Stochastic Optimization with Momentum: A Comparison of "Adam" and Related Methods

Claire Kelling

Penn State University

Department of Statistics

Stochastic Gradient Descent

Stochastic Gradient Descent is often used as an efficient optimization method for stochastic objective functions.

- Proven to be effective in many machine learning applications (Deng et al., 2013; Hinton et al., 2012; Graves et al., 2013)
- Used for speech research, acoustic modeling, and image recognition

Algorithm 1 Stochastic Gradient Descent

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$ (gradient wrt stochastic objective)
2: $\theta_t \leftarrow \theta_{t-1} - \eta g_t$

Computational Challenges:

- Step size can be difficult to tune
- Not always effective for higher-dimensional parameter spaces

Proposed Solution

Adam: Adaptive Moment Estimation [Kingma et al, 2014]

When to use Adam?

- Efficient for stochastic objectives with high-dimensional parameter spaces
- Used in many deep learning applications such as deep adversarial networks, image generation, and image-to-image translation (very popular in classification problems)

Outline

- Classical Momentum
- Nesterov's Accelerated Gradient (NAG, a version of momentum)
- Adam
- Variations of Adam:
  - Adam with NAG, or Nadam
  - Adam and Nadam without bias correction
- Two case studies in logistic regression framework:
  - 2-dimensional case (from homework)
  - 6-dimensional case (email spam classification dataset)

Momentum

Algorithm 1 SGD

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $\theta_t \leftarrow \theta_{t-1} - \eta g_t$

Algorithm 2 Classical Momentum

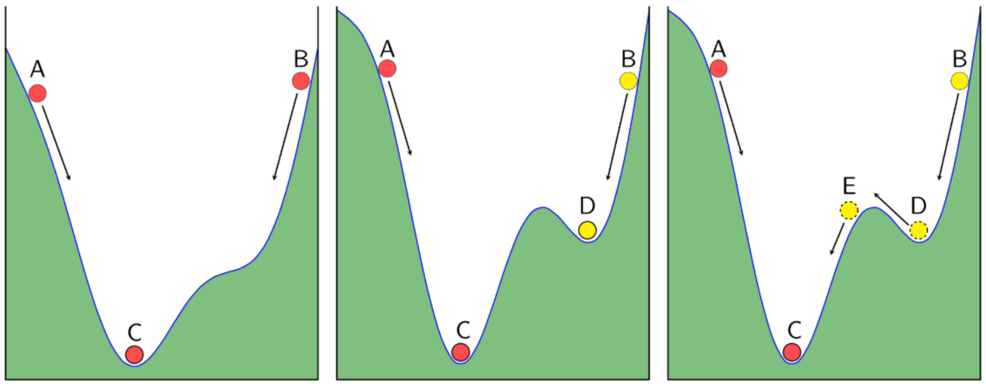
1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $m_t \leftarrow \mu m_{t-1} + g_t$

3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

Why Momentum?
[Polyak 1964, Dozat 2016]

- Gives SGD a short-term memory
- Speeds up convergence
- Smooths and accelerates
- Smaller learning rate



a) GD b) GD c) GD with momentum

https://machinelearningcoban.com/2017/01/16/gradientdescent2/

Accelerated Gradient

Algorithm 2 Classical Momentum (CM)

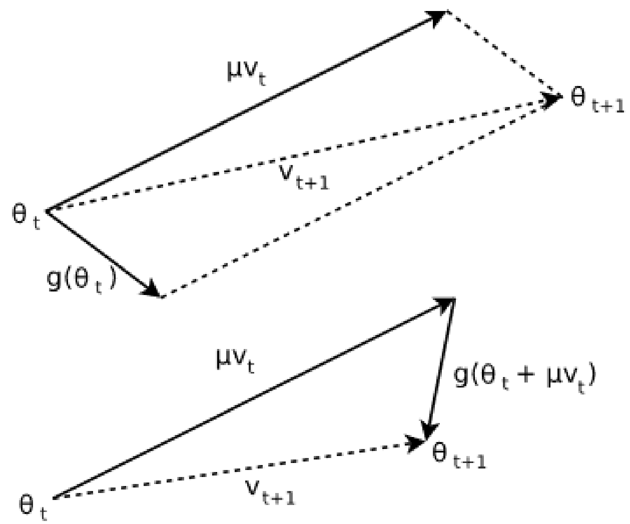
1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + g_t$
3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

Algorithm 3 Nesterov's Accelerated Gradient (NAG)

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta\mu m_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + g_t$
3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

Why NAG? [Sutskever et al 2013]

- Accelerates the convergence
- Better convergence rate guaranteed compared to CM
- It can also be written the same as CM, except adding the step $\bar{m}_t \leftarrow g_t + \mu m_t$ and the update is $\theta_t \leftarrow \theta_{t-1} - \eta \bar{m}_t$



Sutskever et al 2013

Adaptive Learning: Adam

Algorithm 4 Adam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
3: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
4: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
5: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
6: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$

Other L_2 norm methods:
AdaGrad, RMS Prop

Relationship to previous algorithms:

- Incorporates classical momentum with a decaying mean instead of a decaying sum

Details:

- Exponential moving averages of the gradient (m_t) and the squared gradient (n_t)
- Parameters μ and ν control exponential decay rates
- L_2 norm methods allows the algorithm to slow down learning along dimensions that have already changed significantly and speeds up along dimensions only changed slightly

Incorporating NAG: Nadam (Adam with NAG)

Algorithm 4 Adam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$

3: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$

4: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$

5: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$

6: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$

Algorithm 5 Nadam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $\hat{g}_t \leftarrow \frac{g_t}{1-\mu^t}$

3: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$

4: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$

5: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$

6: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$

7: $\bar{m}_t \leftarrow (1-\mu)\hat{g}_t + \mu \hat{m}_t$

8: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$

NAG is proven to converge faster than classical momentum, so we add NAG to Adam, to create Nadam.

Bias Correction

Algorithm 4 Adam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
3: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
4: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
5: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
6: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$

Why bias correction?

- Initialization bias correction
- Offsets some of the instability that initializing m and n to 0 can create

Case Studies

Comparison of 7 algorithms:

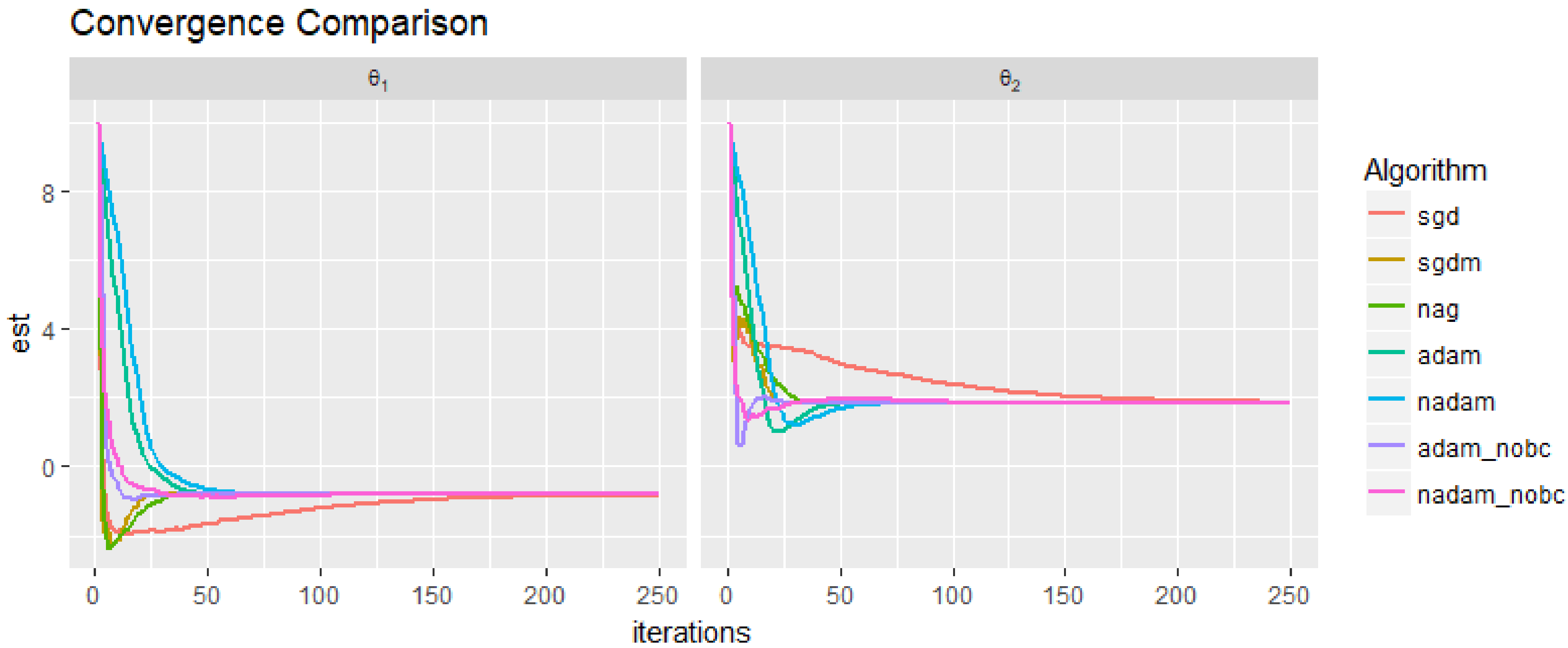
1. SGD
2. SGD with Momentum
3. Nesterov's Accelerated Gradient (NAG)
4. Adam
5. Nadam (Adam with NAG)
6. Adam without bias correction
7. Nadam without bias correction

We will compare using averages over 100 iterations of each algorithm, with a batch size of 1.

- Number of iterations until convergence
- Time until convergence

Logistic Regression, 2-dimensional

	SGD	SGD w/Mom	NAG	Adam	Nadam	Adam w/o BC	Nadam w/o BC
time (s)	0.31	0.04	0.04	0.02	0.10	0.03	0.11
iter	267.81	35.21	40.55	49.43	114.04	30.87	104.96



Logistic Regression, 6-dimensional

Dataset: Spambase Data from UCI Machine Learning Repository [Dua et al 2017]

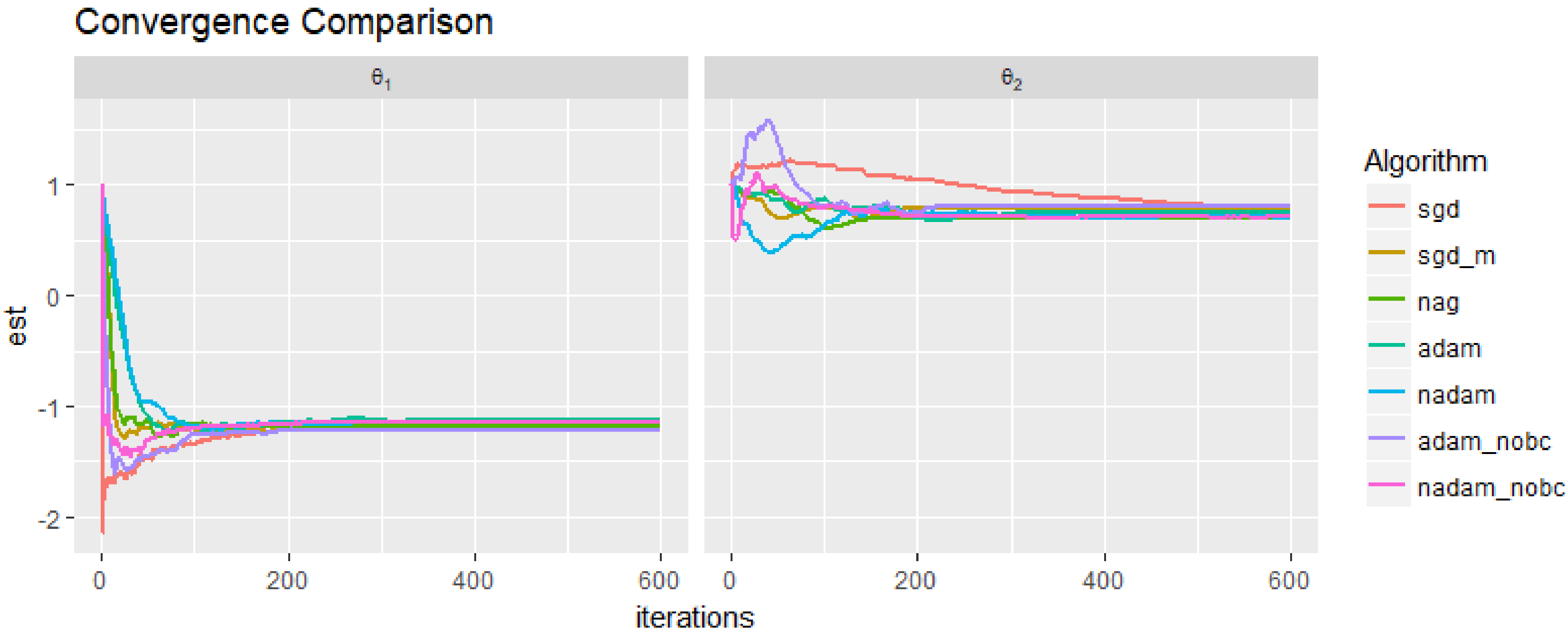
- Common Kaggle/ML Dataset
- Classification of emails as spam or not
- 4,601 emails
- 57 attributes (of which we select 6)
- Word frequency as percentage of total words

Analysis:

- Higher batch size
- Average over 10 iterations

Logistic Regression, 6-dimensional

	SGD	SGD w/Mom	NAG	Adam	Nadam	Adam w/o BC	Nadam w/o BC
time (s)	6.89	0.23	0.29	0.55	4.49	0.41	5.63
iter	749.00	60.67	79.11	213.89	624.67	94.56	704.33



Conclusions and Future Work

In summary,

- All versions of momentum and L2 norm algorithms that we tested are an improvement on SGD
- Nadam (Adam with NAG) takes longer than Adam to converge both in terms of time and the number of iterations
- The bias correction does not seem to have a large impact on convergence, and can slow down convergence

In the future, I would like to consider:

- Other, perhaps noisier objective functions, in addition to logistic regression
- Higher dimensional cases
- Further investigate advantages of bias correction

Proximal Methods

Tobia Boschi

Stat 540 - Spring 2108

Why Proximal Methods?

- **High dimensional convex problems**
  - non-differentiable
  - constrained
  - large-size and parallel implementations

- **Proximal methods to solve LASSO**

Key Idea

- avoid *gradient* and *hessian* computation
- evaluate instead the *proximal operator*:

$\longrightarrow$ small convex optimization problem

Definition

Let $f : R^n \to R$ be a closed proper *convex* function.
The proximal operator $\mathrm{prox}_{\lambda f} : R^n \to R^n$ is defined by:

$$\mathrm{prox}_{\lambda f}(x) = \mathrm{argmin}_z \left(f(z) \quad \frac{1}{2\lambda} ||z - x||_2^2 \right)$$

It balances two goals:

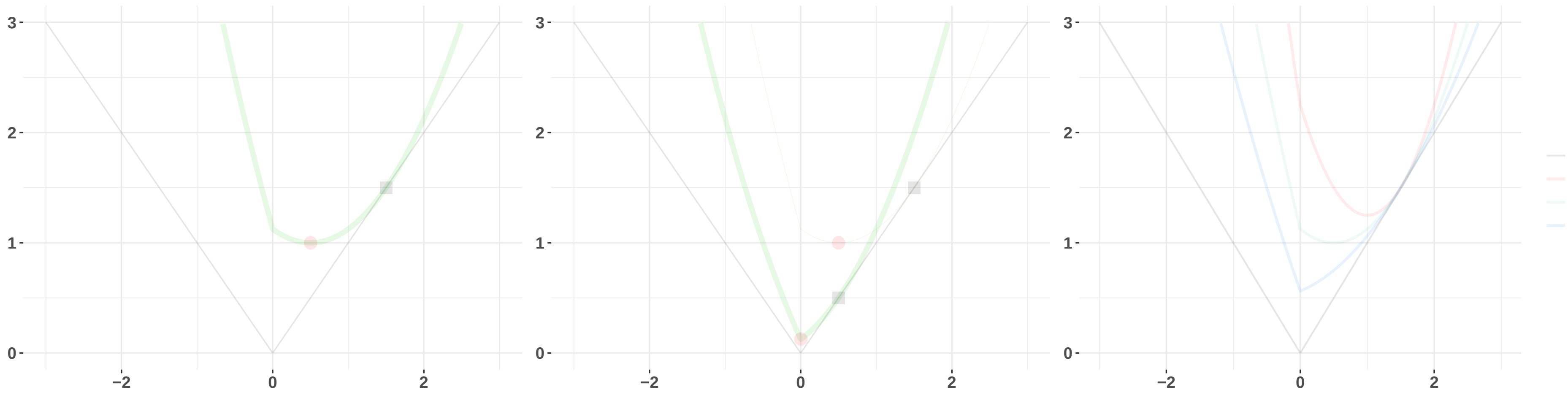
- minimizing f
- staying near x

Possible interpretations

- surrogate method: $f(x) \longrightarrow f(z) \quad \frac{1}{2\lambda} ||z - x||_2^2$
- gradient step for f: $\mathrm{prox}_{\lambda f}(x) \cong x - \lambda \nabla f(x)$

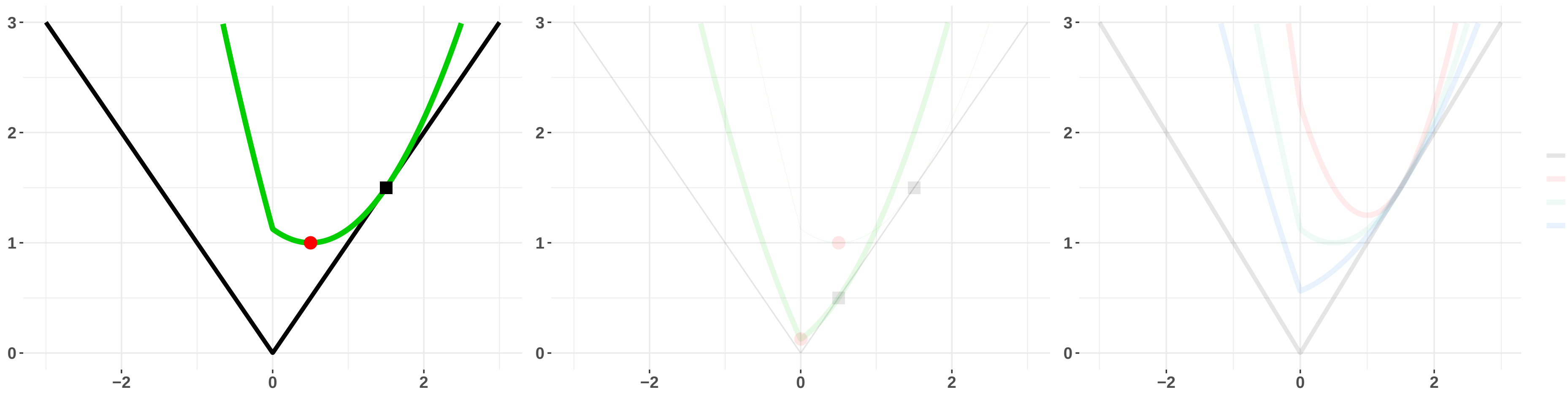
Simple Example

- $f(x) = |x|$
- $|z| \quad \frac{1}{2\lambda}(x - z)^2$
- $\text{prox}_{\lambda f}(x) = \text{sign}(x)(|x| - \lambda)$



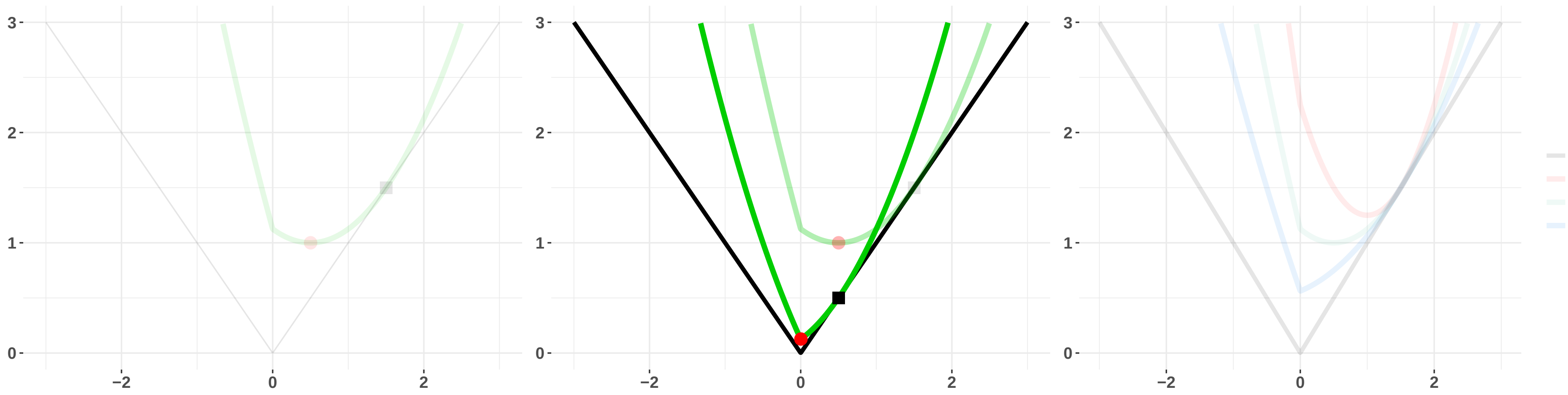
Simple Example

- $f(x) = |x|$
- $|z| \quad \frac{1}{2\lambda}(x - z)^2$
- $\text{prox}_{\lambda f}(x) = \text{sign}(x)(|x| - \lambda)$



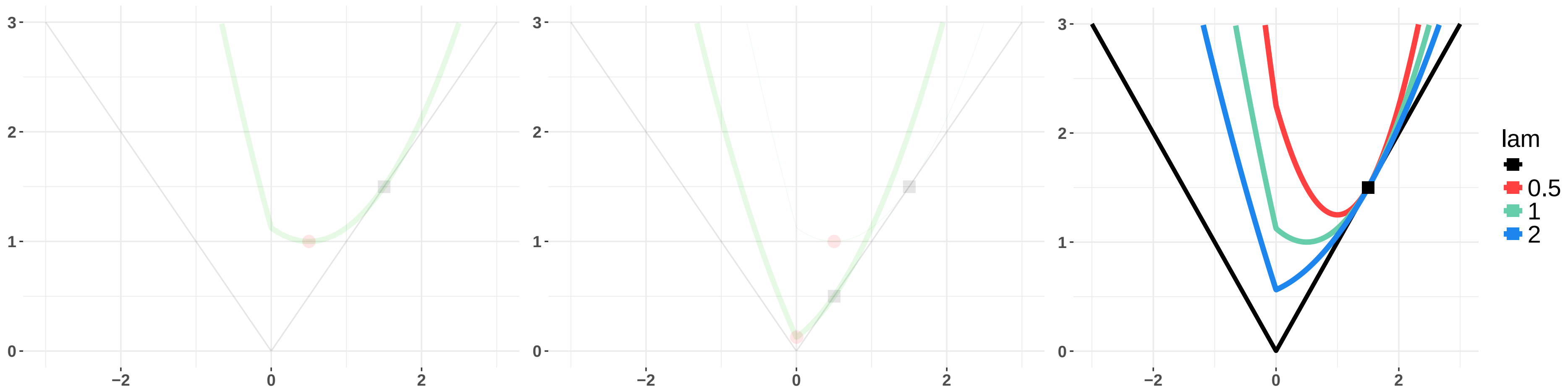
Simple Example

- $f(x) = |x|$
- $|z| \quad \frac{1}{2\lambda}(x - z)^2$
- $\text{prox}_{\lambda f}(x) = \text{sign}(x)(|x| - \lambda)$



Simple Example

- $f(x) = |x|$
- $|z| \quad \frac{1}{2\lambda}(x - z)^2$
- $\text{prox}_{\lambda f}(x) = \text{sign}(x)(|x| - \lambda)$



Proximal Methods and Regression

Let $x \in R^n, b \in R^m, A \in R^{m \times n}$. We want to minimize:

$$g(x) \quad h(x) = \frac{1}{2}||Ax - b||_2^2 \quad h(x)$$

- LASSO: $h(x) = \gamma||x||_1$
- RIDGE: $h(x) = \frac{\gamma}{2}||x||_2^2$
- ELASTIC: $h(x) = \gamma_1||x||_1 \quad \frac{\gamma_2}{2}||x||_2^2$

Implemented Algorithms

- Proximal Gradient
- Proximal ADMM
  (alternating direction method of multipliers)

Proximal Methods and Regression

Let $x \in R^n, b \in R^m, A \in R^{m \times n}$. We want to minimize:

$$g(x) \quad h(x) = \frac{1}{2}||Ax - b||_2^2 \quad h(x)$$

- LASSO: $h(x) = \gamma ||x||_1$
- RIDGE: $h(x) = \frac{\gamma}{2}||x||_2^2$
- ELASTIC: $h(x) = \gamma_1 ||x||_1 \quad \frac{\gamma_2}{2}||x||_2^2$

Implemented Algorithms

- Proximal Gradient
- Proximal ADMM
  (alternating direction method of multipliers)

Proximal Gradient

Pseudo-code

- Gradient step

$$v^k = x^k - \lambda \nabla g(x^k)$$

- Proximal operator step

$$x^{k\quad 1} = \mathrm{prox}_{\lambda h}(v^k)$$

Proximal Gradient

Pseudo-code

- Gradient step

$$v^k = x^k - \lambda \nabla \boldsymbol{g}(x^k)$$

- Proximal operator step

$$x^{k\quad 1} = \mathrm{prox}_{\lambda \boldsymbol{h}}(v^k)$$

- $\nabla g(x) = A'Ax - A'b$
  - precompute $A'A$ and $A'b$
  - at each iteration evaluate $A'Ax$: $\mathcal{O}(n^2)$
- $\left(\mathrm{prox}_{\lambda\gamma \|\boldsymbol{x}\|_1}(x)\right)_i = \mathrm{prox}_{\lambda\gamma |x_i|}(x) = \mathrm{sign}(x)(|x| - \lambda\gamma)$

Proximal Gradient

- Proximal Gradient as an Majorization-Minimization algorithm

$$\hat{g}_\lambda(x, y) = g(y) \quad \nabla g(y)'(x - y) \quad \frac{1}{2\lambda}||x - y||_2^2 \geq g(x)$$

Majorization step

- compute $\hat{g}_\lambda(x, x^k) \quad h(x)$

Minimizaztion step

- $\min_x \left(\hat{g}_\lambda(x, x^k) \quad h(x)\right) = \mathrm{prox}_{\lambda \boldsymbol{h}}\left(x^k - \lambda \nabla \boldsymbol{g}(x^k)\right)$
  $= \mathrm{prox}_{\lambda \boldsymbol{h}}(v^k)$

Proximal Gradient

- Proximal Gradient as an Majorization-Minimization algorithm

$$\hat{g}_\lambda(x, y) = g(y) \quad \nabla g(y)'(x - y) \quad \frac{1}{2\lambda}||x - y||_2^2 \geq g(x)$$

Majorization step

- compute $\hat{g}_\lambda(x, x^k) \quad h(x)$

Minimizaztion step

- $\min_x \left(\hat{g}_\lambda(x, x^k) \quad h(x)\right) = \mathrm{prox}_{\lambda \boldsymbol{h}}\left(x^k - \lambda \nabla \boldsymbol{g}(x^k)\right)$

  $= \mathrm{prox}_{\lambda \boldsymbol{h}}(v^k)$

- **Backtracking of λ**
  - $x = \mathrm{prox}_{\lambda \boldsymbol{h}}\left(x^k - \lambda \nabla \boldsymbol{g}(x^k)\right)$
  - *reduce λ since:* $\quad g(x) \leq \hat{g}_\lambda(x, x^k)$

Proximal ADMM

Note: minimize $g(x) \quad h(x)$ is equivalent to minimize:

$$g(x) \quad h(z) \quad \textit{subject to} \quad x - z = 0$$

Proximal ADMM

Note: minimize $g(x) \quad h(x)$ is equivalent to minimize:

$$g(x) \quad h(z) \quad \textit{subject to} \quad x - z = 0$$

Pseudo-code

- ◎ $x^{k\ \ 1} = \text{prox}_{\lambda g}(z^k - u^k)$
- ◎ $z^{k\ \ 1} = \text{prox}_{\lambda h}(x^{k\ \ 1} \quad u^k)$
- ◎ $u^{k\ \ 1} = u^k \quad x^{k\ \ 1} - z^{k\ \ 1}$

- $g(x) = \frac{1}{2}||Ax - b||_2^2 = \frac{1}{2}(b - Ax)'(b - Ax)$
- $\text{prox}_{\lambda g}(x) = (I_n \quad \lambda A'A)^{-1}(x \quad \lambda A'b)$

Proximal ADMM

How to compute $(I_n \quad \lambda A'A)^{-1}$:

- $\boldsymbol{n > m}$
  - $C = \mathrm{chol}(I_n \quad \lambda A'A)$
  - $(I_n \quad \lambda A'A)^{-1} = (C')^{-1}C^{-1}$
  - $\mathcal{O}(n^3)$

- $\boldsymbol{m > n}$
  - inversion lemma: $(I_n \quad \lambda A'A)^{-1} = A'(I_m \quad \lambda AA')^{-1}A$
  - $C = \mathrm{chol}(I_m \quad \lambda AA')$
  - $\mathcal{O}(m^3)$

Simulation 1

- A = random normal$(m \times n)$
- x_0 = random normal$(n \times 1)$
- $b = Ax_0$ err
- sparsity = 0.95
- $\lambda = 1$
- $\gamma = 0.1||A'b||_\infty$

◎ $m = 500,\ n \uparrow$

	sec			**obj**		
	CVX	grad	ADMM	CVX	err grad	err ADMM
$n = 10^2$	0.10966	0.00136	0.02259	0.4606	0	0
$n = 10^3$	11.10568	0.02289	0.02589	8.5773	0.0010	0.0003
$n = 10^4$	155.49445	3.97270	0.57923	86.8624	0.1288	0.00665
$n = 4 \cdot 10^4$	820.12007	111.32019	4.21288	305.0444	1.0231	0.08624

Simulation 1

- A = random normal$(m \times n)$
- x_0 = random normal$(n \times 1)$
- $b = Ax_0$ err
- sparsity = 0.95
- $\lambda = 1$
- $\gamma = 0.1||A'b||_\infty$

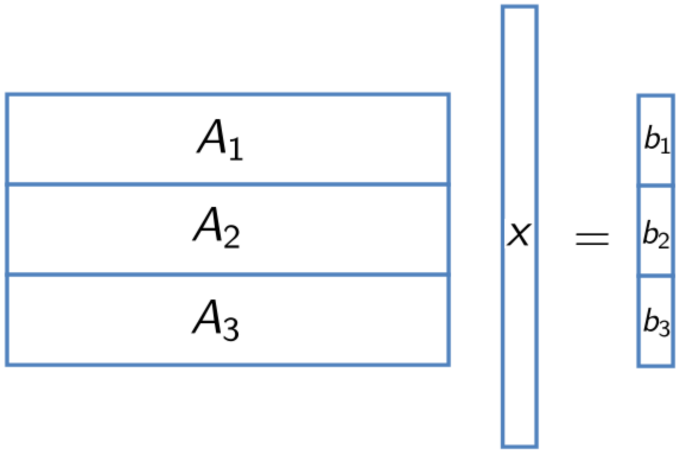
◎ $m = 500,\ n \uparrow$

		sec			**obj**	
	CVX	grad	ADMM	CVX	err grad	err ADMM
$n = 10^2$	0.10966	0.00136	0.02259	0.4606	0	0
$n = 10^3$	11.10568	0.02289	0.02589	8.5773	0.0010	0.0003
$n = 10^4$	155.49445	3.97270	0.57923	86.8624	0.1288	0.00665
$n = 4 \cdot 10^4$	820.12007	111.32019	4.21288	305.0444	1.0231	0.08624

$n >> m \Rightarrow$ ADMM $\mathcal{O}(m^3)$ > Gradient $\mathcal{O}(n^2)$

Distributed ADMM

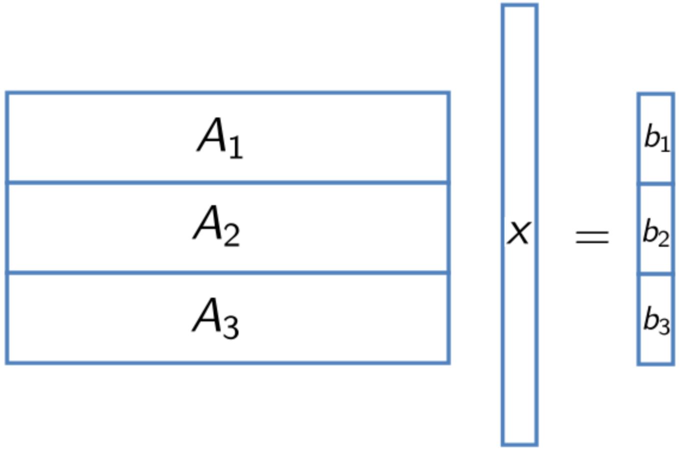
◎ reduce m $\longrightarrow$ divide A in S blocks



- $g(x) = \sum g_i(x)$
  $= \sum \frac{1}{2}||A_i x - b_i||_2^2$
- $\mathcal{O}\left((m/S)^3\right)$

Distributed ADMM

◎ reduce $m \longrightarrow$ divide A in S blocks



- $g(x) = \sum g_i(x)$
  $= \sum \frac{1}{2}||A_i x - b_i||_2^2$
- $\mathcal{O}((m/S)^3)$

Pseudo-code

◎ $x_i^{k\ \ 1} = \text{prox}_{\lambda g_i}(z^k - u_i^k)$

◎ $z^{k\ \ 1} = \text{prox}_{\frac{\lambda h}{S}}(\bar{x}^{k\ \ 1} \quad \bar{u}^k)$

◎ $u_i^{k\ \ 1} = u_i^k \quad x_i^{k\ \ 1} - z^{k\ \ 1}$

Simulation 2

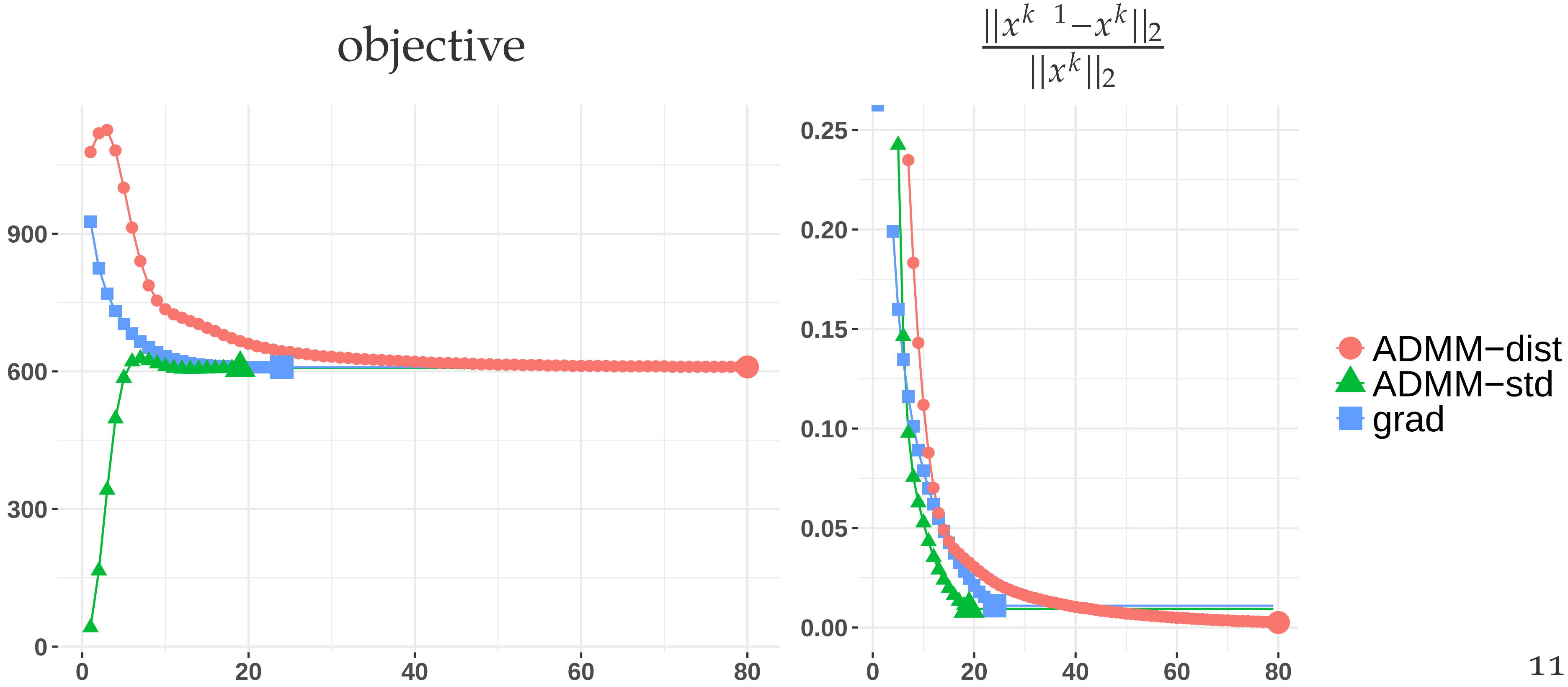
- $m = 10.000, \; n = 50.000$
- $S = 10$

sec			obj		
grad	ADMM	distr	grad	ADMM	distr
3463	423	90	609.012	607.046	609.747

Simulation 2

- $m = 10.000, \; n = 50.000$
- $S = 10$

sec			obj		
grad	ADMM	distr	grad	ADMM	distr
3463	423	90	609.012	607.046	609.747



Conclusions

Pros

- non-smooth problem
- much faster
- easy to parallelize
- good approximations

Cons

- convex problems
- pointwise estimation
- approximations

More work...

- sensitivity study for λ and γ
- add performance indicators *(prediction error)*
- study constrained problems *(matrix decomposition)*