ORF522 – Linear and Nonlinear Optimization

20. Alternating Direction Method of Multipliers

Bartolomeo Stellato — Fall 2022

Recap

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

Solve the dual with proximal point method

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

Method of multipliers (augmented Lagrangian method)

Primal

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

Dual

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

Iterates

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

$$\downarrow$$

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

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

Properties

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

  f^* and g are μ-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

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

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

Useful when R_A and R_B are cheaper than R_F

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

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

Today's lecture

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

Alternating Direction Method of Multipliers

- Alternating Direction Method of Multipliers as Douglas-Rachford splitting in Optimization
- Examples
- Distributed Optimization

Alternating Direction Method of Multipliers

Douglas-Rachford splitting in optimization

Problem

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

Optimality conditions

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

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)

$$\text{minimize} \quad f(x) + g(x)$$

Proximal iterations

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

ADMM iterations

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

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)

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: $t = 1/\lambda$

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

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 $\quad x \in C$

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

ADMM iterates

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

$$\longrightarrow$$

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

- 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

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

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

$\longrightarrow$

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

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

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

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

Optimality conditions

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

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

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

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

$\longrightarrow$

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

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)**: $\|S\|_1 = \sum_{i,j} |S_{ij}|$

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

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

$\longrightarrow$

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

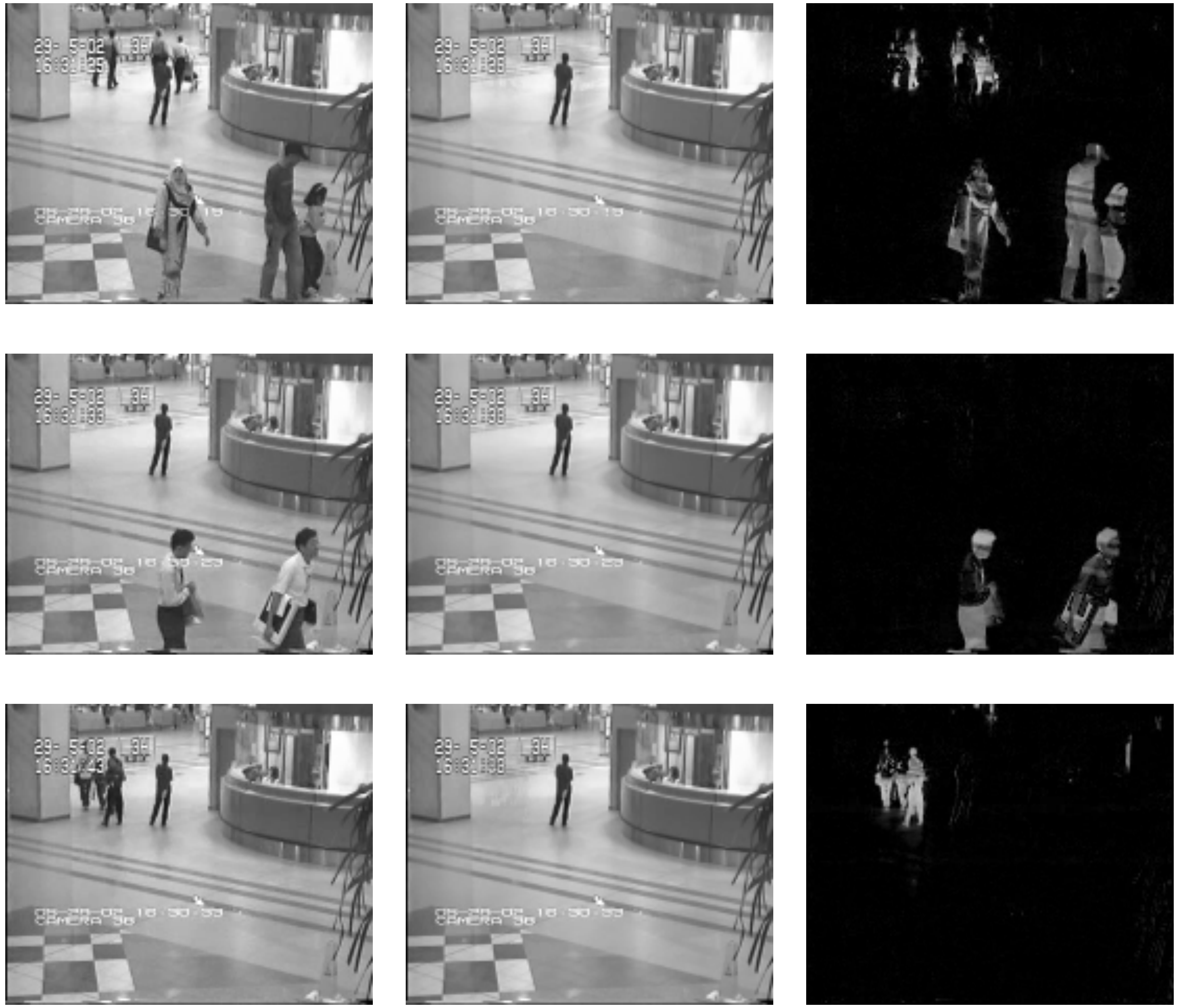
Soft thresholding: $S_\tau(X_i) = (1 - \tau/|X_i|)_+ X_i$ (we saw it in lecture 16)

Singular value thresholding: $ST_\tau(X) = U(\Sigma - \tau I)_+ V^T$ 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

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

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

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

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

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

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

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

By combining,

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

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

Simplified distributed iterations

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

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

- 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”

ADMM iterations

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

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]

Alternating Direction Method of Multipliers (ADMM)

Today, we learned to:

- **Rewrite** Douglas-Rachford splitting for optimization problems: Alternating Directions Method of Multipliers (ADMM)
- **Apply** ADMM to various examples
- **Derive** distributed versions of ADMM

Next lecture

- Acceleration schemes