Anderson acceleration for first-order methods

Michael Garstka[1] · Mark Cannon[1] · Paul Goulart[1]

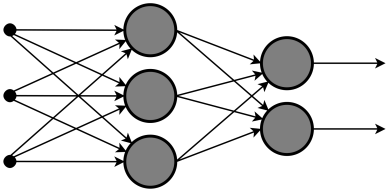
[1]University of Oxford, UK

(virtual) Journal Club, Control Group
26th February 2021

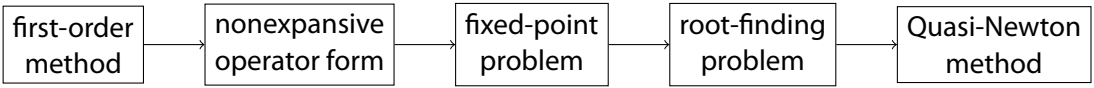
Motivation

When are we using first-order methods?

- non-smooth, large-scale, constrained, (possibly distributed) optimisation problems
- Gradient method, Proximal gradient, Douglas-Rachford / ADMM, ISTA, Consensus, etc.



Motivation / Overview

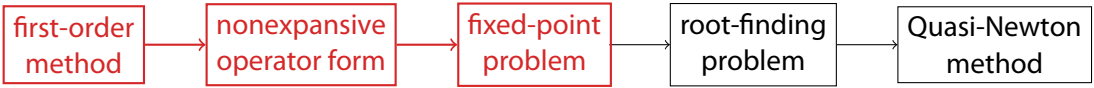


- Gradient descent
- Proximal gradient method
- Douglas-Rachford/ADMM
- Alternating projection method

[. . .]

- Broyden methods
- Anderson acceleration
- Line search

Motivation / Overview



- Gradient descent
- Proximal gradient method
- Douglas-Rachford/ADMM
- Alternating projection method

[. . .]

- Broyden methods
- Anderson acceleration
- Line search

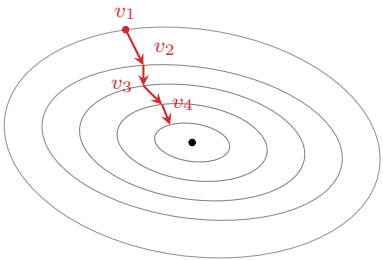
First-order methods

Gradient method:

$$\text{minimize } f(v)$$

with $v \in \mathbb{R}^n$, f strongly convex and strongly smooth with Lipschitz constant L

$$v_{k+1} = v_k - \alpha \nabla f(v)$$



First-order methods

Proximal gradient method:

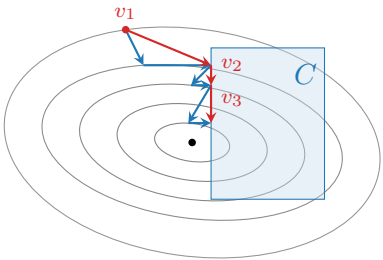
$$\text{minimize } f(v) + g(v)$$

with $v \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to R$ (CCP) and differentiable and $g\colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$v_{k+1} = \mathbf{prox}_g(v_k - \alpha \nabla f(v_k))$$

using the proximity operator:

$$\mathbf{prox}_g(y) := \operatorname*{argmin}_v \left(g(v) + \frac{1}{2}\|v - y\|^2 \right).$$



* CCP: convex, closed, and proper

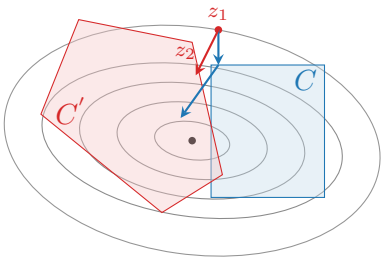
First-order methods

Douglas-Rachford splitting:

$$\text{minimize } f(v) + g(v)$$

with $v \in \mathbb{R}^n$, $f, g \colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\begin{aligned} v^k &= \mathbf{prox}_g(z^k) \\ y^k &= \mathbf{prox}_f(2v^k - z^k) \\ z^{k+1} &= z^k + 2\alpha(y^k - v^k) \end{aligned}$$



* CCP: convex, closed, and proper

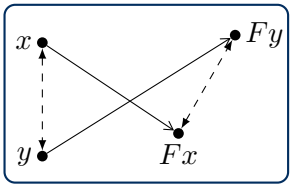
Nonexpansive operators

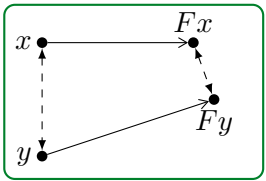
Definition (Nonexpansive operator)

An operator (function) $F\colon \mathcal{D} \to \mathbb{R}^n$ has a Lipschitz constant L if it holds:

$$\|Fx - Fy\| \leq L\|x - y\|, \quad \forall x, y \in \mathcal{D}.$$

When $L = 1$, F is called *nonexpansive* and when $L < 1$, F is called *contractive*.





Finding fixed-points

Fixed-points

We define the set of fixed points of a mapping $F \colon \mathcal{D} \to \mathbb{R}^n$:

$$\textbf{Fix}\, F := \{v \in \textbf{dom}\, F \mid v = Fv\}\,.$$

Theorem (Banach-Picard fixed point theorem*)

Let $(\mathcal{X}, d)$ be a complete metric space and let $F : \mathcal{X} \to \mathcal{X}$ be Lipschitz continuous with constant $L \in [0, 1[$. Given $v_0 \in \mathcal{X}$, set

$$v_{k+1} = Fv_k.$$

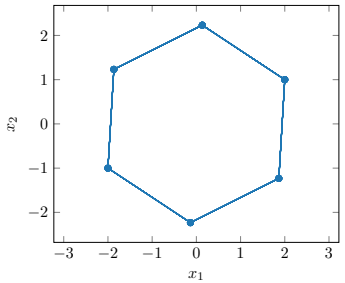
Then there exists $v \in \mathcal{X}$ such that the following hold:

- *v is the unique fixed point of F.*
- *the mapping is contractive.*
- *the sequence $(v_k)_{k\in\mathbb{N}}$ converges linearly to v^*.*

* [Bauschke and Combettes (2011)]

Finding fixed-points

- What about nonexpansive operators, e.g.
  $v = Fv,\ F = \begin{bmatrix} \cos(\frac{\pi}{3}) & -\sin(\frac{\pi}{3}) \\ \sin(\frac{\pi}{3}) & \cos(\frac{\pi}{3}) \end{bmatrix}$?

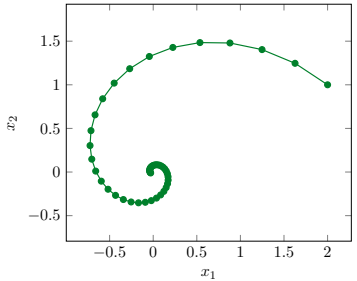


Krasnoselskii-Mann iteration

The Krasnoselskii-Mann iteration

$$v_{k+1} = (1 - \alpha_k)v_k + \alpha_k F v_k,$$

with $\alpha_n \in (0, 1)$ converges to a unique fixed point for nonexpansive operators.



Operator building blocks

- Identity operator: Id $= \{(x, x) \mid x \in \mathbb{R}^n\}$
- Inverse operator: $F^{-1} = \{(x, y) \mid (y, x) \in F\}$
- Subdifferential operator: $\partial f = \{(x, g) \mid x \in \mathbb{R}^n, \forall z \in \mathbb{R}^n,\ f(z) \geq f(x) + g^\top (z - x)\}$
- Zero: $0 \in F(x) \rightarrow$ find $x \in \mathbb{R}^n$ s.t. $(x, 0) \in F$

Resolvent and Cayley operator

- Resolvent operator of F: $R_F = (\mathsf{Id} + \alpha F)^{-1}$
- Cayley operator of F: $C_F = 2R_F - \mathsf{Id}$

- Nonexpansive for monotone F
- Resolvent of ∂f: $R_{\partial f}(x) = (\mathsf{Id} + \alpha \partial f)^{-1}(x) = \mathbf{prox}_f(x)$

- Solutions of $0 \in F(x) \Leftrightarrow$ Fixed points of R_F and C_F:

$$0 \in F(x) \Leftrightarrow x \in (\mathsf{Id} + \alpha F)(x) \Leftrightarrow x = (\mathsf{Id} + \alpha F)^{-1}(x)$$
$$x = R_F(x) \Leftrightarrow x = 2R_F(x) - x = C_F(x)$$

*from survey paper: Ryu and Boyd (2016)

First-order methods in operator form

Gradient method:

$$\text{minimize } f(v)$$

with $v \in \mathbb{R}^n$, f strongly convex and strongly smooth with Lipschitz constant L

$$\Leftrightarrow 0 = \nabla f(v) \Leftrightarrow v = (\mathsf{Id} - \alpha \nabla f)(v)$$

First-order methods in operator form

Gradient method:

$$\text{minimize } f(v)$$

with $v \in \mathbb{R}^n$, f strongly convex and strongly smooth with Lipschitz constant L

$$\Leftrightarrow 0 = \nabla f(v) \Leftrightarrow v = (\ \mathrm{Id} - \alpha \nabla f\ )(v)$$

- $(\mathrm{Id} - \alpha \nabla f)$ has Lipschitz constant $\hat{L} = |1 - \alpha L|$
- contractive for $\alpha \in (0, 2/L)$

First-order methods in operator form

Proximal gradient method:

$$\text{minimize } f(v) + g(v)$$

with $v \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to R$ (CCP) and differentiable and $g\colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\begin{aligned} 0 \in (\nabla f + \partial g)(v) &\Leftrightarrow 0 \in (\mathsf{Id} + \alpha\partial g)(v) - (\mathsf{Id} - \alpha\nabla f)(v) \\ &\Leftrightarrow (\mathsf{Id} + \alpha\partial g)(v) \ni (\mathsf{Id} - \alpha\nabla f)(v) \\ v &= (\mathsf{Id} + \alpha\partial g)^{-1}(\mathsf{Id} - \alpha\nabla f)(v) \\ v &= R_{\partial g}(\mathsf{Id} - \alpha\nabla f)(v) = \mathbf{prox}_g(v - \alpha\nabla f(v)) \end{aligned}$$

First-order methods in operator form

Proximal gradient method:

$$\text{minimize } f(v) + g(v)$$

with $v \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to R$ (CCP) and differentiable and $g\colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\begin{aligned} 0 \in (\nabla f + \partial g)(v) &\Leftrightarrow 0 \in (\mathsf{Id} + \alpha\partial g)(v) - (\mathsf{Id} - \alpha\nabla f)(v) \\ &\Leftrightarrow (\mathsf{Id} + \alpha\partial g)(v) \ni (\mathsf{Id} - \alpha\nabla f)(v) \\ v &= (\mathsf{Id} + \alpha\partial g)^{-1}(\mathsf{Id} - \alpha\nabla f)(v) \\ v &= R_{\partial g}\left(\mathsf{Id} - \alpha\nabla f\right)(v) = \mathbf{prox}_g(v - \alpha\nabla f(v)) \end{aligned}$$

contractive for $\alpha \in (0, 2/L)$

First-order methods in operator form

Proximal gradient method:

$$\text{minimize } f(v) + g(v)$$

with $v \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to R$ (CCP) and differentiable and $g\colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\begin{aligned} 0 \in (\nabla f + \partial g)(v) &\Leftrightarrow 0 \in (\mathsf{Id} + \alpha\partial g)(v) - (\mathsf{Id} - \alpha\nabla f)(v) \\ &\Leftrightarrow (\mathsf{Id} + \alpha\partial g)(v) \ni (\mathsf{Id} - \alpha\nabla f)(v) \\ v &= (\mathsf{Id} + \alpha\partial g)^{-1}(\mathsf{Id} - \alpha\nabla f)(v) \\ v &= R_{\partial g}\left(\mathsf{Id} - \alpha\nabla f\right)(v) = \mathbf{prox}_g(v - \alpha\nabla f(v)) \end{aligned}$$

nonexpansive for CCP g

First-order methods in operator form

Douglas-Rachford splitting:

$$\text{minimize } f(v) + g(v)$$

with $x \in \mathbb{R}^n$, $f, g \colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\Leftrightarrow 0 \in (\partial f + \partial g)(v) \Leftrightarrow z = ((1-\alpha)\mathsf{Id} + \alpha C_{\partial f} C_{\partial g})(z),\ v = R_{\partial g}(z)$$

Fixed-point iteration:

$$\begin{aligned} v^k &= \mathbf{prox}_g(z^k) \\ y^k &= \mathbf{prox}_f(2v^k - z^k) \\ z^{k+1} &= (1-\alpha)z^k + \alpha(2(y^k - v^k) - z^k) \end{aligned}$$

First-order methods in operator form

Douglas-Rachford splitting:

$$\text{minimize } f(v) + g(v)$$

with $x \in \mathbb{R}^n$, $f, g \colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\Leftrightarrow 0 \in (\partial f + \partial g)(v) \Leftrightarrow z = ((1-\alpha)\mathsf{Id} + \alpha C_{\partial f}\, C_{\partial g})(z),\ v = R_{\partial g}(z)$$

Fixed-point iteration:

$$\begin{aligned} v^k &= \mathbf{prox}_g(z^k) \\ y^k &= \mathbf{prox}_f(2v^k - z^k) \\ z^{k+1} &= (1-\alpha)z^k + \alpha(2(y^k - v^k) - z^k) \end{aligned}$$

First-order methods in operator form

Douglas-Rachford splitting:

$$\text{minimize } f(v) + g(v)$$

with $x \in \mathbb{R}^n$, $f, g \colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

$$\Leftrightarrow 0 \in (\partial f + \partial g)(v) \Leftrightarrow z = ((1-\alpha)\mathsf{Id} + \alpha\, C_{\partial f}\, C_{\partial g})(z),\ v = R_{\partial g}(z)$$

Fixed-point iteration:

$$\begin{aligned} v^k &= \mathbf{prox}_g(z^k) \\ y^k &= \mathbf{prox}_f(2v^k - z^k) \\ z^{k+1} &= (1-\alpha)z^k + \alpha(\, 2(y^k - v^k) + z^k \,) \end{aligned}$$

First-order methods in operator form

Douglas-Rachford splitting:

$$\text{minimize } f(v) + g(v)$$

with $x \in \mathbb{R}^n$, $f, g \colon \mathbb{R}^n \to R \cup \{\infty\}$ (CCP)

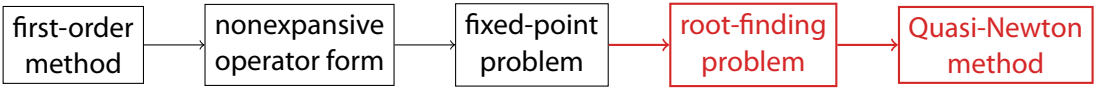
$$\Leftrightarrow 0 \in (\partial f + \partial g)(v) \Leftrightarrow z = ((1-\alpha)\mathsf{Id} + \alpha C_{\partial f} C_{\partial g})(z),\ v = R_{\partial g}(z)$$

Fixed-point iteration:

$$\begin{aligned} v^k &= \mathbf{prox}_g(z^k) \\ y^k &= \mathbf{prox}_f(2v^k - z^k) \\ z^{k+1} &= (1-\alpha)z^k + \alpha(2(y^k - v^k) - z^k) \end{aligned}$$

First-order methods in operator form

Method	Fixed point iteration
Gradient method	$x^{k+1} = (\mathsf{Id} - \alpha\nabla f)(x^k)$
$\hookrightarrow$ Dual ascent	
Proximal point method	$x^{k+1} = R_A(x^k) = (\mathsf{Id} + \alpha A)^{-1}$
$\hookrightarrow$ Method of multipliers	
$\hookrightarrow$ Iterative refinement	
Forward-backward splitting	$x^{k+1} = R_B(x^k - \alpha A x^k)$
$\hookrightarrow$ Proximal gradient method	
$\hookrightarrow$ Alternating projection method	
$\hookrightarrow$ Projected gradient method	
$\hookrightarrow$ Iterative shrinkage-thresholding algorithm (ISTA)	
Peaceman-Rachford splitting	$z^{k+1} = C_A C_B(z^k), x^{k+1} = R_B(z^{k+1})$
Douglas-Rachford splitting	$z^{k+1} = 1/2\mathsf{Id} + 1/2 C_A C_B(z^k)$ $x^{k+1} = R_B(z^{k+1})$
$\hookrightarrow$ ADMM	
$\hookrightarrow$ Consensus	



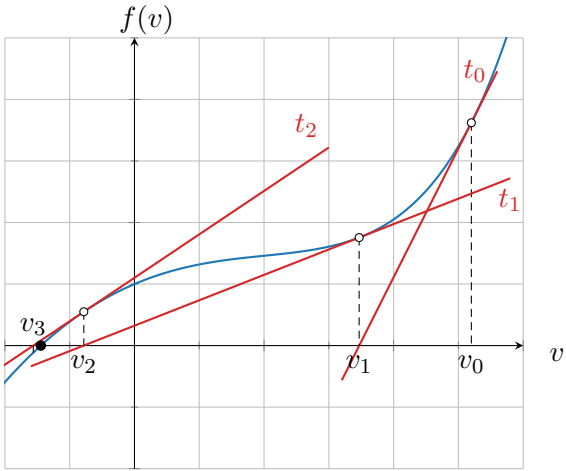
- Gradient descent
- Proximal gradient method
- Douglas-Rachford/ADMM
- Alternating projection method

[. . .]

- Broyden methods
- Anderson acceleration
- Line search

Quasi-Newton methods

- Our algorithm:
  $v_{k+1} = (1 - \alpha_k)v_k + \alpha_k T(v_k) = g(v_k)$
- Define residual operator:
  $f(v) := v - g(v)$
- Newton method:
  $v_{k+1} = v_k - J_f(v_k)^{-1} f(v_k)$
- Quasi-Newton method:
  $v_{k+1} = v_k - B_f(v_k)^{-1} f(v_k)$ or
  $v_{k+1} = v_k - H_f(v_k) f(v_k)$



Broyden's second method

- Improve the approximation of inverse Jacobian $H_k \approx J^{-1}(v_k)$ by taking the solution to the secant equation that is a minimal modification to H_k.

$$\begin{aligned} &\text{minimize} && \|H_{k+1} - H_k\|_F^2 \\ &\text{subject to} && H_{k+1} \underbrace{(f_{k+1} - f_k)}_{=\Delta f_k} = \underbrace{v_{k+1} - v_k}_{=\Delta v_k} \end{aligned}$$

- Analytic solution yields Broyden's second method:

$$\begin{aligned} v_{k+1} &= v_k - H_k(v_k) f(v_k) \\ H_{k+1} &= H_k + \frac{\Delta v_k - H_k \Delta f_k}{\|\Delta f_k\|^2} \Delta f_k^\top, \end{aligned}$$

Generalized Broyden's second method

- Now: Satisfy m multiple secant equations at the same time:

$$H_k \Delta f_i = \Delta v_i, \quad \text{for } i = k-m, \ldots, k-1$$

- In matrix form:

$$H_k \mathcal{F}_k = \mathcal{V}_k,$$

where $\mathcal{F}_k = [\Delta f_{k-m} \cdots \Delta f_{k-1}]$, $\mathcal{V}_k = [\Delta v_{k-m} \cdots \Delta v_{k-1}] \in \mathbb{R}^{n \times m}$

- One can derive a rank-m update for the inverse Jacobian H_k which minimizes $\|H_k - H_{k-m}\|_F^2$:

$$H_k = H_{k-m} + (\mathcal{V}_k - H_{k-m}\mathcal{F}_k)(\mathcal{F}_k^\top \mathcal{F}_k)^{-1} \mathcal{F}_k^\top.$$

Generalized Broyden's second method

This gives the update formula for v_{k+1}

$$\begin{aligned} v_{k+1} &= v_k - H_k f_k \\ &= v_k - H_{k-m} f_k - (\mathcal{V}_k - H_{k-m}\mathcal{F}_k)(\mathcal{F}_k^\top \mathcal{F}_k)^{-1}\mathcal{F}_k^\top f_k \\ &= v_k - H_{k-m} f_k - (\mathcal{V}_k - H_{k-m}\mathcal{F}_k)\gamma_k \end{aligned}$$

with $\gamma_k = \text{argmin}_\gamma \|\mathcal{F}_k\gamma - f_k\|_2$.

Pros:

- superlinear convergence in practice
- cheap update rule, efficient QR decomposition
- only need access to iterates

Cons:

- inverse has to stay well defined
- only local convergence guarantees

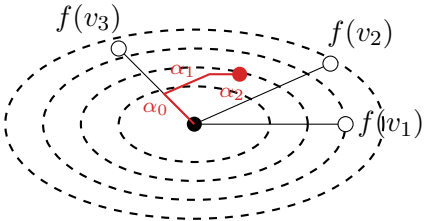
Anderson Acceleration

- Given fixed point problem $g(v) = v$ and the residual $f(v) = v - g(v)$

Input: v_0 and fixed-point iteration $g\colon \mathbb{R}^n \to \mathbb{R}^n$.
for *k = 0,1,...* **do**
  Set $m_k = \min\{m, k\}$;
  Select $m_k + 1$ weights α_j that satisfy $\sum_{j=0}^{m_k} \alpha_j = 1$;
  Update $v_{k+1} = \sum_{j=0}^{m_k} \alpha_j g(v_{k-m_k+j})$;
end

- Choose weights α_j to minimize weighted residuals:

  minimize $\quad \|\sum_{j=0}^{m_k} \alpha_j f(v_{k-m_k+j})\|_2^2$
  subject to $\quad \sum_{j=0}^{m_k} \alpha_j = 1$



* Anderson (1965)

Anderson Acceleration as a Broyden (type-II) update

- Constrained minimization problem:

$$\begin{array}{ll} \text{minimize} & \|\sum_{j=0}^{m_k} \alpha_j f(v_{k-m_k+j})\|_2^2 \\ \text{subject to} & \sum_{j=0}^{m_k} \alpha_j = 1 \end{array}$$

- This can be rewritten as an unconstrained least squares problem:

$$\gamma_k = \underset{\gamma}{\text{argmin}} \|\mathcal{F}_k \gamma - f_k\|_2^2 = (\mathcal{F}_k^\top \mathcal{F}_k)^{-1} \mathcal{F}_k^\top f_k$$

with

- $\alpha_0 = \gamma_0$, $\alpha_i = \gamma_i - \gamma_{i-1}$, $\alpha_{m_k} = 1 - \gamma_{m_k - 1}$
- $\mathcal{F}_k = [\Delta f_{k-m} \cdots \Delta f_{k-1}]$ (as before)

Anderson Acceleration as a Broyden (type-II) update

$$
\begin{aligned}
v_{k+1} &= \sum_{j=0}^{m_k} \alpha_j g(v_{k-m_k+j}) \\
&= g(v_k) - \sum_{i=0}^{m_k-1} \gamma_i \left(g(v_{k-m_k+i+1}) - g(v_{k-m_k+i}) \right) \\
&= v_k - f_k - (\mathcal{V}_k - \mathcal{F}_k)\gamma_k \\
&= v_k - \underbrace{\left(I + (\mathcal{V}_k - \mathcal{F}_k)(\mathcal{F}_k^\top \mathcal{F}_k)^{-1} \mathcal{F}_k^\top\right)}_{=H_k} f_k
\end{aligned}
$$

from previous slides:

- $f(v) = v - g(v)$
- $\mathcal{V}_k \coloneqq [\Delta v_{k-m} \cdots \Delta v_{k-1}]$
- $\mathcal{F}_k \coloneqq [\Delta f_{k-m} \cdots \Delta f_{k-1}]$
- $\gamma = (\gamma_0, \ldots, \gamma_{m_k-1})$
  $\alpha_j = \gamma_j - \gamma_{j-1}$
- $\gamma_k = (\mathcal{F}_k^\top \mathcal{F}_k)^{-1} \mathcal{F}_k^\top f_k$

- AA is performing a rank-m Broyden (type-II) update of I, and H_k minimizes $\|H_k - I\|_F$.

* Eyert (1996)

Our ADMM-algorithm

minimize $\frac{1}{2}x^\top Px + q^\top x$

subject to $Ax + s = b$

$s \in \mathcal{K}$

Our algorithm $v^{k+1} = g_\rho(v^k)$:

$$x^k = \Pi_{\mathbb{R}^n \times \mathcal{K}}(v^k)$$

$$y^k = \text{argmin}\{f(y) + \frac{\rho}{2}\|y - 2x^k + v^k\|_2^2\}$$

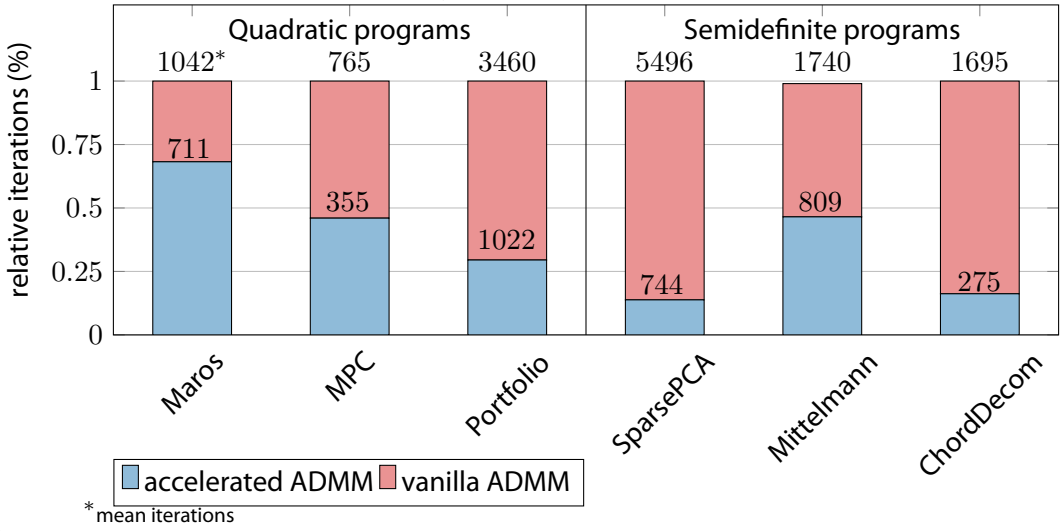
$$v^{k+1} = v^k + 2\alpha(y^k - x^k)$$

Safeguarded acceleration

Input: v_0, fixed-point iteration $g_\rho \colon \mathbb{R}^n \to \mathbb{R}^n$
while $f(v_k) \leq \epsilon$ **do**
  Accelerate: $v_k^{acc} = v_k - (I + (\mathcal{V}_k - \mathcal{F}_k)(\mathcal{F}_k^\top \mathcal{F}_k)^{-1}\mathcal{F}_k^\top) f_k$;
  ADMM step: $v_{k+1}^{acc} = g_\rho(v_k^{acc})$;
  if $\|f(v_k^{acc})\| \leq \tau \|f(v_{k-1})\|$ **then**
    $v_{k+1} = v_k^{acc}$
  else
    Safeguarding step: $v_{k+1} = g_\rho(v_k)$;
  end
  $\mathcal{V}_{k+1} \leftarrow [\mathcal{V}_k, \Delta v_k]$, $\mathcal{F}_{k+1} \leftarrow [\mathcal{F}_k, \Delta f_k]$;
  if *history full* **then**
    delete history;
  end
end

Numerical results

- Hardware: Oxford ARC-HTC 16 logical Intel Xeon E5-2560 cores, 64GB RAM



Numerical results

Table: Maros Meszaros QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	75	1042 (125)	1.62s (0.21s)	0
accelerated	97	711 (109)	1.10s (0.21s)	22 %

Table: Ferreau MPC QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	256	765 (125)	0.018 (0.0021s)	0
accelerated	278	355 (85)	0.013 (0.0021s)	24 %

* mean (median)

Numerical results

Table: Maros Meszaros QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	75	1042 (125)	1.62s (0.21s)	0
accelerated	97	711 (109)	1.10s (0.21s)	22 %

Table: Ferreau MPC QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	256	765 (125)	0.018 (0.0021s)	0
accelerated	278	355 (85)	0.013 (0.0021s)	24 %

* mean (median)

Numerical results

Table: Maros Meszaros QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	75	1042 (125)	1.62s (0.21s)	0
accelerated	97	711 (109)	1.10s (0.21s)	22%

Table: Ferreau MPC QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	256	765 (125)	0.018 (0.0021s)	0
accelerated	278	355 (85)	0.013 (0.0021s)	24%

* mean (median)

Numerical results

Table: Maros Meszaros QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	75	1042 (125)	1.62s (0.21s)	0
accelerated	97	711 (109)	1.10s (0.21s)	22 %

Table: Ferreau MPC QP set, $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	256	765 (125)	0.018 (0.0021s)	0
accelerated	278	355 (85)	0.013 (0.0021s)	24%

* mean (median)

Numerical results

Table: Portfolio Optimisation (QP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	10	3460 (3550)	178s (140s)	0
accelerated	10	1022 (918)	70s (41.5s)	3.1 %

Table: Sparse PCA (SDP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	9	5396 (5750)	97.7s (76.4s)	0
accelerated	9	744 (736)	14.6s (13.1s)	13%

* mean (median)

Numerical results

Table: Portfolio Optimisation (QP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	10	3460 (3550)	178s (140s)	0
accelerated	10	1022 (918)	70s (41.5s)	3.1 %

Table: Sparse PCA (SDP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	9	5396 (5750)	97.7s (76.4s)	0
accelerated	9	744 (736)	14.6s (13.1s)	13%

* mean (median)

Numerical results

Table: Portfolio Optimisation (QP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	10	3460 (3550)	178s (140s)	0
accelerated	10	1022 (918)	70s (41.5s)	3.1 %

Table: Sparse PCA (SDP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	9	5396 (5750)	97.7s (76.4s)	0
accelerated	9	744 (736)	14.6s (13.1s)	13%

* mean (median)

Numerical results

Table: Portfolio Optimisation (QP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	10	3460 (3550)	178s (140s)	0
accelerated	10	1022 (918)	70s (41.5s)	3.1 %

Table: Sparse PCA (SDP), $\epsilon = 1e^{-6}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	9	5396 (5750)	97.7s (76.4s)	0
accelerated	9	744 (736)	14.6s (13.1s)	13%

* mean (median)

Numerical results

Table: Mittelmann SDP collection, $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	15	1740 (1375)	207s (35.9s)	0
accelerated	22	809 (629)	97.6s (52.55s)	3.1 %

Table: Chordal decomposition (SDP), $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	16	1695 (1400)	38s (26.5s)	0
accelerated	21	275 (254)	7.9s (6.9s)	3.1%

* mean (median)

Numerical results

Table: Mittelmann SDP collection, $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	15	1740 (1375)	207s (35.9s)	0
accelerated	22	809 (629)	97.6s (52.55s)	3.1 %

Table: Chordal decomposition (SDP), $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	16	1695 (1400)	38s (26.5s)	0
accelerated	21	275 (254)	7.9s (6.9s)	3.1%

* mean (median)

Numerical results

Table: Mittelmann SDP collection, $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	15	1740 (1375)	207s (35.9s)	0
accelerated	22	809 (629)	97.6s (52.55s)	3.1 %

Table: Chordal decomposition (SDP), $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	16	1695 (1400)	38s (26.5s)	0
accelerated	21	275 (254)	7.9s (6.9s)	3.1%

* mean (median)

Numerical results

Table: Mittelmann SDP collection, $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter*	solve time*	% acc time
vanilla	15	1740 (1375)	207s (35.9s)	0
accelerated	22	809 (629)	97.6s (52.55s)	3.1 %

Table: Chordal decomposition (SDP), $\epsilon = 1e^{-5}$, memory size: 10

algorithm	solved	iter	solve time	% acc time
vanilla	16	1695 (1400)	38s (26.5s)	0
accelerated	21	275 (254)	7.9s (6.9s)	3.1%

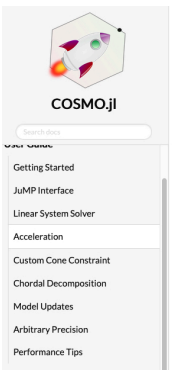
* mean (median)

Implementation in COSMO v0.8

```
------------------------------------------------------------------
          COSMO v0.8.0 - A Quadratic Objective Conic Solver
                       Michael Garstka
             University of Oxford, 2017 - 2021
------------------------------------------------------------------

Problem:  x ∈ R^{2},
          constraints: A ∈ R^{3x2} (4 nnz),
          matrix size to factor: 5x5,
          Floating-point precision: Float64
Sets:     Box of dim: 3
Settings: ϵ_abs = 1.0e-05, ϵ_rel = 1.0e-05,
          ϵ_prim_inf = 1.0e-04, ϵ_dual_inf = 1.0e-04,
          ρ = 0.1, σ = 1e-06, α = 1.6,
          max_iter = 5000,
          scaling iter = 10 (on),
          check termination every 25 iter,
          check infeasibility every 40 iter,
          KKT system solver: QDLDL
Acc:      Anderson Type2{QRDecomp},
          Memory size = 5, RestartedMemory,
          Safeguarded: true, tol: 2.0
Setup Time: 0.04ms

Iter:   Objective:      Primal Res:     Dual Res:       Rho:
1       -7.8808e-03     1.0079e+00      2.0033e+02      1.0000e-01
25      1.8800e+00      1.5712e-16      8.8818e-16      1.0000e-01

------------------------------------------------------------------
>>> Results
Status: Solved
Iterations: 26 (incl. 1 safeguarding iter)
Optimal objective: 1.88
Runtime: 0.001s (0.7ms)
```

Code and Documentation

`https://github.com/oxfordcontrol/COSMO.jl`
`https://github.com/oxfordcontrol/COSMOAccelerators.jl`



Conclusion:

- Acceleration effective for large QPs and SDPs
- Increased robustness and reduced solve times
- Works with ρ adaptation and chordal decomposition

Conclusion:

- Acceleration effective for large QPs and SDPs
- Increased robustness and reduced solve times
- Works with ρ adaptation and chordal decomposition

Open questions:

- Most effective acceleration variant
- Alternative safeguarding strategies
- Apply acceleration only to part of the operator
- Delayed acceleration
- Line search along Anderson direction

Conclusion:

- Acceleration effective for large QPs and SDPs
- Increased robustness and reduced solve times
- Works with ρ adaptation and chordal decomposition

Open questions:

- Most effective acceleration variant
- Alternative safeguarding strategies
- Apply acceleration only to part of the operator
- Delayed acceleration
- Line search along Anderson direction

Questions?