ORF522 – Linear and Nonlinear Optimization

18. Operator splitting algorithms

Bartolomeo Stellato — Fall 2021

$\boxed{R_T, C_T}$

Ed Forum

- Why do we try to only use maximal monotone operators (slide 42, theory)?

If an operator T is maximal monotone, it's domain is the whole $\mathbf{R}^n$. This means that, if we apply iterations of the form $x^{k+1} = R_T(x^k)$, we never risk to go outside the domain of R_T (and map x^k to the empty set). In practice, non-maximal monotone operators are are usually the ones we cannot efficiently deal with (e.g., evaluate the resolvent of the subdifferential of a nonconvex function f: $(I + \partial f)^{-1}$).

- Why does nonexpansiveness of an operator tell us that it is a function?

An operator T is L-Lipschitz if

$$\|T(x) - T(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbf{dom}\, T$$

(lec 16 slide 35)

Fact If T is Lipschitz, then it is single-valued

Proof If $y = T(x), z = T(x)$, then $\|y - z\| \leq L\|x - x\| = 0 \Longrightarrow y = z$ ■

Recap

Resolvent and Cayley operators

The **resolvent** of operator A is defined as

$$R_A = (I + A)^{-1}$$

The **Cayley (reflection) operator** of A is defined as

$$C_A = 2R_A - I = 2(I + A)^{-1} - I$$

Properties

- If A is maximal monotone, $\mathbf{dom}\, R_A = \mathbf{dom}\, C_A = \mathbf{R}^n$ (Minty's theorem)
- If A is **monotone**, R_A and C_A are **nonexpansive** (thus functions)
- **Zeros** of A are **fixed points** of R_A and C_A

Resolvent and Cayley operators

The **resolvent** of operator A is defined as

$$R_A = (I + A)^{-1}$$

The **Cayley (reflection) operator** of A is defined as

$$C_A = 2R_A - I = 2(I + A)^{-1} - I$$

Properties

- If A is maximal monotone, $\mathbf{dom}\, R_A = \mathbf{dom}\, C_A = \mathbf{R}^n$ (Minty's theorem)
- If A is **monotone**, R_A and C_A are **nonexpansive** (thus functions)
- **Zeros** of A are **fixed points** of R_A and C_A

Key result we can solve $0 \in A(x)$ by finding fixed points of C_A or R_A

“multiplier to residual” mapping

minimize $f(x)$
subject to $Ax = b$ $\longrightarrow$

Lagrangian

$$L(x,y) = f(x) + y^T(Ax - b)$$

Dual problem

maximize $g(y) = \min_x L(x,y) = -\max_x -L(x,y) = -(f^*(-A^Ty) + y^Tb)$

Operator

$T(y) = b - Ax$, where $x = \mathrm{argmin}_z\, L(z,y)$ $\longrightarrow$

Monotonicity

If f CCP, then T is monotone

Proof

$$0 \in \partial f(x) + A^Ty \quad \Longleftrightarrow \quad x = (\partial f)^{-1}(-A^Ty)$$

Therefore, $T(y) = b - A(\partial f)^{-1}(-A^Ty) = \partial_y\left(b^Ty + f^*(-A^Ty)\right) = \boxed{\partial(-g)}$ ■

Summary of monotone and cocoercive operators

Monotone

$$(T(x) - T(y))^T(x - y) \geq 0$$

$$\uparrow \; \mu = 0$$

Strongly monotone

$$(T(x) - T(y))^T(x - y) \geq \mu\|x - y\|^2$$

$$\xleftrightarrow[F = T^{-1}]{}$$

Lipschitz

$$\|F(x) - F(y)\| \leq L\|x - y\|$$

$$\uparrow \; L = 1/\mu$$

Cocoercive

$$(F(x) - F(y))^T(x - y) \geq \mu\|F(x) - F(y)\|^2$$

$$\updownarrow \; G = I - 2\mu F$$

Nonexpansive

$$\|G(x) - G(y)\| \leq \|x - y\|$$

Strongly monotone and cocoercive subdifferential

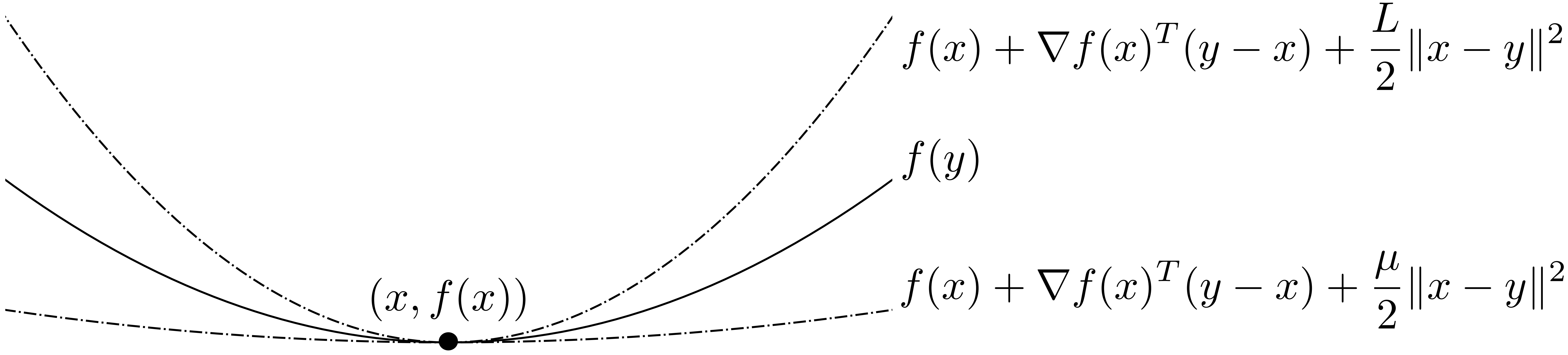
f is μ **-strongly convex** $\iff$ ∂f μ**-strongly monotone**

$$(\partial f(x) - \partial f(y))^T(x - y) \geq \mu\|x - y\|^2$$

f is L**-smooth**

$\iff$ ∂f L**-Lipschitz** and $\partial f = \nabla f$: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$

$\iff$ ∂f $(1/L)$**-cocoercive**: $(\nabla f(x) - \nabla f(y))^T(x - y) \geq (1/L)\|\nabla f(x) - \nabla f(y)\|^2$



Inverse of subdifferential

If f is CCP, then, $\quad (\partial f)^{-1} = \partial f^*$

Proof

$$
\begin{aligned}
(u, v) \in \mathbf{gph}(\partial f)^{-1} &\iff (v, u) \in \mathbf{gph}\partial f \\
&\iff u \in \partial f(v) \\
&\iff 0 \in \partial f(v) - u \\
&\iff v \in \underset{x}{\operatorname{argmin}}\, f(x) - u^T x \\
&\iff f^*(u) = u^T v - f(v)
\end{aligned}
$$

Therefore, $f(v) + f^*(u) = u^T v$. If f is CCP, then $f^{**} = f$ and we can write

$$
f^{**}(v) + f^*(u) = u^T v \quad \iff \quad (u, v) \in \mathbf{gph}\partial f^* \qquad \blacksquare
$$

Strong convexity is the dual of smoothness

f is μ-strongly convex $\quad\Longleftrightarrow\quad$ f^* is $(1/\mu)$-smooth

Proof

$$\begin{aligned} f \quad \mu\text{-strongly convex} \quad &\Longleftrightarrow \quad \partial f \quad \mu\text{-strongly monotone} \\ &\Longleftrightarrow \quad (\partial f)^{-1} = \partial f^* \quad \mu\text{-cocoercive} \\ &\Longleftrightarrow \quad f^* \quad (1/\mu)\text{-smooth} \qquad \blacksquare \end{aligned}$$

Remark: strong convexity and (strong) smoothness are **dual**

Forward step contractions

Given T L-Lipschitz and μ-strongly monotone, then $I - \gamma T$ converges linearly at rate $\sqrt{1 - 2\gamma\mu + \gamma^2 L^2}$, with optimal step $\gamma = \mu / L^2$.

Forward step contractions

Given T L-Lipschitz and μ-strongly monotone, then $I - \gamma T$ converges linearly at rate $\sqrt{1 - 2\gamma\mu + \gamma^2 L^2}$, with optimal step $\gamma = \mu / L^2$.

Proof

$$\begin{aligned}
\|(I - \gamma T)(x) - (I - \gamma T)(y)\|^2 &= \|x - y + \gamma T(x) - \gamma T(y)\|^2 \\
&= \|x - y\|^2 - 2\gamma(T(x) - T(y))^T(x - y) + \gamma^2\|T(x) - T(y)\|^2 \\
&\leq (1 - 2\gamma\mu + \gamma^2 L^2)\|x - y\|^2
\end{aligned}$$

$\blacksquare$

Forward step contractions

Given T L-Lipschitz and μ-strongly monotone, then $I - \gamma T$ converges linearly at rate $\sqrt{1 - 2\gamma\mu + \gamma^2 L^2}$, with optimal step $\gamma = \mu / L^2$.

Proof

$$\begin{aligned} \|(I - \gamma T)(x) - (I - \gamma T)(y)\|^2 &= \|x - y + \gamma T(x) - \gamma T(y)\|^2 \\ &= \|x - y\|^2 - 2\gamma \underbrace{(T(x) - T(y))^T (x - y)}_{\text{strongly monotone}} + \gamma^2 \underbrace{\|T(x) - T(y)\|^2}_{\text{Lipschitz}} \\ &\le (1 - 2\gamma\mu + \gamma^2 L^2)\|x - y\|^2 \end{aligned}$$

■

Forward step contractions

Given T L-Lipschitz and μ-strongly monotone, then $\quad I - \gamma T$

converges linearly at rate $\sqrt{1 - 2\gamma\mu + \gamma^2 L^2}$, with optimal step $\gamma = \mu/L^2$.

Proof

strongly monotone

Lipschitz

$$
\begin{aligned}
\|(I-\gamma T)(x) - (I-\gamma T)(y)\|^2 &= \|x - y + \gamma T(x) - \gamma T(y)\|^2 \\
&= \|x-y\|^2 - 2\gamma (T(x) - T(y))^T (x-y) + \gamma^2 \|T(x) - T(y)\|^2 \\
&\leq (1 - 2\gamma\mu + \gamma^2 L^2)\|x-y\|^2
\end{aligned}
$$

■

Remarks

- It applies to **gradient descent** with L-smooth and μ-strongly convex f
- Better rate in gradient descent lecture. We can get it by bounding derivative: $\|D(I - \gamma \nabla^2 f(x))\|_2 \leq \max\{|1-\gamma L|, |1-\gamma\mu|\}$. Optimal step $\gamma = 2/(\mu + L)$ and factor $(\mu/L - 1)(\mu/L + 1)$.

Resolvent contractions

If A is μ-strongly monotone, then

$$R_A = (I + A)^{-1}$$

is a contraction with Lipschitz parameter $1/(1 + \mu)$

Resolvent contractions

If A is μ-strongly monotone, then

$$R_A = (I + A)^{-1}$$

is a contraction with Lipschitz parameter $1/(1+\mu)$

Proof

$$\begin{aligned}
A\ \mu\text{-strongly monotone} \quad &\implies \quad (I + A) \qquad (1+\mu)\text{-strongly monotone} \\
&\implies \quad R_A = (I + A)^{-1} \quad (1+\mu)\text{-cocoercive} \\
&\implies \quad R_A \qquad (1/(1+\mu))\text{-Lipschitz} \qquad \blacksquare
\end{aligned}$$

Cayley contractions

If A is μ-strongly monotone and L-Lipschitz, then

$$C_{\gamma A} = 2R_{\gamma A} - I = 2(I + \gamma A)^{-1} - I$$

is a contraction with factor $\sqrt{1 - 4\gamma\mu/(1+\gamma L)^2}$

Remark need also Lipschitz condition

Proof [Page 20, PMO]

Cayley contractions

If A is μ-strongly monotone and L-Lipschitz, then

$$C_{\gamma A} = 2R_{\gamma A} - I = 2(I + \gamma A)^{-1} - I$$

is a contraction with factor $\sqrt{1 - 4\gamma\mu/(1+\gamma L)^2}$

Remark need also Lipschitz condition

Proof [Page 20, PMO]

If, in addition, $A = \partial f$ where f is CCP, then $C_{\gamma A}$ converges with factor $(\sqrt{\mu/L} - 1)/(\sqrt{\mu/L} + 1)$ and optimal step $\gamma = 1/\sqrt{\mu L}$

Proof
[Linear Convergence and Metric Selection for Douglas-Rachford Splitting and ADMM, Giselsson and Boyd]

Requirements for contractions

	Operator A	**Function** f ($A = \partial f$)
Forward step $I - \gamma A$	μ-strongly monotone	μ-strongly convex L-smooth
Resolvent $R_A = (I + A)^{-1}$	μ-strongly monotone	μ-strongly convex L-smooth
Cayley $C_A = 2(I + A)^{-1} - I$	μ-strongly monotone L-Lipschitz	μ-strongly convex L-smooth
		faster convergence

Key to contractions: strong monotonicity/convexity

Today's lecture

[PMO][LSMO][PA][ADMM]

Operator splitting algorithms

- Proximal point method
- Forward-backward splitting
- Douglas-Rachford splitting
- Alternating Direction Method of Multipliers
- Examples
- Distributed optimization

Proximal point method

Proximal point method

Resolvent iterations

$$x^{k+1} = R_A(x^k) = (I + A)^{-1}(x^k)$$

Many traditional algorithms are **proximal point method** with a specific A

Proximal point method

Resolvent iterations

$$x^{k+1} = R_A(x^k) = (I + A)^{-1}(x^k)$$

Many traditional algorithms are **proximal point method** with a specific A

If $A = \partial t f$, we get **proximal minimization algorithm**

$$x^{k+1} = \mathbf{prox}_{tf}(x^k) = \underset{z}{\operatorname{argmin}} \left(t f(z) + \frac{1}{2} \| z - x^k \|_2^2 \right)$$

Proximal point method

Resolvent iterations

$$x^{k+1} = R_A(x^k) = (I + A)^{-1}(x^k)$$

Many traditional algorithms are **proximal point method** with a specific A

If $A = \partial t f$, we get **proximal minimization algorithm**

$$x^{k+1} = \mathbf{prox}_{tf}(x^k) = \underset{z}{\operatorname{argmin}} \left(tf(z) + \frac{1}{2}\|z - x^k\|_2^2 \right)$$

Proximal minimization properties

- R_A is $1/2$ averaged: $R_A = (1/2)I + (1/2)C_A \quad \Longrightarrow \quad R_{t\partial f}$ converges $\forall t$
- $\mathbf{fix}\, R_{\partial t f}$ are zeros of ∂f: **optimal solutions**
- If f μ-strongly convex, $R_{\partial t f}$ contraction: **linear convergence**
- Useful only if you can evaluate $\mathbf{prox}_{tf}$ efficiently

Method of multipliers

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

Lagrangian

$$L(x, y) = f(x) + y^T(Ax - b)$$

Method of multipliers

minimize $\quad f(x)$

subject to $\quad Ax = b$

Lagrangian

$$L(x, y) = f(x) + y^T(Ax - b)$$

Dual problem

maximize $\quad g(y) = -(f^*(-A^T y) + y^T b)$

Method of multipliers

minimize $\quad f(x)$

subject to $\quad Ax = b$

Lagrangian

$$L(x, y) = f(x) + y^T(Ax - b)$$

Dual problem

maximize $\quad g(y) = -(f^*(-A^T y) + y^T b)$

Multiplier to residual map operator

$T(y) = b - Ax$, where $x = \operatorname{argmin}_z L(z, y) \longrightarrow T(y) = \partial(-g)$

Therefore, $\quad \partial(-g)(y) = b - Ax, \qquad 0 \in \partial f(x) + A^T y$

Method of multipliers

minimize $\quad f(x)$
subject to $\quad Ax = b$

Lagrangian

$$L(x, y) = f(x) + y^T(Ax - b)$$

Dual problem

maximize $\quad g(y) = -(f^*(-A^T y) + y^T b)$

Multiplier to residual map operator

$T(y) = b - Ax$, where $x = \operatorname{argmin}_z L(z, y) \longrightarrow T(y) = \partial(-g)$

Therefore, $\quad \partial(-g)(y) = b - Ax, \quad 0 \in \partial f(x) + A^T y$

Solve the dual with proximal point method

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

Method of multipliers

Derivation

Solve the dual with proximal point method

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

where $\quad \partial(-g)(y) = b - Ax, \qquad$ with x such that $\quad 0 \in \partial f(x) + A^T y$

Method of multipliers

Derivation

Solve the dual with proximal point method

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

where $\quad \partial(-g)(y) = b - Ax, \qquad$ with x such that $\quad 0 \in \partial f(x) + A^T y$

Resolvent reformulation

$$y^{k+1} = R_{t\partial(-g)}(y^k) \iff y^{k+1} + t\boxed{\partial(-g)(y^{k+1})} = y^k$$

$$\iff y^{k+1} + t(b - Ax^{k+1}) = y^k, \quad \text{with} \quad 0 \in \partial f(x^{k+1}) + A^T y^{k+1}$$

Method of multipliers

Derivation

Solve the dual with proximal point method

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

where $\quad \partial(-g)(y) = b - Ax, \qquad$ with x such that $\quad 0 \in \partial f(x) + A^T y$

Resolvent reformulation

$$\begin{aligned} y^{k+1} = R_{t\partial(-g)}(y^k) \iff & y^{k+1} + t\partial(-g)(y^{k+1}) = y^k \\ \iff & y^{k+1} + t(b - Ax^{k+1}) = y^k, \quad \text{with} \quad 0 \in \partial f(x^{k+1}) + A^T \boxed{y^{k+1}} \end{aligned}$$

x^{k+1} minimizes the **augmented Lagrangian** $L_t(x, y^{k+1})$

$$0 \in \partial f(x^{k+1}) + A^T(y^k + t(Ax^{k+1} - b))$$

$$\implies \quad x^{k+1} \in \underset{x}{\operatorname{argmin}}\, f(x) + (y^k)^T(Ax - b) + \boxed{(t/2)\|Ax - b\|^2} = \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$

Method of multipliers (augmented Lagrangian method)

Primal

minimize $f(x)$

subject to $Ax = b$

Dual

maximize $g(y) = -(f^*(-A^T y) + y^T b)$

Iterates

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

$$\downarrow$$

$$x^{k+1} \in \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$

$$y^{k+1} = y^k + t(Ax^{k+1} - b)$$

Method of multipliers (augmented Lagrangian method)

Primal

minimize $f(x)$

subject to $Ax = b$

Iterates

$$y^{k+1} = R_{t\partial(-g)}(y^k)$$

$$\downarrow$$

$$x^{k+1} \in \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$

$$y^{k+1} = y^k + t(Ax^{k+1} - b)$$

Dual

maximize $g(y) = -(f^*(-A^T y) + y^T b)$

Properties

- Always converges with CCP f for any $t > 0$
- If f L-smooth

  f^* and g are $(1/L)$-strongly convex

  $R_{\partial(-g)}$ is a contraction: **linear convergence**
- If f strictly convex ($>$), then argmin has a unique solution ($\in$ becomes $=$)
- Useful when f L-smooth and A sparse

Method of multipliers dual feasibility

minimize $\quad f(x)$

subject to $\quad Ax = b$

$$x^{k+1} \in \operatorname*{argmin}_x L_t(x, y^k)$$

$$y^{k+1} = y^k + t(Ax^{k+1} - b)$$

Optimality conditions (primal and dual feasibility)

$$Ax = b, \qquad \partial f(x) + A^T y \ni 0$$

Method of multipliers dual feasibility

$$x^{k+1} \in \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$

$$y^{k+1} = y^k + t(Ax^{k+1} - b)$$

minimize	$f(x)$
subject to	$Ax = b$

Optimality conditions (primal and dual feasibility)

$$Ax = b, \qquad \partial f(x) + A^T y \ni 0$$

From x^{k+1} update

$$\begin{aligned} 0 &\in \partial f(x^{k+1}) + A^T y^k + tA^T(Ax^{k+1} - b) \\ &= \partial f(x^{k+1}) + A^T y^{k+1} \end{aligned} \longrightarrow \quad (x^{k+1}, y^{k+1})$$

dual feasible

primal feasible in the limit, i.e. $Ax^k - b \to 0$

Forward-backward splitting

Operator splitting
Main idea

We would like to solve

$$0 \in F(x), \qquad F \text{ maximal monotone}$$

Split the operator

$$F = A + B, \qquad A \text{ and } B \text{ are maximal monotone}$$

Solve by evaluating

$$R_A = (I + A)^{-1} \qquad\qquad C_A = 2R_A - I$$

or

$$R_B = (I + B)^{-1} \qquad\qquad C_B = 2R_B - I$$

Useful when R_A and R_B are cheaper than R_F

Forward-backward splitting

Goal

Find x such that $0 \in A(x) + B(x)$

Forward-backward splitting

Goal

Find x such that $0 \in A(x) + B(x)$

Rewrite optimality condition

$$
\begin{aligned}
0 \in (A + B)(x) &\iff 0 \in t(A + B)(x) \\
&\iff 0 \in (I + tB)(x) - (I - tA)(x) \\
&\iff (I + tB)(x) \ni (I - tA)(x) \\
&\iff x = (I + tB)^{-1}(I - tA)(x) \\
&\iff x = R_{tB}(I - tA)(x)
\end{aligned}
$$

Forward-backward splitting

Goal

Find x such that $0 \in A(x) + B(x)$

Rewrite optimality condition

$$\begin{aligned} 0 \in (A+B)(x) &\iff 0 \in t(A+B)(x) \\ &\iff 0 \in (I+tB)(x) - (I-tA)(x) \\ &\iff (I+tB)(x) \ni (I-tA)(x) \\ &\iff x = (I+tB)^{-1}(I-tA)(x) \\ &\iff x = R_{tB}(I-tA)(x) \end{aligned}$$

Iterations

$$x^{k+1} = R_{tB}(I-tA)(x^k)$$

Forward-backward splitting

Properties

Iterations

$$x^{k+1} = \underbrace{R_{tB}}_{\text{resolvent}}\underbrace{(I - tA)}_{\text{forward step}}(x^k)$$

Properties

- R_{tB} is $1/2$ averaged of C_{tB}
- If A is μ-cocoercive then $I - 2\mu A$ is nonexpansive
  $\Rightarrow I - tA$ is averaged for $t \in (0, 2\mu)$
- Therefore forward-backward splitting converges
- If either A or B is strongly monotone, then **linear convergence**

Proximal gradient descent as forward-backward splitting

minimize $\quad f(x) + g(x)$

f is L-smooth
g is nonsmooth but proxable

Therefore, ∇f is $(1/L)$-cocoercive and ∂g maximal monotone

Proximal gradient descent

$$
\begin{aligned}
x^{k+1} &= R_{t\partial g}(I - t\nabla f)(x^k) \\
&= \mathbf{prox}_{tg}(x^k - t\nabla f(x^k))
\end{aligned}
$$

Proximal gradient descent as forward-backward splitting

minimize $\quad f(x) + g(x)$

f is L-smooth
g is nonsmooth but proxable

Therefore, ∇f is $(1/L)$-cocoercive and ∂g maximal monotone

Proximal gradient descent

$$\begin{aligned} x^{k+1} &= R_{t\partial g}(I - t\nabla f)(x^k) \\ &= \mathbf{prox}_{tg}(x^k - t\nabla f(x^k)) \end{aligned}$$

Remarks
- Converges for $t \in (0, 2/L)$
- If either f or g strongly convex **linear convergence**
- If $g = \mathcal{I}_C$, then it's projected gradient descent

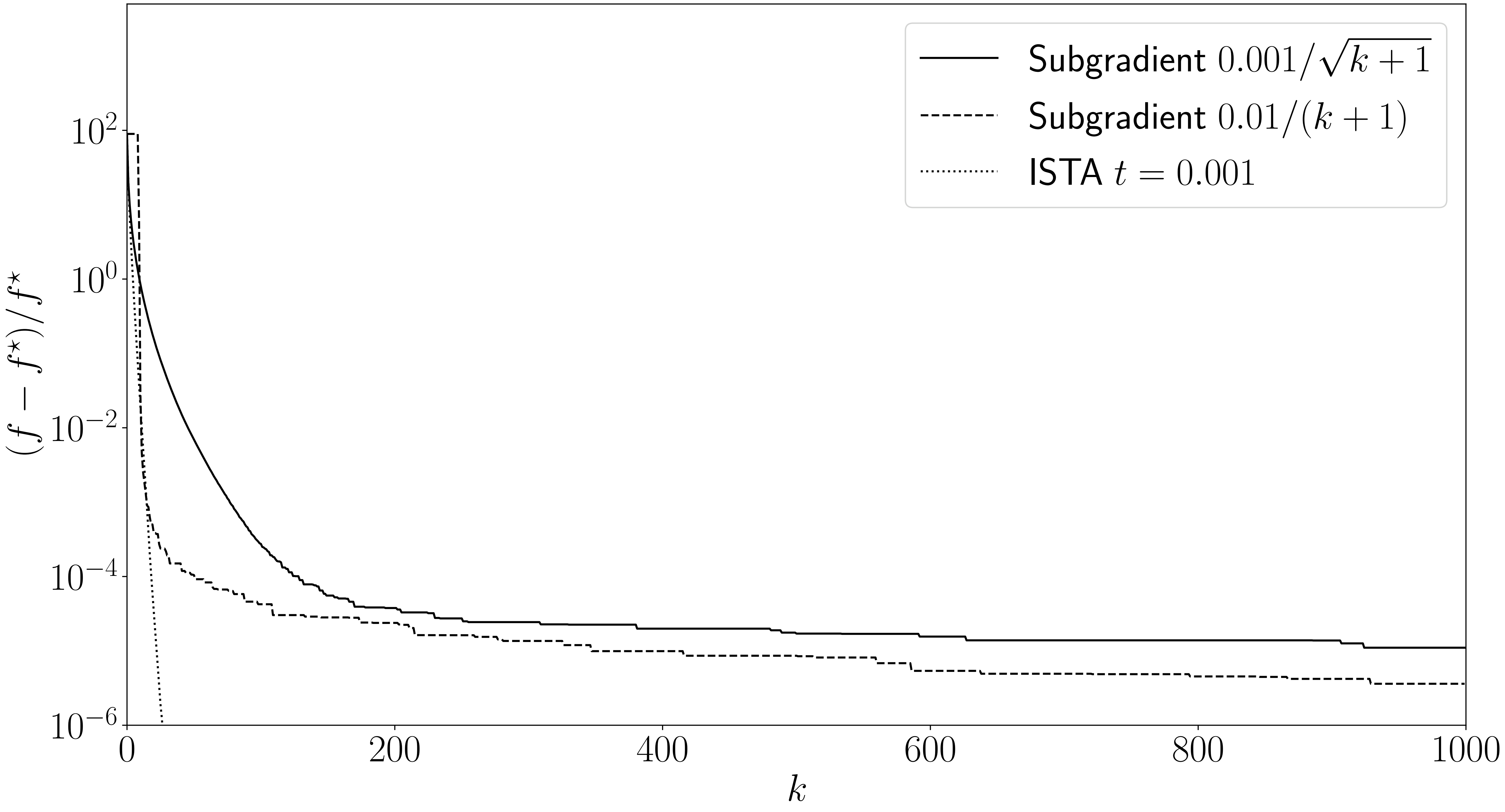
Example: Lasso with linear convergence

Iterative Soft Thresholding Algorithm (ISTA)

minimize $\underbrace{(1/2)\|Ax - b\|_2^2}_{f(x)} + \underbrace{\lambda\|x\|_1}_{g(x)}$

Proximal gradient descent

$$x^{k+1} = S_{\lambda t}\left(x^k - tA^T(Ax^k - b)\right)$$



Example

randomly generated $A \in \mathbf{R}^{500\times 300}$

$\Rightarrow \quad \nabla^2 f = A^T A \succ 0$

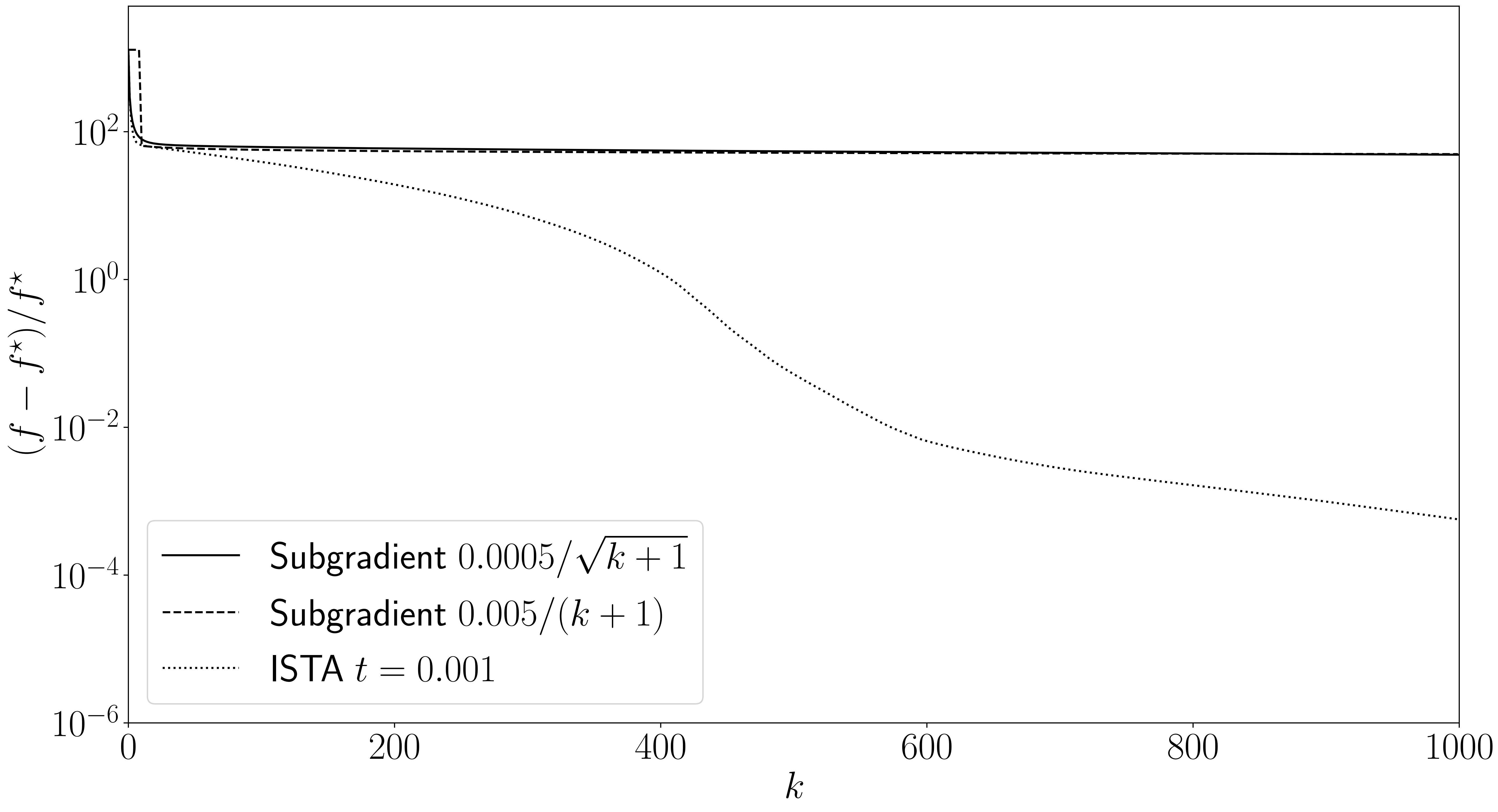
$\Rightarrow \quad f$ strongly convex

linear convergence

Example: Lasso without linear convergence

Iterative Soft Thresholding Algorithm (ISTA)

minimize $\underbrace{(1/2)\|Ax - b\|_2^2}_{f(x)} + \underbrace{\lambda\|x\|_1}_{g(x)}$



Proximal gradient descent

$$x^{k+1} = S_{\lambda t}\left(x^k - tA^T(Ax^k - b)\right)$$

Example

randomly generated $A \in \mathbf{R}^{300\times 500}$

$\Rightarrow \quad \nabla^2 f = A^T A \succeq 0$

$\Rightarrow \quad f$ not strongly convex

sublinear convergence

Douglas-Rachford splitting

Operator splitting

Main idea

We would like to solve

$$0 \in F(x), \qquad F \text{ maximal monotone}$$

Split the operator

$$F = A + B, \qquad A \text{ and } B \text{ are maximal monotone}$$

Solve by evaluating

$$R_A = (I + A)^{-1} \qquad\qquad C_A = 2R_A - I$$
or
$$R_B = (I + B)^{-1} \qquad\qquad C_B = 2R_B - I$$

Useful when R_A and R_B are cheaper than R_F

Splitting Cayley iterations

Key result

$$0 \in A(x) + B(x) \quad \Longleftrightarrow \quad C_A C_B(z) = z, \quad x = R_B(z)$$

Goal

Apply C_A and C_B sequentially instead of computing R_{A+B} directly

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z$$

$$x = R_B(z)$$

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z$$

$$x = R_B(z)$$

$$\longrightarrow$$

$$x = R_B(z)$$

$$\tilde{z} = 2x - z$$

$$\tilde{x} = R_A(\tilde{z})$$

$$z = 2\tilde{x} - \tilde{z}$$

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z$$

$$x = R_B(z)$$

$$\longrightarrow$$

$$x = R_B(z)$$

$$\tilde{z} = 2x - z$$

$$\tilde{x} = R_A(\tilde{z})$$

$$z = 2\tilde{x} - \tilde{z}$$

combine

$$\tilde{x} = x$$

last
equation

$$2x = z + \tilde{z}$$

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z$$

$$x = R_B(z)$$

$$\longrightarrow$$

$$\begin{aligned} x &= R_B(z) \\ \tilde{z} &= 2x - z \\ \tilde{x} &= R_A(\tilde{z}) \\ z &= 2\tilde{x} - \tilde{z} \end{aligned} \xrightarrow{\text{combine}} \tilde{x} = x$$

last equation

$$2x = z + \tilde{z}$$

Since $x = R_B(z)$, we have $z \in x + B(x)$

Since $\tilde{x} = R_A(\tilde{z})$, we have $\tilde{z} \in \tilde{x} + A(\tilde{x}) = x + A(x)$

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z \qquad \longrightarrow \qquad \begin{aligned} x &= R_B(z) \\ \tilde{z} &= 2x - z \\ \tilde{x} &= R_A(\tilde{z}) \\ z &= 2\tilde{x} - \tilde{z} \end{aligned} \xrightarrow{\text{combine}} \tilde{x} = x$$

$$x = R_B(z)$$

last equation

$$2x = z + \tilde{z}$$

Since $x = R_B(z)$, we have $z \in x + B(x)$

Since $\tilde{x} = R_A(\tilde{z})$, we have $\tilde{z} \in \tilde{x} + A(\tilde{x}) = x + A(x)$

By adding them, we obtain $z + \tilde{z} \in 2x + A(x) + B(x)$

Therefore, $0 \in A(x) + B(x)$ $\blacksquare$

Splitting Cayley iterations

Proof of key result

$$C_A C_B(z) = z$$

$$x = R_B(z)$$

$$x = R_B(z)$$

$$\tilde{z} = 2x - z$$

$$\tilde{x} = R_A(\tilde{z})$$

$$z = 2\tilde{x} - \tilde{z}$$

combine

$$\tilde{x} = x$$

last equation

$$2x = z + \tilde{z}$$

Since $x = R_B(z)$, we have $z \in x + B(x)$

Since $\tilde{x} = R_A(\tilde{z})$, we have $\tilde{z} \in \tilde{x} + A(\tilde{x}) = x + A(x)$

By adding them, we obtain $z + \tilde{z} \in 2x + A(x) + B(x)$

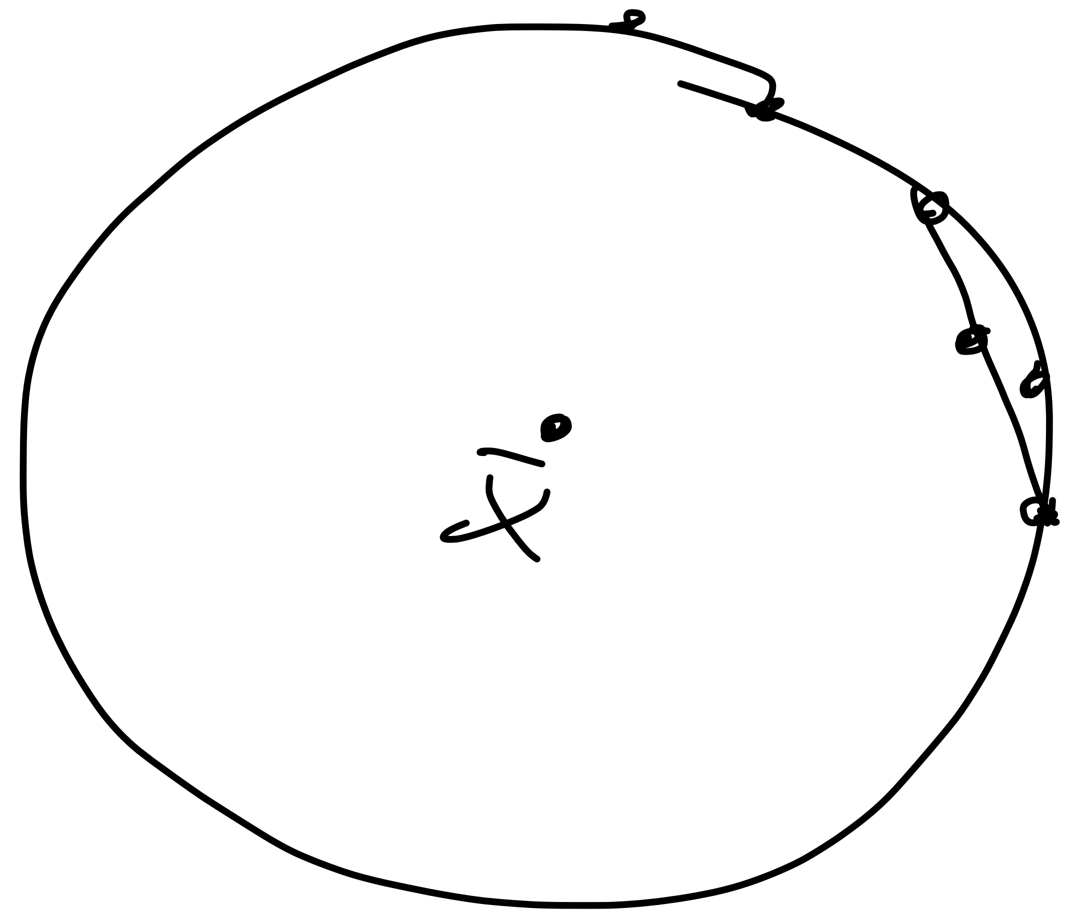
Therefore, $0 \in A(x) + B(x)$ ■

Note the arguments also holds the other way but we do not need it

Peaceman-Rachford and Douglas Rachford splitting

Peaceman-Rachford splitting

$$w^{k+1} = C_A C_B(w^k)$$



It does not converge in general (product of nonexpansive).
Need C_A or C_B to be a contraction

Peaceman-Rachford and Douglas Rachford splitting

Peaceman-Rachford splitting

$$w^{k+1} = C_A C_B(w^k)$$

It does not converge in general (product of nonexpansive).
Need C_A or C_B to be a contraction

Douglas-Rachford splitting (averaged iterations)

$$w^{k+1} = (1/2)(I + C_A C_B)(w^k)$$

- **Always converges** when $0 \in A(x) + B(x)$ has a solution
- If A or B strongly monotone and Lipschitz, then $C_A C_B$ is a contraction: **linear convergence**
- This method traces back to the 1950s

Douglas-Rachford splitting

$$w^{k+1} = (1/2)(I + C_A C_B)(w^k) \longrightarrow$$

Iterations

$$z^{k+1} = R_B(w^k)$$

$$\tilde{w}^{k+1} = 2z^{k+1} - w^k$$

$$x^{k+1} = R_A(\tilde{w}^{k+1})$$

$$w^{k+1} = w^k + x^{k+1} - z^{k+1}$$

Douglas-Rachford splitting

$$w^{k+1} = (1/2)(I + C_A C_B)(w^k) \longrightarrow$$

Iterations

$$\begin{aligned} z^{k+1} &= R_B(w^k) \\ \tilde{w}^{k+1} &= 2z^{k+1} - w^k \\ x^{k+1} &= R_A(\tilde{w}^{k+1}) \\ w^{k+1} &= w^k + x^{k+1} - z^{k+1} \end{aligned}$$

$$x^{k+1} = R_A(2z^{k+1} - w^k)$$

Last update (averaging) follows from:

$$\begin{aligned} w^{k+1} &= (1/2)w^k + (1/2)(2x^{k+1} - \tilde{w}^{k+1}) \\ &= (1/2)w^k + x^{k+1} - (1/2)(2z^{k+1} - w^k) \\ &= w^k + x^{k+1} - z^{k+1} \end{aligned}$$

Simplified iterations of Douglas-Rachford splitting

DR iterations

(simplify two inner steps)

$$z^{k+1} = R_B(w^k)$$

$$w^{k+1} = w^k + R_A(2z^{k+1} - w^k) - z^{k+1}$$

Simplified iterations of Douglas-Rachford splitting

DR iterations

(simplify two inner steps)

$$z^{k+1} = R_B(w^k)$$

$$w^{k+1} = w^k + R_A(2z^{k+1} - w^k) - z^{k+1}$$

1 Swap iterations and counter

$$w^{k+1} = w^k + R_A(2z^k - w^k) - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

Simplified iterations of Douglas-Rachford splitting

DR iterations

(simplify two inner steps)

$$z^{k+1} = R_B(w^k)$$

$$w^{k+1} = w^k + R_A(2z^{k+1} - w^k) - z^{k+1}$$

1 Swap iterations and counter

$$w^{k+1} = w^k + \underbrace{R_A(2z^k - w^k)} - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

2 Introduce x^{k+1}

$$\underline{x^{k+1}} = R_A(2z^k - w^k)$$

$$w^{k+1} = w^k + x^{k+1} - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

Simplified iterations of Douglas-Rachford splitting

DR iterations

(simplify two inner steps)

$$z^{k+1} = R_B(w^k)$$

$$w^{k+1} = w^k + R_A(2z^{k+1} - w^k) - z^{k+1}$$

1 Swap iterations and counter

$$w^{k+1} = w^k + R_A(2z^k - w^k) - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

2 Introduce x^{k+1}

$$x^{k+1} = R_A(2z^k - w^k)$$

$$w^{k+1} = w^k + x^{k+1} - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

3 Update w^{k+1} at the end

$$x^{k+1} = R_A(2z^k - w^k)$$

$$z^{k+1} = R_B(w^k + x^{k+1} - z^k)$$

$$w^{k+1} = w^k + x^{k+1} - z^k$$

Simplified iterations of Douglas-Rachford splitting

DR iterations

(simplify two inner steps)

$$z^{k+1} = R_B(w^k)$$

$$w^{k+1} = w^k + R_A(2z^{k+1} - w^k) - z^{k+1}$$

1 Swap iterations and counter

$$w^{k+1} = w^k + R_A(2z^k - w^k) - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

2 Introduce x^{k+1}

$$x^{k+1} = R_A(2z^k - w^k)$$

$$w^{k+1} = w^k + x^{k+1} - z^k$$

$$z^{k+1} = R_B(w^{k+1})$$

3 Update w^{k+1} at the end

$$x^{k+1} = R_A(2z^k - w^k)$$

$$z^{k+1} = R_B(w^k + x^{k+1} - z^k)$$

$$w^{k+1} = w^k + x^{k+1} - z^k$$

4 Define $u^k = w^k - z^k$

$$x^{k+1} = R_A(z^k - u^k)$$

$$z^{k+1} = R_B(x^{k+1} + u^k)$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Douglas-Rachford splitting

Simplified iterations

$$x^{k+1} = R_A(z^k - u^k)$$

$$z^{k+1} = R_B(x^{k+1} + u^k)$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Douglas-Rachford splitting

Simplified iterations

$$x^{k+1} = R_A(z^k - u^k)$$
$$z^{k+1} = R_B(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

$\longrightarrow$

Residual: $x^{k+1} - z^{k+1}$

running sum of residuals

u^k

Interpretation as integral control

Douglas-Rachford splitting

Simplified iterations

$$x^{k+1} = R_A(z^k - u^k)$$
$$z^{k+1} = R_B(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

$\longrightarrow$

Residual: $x^{k+1} - z^{k+1}$

running sum of residuals
u^k

Interpretation as integral control

Remarks

- *many* ways to rearrange the D-R algorithm
- Equivalent to many other algorithms (proximal point, Spingarn's partial inverses, Bregman iterative methods, etc.)
- Need very little to converge: A, B maximal monotone
- Splitting A and B, we can uncouple and evaluate R_A and R_B separately

Alternating Direction Method of Multipliers

Douglas-Rachford splitting in optimization

Problem

minimize $\quad f(x) + g(x)$

Optimality conditions

$$0 \in \partial f(x) + \partial g(x)$$

Douglas-Rachford splitting in optimization

Problem

minimize $\quad f(x) + g(x)$

Optimality conditions

$$0 \in \partial f(x) + \partial g(x)$$

Scaling by $\lambda > 0$

$\longrightarrow$

Problem

minimize $\quad \lambda f(x) + \lambda g(x)$

Optimality conditions

$$0 \in \lambda \partial f(x) + \lambda \partial g(x)$$

Douglas-Rachford splitting in optimization

Problem

minimize $\quad f(x) + g(x)$

Optimality conditions

$$0 \in \partial f(x) + \partial g(x)$$

Scaling by $\lambda > 0$

$\longrightarrow$

Problem

minimize $\quad \lambda f(x) + \lambda g(x)$

Optimality conditions

$$0 \in \underbrace{\lambda \partial f(x)}_{A(x)} + \underbrace{\lambda \partial g(x)}_{B(x)}$$

Douglas-Rachford splitting in optimization

Problem

minimize $\quad f(x) + g(x)$

Optimality conditions

$$0 \in \partial f(x) + \partial g(x)$$

Scaling by $\lambda > 0$

$\longrightarrow$

Problem

minimize $\quad \lambda f(x) + \lambda g(x)$

Optimality conditions

$$0 \in \underbrace{\lambda \partial f(x)}_{A(x)} + \underbrace{\lambda \partial g(x)}_{B(x)}$$

Douglas-Rachford splitting

$$x^{k+1} = R_{\lambda \partial f}(z^k - u^k)$$
$$z^{k+1} = R_{\lambda \partial g}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Proximal operators

$$x^{k+1} = \mathbf{prox}_{\lambda f}(z^k - u^k)$$
$$z^{k+1} = \mathbf{prox}_{\lambda g}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Alternating direction method of multipliers (ADMM)

minimize $\quad f(x) + g(x)$

Proximal iterations

$$x^{k+1} = \mathbf{prox}_{\lambda f}(z^k - u^k)$$

$$z^{k+1} = \mathbf{prox}_{\lambda g}(x^{k+1} + u^k)$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Alternating direction method of multipliers (ADMM)

minimize $\quad f(x) + g(x)$

Proximal iterations

$$x^{k+1} = \mathbf{prox}_{\lambda f}(z^k - u^k)$$
$$z^{k+1} = \mathbf{prox}_{\lambda g}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

$\longrightarrow$

ADMM iterations

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left(\lambda f(x) + (1/2)\|x - z^k + u^k\|^2\right)$$
$$z^{k+1} = \underset{z}{\operatorname{argmin}} \left(\lambda g(z) + (1/2)\|z - x^{k+1} - u^k\|^2\right)$$
$$u^{k+1} = u^k + z^{k+1} - x^{k+1}$$

Alternating direction method of multipliers (ADMM)

minimize $\quad f(x) + g(x) \longrightarrow$ $\min f(x) + g(z)$ s.t. $x = z$

Proximal iterations

$$x^{k+1} = \mathbf{prox}_{\lambda f}(z^k - u^k)$$
$$z^{k+1} = \mathbf{prox}_{\lambda g}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

$\longrightarrow$

ADMM iterations

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left(\lambda f(x) + (1/2)\|x - z^k + u^k\|^2\right)$$
$$z^{k+1} = \underset{z}{\operatorname{argmin}} \left(\lambda g(z) + (1/2)\|z - x^{k+1} - u^k\|^2\right)$$
$$u^{k+1} = u^k + z^{k+1} - x^{k+1}$$

Remarks

- It works for any $\lambda > 0$
- The choice of λ can greatly change performance
- It recently gained a **wide popularity** in various fields: Machine Learning, Imaging, Control, Finance

ADMM and the Augmented Lagrangian

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

(more generic form)
than $x=z$

Augmented Lagrangian

$$f(x) + g(z) + y^T(Ax + Bz - c) + \boxed{(t/2)\|Ax + Bz - c\|^2} =$$

$$= f(x) + g(z) + (t/2)\|Ax + Bz - c + u\|^2 - (t/2)\|u\|^2 = L_t(x, z, u)$$

ADMM and the Augmented Lagrangian

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

(more generic form)

Augmented Lagrangian

$$f(x) + g(z) + y^T(Ax + Bz - c) + (t/2)\|Ax + Bz - c\|^2 =$$

$$= f(x) + g(z) + (t/2)\|Ax + Bz - c + u\|^2 - (t/2)\|u\|^2 = L_t(x, z, u)$$

scaled dual variable

$u = y/t$

Note: $t = 1/\lambda$

ADMM and the Augmented Lagrangian

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

(more generic form)

$A = I$, $B = -I$, $c = 0$

Augmented Lagrangian

$$f(x) + g(z) + y^T(Ax + Bz - c) + (t/2)\|Ax + Bz - c\|^2 =$$

$$= f(x) + g(z) + (t/2)\|Ax + Bz - c + u\|^2 \cancel{- (t/2)\|u\|^2} = L_t(x, z, u)$$

scaled dual variable

$$u = y/t$$

Note: $t = 1/\lambda$

Rewritten ADMM iterations

$$x^{k+1} = \underset{x}{\operatorname{argmin}}\, L_t(x, z^k, u^k)$$

$$z^{k+1} = \underset{z}{\operatorname{argmin}}\, L_t(x^{k+1}, z, u^k)$$

$$u^{k+1} = u^k + Ax^{k+1} + Bz^{k+1} - c$$

Comparison with method of multipliers

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

Method of Multipliers

$$x^{k+1} \in \operatorname*{argmin}_x L_t(x, y^k)$$

$$u^{k+1} = u^k + Ax^{k+1} - b$$

Comparison with method of multipliers

minimize $\quad f(x)$

subject to $\quad Ax = b$

Method of Multipliers

$$x^{k+1} \in \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$

$$u^{k+1} = u^k + Ax^{k+1} - b$$

minimize $\quad f(x) + g(z)$

subject to $\quad Ax + Bz = c$

ADMM

$$x^{k+1} = \underset{x}{\operatorname{argmin}}\, L_t(x, z^k, u^k)$$

$$z^{k+1} = \underset{z}{\operatorname{argmin}}\, L_t(x^{k+1}, z, u^k)$$

$$u^{k+1} = u^k + Ax^{k+1} + Bz^{k+1} - c$$

$u^{k+1} = y^{k+1}/t$

Comparison with method of multipliers

minimize $\quad f(x)$
subject to $\quad Ax = b$

minimize $\quad f(x) + g(z)$
subject to $\quad Ax + Bz = c$

Method of Multipliers

$$x^{k+1} \in \underset{x}{\operatorname{argmin}}\, L_t(x, y^k)$$
$$u^{k+1} = u^k + Ax^{k+1} - b$$

ADMM

$$x^{k+1} = \underset{x}{\operatorname{argmin}}\, L_t(x, z^k, u^k)$$
$$z^{k+1} = \underset{z}{\operatorname{argmin}}\, L_t(x^{k+1}, z, u^k)$$
$$u^{k+1} = u^k + Ax^{k+1} + Bz^{k+1} - c$$

Remarks

- Same dual variable update u^{k+1}
- Augmented Lagrangian does not split f and g: argmin can be expensive
- ADMM splits f and g making steps **easier**
- We can derive ADMM by **splitting the dual subdifferential operator** [page 35, A Primer on Monotone Operator Methods]

Examples

Constrained optimization

minimize $\quad f(x)$

subject to $\quad x \in C$

$$\longrightarrow \quad g(x) = \mathcal{I}_C(x)$$

Constrained optimization

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in C \end{array} \longrightarrow \quad g(x) = \mathcal{I}_C(x)$$

ADMM iterates

$$\begin{aligned} x^{k+1} &= \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &= \mathbf{prox}_{\lambda g}(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1} \end{aligned} \longrightarrow \quad \begin{aligned} x^{k+1} &= \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &= \Pi_C(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

- Easy if $\mathbf{prox}_{\lambda f}$ and Π_C are easy
- Many ways to split (we can include some constraints also in f)

Linear/Quadratic Optimization

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & Ax = b \\
& x \geq 0
\end{array}
$$

$A \in \mathbf{R}^{m \times n}$

Linear/Quadratic Optimization

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

$$\longrightarrow$$

$$f(x) = (1/2)x^T P x + q^T x$$
$$\textbf{dom}\, f = \{x \mid Ax = b\}$$

$$g(z) = \mathcal{I}_{\mathbf{R}_+}(z)$$

$A \in \mathbf{R}^{m \times n}$

Linear/Quadratic Optimization

minimize	$(1/2)x^T P x + q^T x$
subject to	$Ax = b$
	$x \geq 0$

$A \in \mathbf{R}^{m \times n}$

$$f(x) = (1/2)x^T P x + q^T x$$

$$\mathbf{dom}\, f = \{x \mid Ax = b\}$$

$$g(z) = \mathcal{I}_{\mathbf{R}_+}(z)$$

ADMM iterations

$$x^{k+1} = \underset{\{x \mid Ax = b\}}{\operatorname{argmin}} \left(\lambda f(x) + (1/2)\|x - z^k + u^k\|^2\right)$$

$$z^{k+1} = (x^{k+1} + u^k)_+$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Linear/Quadratic Optimization

Rewriting prox

Equality constrained QP

$$
\begin{array}{rll}
x^{k+1} = & \text{argmin} & (\lambda/2)x^T P x + \lambda q^T x + (1/2)\|x - z^k + u^k\|^2 \\
& \text{subject to} & Ax = b
\end{array}
$$

Linear/Quadratic Optimization

Rewriting prox

Equality constrained QP

$$
\begin{array}{rll}
x^{k+1} = & \text{argmin} & (\lambda/2)x^T P x + \lambda q^T x + (1/2)\|x - z^k + u^k\|^2 \\
& \text{subject to} & Ax = b
\end{array}
$$

Optimality conditions

$$
\begin{bmatrix} \lambda P + I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu \end{bmatrix} = \begin{bmatrix} -\lambda q + z^k - u^k \\ b \end{bmatrix}
$$

STATIONARITY

PRIMAL FEAS

- Symmetric, possibly sparse, linear system $O((n+m)^3)$
- We can factor only once (it does not depend on the iterates)

Linear/Quadratic Optimization

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

Iterations

1. $x^{k+1} = \text{Solve} \begin{bmatrix} \lambda P + I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu \end{bmatrix} = \begin{bmatrix} -\lambda q + z^k - u^k \\ b \end{bmatrix}$
2. $z^{k+1} = (x^{k+1} + u^k)_+$
3. $u^{k+1} = u^k + x^{k+1} - z^{k+1}$

Linear/Quadratic Optimization

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

Iterations

1. $x^{k+1} = \text{Solve} \begin{bmatrix} \lambda P + I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu \end{bmatrix} = \begin{bmatrix} -\lambda q + z^k - u^k \\ b \end{bmatrix}$
2. $z^{k+1} = (x^{k+1} + u^k)_+$
3. $u^{k+1} = u^k + x^{k+1} - z^{k+1}$

Remarks

- Cheap iterations (after factorization) $O((n+m)^2)$
- Projection is just variables clipping
- Dual variables $y = \lambda u$
- More sophisticated version
  [OSQP: An Operator Splitting Solver for Quadratic Programs, Stellato, Banjac, Goulart, Bemporad, Boyd]

Find point at the intersection of two sets

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & x \in C \cap D \end{array} \longrightarrow \begin{aligned} x^{k+1} &= \Pi_C(z^k - u^k) \\ z^{k+1} &= \Pi_D(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

Find point at the intersection of two sets

find $\quad x$

subject to $\quad x \in C \cap D$

$\longrightarrow$

$$
\begin{aligned}
x^{k+1} &= \Pi_C(z^k - u^k) \\
z^{k+1} &= \Pi_D(x^{k+1} + u^k) \\
u^{k+1} &= u^k + x^{k+1} - z^{k+1}
\end{aligned}
$$

Remarks

- Much more robust convergence than simple alternating projections
- Useful when projections are cheap
- Similar to **Dykstra's alternating projections**
- It can be used to **solve optimization problems**
[Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding, O'Donoghue, Chu, Parikh, Boyd]

Matrix decomposition

Given $M \in \mathbf{R}^{m \times n}$, consider the **sparse + low rank** decomposition

$$\begin{array}{ll} \text{minimize} & \|L\|_* + \gamma \|S\|_1 \\ \text{subject to} & L + S = M \end{array}$$

- **Nuclear norm (low-rank)**: $\|L\|_* = \sum_{i=1}^{n} \sigma_i(L)$ (1-norm on singular values)
- **Elementwise 1-norm (sparse)**: $\|S\|_1 = \sum_{i,j} |S_{ij}|$

[Robust Principal Component Analysis?, Candes et al.]

Matrix decomposition

Given $M \in \mathbf{R}^{m \times n}$, consider the **sparse + low rank** decomposition

$$\begin{array}{ll} \text{minimize} & \|L\|_* + \gamma \|S\|_1 \\ \text{subject to} & L + S = M \end{array}$$

- **Nuclear norm (low-rank)**: $\|L\|_* = \sum_{i=1}^n \sigma_i(L)$ (1-norm on singular values)
- **Elementwise 1-norm (sparse)**: $\|S\|_1 = \sum_{i,j} |S_{ij}|$

ADMM Iterations

$$L^{k+1} = \mathbf{prox}_{\lambda\|\cdot\|_*}(M - S^{k-1} - W^k)$$
$$S^{k+1} = \mathbf{prox}_{\lambda\gamma\|\cdot\|_1}(M - L^{k+1} + W^k)$$
$$W^{k+1} = W^k + M - L^{k+1} - S^{k+1}$$

[Robust Principal Component Analysis?, Candes et al.]

Matrix decomposition

Explicit iterations

$$L^{k+1} = \mathbf{prox}_{\lambda\|\cdot\|_*}(M - S^{k-1} - W^k)$$
$$S^{k+1} = \mathbf{prox}_{\lambda\gamma\|\cdot\|_1}(M - L^{k+1} + W^k)$$
$$W^{k+1} = W^k + M - L^{k+1} - S^{k+1}$$

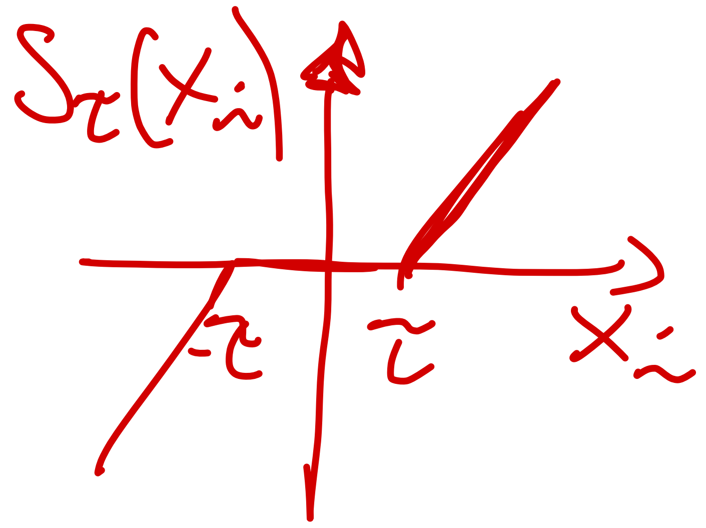
$\longrightarrow$

$$L^{k+1} = ST_\lambda(M - S^{k-1} - W^k)$$
$$S^{k+1} = S_{\lambda\gamma}(M - L^{k+1} + W^k)$$
$$W^{k+1} = W^k + M - L^{k+1} - S^{k+1}$$

Matrix decomposition

Explicit iterations

$$
\begin{aligned}
L^{k+1} &= \mathbf{prox}_{\lambda\|\cdot\|_*}(M - S^{k-1} - W^k) \\
S^{k+1} &= \mathbf{prox}_{\lambda\gamma\|\cdot\|_1}(M - L^{k+1} + W^k) \\
W^{k+1} &= W^k + M - L^{k+1} - S^{k+1}
\end{aligned}
\quad \longrightarrow \quad
\begin{aligned}
L^{k+1} &= ST_\lambda(M - S^{k-1} - W^k) \\
S^{k+1} &= S_{\lambda\gamma}(M - L^{k+1} + W^k) \\
W^{k+1} &= W^k + M - L^{k+1} - S^{k+1}
\end{aligned}
$$



Soft thresholding: $S_\tau(X_i) = (1 - \tau/|X_i|)_+ X_i$ (we saw it in lecture 16)

Singular value thresholding: $ST_\tau(X) = U\underbrace{(\Sigma - \tau I)_+}_{\begin{bmatrix} (\sigma_1-\tau)_+ & & \\ & (\sigma_2-\tau)_+ & \\ & & \ddots \end{bmatrix}} V^T$ where $X = U\Sigma V^T$

Note it involves an SVD!

Matrix decomposition surveillance example

Original M | Estimated Low-rank $\hat{L}$ | Estimated Sparse $\hat{S}$

[Robust Principal Component Analysis?, Candes et al.]

Distributed optimization

Consensus optimization

Goal solve

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} f_i(x)$$

Consensus optimization

Goal solve

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} f_i(x)$$

Rewrite as **consensus problem**

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & x \in C \end{array}$$

Consensus set

$$C = \{(x_1, \dots, x_N) \mid x_1 = x_2 = \dots = x_N\}$$

Consensus optimization

Goal solve

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} f_i(x)$$

Rewrite as **consensus problem**

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & x \in C \end{array}$$

Consensus set

$$C = \{(x_1, \dots, x_N) \mid x_1 = x_2 = \cdots = x_N\}$$

Constrained ADMM

$$x^{k+1} = \mathbf{prox}_{\lambda f}(z^k - u^k)$$

$$z^{k+1} = \Pi_C(x^{k+1} + u^k)$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Consensus optimization

Goal solve

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} f_i(x)$$

Rewrite as **consensus problem**

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & x \in C \end{array}$$

Consensus set

$$C = \{(x_1, \dots, x_N) \mid x_1 = x_2 = \cdots = x_N\}$$

Constrained ADMM

$$\begin{aligned} x^{k+1} &= \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &= \Pi_C(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

$\longrightarrow$

$$\begin{aligned} x_i^{k+1} &= \mathbf{prox}_{\lambda f_i}(z^k - u^k) & \textbf{separable} \\ z^{k+1} &= (1/N) \sum_{i=1}^{N} (x_i^{k+1} + u_i^k) & \textbf{averaging} \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z^{k+1} \end{aligned}$$

Distributed consensus optimization

$$x_i^{k+1} = \mathbf{prox}_{\lambda f_i}(z^k - u^k)$$

$$z^{k+1} = (1/N)\sum_{i=1}^{N}(x_i^{k+1} + u_i^k)$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1}$$

Distributed consensus optimization

$$x_i^{k+1} = \mathbf{prox}_{\lambda f_i}(z^k - u^k)$$

$$z^{k+1} = (1/N)\sum_{i=1}^{N}(x_i^{k+1} + u_i^k) \xrightarrow{\text{rewrite}} z^{k+1} = \bar{x}^{k+1} + \bar{u}^k$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1} \xrightarrow{\text{average}} \bar{u}^{k+1} = \bar{u}^k + \bar{x}^{k+1} - z^{k+1}$$

Distributed consensus optimization

$$x_i^{k+1} = \mathbf{prox}_{\lambda f_i}(z^k - u^k)$$

$$z^{k+1} = (1/N)\sum_{i=1}^{N}(x_i^{k+1} + u_i^k) \xrightarrow{\text{rewrite}} z^{k+1} = \bar{x}^{k+1} + \bar{u}^k$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1} \xrightarrow{\text{average}} \bar{u}^{k+1} = \bar{u}^k + \bar{x}^{k+1} - z^{k+1}$$

$\longrightarrow$ By combining,

$$\bar{u}^{k+1} = 0$$

$$\downarrow$$

$$z^{k+1} = \bar{x}^{k+1}$$

Distributed consensus optimization

$$x_i^{k+1} = \mathbf{prox}_{\lambda f_i}(z^k - u^k)$$

$$z^{k+1} = (1/N)\sum_{i=1}^{N}(x_i^{k+1} + u_i^k) \xrightarrow{\text{rewrite}} z^{k+1} = \bar{x}^{k+1} + \bar{u}^k$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1} \xrightarrow{\text{average}} \bar{u}^{k+1} = \bar{u}^k + \bar{x}^{k+1} - z^{k+1}$$

$\longrightarrow$ By combining,

$$\bar{u}^{k+1} = 0$$

$$\downarrow$$

$$z^{k+1} = \bar{x}^{k+1}$$

Simplified distributed iterations

$$x_i^{k+1} = \mathbf{prox}_{\lambda f_i}(\bar{x}^k - u^k)$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - \bar{x}^{k+1}$$

- Fully distributed $\mathbf{prox}$ between processors/cores/agents
- Gather x_i's to compute $\bar{x}$, which is then scattered

Global exchange problem

$$
\begin{array}{ll}
\text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\
\text{subject to} & \sum_{i=1}^{N} x_i = 0
\end{array}
\qquad x_i \in \mathbf{R}^n
$$

- $(x_i)_j$: quantity of commodity received (> 0) or contributed by (< 0) agent i
- f_i: utility function of each agent
- **equilibrium constraint** (market clearing) “supply” = “demand”

Global exchange problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} x_i = 0 \end{array} \qquad x_i \in \mathbf{R}^n$$

- $(x_i)_j$: quantity of commodity received (> 0) or contributed by (< 0) agent i
- f_i: utility function of each agent
- **equilibrium constraint** (market clearing) "supply" = "demand"

ADMM iterations

$$\begin{aligned} x_i^{k+1} &= \mathbf{prox}_{\lambda f_i}(x_i^k - \bar{x}^k - u^k) \\ u^{k+1} &= u^k + \bar{x}^{k+1} \end{aligned}$$

proximal exchange algrithm

Summary of ADMM

Convergence

- Slow to converge to high accuracy
- It often converges to modest accuracy in a few tens of iterations
- Step size λ (also called $1/\rho$) can greatly influence convergence
- If f or g is strongly convex, it converges linearly

Applications

Machine learning, control, finance, parallel computing, advertising, imaging, robotics, etc…

Surveys

- [Proximal Algorithms, Parikh and Boyd]
- [Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, Boyd, Parikh, Chu, Peleato, Eckstein]

Operator splitting algorithms

Today, we learned to:

- **Apply** the proximal point method to the “multiplier to residual” mapping obtaining the Method of Multipliers (Augmented Lagrangian)

- **Derive** proximal gradient from forward-backward splitting

- **Split** operators to obtain simpler averaged iterations with Douglas-Rachford splitting

- **Rewrite** Douglas-Rachford splitting for optimization problems obtaining the Alternating Directions Method of Multipliers

- **Apply** ADMM to various examples

- **Develop** distributed algorithms

Next lecture

- Acceleration schemes