a2dr: Anderson Accelerated Douglas-Rachford Splitting

Open-sourced Python Solver for Prox-Affine Distributed Convex Optimization

https://github.com/cvxgrp/a2dr

Junzi Zhang

Stanford ICME, *junziz@stanford.edu*

Joint work with Anqi Fu and Stephen P. Boyd

Special Session on Nonlinear Solvers and Acceleration Methods, I
University of Florida

November 2, 2019

Overview

1. Motivation and Problem Statement
2. Douglas-Rachford Splitting
3. Anderson Acceleration & A2DR
4. Numerical experiments
5. Conclusion

1. Motivation and Problem Statement
2. Douglas-Rachford Splitting
3. Anderson Acceleration & A2DR
4. Numerical experiments
5. Conclusion

Prox-affine form of generic convex optimization

We consider the following **prox-affine** representation/formulation of a **generic** convex optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & \sum_{i=1}^N A_i x_i = b. \end{array}$$

with variable $x = (x_1, \ldots, x_N) \in \mathbf{R}^{n_1+\cdots+n_N}$, $A_i \in \mathbf{R}^{m \times n_i}$, $b \in \mathbf{R}^m$.

Prox-affine form of generic convex optimization

We consider the following **prox-affine** representation/formulation of a **generic** convex optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & \sum_{i=1}^N A_i x_i = b. \end{array}$$

with variable $x = (x_1, \ldots, x_N) \in \mathbf{R}^{n_1+\cdots+n_N}$, $A_i \in \mathbf{R}^{m\times n_i}$, $b \in \mathbf{R}^m$.

- $f_i : \mathbf{R}^{n_i} \to \mathbf{R} \cup \{+\infty\}$ is closed, convex and proper (CCP).
- Each f_i can **only** be accessed through its proximal operator:

$$\mathbf{prox}_{tf_i}(v_i) = \mathrm{argmin}_{x_i} \ \left(f_i(x_i) + \frac{1}{2t}\|x_i - v_i\|_2^2 \right).$$

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b. \end{array}$$

- **Separability:** suitable for parallel and distributed implementation.

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b. \end{array}$$

- **Separability:** suitable for parallel and distributed implementation.
- **Black-box proximal:** suitable for peer-to-peer optimization with privacy requirements.

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b. \end{array}$$

- **Separability:** suitable for parallel and distributed implementation.
- **Black-box proximal:** suitable for peer-to-peer optimization with privacy requirements.
- **New interface:** good substitute for the **conic** standard form.

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b. \end{array}$$

- **Separability:** suitable for parallel and distributed implementation.
- **Black-box proximal:** suitable for peer-to-peer optimization with privacy requirements.
- **New interface:** good substitute for the **conic** standard form.
    - Cone programs can be represented in prox-affine form by consensus without complication (but NOT vice versa).

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll}\text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b.\end{array}$$

- **Separability:** suitable for parallel and distributed implementation.
- **Black-box proximal:** suitable for peer-to-peer optimization with privacy requirements.
- **New interface:** good substitute for the **conic** standard form.
    - Cone programs can be represented in prox-affine form by consensus without complication (but NOT vice versa).
    - With log, exp, det involved, prox-affine form is much more compact.

Prox-affine form of generic convex optimization

Why **prox-affine** form?

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & \sum_{i=1}^{N} A_i x_i = b. \end{array}$$

- **Separability:** suitable for parallel and distributed implementation.
- **Black-box proximal:** suitable for peer-to-peer optimization with privacy requirements.
- **New interface:** good substitute for the **conic** standard form.
  - Cone programs can be represented in prox-affine form by consensus without complication (but NOT vice versa).
  - With log, exp, det involved, prox-affine form is much more compact.

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (p_list), list of A_i's (A_list), and b (b), and you are done!

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (p_list), list of A_i's (A_list), and b (b), and you are done!

Why a2dr?

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (p_list), list of A_i's (A_list), and b (b), and you are done!

Why a2dr?

- Hundreds of papers on distributed/parallel optimization every year

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (p_list), list of A_i's (A_list), and b (b), and you are done!

Why a2dr?

- Hundreds of papers on distributed/parallel optimization every year
- Few solvers/softwares are written

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (`p_list`), list of A_i's (`A_list`), and b (`b`), and you are done!

Why a2dr?

- Hundreds of papers on distributed/parallel optimization every year
- Few solvers/softwares are written
- Existing good ones: CoCoA(+), TMAC, etc.
  - Efficient in communication cost
  - But hard to extend and use for general purposes.
  - Intended mostly for optimization experts.

a2dr: Solver interface

Interface of a2dr:

```
x_vals, primal, dual, num_iters, solve_time = a2dr(p_list, A_list, b)
```

Try it out! Simply provide a list of proximal functions $\mathbf{prox}_{tf_i}(v_i)$ (p_list), list of A_i's (A_list), and b (b), and you are done!

Why a2dr?

- Hundreds of papers on distributed/parallel optimization every year
- Few solvers/softwares are written
- Existing good ones: CoCoA(+), TMAC, etc.
  - Efficient in communication cost
  - But hard to extend and use for general purposes.
  - Intended mostly for optimization experts.

Finally: CVXPY + a2dr – Expression tree complier exists: Epsilon (Wytock et al., 2015).

Previous Work

Most common approaches for prox-affine formulation (sometimes goes by the name "distributed optimization"):

- Alternating direction method of multipliers (ADMM).
- Douglas-Rachford splitting (DRS).
- Augmented Lagrangian method (ALM).

Previous Work

Most common approaches for prox-affine formulation (sometimes goes by the name "distributed optimization"):

- Alternating direction method of multipliers (ADMM).
- Douglas-Rachford splitting (DRS).
- Augmented Lagrangian method (ALM).

These are typically slow to converge – acceleration techniques:

- Adaptive penalty parameters.
- Momentum methods.
- Quasi-Newton or Newton-type method with line search.

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
  - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
  - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free
  - *Flexibility:* Applicable to general non-expansive fixed-point (NEFP) iterations (Zhang et al., 2018):

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
  - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free
  - *Flexibility:* Applicable to general non-expansive fixed-point (NEFP) iterations (Zhang et al., 2018):
    - projected/proximal gradient descent, **DRS**, value iteration, etc.

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
    - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free
    - *Flexibility:* Applicable to general non-expansive fixed-point (NEFP) iterations (Zhang et al., 2018):
        - projected/proximal gradient descent, **DRS**, value iteration, etc.
        - globalized type-I AA proposed in (Zhang et al., 2018) used in SCS 2.x.

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
  - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free
  - *Flexibility:* Applicable to general non-expansive fixed-point (NEFP) iterations (Zhang et al., 2018):
    - projected/proximal gradient descent, **DRS**, value iteration, etc.
    - globalized type-I AA proposed in (Zhang et al., 2018) used in SCS 2.x.
- Why **DRS**?

Our Method

A2DR: Anderson acceleration (AA) applied to DRS

- Why **AA**?
  - *Fast and cheap:* As fast as (quasi-)Newton acceleration, but as memory efficient as adaptive penalty and momentum, and line-search free
  - *Flexibility:* Applicable to general non-expansive fixed-point (NEFP) iterations (Zhang et al., 2018):
    - projected/proximal gradient descent, **DRS**, value iteration, etc.
    - globalized type-I AA proposed in (Zhang et al., 2018) used in SCS 2.x.
- Why **DRS**?
  - Allows for a natural NEFP representation (ADMM not), and amenable to proximal evaluation (ALM not).

Challenges and contribution

Challenges:

Challenges and contribution

Challenges:

- **Instability:** AA is unstable without modifications (Scieur et al., 2016, Zhang et al., 2018).

Challenges and contribution

Challenges:

- **Instability:** AA is unstable without modifications (Scieur et al., 2016, Zhang et al., 2018).
- **Need for globalized type-II AA:** globalized type-I AA for SCS 2.x (Zhang et al., 2018) does not work that well with DRS + prox-affine.

Challenges and contribution

Challenges:

- **Instability:** AA is unstable without modifications (Scieur et al., 2016, Zhang et al., 2018).
- **Need for globalized type-II AA:** globalized type-I AA for SCS 2.x (Zhang et al., 2018) does not work that well with DRS + prox-affine.
- **Non-smoothness and pathology:** DRS is non-smooth, and does not always have a fixed-point solution (unlike SCS).

Challenges and contribution

Challenges:

- **Instability:** AA is unstable without modifications (Scieur et al., 2016, Zhang et al., 2018).
- **Need for globalized type-II AA:** globalized type-I AA for SCS 2.x (Zhang et al., 2018) does not work that well with DRS + prox-affine.
- **Non-smoothness and pathology:** DRS is non-smooth, and does not always have a fixed-point solution (unlike SCS).

Theory: First **globally** convergent type-II AA variant in **non-smooth**, potentially **pathological** settings.

Challenges and contribution

Challenges:

- **Instability:** AA is unstable without modifications (Scieur et al., 2016, Zhang et al., 2018).
- **Need for globalized type-II AA:** globalized type-I AA for SCS 2.x (Zhang et al., 2018) does not work that well with DRS + prox-affine.
- **Non-smoothness and pathology:** DRS is non-smooth, and does not always have a fixed-point solution (unlike SCS).

Theory: First **globally** convergent type-II AA variant in **non-smooth**, potentially **pathological** settings.
Practice: An open-source Python solver `a2dr` based on **A2DR**:

`https://github.com/cvxgrp/a2dr`.

1. Motivation and Problem Statement
2. Douglas-Rachford Splitting
3. Anderson Acceleration & A2DR
4. Numerical experiments
5. Conclusion

DRS Algorithm

- Rewrite problem as ($\mathcal{I}_S$ is the indicator of set S)

$$\text{minimize} \quad \overbrace{\sum_{i=1}^{N} f_i(x_i)}^{f(x)} + \overbrace{\mathcal{I}_{Ax=b}(x)}^{g(x)}.$$

DRS Algorithm

- Rewrite problem as ($\mathcal{I}_S$ is the indicator of set S)

$$\text{minimize} \quad \overbrace{\sum_{i=1}^{N} f_i(x_i)}^{f(x)} + \overbrace{\mathcal{I}_{Ax=b}(x)}^{g(x)}.$$

- DRS iterates for $k = 1, 2, \ldots,$

$$\begin{aligned} x_i^{k+1/2} &= \mathbf{prox}_{tf_i}(v^k), \quad i = 1, \ldots, N \\ v^{k+1/2} &= 2x^{k+1/2} - v^k \\ x^{k+1} &= \Pi_{Av=b}(v^{k+1/2}) \\ v^{k+1} &= v^k + x^{k+1} - x^{k+1/2} \end{aligned}$$

$\Pi_S(v)$ is Euclidean projection of v onto S.

Convergence of DRS

- DRS iterations can be conceived as a fixed point (FP) mapping

$$v^{k+1} = F(v^k)$$

- F is **firmly non-expansive**.
- v^k converges to a fixed point of F (if it exists).
- x^k and $x^{k+1/2}$ converge to a solution of our problem.

Convergence of DRS

- DRS iterations can be conceived as a fixed point (FP) mapping

$$v^{k+1} = F(v^k)$$

- F is **firmly non-expansive**.
- v^k converges to a fixed point of F (if it exists).
- x^k and $x^{k+1/2}$ converge to a solution of our problem.

In practice, this convergence is often rather slow.

1. Motivation and Problem Statement
2. Douglas-Rachford Splitting
3. Anderson Acceleration & A2DR
4. Numerical experiments
5. Conclusion

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
    - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
    - Type-I AA: approximate the Jacobian of the FP mapping

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).
- Why AA (but not other quasi-newton methods)?

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).
- Why AA (but not other quasi-newton methods)?
  - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).

- Why AA (but not other quasi-newton methods)?
  - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).
  - AA is more memory-efficient (AA with $M = 5 \sim 10$ beats LBFGS/restarted Broyden with $M = 200 \sim 500$).

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).

- Why AA (but not other quasi-newton methods)?
  - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).
  - AA is more memory-efficient (AA with $M = 5 \sim 10$ beats LBFGS/restarted Broyden with $M = 200 \sim 500$).
  - AA is line-search free: just accept or reject is the best practice.

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
    - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
    - Type-I AA: approximate the Jacobian of the FP mapping
    - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
        - Also has an intuitive **extrapolation** formulation (used later).
- Why AA (but not other quasi-newton methods)?
    - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).
    - AA is more memory-efficient (AA with $M = 5 \sim 10$ beats LBFGS/restarted Broyden with $M = 200 \sim 500$).
    - AA is line-search free: just accept or reject is the best practice.
- Why type-II AA?

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).

- Why AA (but not other quasi-newton methods)?
  - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).
  - AA is more memory-efficient (AA with $M = 5 \sim 10$ beats LBFGS/restarted Broyden with $M = 200 \sim 500$).
  - AA is line-search free: just accept or reject is the best practice.

- Why type-II AA?
  - Work better with DRS + prox-affine than type-I AA

Type-II AA

- Quasi-Newton type method for accelerating FP iterations.
  - **Multi-secant** quasi-Newton method (Fang & Saad, 2009).
  - Type-I AA: approximate the Jacobian of the FP mapping
  - **Type-II AA:** approximate the **inverse** Jacobian of the FP mapping
    - Also has an intuitive **extrapolation** formulation (used later).
- Why AA (but not other quasi-newton methods)?
  - Successful applications in SCS 2.x (type-I) and SuperSCS (type-II).
  - AA is more memory-efficient (AA with $M = 5 \sim 10$ beats LBFGS/restarted Broyden with $M = 200 \sim 500$).
  - AA is line-search free: just accept or reject is the best practice.
- Why type-II AA?
  - Work better with DRS + prox-affine than type-I AA
  - Better stability for **general purpose** solvers and distributed settings.
    - **prox** operators have much larger diversity than solvable cones in SCS.

Type-II AA

Extrapolation perspective of type-II AA:

- *Extrapolates* next iterate using $M+1$ most recent iterates

$$v^{k+1} = \sum_{j=0}^{M} \alpha_j^k F(v^{k-M+j})$$

Type-II AA

Extrapolation perspective of type-II AA:

- *Extrapolates* next iterate using $M+1$ most recent iterates

$$v^{k+1} = \sum_{j=0}^{M} \alpha_j^k F(v^{k-M+j})$$

- Let $G(v) = v - F(v)$ (**FP residual**), then $\alpha^k \in \mathbf{R}^{M+1}$ is solution to

$$\begin{array}{ll} \text{minimize} & \|\sum_{j=0}^{M} \alpha_j^k G(v^{k-M+j})\|_2^2 \\ \text{subject to} & \sum_{j=0}^{M} \alpha_j^k = 1 \end{array} \qquad \text{(constrained LS)}$$

Type-II AA

Extrapolation perspective of type-II AA:

- *Extrapolates* next iterate using $M+1$ most recent iterates

$$v^{k+1} = \sum_{j=0}^{M} \alpha_j^k F(v^{k-M+j})$$

- Let $G(v) = v - F(v)$ (**FP residual**), then $\alpha^k \in \mathbf{R}^{M+1}$ is solution to

$$\begin{array}{ll} \text{minimize} & \|\sum_{j=0}^{M} \alpha_j^k G(v^{k-M+j})\|_2^2 \\ \text{subject to} & \sum_{j=0}^{M} \alpha_j^k = 1 \end{array} \qquad \text{(constrained LS)}$$

- Minimizing the FP residual of extrapolated point $\sum_{j=0}^{M} \alpha_j^k v^{k-M+j}$ when F is affine.

Regularization

Type-II AA is **unstable** (Scieur et al., 2016), and can even provably diverge when applied to the gradient descent on a one-dimensional smooth unconstrained optimization problem (Mai & Johansson, 2019):

Regularization

Type-II AA is **unstable** (Scieur et al., 2016), and can even provably diverge when applied to the gradient descent on a one-dimensional smooth unconstrained optimization problem (Mai & Johansson, 2019):

- (Scieur et al., 2016) showed that adding **constant quadratic regularization** to the objective leads to local convergence improvement.

Regularization

Type-II AA is **unstable** (Scieur et al., 2016), and can even provably diverge when applied to the gradient descent on a one-dimensional smooth unconstrained optimization problem (Mai & Johansson, 2019):

- (Scieur et al., 2016) showed that adding **constant quadratic regularization** to the objective leads to local convergence improvement.
- **Insufficient** for global convergence both in theory and practice.

Adaptive Regularization

Our approach:

- Add *adaptive* regularization to the *unconstrained* formulation.

Adaptive Regularization

Our approach:

- Add *adaptive* regularization to the *unconstrained* formulation.
- Change variables to $\gamma^k \in \mathbf{R}^M$ (unconstrained LS):

$$\alpha_0^k = \gamma_0^k,\ \alpha_i^k = \gamma_i^k - \gamma_{i-1}^k,\ (i = 1, \ldots, M-1),\ \alpha_M^k = 1 - \gamma_{M-1}^k$$

Adaptive Regularization

Our approach:

- Add *adaptive* regularization to the *unconstrained* formulation.
- Change variables to $\gamma^k \in \mathbf{R}^M$ (unconstrained LS):

$$\alpha_0^k = \gamma_0^k,\ \alpha_i^k = \gamma_i^k - \gamma_{i-1}^k,\ (i = 1, \ldots, M-1),\ \alpha_M^k = 1 - \gamma_{M-1}^k$$

- Adaptive quadratic regularization: (adaptive LS)

$$\text{minimize} \quad \|g^k - Y_k\gamma^k\|_2^2 + \eta\left(\|S_k\|_F^2 + \|Y_k\|_F^2\right)\|\gamma^k\|_2^2,$$

where $\eta \geq 0$ is a regularization parameter and

$$g^k = G(v^k), \quad y^k = g^{k+1} - g^k, \quad Y_k = [y^{k-M} \ \ \ldots \ \ y^{k-1}]$$
$$s^k = v^{k+1} - v^k, \quad S_k = [s^{k-M} \ \ \ldots \ \ s^{k-1}]$$

A2DR

- A2DR iterates for $k = 1, 2, \ldots,$ ($\epsilon > 0$, M positive integer)

  1. Compute $v_{\text{DRS}}^{k+1} = F(v^k)$, $\quad g^k = v^k - v_{\text{DRS}}^{k+1}$.
  2. Update Y_k and S_k to include the new information

     & Compute α^k by solving the **adaptive LS** w.r.t. γ^k.
  3. Compute $v_{\text{AA}}^{k+1} = \sum_{j=0}^{M} \alpha_j^k v_{\text{DRS}}^{k-M+j+1}$.
  4. If the residual $\|G(v^k)\|_2 = O(1/n_{\text{AA}}^{1+\epsilon})$: (**safeguard**)

     Adopt $v^{k+i} = v_{\text{AA}}^{k+i}$ for $i = 1, \ldots, M$.

     (n_{AA}: # of adopted AA candidates)
  5. Otherwise, take $v^{k+1} = v_{\text{DRS}}^{k+1}$.

Stopping Criterion of A2DR

- Stop and output $x^{k+1/2}$ when $\|r^k\|_2 \leq \epsilon_{\mathsf{tol}} = \epsilon_{\mathrm{abs}} + \epsilon_{\mathrm{rel}}\|r^0\|_2$:

$$\begin{aligned} r^k_{\mathsf{prim}} &= Ax^{k+1/2} - b, \\ r^k_{\mathsf{dual}} &= \tfrac{1}{t}(v^k - x^{k+1/2}) + A^T\lambda^k, \\ r^k &= (r^k_{\mathrm{prim}}, r^k_{\mathrm{dual}}). \end{aligned}$$

Stopping Criterion of A2DR

- Stop and output $x^{k+1/2}$ when $\|r^k\|_2 \leq \epsilon_{\text{tol}} = \epsilon_{\mathrm{abs}} + \epsilon_{\mathrm{rel}}\|r^0\|_2$:

$$\begin{aligned} r^k_{\text{prim}} &= Ax^{k+1/2} - b, \\ r^k_{\text{dual}} &= \tfrac{1}{t}(v^k - x^{k+1/2}) + A^T\lambda^k, \\ r^k &= (r^k_{\mathrm{prim}}, r^k_{\mathrm{dual}}). \end{aligned}$$

- Remark:
  - Just KKT conditions. Notice that $(v^k - x^{k+1/2})/t \in \partial f(x^{k+1/2})$.
  - $\mathbf{prox}_f$ is enough, and no need for access to f or its sub-gradient.
- Dual variable is solution to least-squares problem

$$\lambda^k = \text{argmin}_\lambda \ \|r^k_{\text{dual}}\|_2$$

Key lemmas to the proof

Lemma (Bounded approximate inverse Jacobian)

We have $v_{AA}^{k+1} = v^k - H_k g^k$, *where* $g^k = G(v^k)$ *is the FP residual at* v^k, *and* $\|H_k\|_2 \leq 1 + 2/\eta$, *where* $\eta > 0$ *is the regularization parameter in the adaptive LS subproblem.*

Lemma (Connecting FP residuals with OPT residuals)

Suppose that $\liminf_{j\to\infty} \|G(v^j)\|_2 \leq \epsilon$ *for some* $\epsilon > 0$, *then*

$$\liminf_{j\to\infty} \|r_{\mathrm{prim}}^j\|_2 \leq \|A\|_2\epsilon, \quad \liminf_{j\to\infty} \|r_{\mathrm{dual}}^j\|_2 \leq \frac{1}{t}\epsilon.$$

Convergence of A2DR

Theorem (Solvable Case)

If the problem is solvable (e.g., feasible and bounded), then

$$\liminf_{k\to\infty} \|r^k\|_2 = 0$$

and the AA candidates are adopted infinitely often. Furthermore, if F has a fixed point, then

$$\lim_{k\to\infty} v^k = v^\star \textit{ and } \lim_{k\to\infty} x^{k+1/2} = x^\star,$$

where $v^\star$ is a fixed-point of F and $x^\star$ is a solution to our problem.

Remark. when the proximal operators and projections are evaluated with *errors* bounded by ϵ, then $\liminf_{k\to\infty} \|r^k\|_2 = O(\sqrt{\epsilon})$.

Convergence of A2DR

Theorem (Pathological Case)

If the problem is pathological (strongly primal infeasible or strongly dual infeasible), then

$$\lim_{k\to\infty} \left(v^k - v^{k+1}\right) = \delta v \neq 0.$$

Furthermore, if $\lim_{k\to\infty} Ax^{k+1/2} = b$*, then the problem is unbounded and* $\|\delta v\|_2 = t\,\mathbf{dist}(\mathbf{dom}\, f^*, \mathbf{range}(A^T))$*.*
Otherwise, it is infeasible and $\|\delta v\|_2 \geq \mathbf{dist}(\mathbf{dom}\, f, \{x : Ax = b\})$ *with equality when the dual problem is feasible.*

Implementation

Pre-conditioning (convergence greatly improved by rescaling problem):

- Replace original A, b, f_i with

$$\hat{A} = DAE, \quad \hat{b} = Db, \quad \hat{f}_i(\hat{x}_i) = f_i(e_i \hat{x}_i)$$

- D and E are diagonal positive, $e_i > 0$ corresponds to ith block diagonal entry of E, and chosen by equilibrating A
- Proximal operator of $\hat{f}_i$ can be evaluated using proximal operator of f_i

$$\mathbf{prox}_{t\hat{f}_i}(\hat{v}_i) = \tfrac{1}{e_i}\mathbf{prox}_{(e_i^2 t)f_i}(e_i \hat{v}_i)$$

Implementation

Pre-conditioning (convergence greatly improved by rescaling problem):

- Replace original A, b, f_i with

$$\hat{A} = DAE, \quad \hat{b} = Db, \quad \hat{f}_i(\hat{x}_i) = f_i(e_i \hat{x}_i)$$

- D and E are diagonal positive, $e_i > 0$ corresponds to ith block diagonal entry of E, and chosen by equilibrating A
- Proximal operator of $\hat{f}_i$ can be evaluated using proximal operator of f_i

$$\mathbf{prox}_{t\hat{f}_i}(\hat{v}_i) = \frac{1}{e_i}\mathbf{prox}_{(e_i^2 t)f_i}(e_i \hat{v}_i)$$

Choice of t (in DRS, $\mathbf{prox}_{tf_i}$): $t = \frac{1}{10}\left(\prod_{j=1}^N e_j\right)^{-2/N}$.

Implementation

Pre-conditioning (convergence greatly improved by rescaling problem):

- Replace original A, b, f_i with

$$\hat{A} = DAE, \quad \hat{b} = Db, \quad \hat{f}_i(\hat{x}_i) = f_i(e_i \hat{x}_i)$$

- D and E are diagonal positive, $e_i > 0$ corresponds to ith block diagonal entry of E, and chosen by equilibrating A
- Proximal operator of $\hat{f}_i$ can be evaluated using proximal operator of f_i

$$\mathbf{prox}_{t\hat{f}_i}(\hat{v}_i) = \frac{1}{e_i}\mathbf{prox}_{(e_i^2 t)f_i}(e_i \hat{v}_i)$$

Choice of t (in DRS, $\mathbf{prox}_{tf_i}$): $t = \frac{1}{10}\left(\prod_{j=1}^{N} e_j\right)^{-2/N}$.

Parallelization: `multiprocessing` package in Python.

1. Motivation and Problem Statement

2. Douglas-Rachford Splitting

3. Anderson Acceleration & A2DR

4. Numerical experiments

5. Conclusion

Nonnegative Least Squares (NNLS)

$$\begin{array}{ll} \text{minimize} & \|Fz - g\|_2^2 \\ \text{subject to} & z \geq 0 \end{array}$$

with respect to $z \in \mathbf{R}^q$

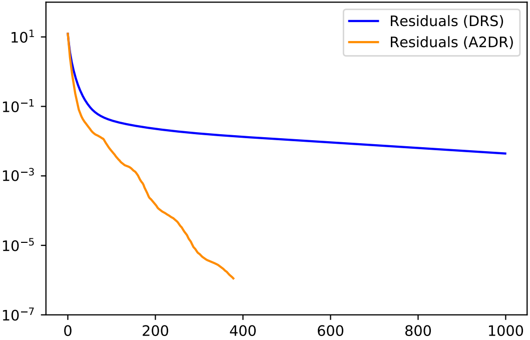
- Problem data: $F \in \mathbf{R}^{p \times q}$ and $g \in \mathbf{R}^p$
- Can be written in standard form with

$$f_1(x_1) = \|Fx_1 - g\|_2^2, \quad f_2(x_2) = \mathcal{I}_{\mathbf{R}_+^n}(x_2)$$
$$A_1 = I, \quad A_2 = -I, \quad b = 0$$

- We evaluate proximal operator of f_1 using LSQR

NNLS: Convergence of $\|r^k\|_2$

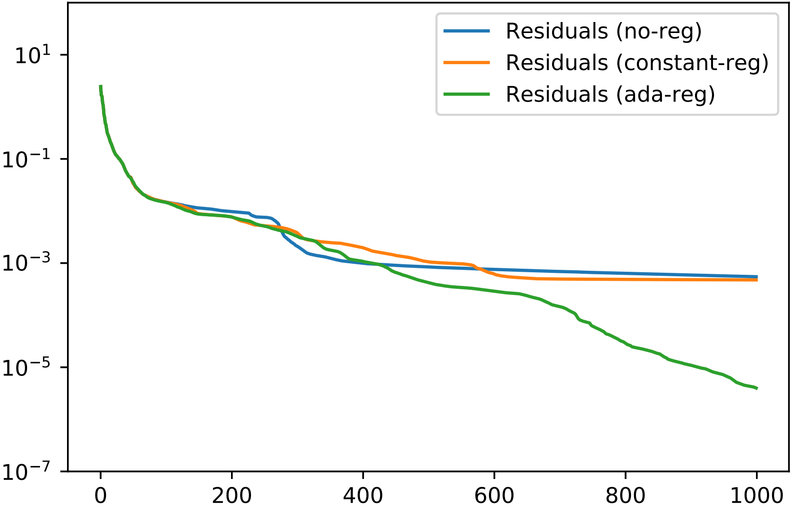
$p = 10^4$, $q = 8000$, F has 0.1% nonzeros



OSQP and SCS took respectively 349 and 327 seconds, while A2DR only took 55 seconds.

NNLS: Effect of regularization

$p = 300,\ q = 500,\ F$ has 0.1% nonzeros



Sparse Inverse Covariance Estimation

- Samples $z_1, \dots, z_p$ IID from $\mathcal{N}(0, \Sigma)$
- Know covariance $\Sigma \in \mathbf{S}_+^q$ has **sparse** inverse $S = \Sigma^{-1}$
- One way to estimate S is by solving the penalized log-likelihood problem

$$\text{minimize} \quad -\log\det(S) + \mathrm{tr}(SQ) + \alpha\|S\|_1,$$

where Q is the sample covariance, $\alpha \geq 0$ is a parameter

- Note $\log\det(S) = -\infty$ when $S \nsucc 0$

Sparse Inverse Covariance Estimation

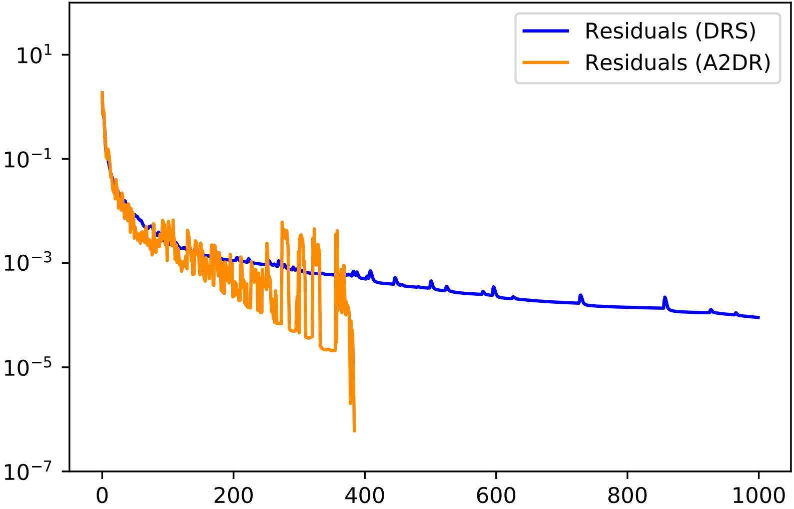
- Problem can be written in standard form with

$$f_1(S_1) = -\log\det(S_1) + \mathrm{tr}(S_1 Q), \quad f_2(S_2) = \alpha \|S_2\|_1,$$
$$A_1 = I, \quad A_2 = -I, \quad b = 0.$$

- Both proximal operators have closed-form solutions.

Covariance Estimation: Convergence of $\|r^k\|_2$

$p = 1000,\ q = 100,\ S$ has 10% nonzeros



Covariance Estimation: larger examples

Ran A2DR on instances with $q = 1200$ and $q = 2000$ (vectorizations on the order of 10^6) and compared its performance to SCS:

Covariance Estimation: larger examples

Ran A2DR on instances with $q = 1200$ and $q = 2000$ (vectorizations on the order of 10^6) and compared its performance to SCS:

- In the former case, A2DR took 1 hour to converge to a tolerance of 10^{-3}, while SCS took 11 hours to achieve a tolerance of 10^{-1} and yielded a much worse objective value.

Covariance Estimation: larger examples

Ran A2DR on instances with $q = 1200$ and $q = 2000$ (vectorizations on the order of 10^6) and compared its performance to SCS:

- In the former case, A2DR took 1 hour to converge to a tolerance of 10^{-3}, while SCS took 11 hours to achieve a tolerance of 10^{-1} and yielded a much worse objective value.
- In the latter case, A2DR converged in 2.6 hours to a tolerance of 10^{-3}, while SCS failed immediately with an out-of-memory error.

Multi-Task Logistic Regression

$$\text{minimize} \quad \phi(W\theta, Y) + \alpha \textstyle\sum_{l=1}^{L} \|\theta_l\|_2 + \beta\|\theta\|_*$$

with respect to $\theta = [\theta_1 \cdots \theta_L] \in \mathbf{R}^{s \times L}$

- Problem data: $W \in \mathbf{R}^{p \times s}$ and $Y = [y_1 \cdots y_L] \in \mathbf{R}^{p \times L}$
- Regularization parameters: $\alpha \geq 0, \beta \geq 0$
- Logistic loss function

$$\phi(Z, Y) = \sum_{l=1}^{L} \sum_{i=1}^{p} \log\left(1 + \exp(-Y_{il} Z_{il})\right)$$

Multi-Task Logistic Regression

- Rewrite problem in standard form with:

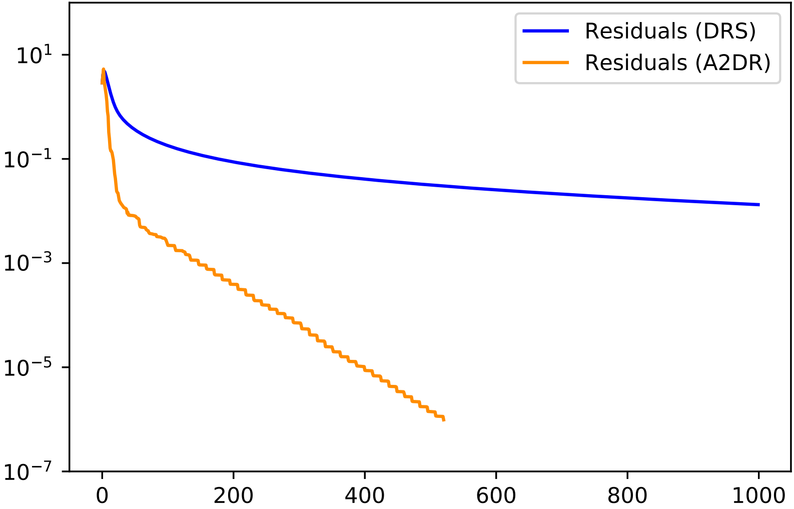
$$f_1(Z) = \phi(Z, Y), \quad f_2(\theta) = \alpha \sum_{l=1}^{L} \|\theta_l\|_2, \quad f_3(\tilde{\theta}) = \beta \|\tilde{\theta}\|_*,$$

$$A = \begin{bmatrix} I & -W & 0 \\ 0 & I & -I \end{bmatrix}, \quad x = \begin{bmatrix} Z \\ \theta \\ \tilde{\theta} \end{bmatrix}, \quad b = 0$$

- We evaluate proximal operator of f_1 using Newton-CG method, and the rest with closed-form formulae.

Multi-Task Logistic: Convergence of $\|r^k\|_2$

$p = 300,\ s = 500,\ L = 10,\ \alpha = \beta = 0.1$



Other examples

A (very) brief summary of other examples (see the paper for more details):

- l_1 trend filtering.
- Stratified models.
- Single commodity flow optimization (match the performance of OSQP, and largely outperform SCS).
- Optimal control (largely outperform both SCS and OSQP).
- Coupled quadratic program (match the performance of OSQP and SCS).

Other examples

A (very) brief summary of other examples (see the paper for more details):

- l_1 trend filtering.
- Stratified models.
- Single commodity flow optimization (match the performance of OSQP, and largely outperform SCS).
- Optimal control (largely outperform both SCS and OSQP).
- Coupled quadratic program (match the performance of OSQP and SCS).

Remark. The advantage compared to OSQP probably comes from the inclusion of AA, while the advantage compared to SCS (which includes type-I AA) is probably due to the more compact standard form representation.

Conclusion

- A2DR is a fast, robust algorithm for solving generic (non-smooth) convex optimization problems in the prox-affine form.

Conclusion

- A2DR is a fast, robust algorithm for solving generic (non-smooth) convex optimization problems in the prox-affine form.
- Parallelized, scalable and memory-efficient.

Conclusion

- A2DR is a fast, robust algorithm for solving generic (non-smooth) convex optimization problems in the prox-affine form.
- Parallelized, scalable and memory-efficient.
- Consistent and fast convergence with no parameter tuning, and beat SOTA open source solvers like SCS (2.x) and OSQP in many cases.

Conclusion

- A2DR is a fast, robust algorithm for solving generic (non-smooth) convex optimization problems in the prox-affine form.
- Parallelized, scalable and memory-efficient.
- Consistent and fast convergence with no parameter tuning, and beat SOTA open source solvers like SCS (2.x) and OSQP in many cases.
- Produces primal and dual solutions, or a certificate of infeasibility/unboundedness.

Conclusion

- A2DR is a fast, robust algorithm for solving generic (non-smooth) convex optimization problems in the prox-affine form.
- Parallelized, scalable and memory-efficient.
- Consistent and fast convergence with no parameter tuning, and beat SOTA open source solvers like SCS (2.x) and OSQP in many cases.
- Produces primal and dual solutions, or a certificate of infeasibility/unboundedness.
- Python library:

  `https://github.com/cvxgrp/a2dr`

Future Work

- More work on feasibility detection.
- Expand library of proximal operators (non-convex proximal).
- User-friendly interface with CVXPY (with the help of `Epsilon`).
- GPU parallelization and cloud computing,

References

Fu, A.*, Zhang, J.* and Boyd, S. P. (2019). (*equal contribution) Anderson Accelerated Douglas-Rachford Splitting. *arXiv preprint arXiv:1908.11482.*

Acknowledgment

- Thanks to Brendan ODonoghue for his advice on pre-conditioning and his inspirational ideas of developing solvers with Anderson acceleration, pioneered by SCS 2.x:
    - Zhang, J., O'Donoghue, B. and Boyd, S. P. (2018).
- Thanks to Anqi Fu for the input to the slides.

Thanks for listening!

Any questions?