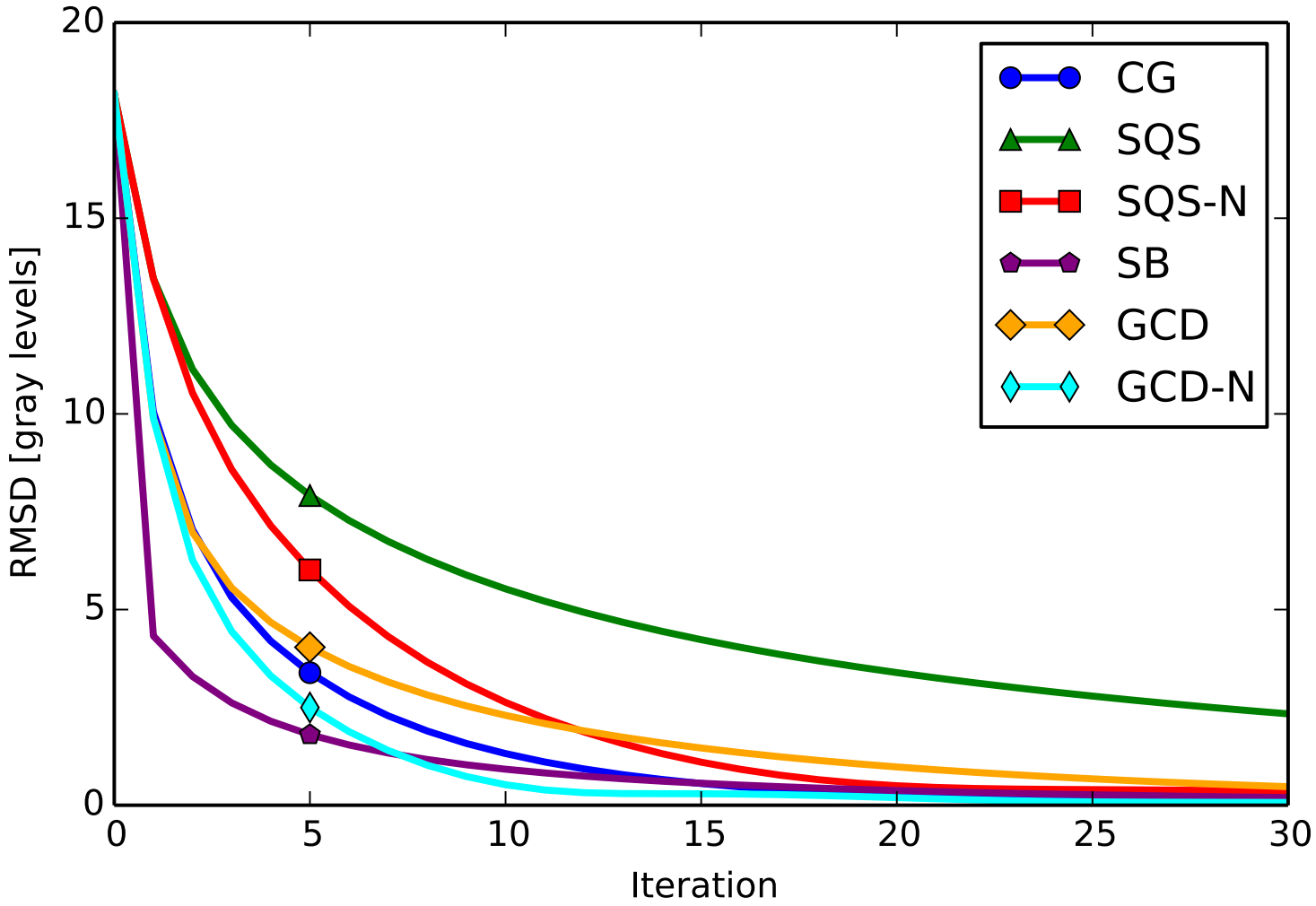
- CG: Conjugate gradients
- SQS and SQS-N: coordinate-separable majorize-minimize using Huber's curvatures and Nesterov's momentum
- SB: Split Bregman algorithm
- GCD and GCD-N: Proposed algorithm with Nesterov's momentum

True image

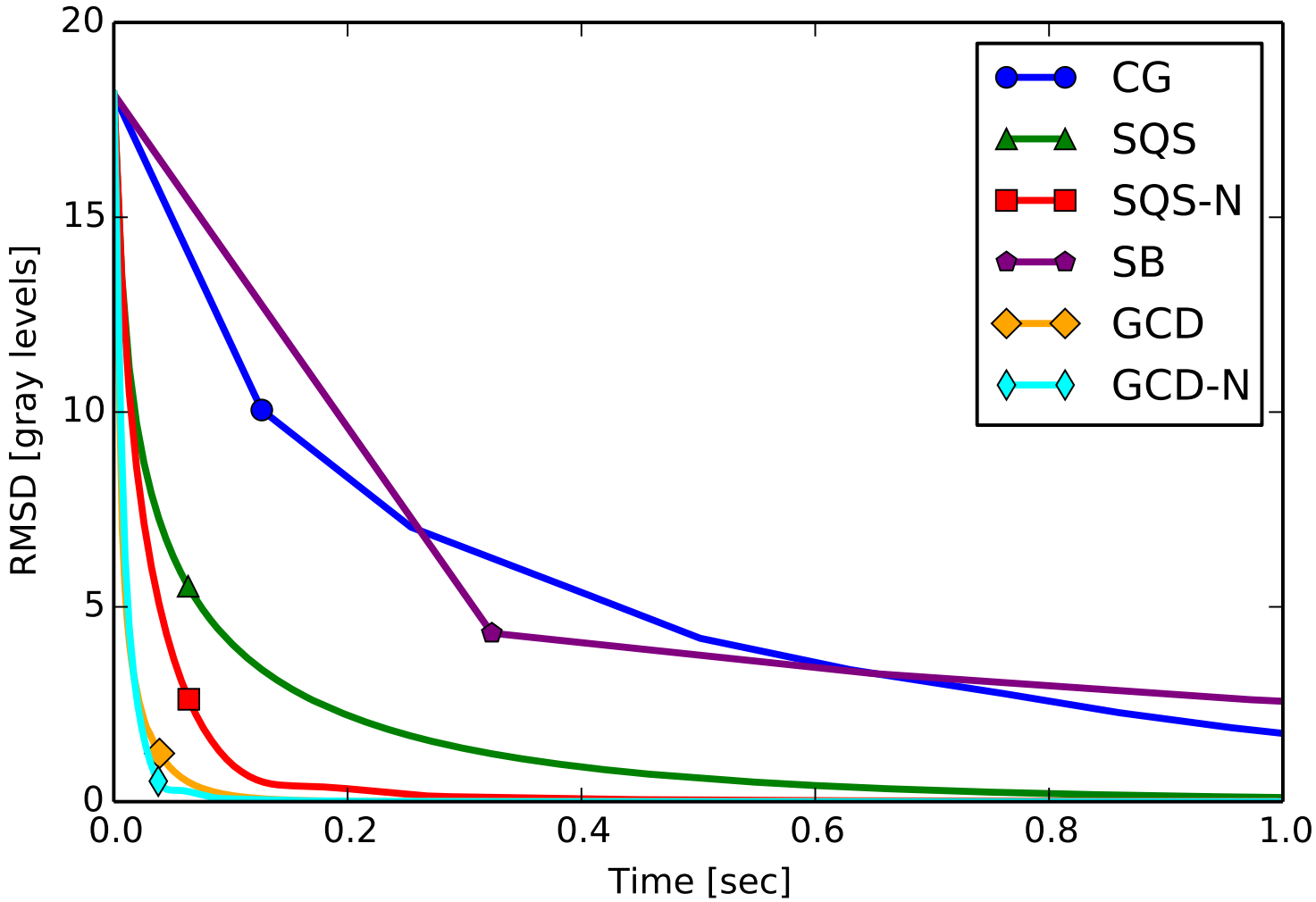
Noisy image

Denoised image

RMSD by iteration



RMSD by time


20
15
10
5
0
RMSD [gray levels]
0.0
0.2
0.4
0.6
0.8
1.0
Time [sec]
CG
SQS
SQS-N
SB
GCD
GCD-N

The 1D subproblem

$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k \geq \mathbf{0}}{\operatorname{argmin}} \sum_{m \in S_k} \psi_m^{(n)}(x_m)$$

$$\psi_m^{(n)}(x) = \frac{w_m}{2}(x - y_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi(x - x_j)$$

Is ϕ differentiable?

- we can use a simple majorize-minimize algorithm to minimize ψ.

But what if $\phi(t) = |t|$? A couple choices:

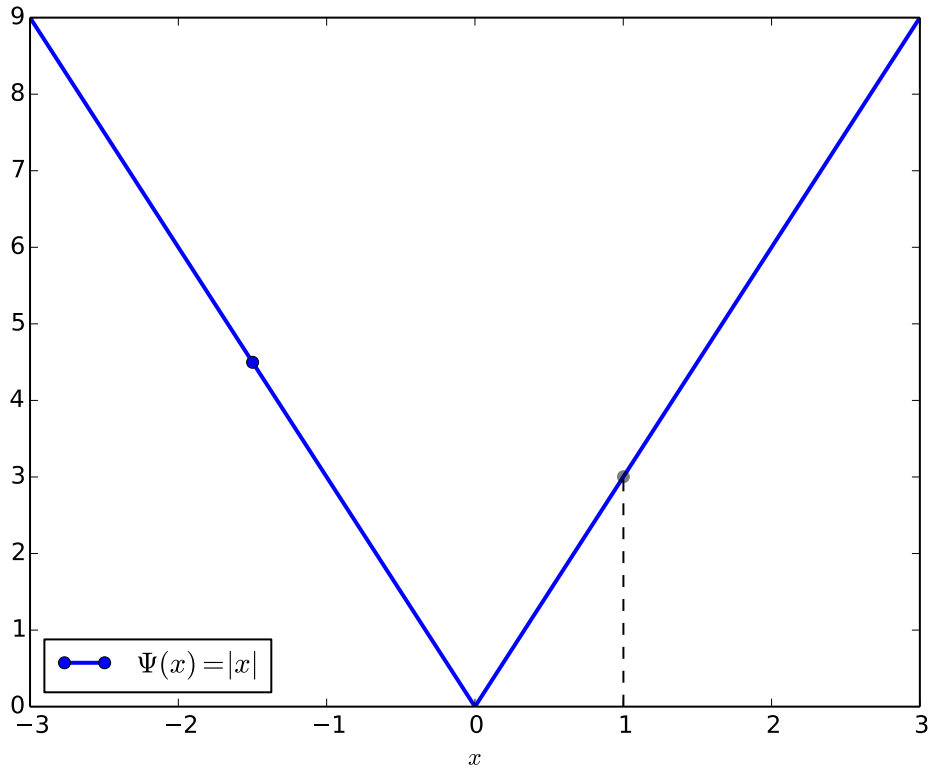
- "corner rounding"
- "Huber-and-pray:" assume $x \neq x_j$; possible numerical problems
- split Bregman: high memory cost, but pretty quick.
- our choice: dual majorize-minimize approach.

The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

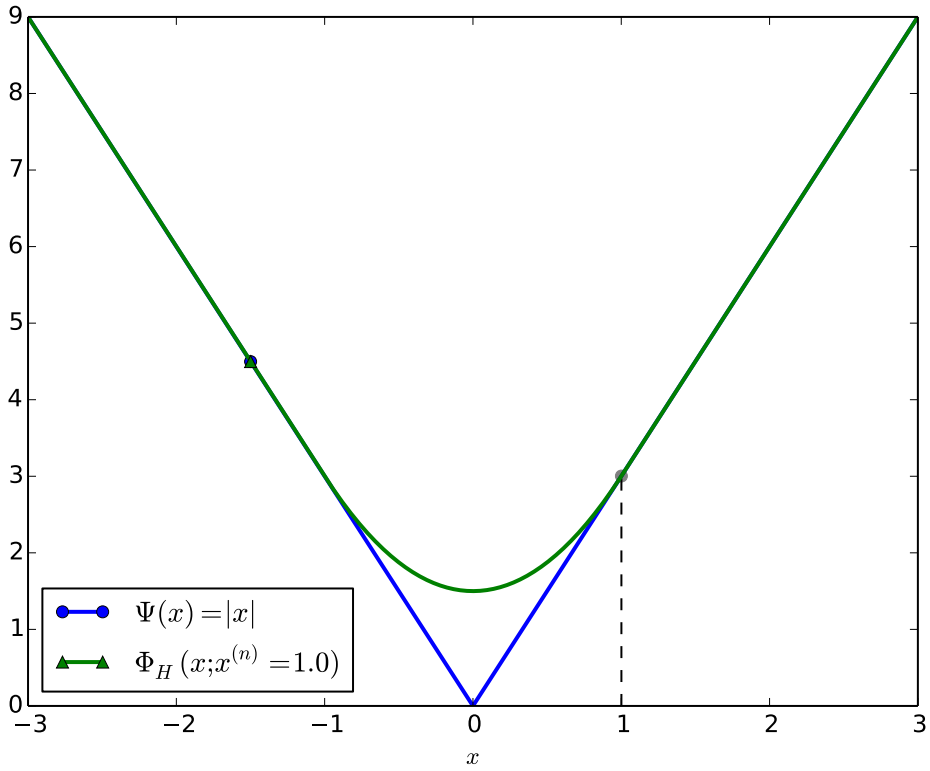


The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

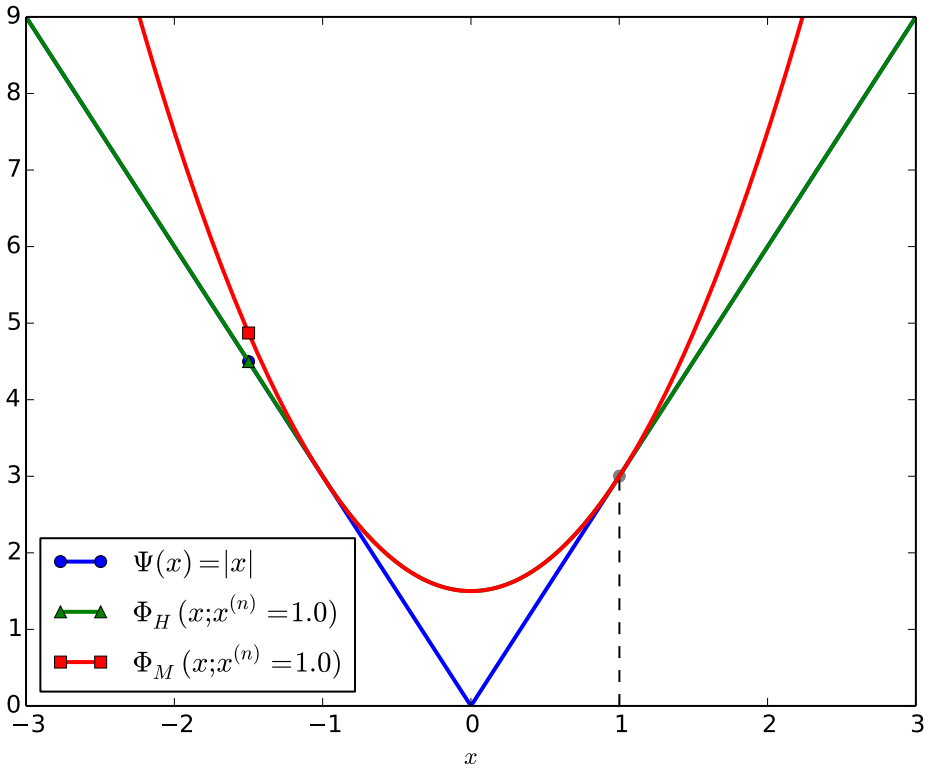


The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

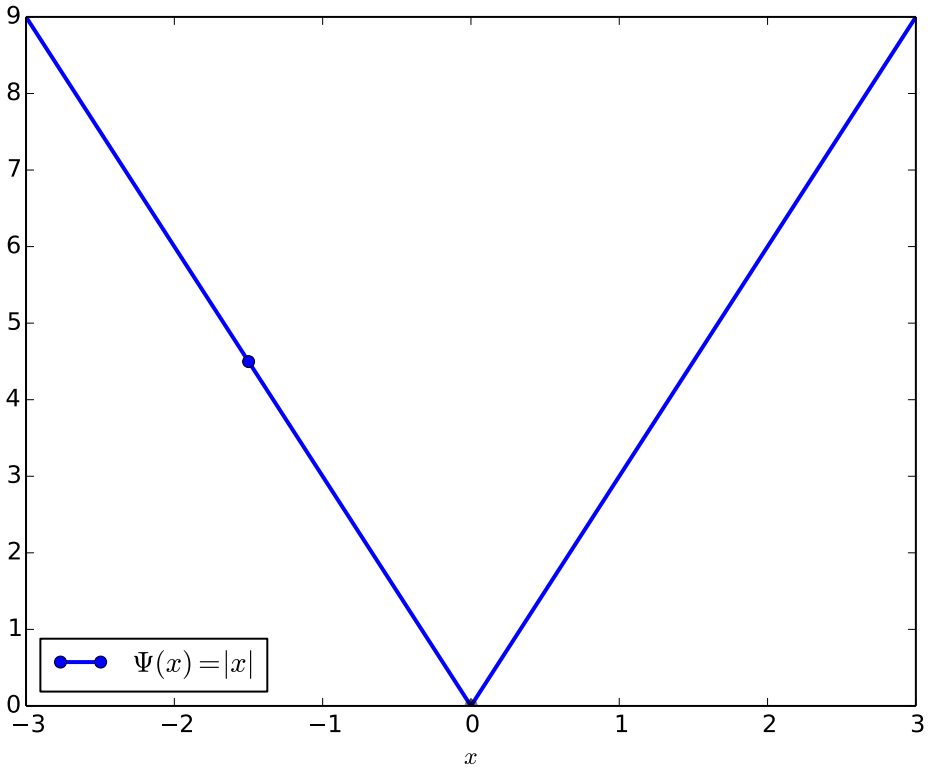


The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

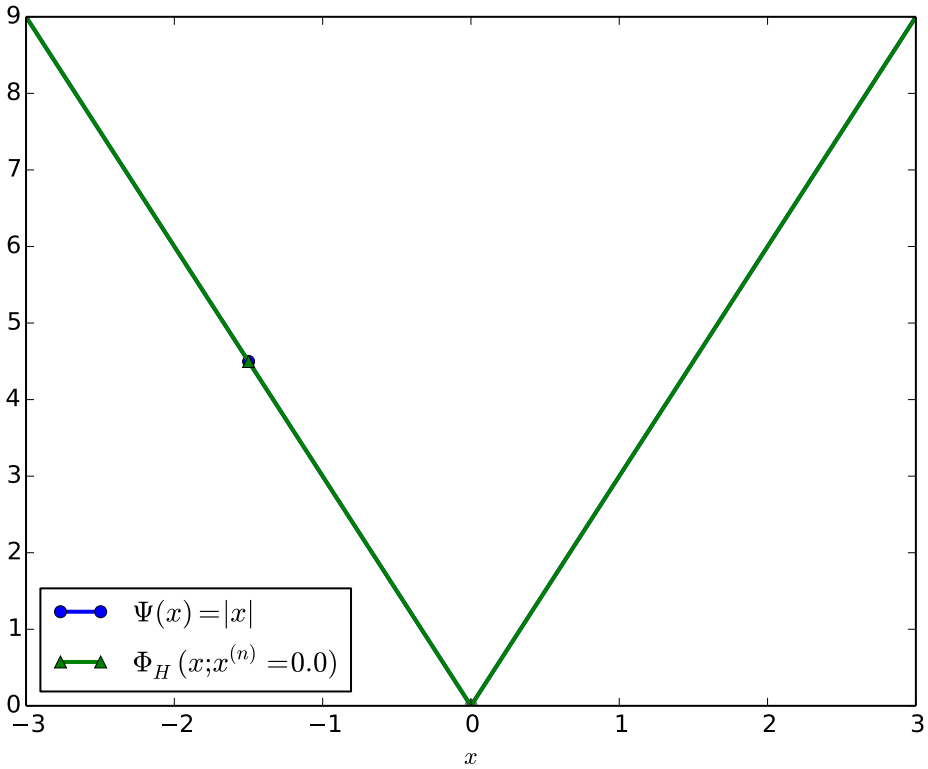


The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

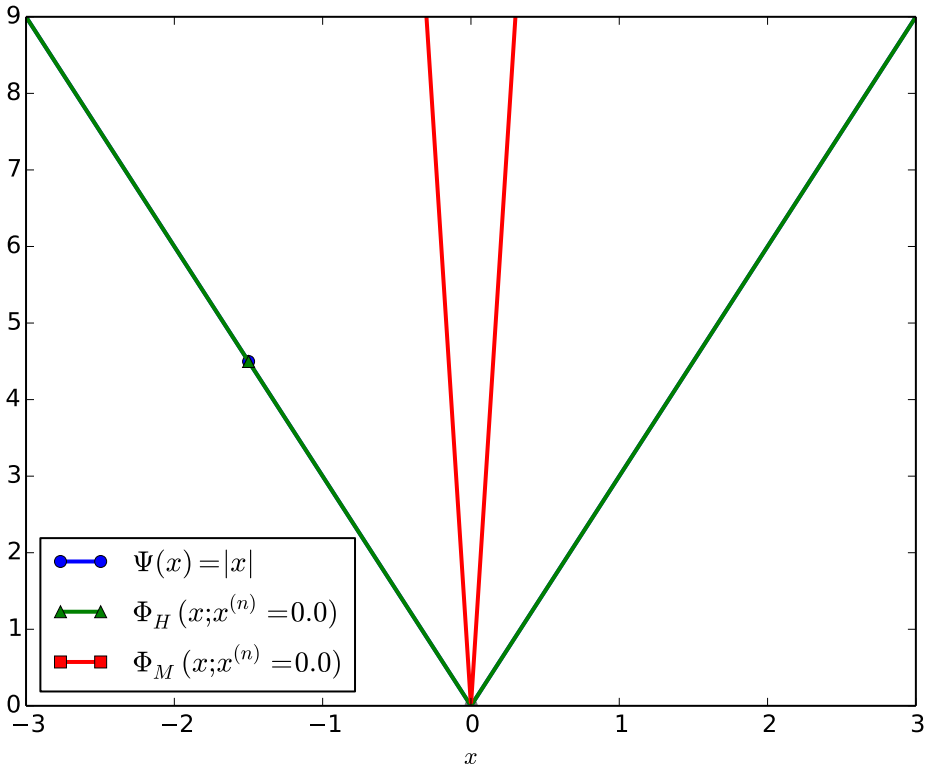


The 1D subproblem

$$\Psi(x) = |x| = \max_{\gamma \in [-1,1]} \gamma x$$

$$\Phi_M(x) = \max_{\gamma \in [-1,1]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$

$$\Phi_H(x) = \max_{\gamma \in [-10,10]} \gamma x - \frac{x^{(n)}}{2}\left(\gamma^2 - 1\right)$$



Relaxed Huber function majorizer

$$\Psi(x) = \frac{w}{2}(x-y)^2 + \sum_{j \in \mathcal{N}} \beta_j |x - x_j|$$

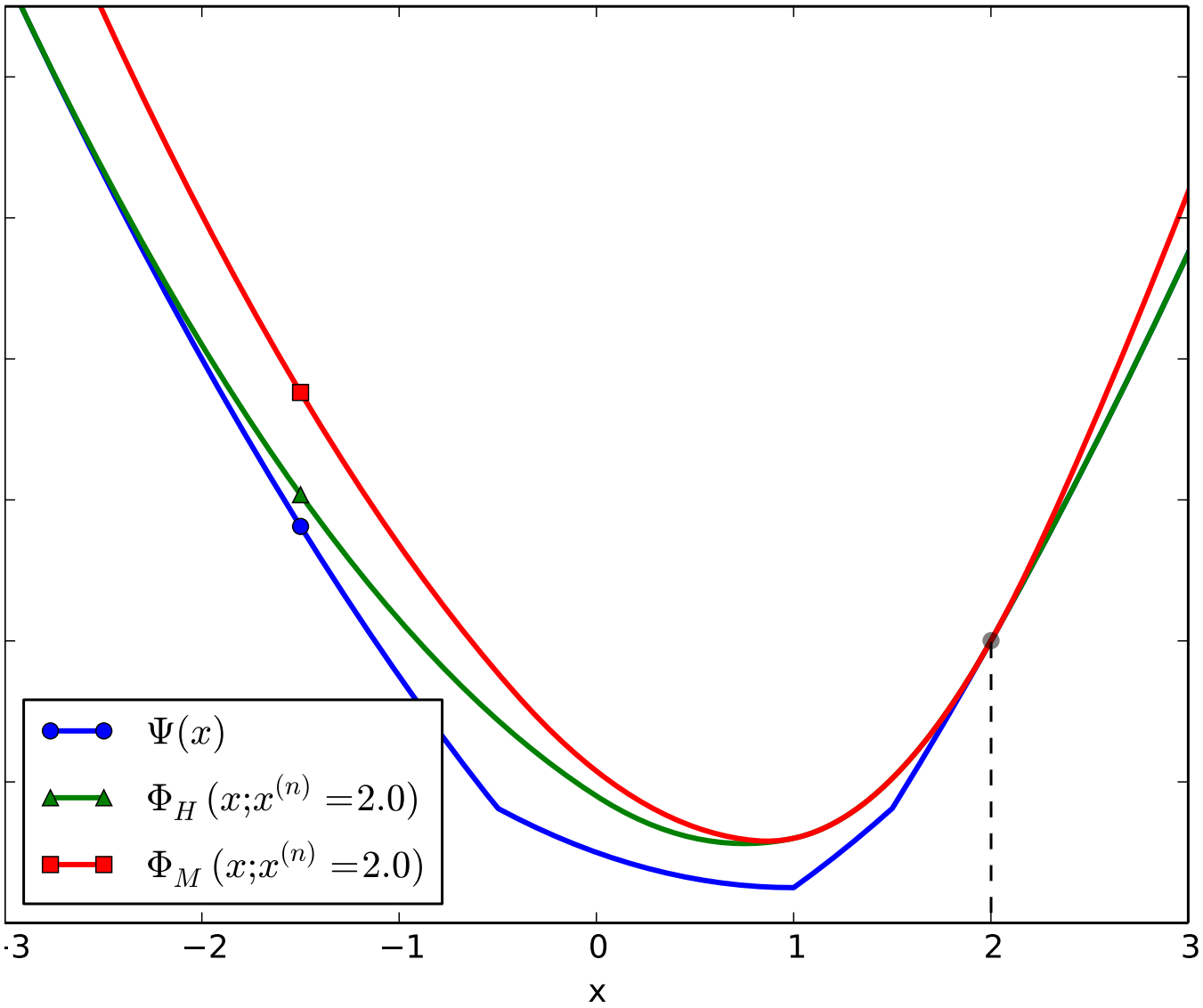
$$\Phi_m(x) = \max_{\gamma \in \Omega^{|\mathcal{N}|}} S(x;\gamma)$$

$$S(x;\gamma) = \frac{1}{2}(x-y)^2 + \sum_{j \in \mathcal{N}} \beta_j s\left(x - x_j, \gamma_j; \left|x^{(n)} - x_j\right|\right)$$

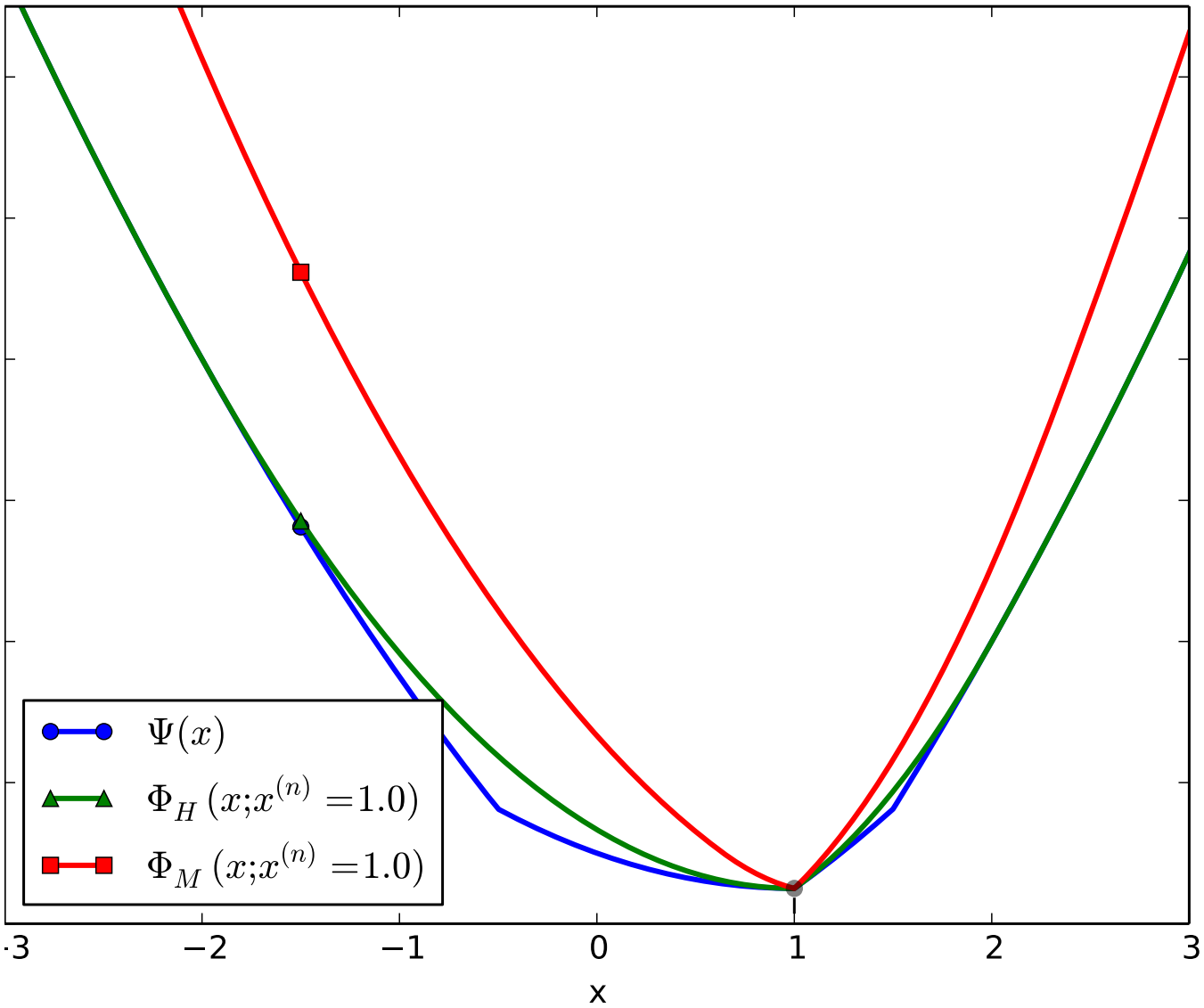
$$s(t,\gamma;\delta) = \gamma t - \frac{\delta}{2}\left(\gamma^2 - 1\right)$$

Assertion: Φ_m is a majorizer for Ψ at $x = x^{(n)}$. Any Ω which contains $[-1, 1]$ is valid.

Relaxed Huber function majorizer



Relaxed Huber function majorizer



Duality trick

To solve

$$x^{(n+1)} = \underset{x \geq 0}{\operatorname{argmin}} \ \Phi(x) = \underset{x \geq 0}{\operatorname{argmin}} \ \max_{\gamma \in \Omega^{|\mathcal{N}|}} \ S(x; \gamma),$$

reverse the order of minimization and maximization:

$$x^{(n+1)} = \max\{x(\gamma_*), 0\},$$

$$x(\gamma) = \underset{x}{\operatorname{argmin}} \ S(x; \gamma) = y - \frac{1}{w}\beta'\gamma$$

$$\gamma_* \in \underset{\gamma \in \Omega^{|\mathcal{N}|}}{\operatorname{argmax}} \ S(x(\gamma); \gamma).$$

$$S(x(\gamma); \gamma) = -\frac{1}{2}||\gamma||^2_{\mathbf{D}+\beta\beta'} + \gamma'\mathbf{W}\beta,$$

where $\mathbf{D} = \operatorname{diag}_d\left\{\left|x^{(n)} - x_d\right|\right\}$, and $\beta = \operatorname{vec}_d\{\beta_d\}$.

Duality trick

$$\gamma_* \in \underset{\gamma \in \Omega^{|\mathcal{N}|}}{\operatorname{argmax}} \left\{ S(x(\gamma);\gamma) = -\frac{1}{2}||\gamma||^2_{\mathbf{D}+\beta\beta'} + \gamma'\mathbf{W}\beta, \right\}$$

A couple more tricks:

- Expand Ω until pseudoinverse lies in $\Omega^{|\mathcal{N}|}$; relax to unconstrained optimization.
- Solve with *e.g.*, matrix factorization (too expensive?) or iterative method.

Duality trick

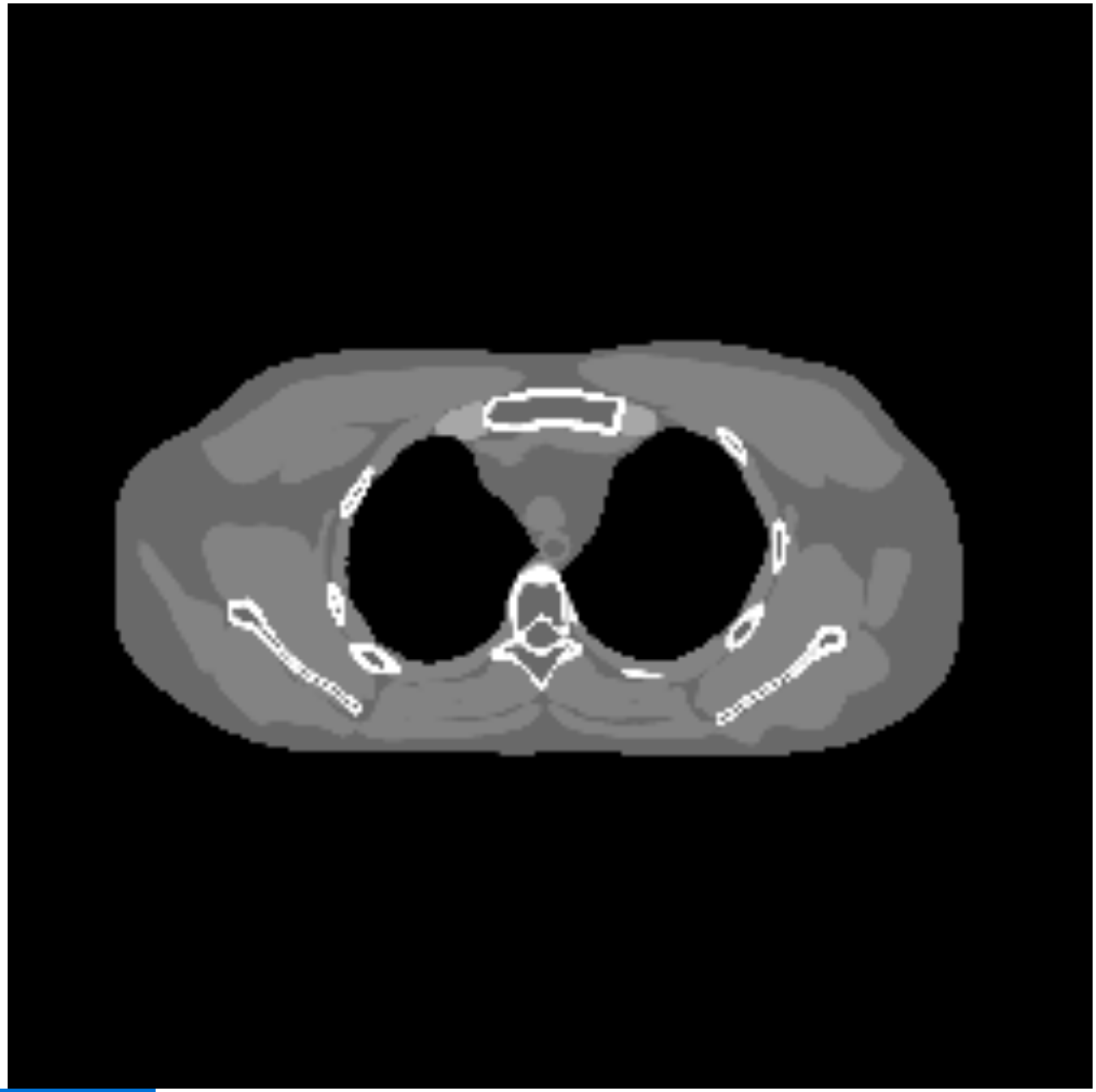
$$\gamma_* \in \underset{\gamma}{\operatorname{argmin}} \left\{ S(x(\gamma);\gamma) = \frac{1}{2}||\gamma||^2_{\mathbf{D}+\beta\beta'} - \gamma'\mathbf{W}\beta, \right\}$$

Majorize-minimize (minorize maximimize):

$$\gamma^{(n+1)} = \gamma^{(n)} - \left(\mathbf{D}_\epsilon + \beta\beta'\right)^{-1}\left(\left(\mathbf{D} + \beta\beta'\right)\gamma^{(n)} + \mathbf{W}\beta\right)$$

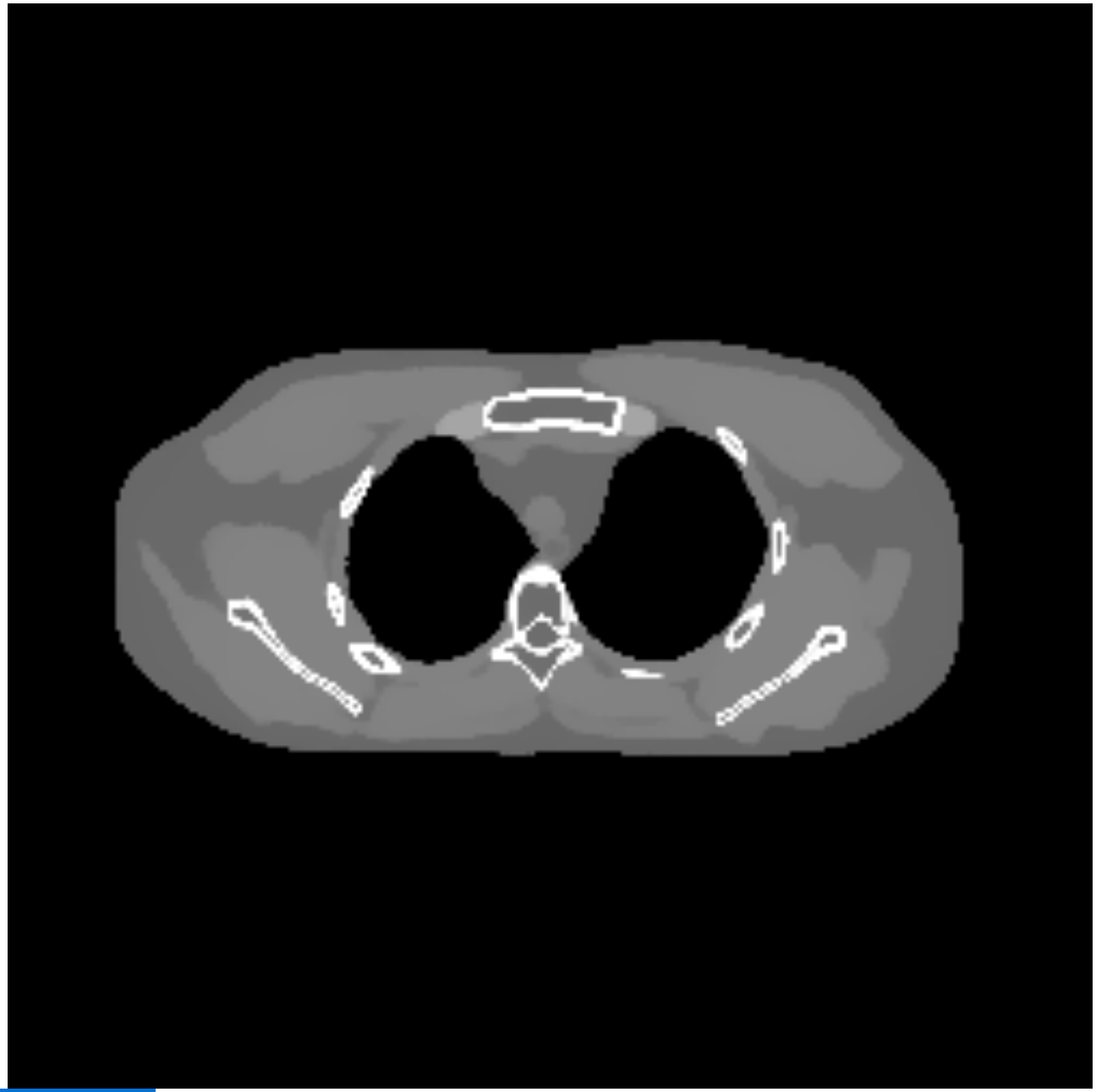
- $\mathbf{D}_\epsilon = \operatorname{diag}_d\left\{\max\left\{\epsilon, \mathbf{D}_{dd}\right\}\right\} \succeq \mathbf{D}$
- Matrix inverse with matrix inversion lemma.

True image

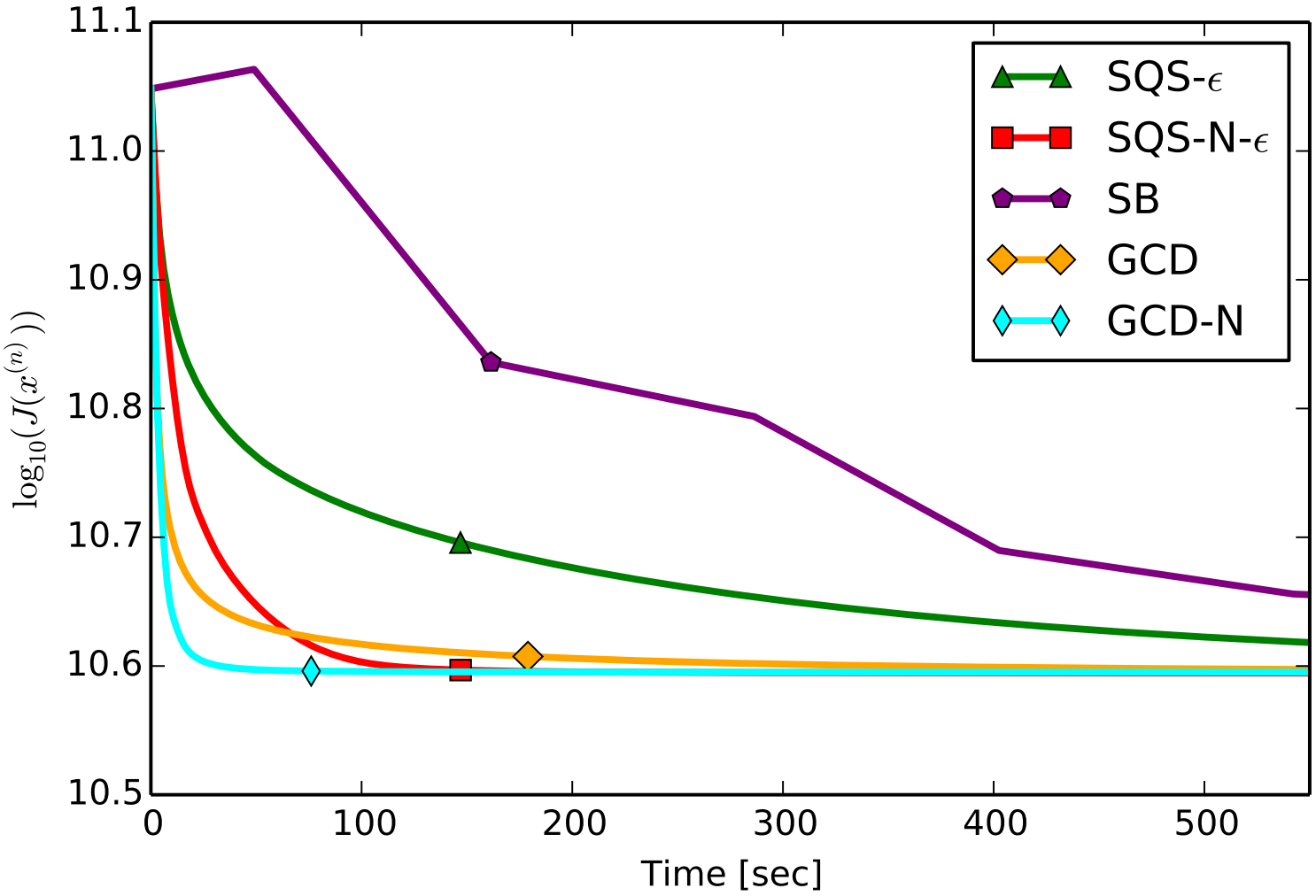


Noisy image

Denoised image



Cost function by time



Conclusions

The proposed algorithms

- iteratively solve a set of independent 1D subproblems,
- support differentiable and nondifferentiable cost functions,
- and play to the strengths of GPUs (*e.g.*, high parallelism, low memory requirements, no branching, . . .)

Thank you!