

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

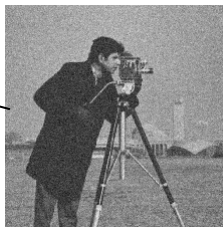
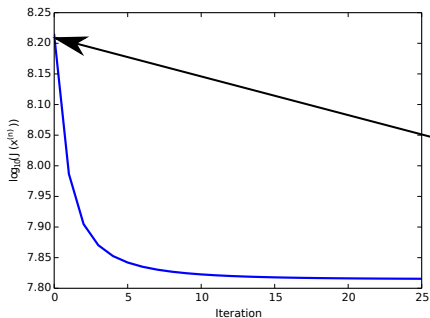
Madison G. McGaffin   Jeffrey A. Fessler  
University of Michigan

SPIE Computational Imaging XII  
February 7th, 2014

# Edge-preserving image denoising

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

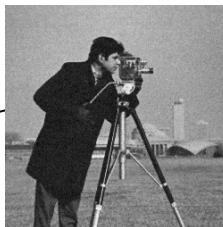
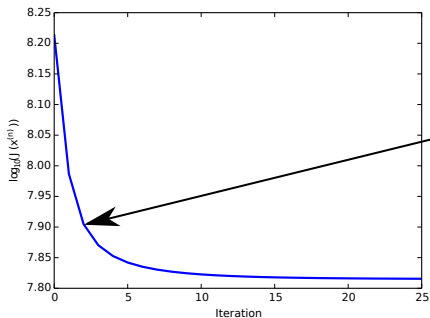
Regularizer:  $R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^N \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$



# Edge-preserving image denoising

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

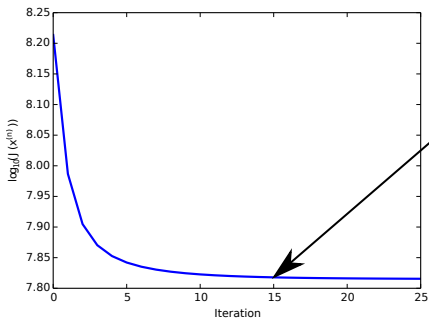
Regularizer:  $R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^N \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$



# Edge-preserving image denoising

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

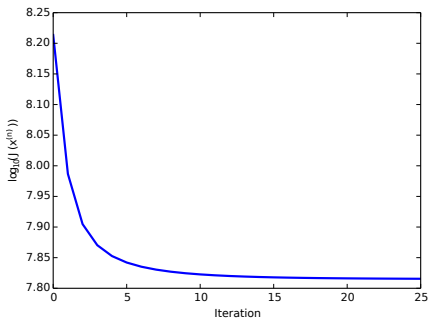
Regularizer:  $R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^N \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$



# Edge-preserving image denoising

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

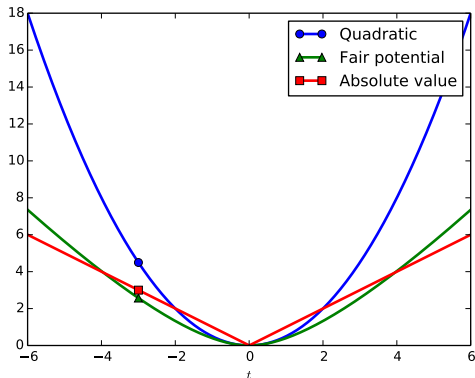
Regularizer:  $R(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^N \sum_{k \in \mathcal{N}_j} \kappa_{kj} \phi(x_j - x_k)$



# Edge-preserving image denoising

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

Different **potential functions** yield different solutions.



# Edge-preserving image denoising

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

Different **potential functions** yield different solutions.



: Quadratic



: Fair potential



: Absolute value

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

Iterative optimization algorithm design goals:

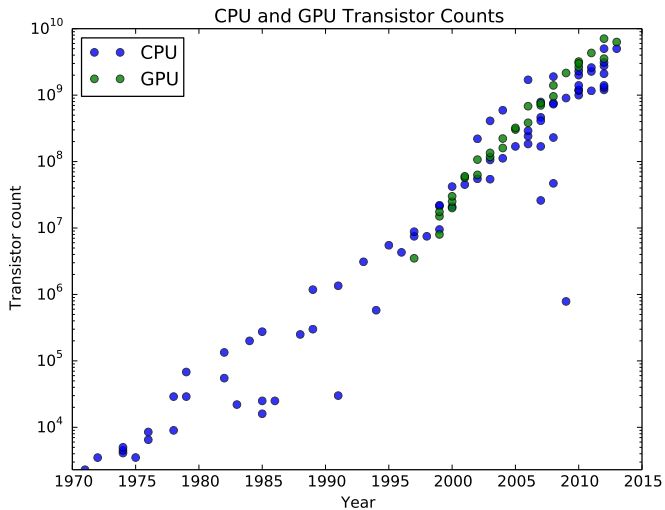


$$\hat{\mathbf{x}} = \underset{\mathbf{x} \geq \mathbf{0}}{\operatorname{argmin}} \left\{ J(\mathbf{x}) = \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2 + R(\mathbf{x}) \right\}$$

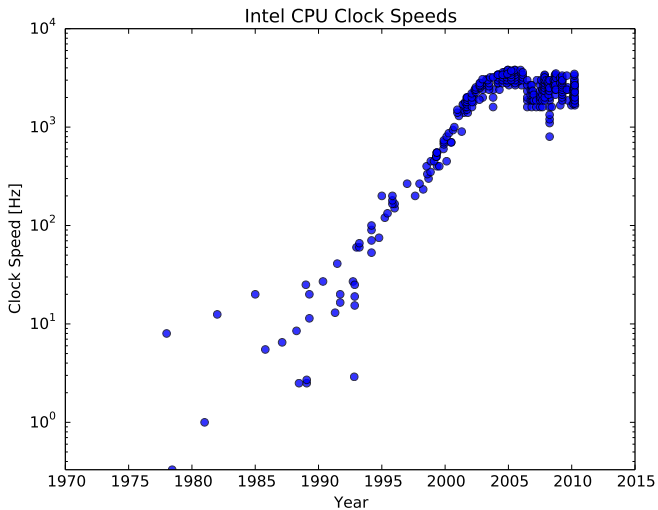
Iterative optimization algorithm design goals:

1. use a computer

# 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

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

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

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

## Group Coordinate Descent

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

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

# Group coordinate descent

```
1: procedure GCD-OUTER
2:   for  $n = 1$  up to  $N_{\text{iter}}$  do
3:     for  $k = 1$  up to  $K$  do
4:       
$$\mathbf{x}_k^{(n+1)} = \underset{\mathbf{x}_k}{\operatorname{argmin}} J\left(\mathbf{x}_1^{(n+1)}, \dots, \mathbf{x}_k, \mathbf{x}_{k+1}^{(n)}, \dots, \mathbf{x}_K^{(n)}\right)$$

5:     end for
6:   end for
7: end procedure
```

Figure: Group coordinate descent outline.

# Group coordinate descent

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

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

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

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

# Group coordinate descent

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

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

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

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

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

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

```
1: procedure GCD-OUTER
2:   for  $n = 1$  up to  $N_{\text{iter}}$  do
3:     for  $k = 1$  up to  $K$  do
4:       Parfor  $j \in S_k$ 
5:         Run inner algorithm to solve  $j$ th 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

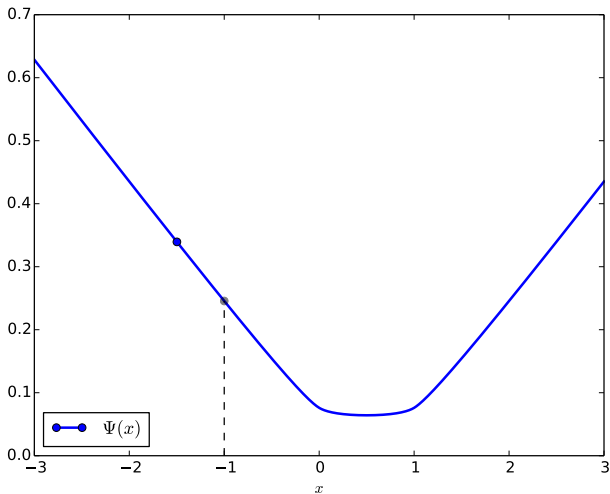
$$\mathbf{x}_k^{(n+1)} = \operatorname{argmin}_{\mathbf{x}_k \geq \mathbf{0}} \sum_{m \in S_k} \Psi_m^{(n)}(\mathbf{x}_m)$$

$$\Psi_m^{(n)}(\mathbf{x}) = \frac{W_m}{2}(\mathbf{x} - \mathbf{y}_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi(\mathbf{x} - \mathbf{x}_j)$$

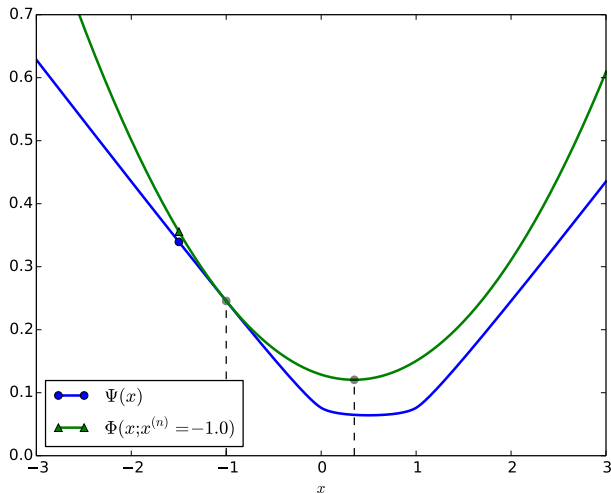
Is  $\phi$  differentiable?

- we can use a simple **majorize-minimize algorithm** to minimize  $\Psi$ .

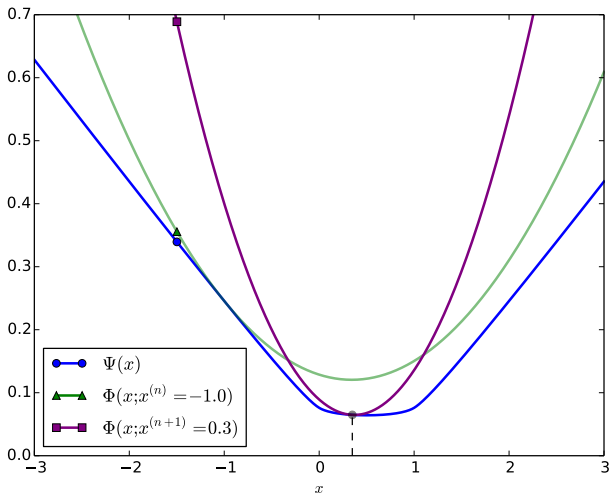
# The 1D subproblem



# The 1D subproblem



# The 1D subproblem



# The 1D subproblem

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

$$\Psi_m^{(n)}(\mathbf{x}) = \frac{w_m}{2} (\mathbf{x} - \mathbf{y}_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi(\mathbf{x} - \mathbf{x}_j)$$

Is  $\phi$  differentiable?

- we can use a simple **majorize-minimize algorithm** to minimize  $\Psi$ . **Huber's curvatures**:

$$\mathbf{g}_j = \phi'(\mathbf{x}^{(n)} - \mathbf{x}_j), \quad \mathbf{c}_j^{(m)} = \omega(\mathbf{x}^{(n)} - \mathbf{x}_j) = \frac{\mathbf{g}_j}{|\mathbf{x}^{(n)} - \mathbf{x}_j|}$$

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



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



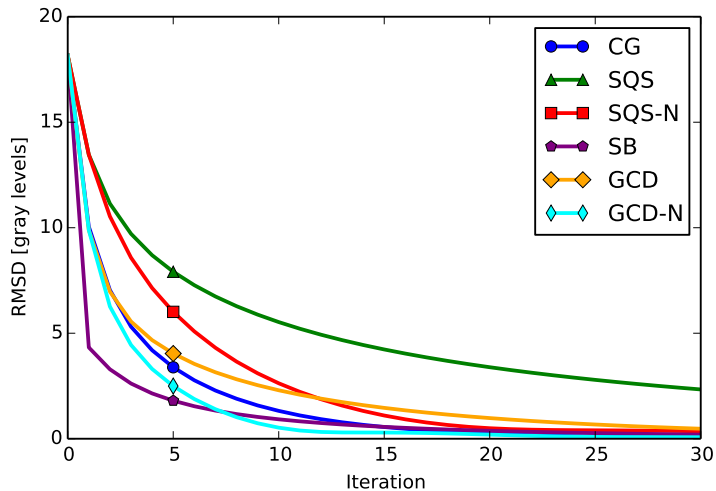
## Noisy image



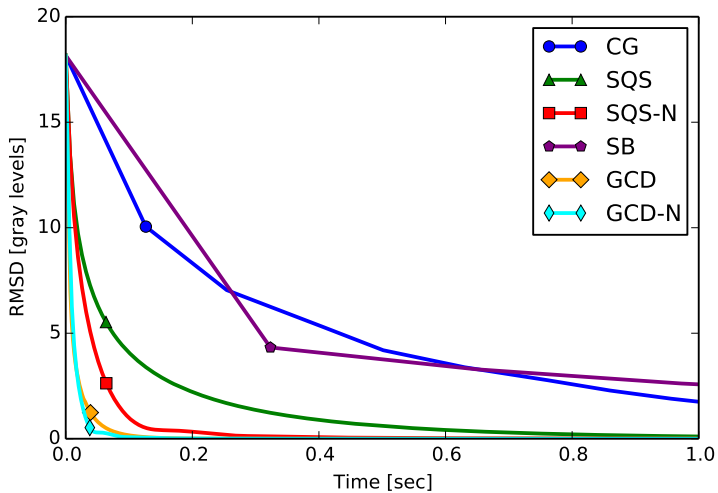
## Denoised image



# RMSD by iteration



# RMSD by time



# The 1D subproblem

$$\mathbf{x}_k^{(n+1)} = \operatorname{argmin}_{\mathbf{x}_k \geq \mathbf{0}} \sum_{m \in S_k} \Psi_m^{(n)}(\mathbf{x}_m)$$
$$\Psi_m^{(n)}(\mathbf{x}) = \frac{w_m}{2} (\mathbf{x} - \mathbf{y}_m)^2 + \sum_{j \in \mathcal{N}_m} \beta_j \phi(\mathbf{x} - \mathbf{x}_j)$$

Is  $\phi$  differentiable?

- we can use a simple majorize-minimize algorithm to minimize  $\Psi$ .

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

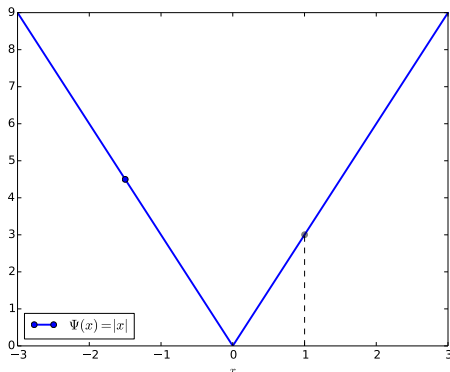
- “corner rounding”
- “Huber-and-pray:” assume  $x \neq x_j$ ; possible numerical problems
- split Bregman: high memory cost, but pretty quick.
- our choice: dual majorize-minimize approach.

# The 1D subproblem

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

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

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



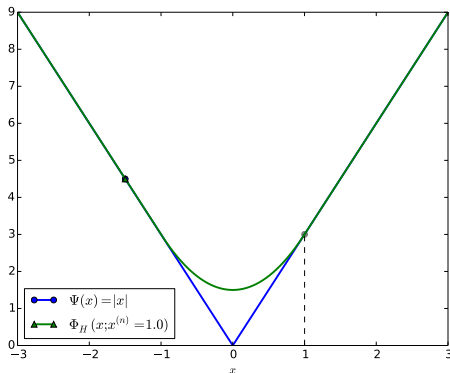


# The 1D subproblem

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

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

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

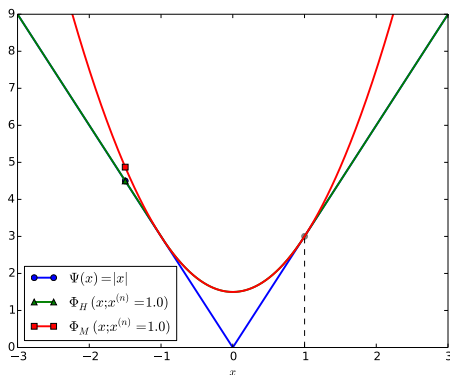


# The 1D subproblem

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

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

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

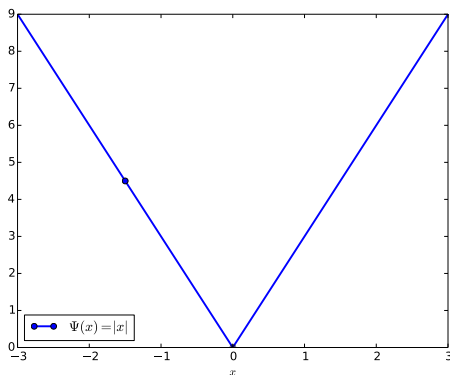


# The 1D subproblem

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

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

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

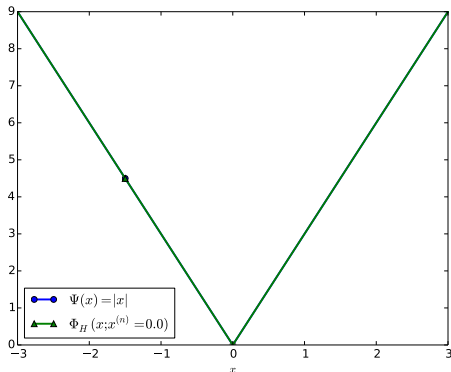


# The 1D subproblem

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

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

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

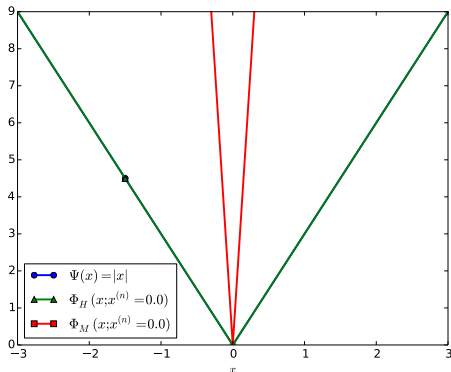


# The 1D subproblem

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

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

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



## Relaxed Huber function majorizer

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

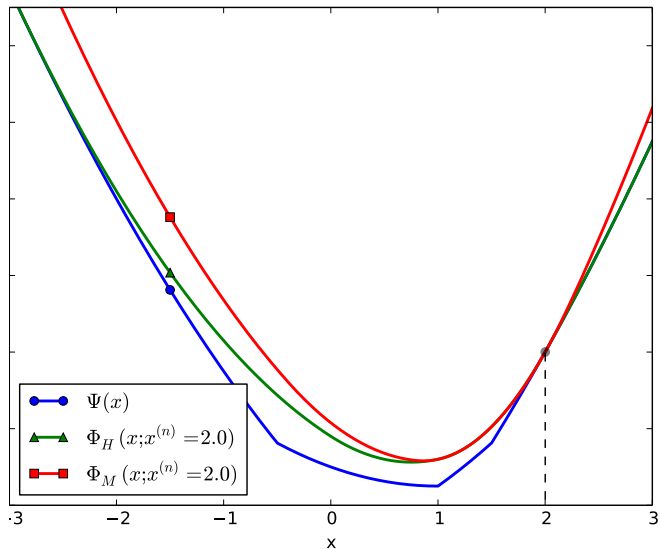
$$\Phi_m(x) = \max_{\gamma \in \Omega^{|\mathcal{N}|}} S(x; \gamma)$$

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

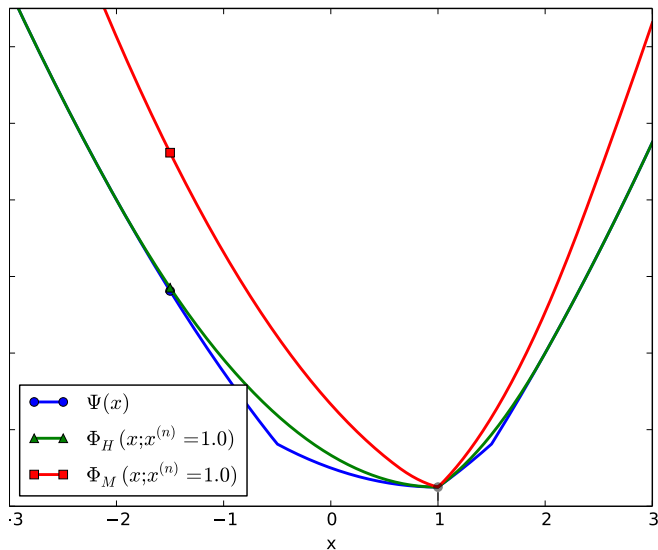
$$s(t, \gamma; \delta) = \gamma t - \frac{\delta}{2}(\gamma^2 - 1)$$

Assertion:  $\Phi_m$  is a majorizer for  $\Psi$  at  $x = x^{(n)}$ . Any  $\Omega$  which contains  $[-1, 1]$  is valid.

# Relaxed Huber function majorizer



# Relaxed Huber function majorizer





# Duality trick

To solve

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

reverse the order of minimization and maximization:

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

$$x(\gamma) = \operatorname{argmin}_x S(x; \gamma) = y - \frac{1}{\mathbf{W}} \beta' \gamma$$

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

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

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

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

A couple more tricks:

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

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

Majorize-minimize (minorize maximize):

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

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

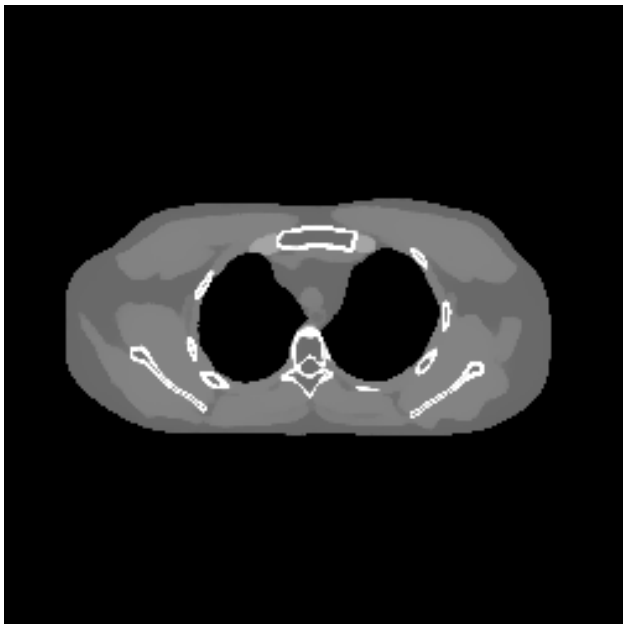
# True image



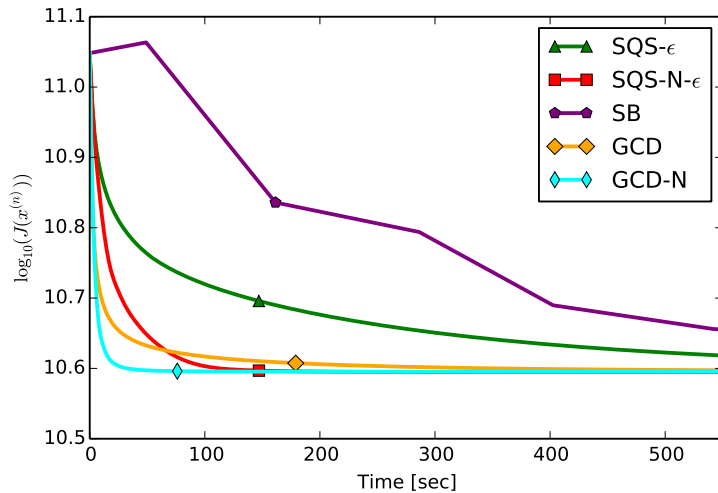
# Noisy image



## Denoised image



# Cost function by time



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!**