Perspective on the analysis and design of optimization algorithms

Adrien Taylor

AGIMUS Winter School — December 2023

Sierra team at Inria Paris

Main research themes: optimization, statistical learning, and interactions.

- ◇ Team leader: Francis Bach,
- ◇ Assistant: Marina Kovacic,
- ◇ 5 permanent researchers,
- ◇ 4 postdocs,
- ◇ 15 PhD students.

Perspective on the analysis and design of optimization algorithms

Adrien Taylor

AGIMUS Winter School — December 2023

Key messages

- fixed-point perspective on optimization algorithms

  many references on the topic.

Key messages

◇ fixed-point perspective on optimization algorithms

many references on the topic.[1,2,3,4]

[1] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[2] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[3] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[4] Ryu, Boyd. (2016). "Primer on monotone operator methods."

Key messages

◇ fixed-point perspective on optimization algorithms

many references on the topic.[1,2,3,4]

◇ a few (standard) ideas for designing methods,

[1] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[2] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[3] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[4] Ryu, Boyd. (2016). "Primer on monotone operator methods."

Key messages

◇ fixed-point perspective on optimization algorithms

many references on the topic.[1,2,3,4]

◇ a few (standard) ideas for designing methods,

◇ a constructive approach to algorithm analysis and design

... if time allows.

[1] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[2] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[3] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[4] Ryu, Boyd. (2016). "Primer on monotone operator methods."


François Glineur


Julien Hendrickx


Etienne de Klerk


Ernest Ryu


Carolina Bergeling


Pontus Giselsson


Francis Bach


Jérôme Bolte


Yoel Drori


Alexandre d'Aspremont


Mathieu Barré

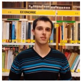

Radu Dragomir


Bryan Van Scoy


Laurent Lessard

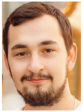

Aymeric Dieuleveut


Céline Moucer


Baptiste Goujaud


Sebastian Banert


Manu Uphadyaya


Eduard Gorbunov


Gauthier Gidel


Antoine Bambade


Sarah Kazdadi


Justin Carpentier

Convex optimization: minimize function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$

$$f(x_\star) \triangleq \min_{x \in \mathcal{D}} f(x),$$

with f, $\mathcal{D}$ convex.

Convex optimization: minimize function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$

$$f(x_\star) \triangleq \min_{x \in \mathcal{D}} f(x),$$

with f, $\mathcal{D}$ convex.

Use **iterative algorithm** generating a sequence $x_0, x_1, \ldots, x_N$.

Convex optimization: minimize function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$

$$f(x_\star) \triangleq \min_{x \in \mathcal{D}} f(x),$$

with f, $\mathcal{D}$ convex.

Use **iterative algorithm** generating a sequence $x_0, x_1, \ldots, x_N$.

Example: gradient descent iterates $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$.

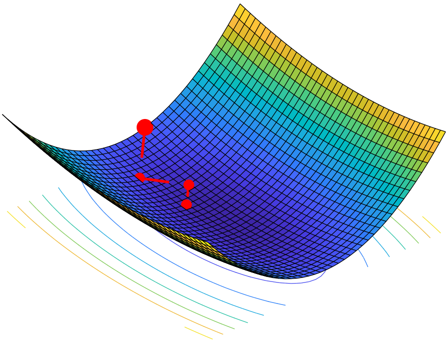
Convex optimization: minimize function $f : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$

$$f(x_\star) \triangleq \min_{x \in \mathcal{D}} f(x),$$

with f, $\mathcal{D}$ convex.

Use **iterative algorithm** generating a sequence $x_0, x_1, \ldots, x_N$.

Example: gradient descent iterates $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$.



Trust in numerical algorithms is a desirable luxury.

Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

Trust in numerical algorithms is a desirable luxury.

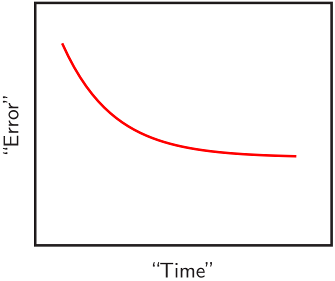
Question: what *a priori* guarantees after N iterations?

Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?

Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

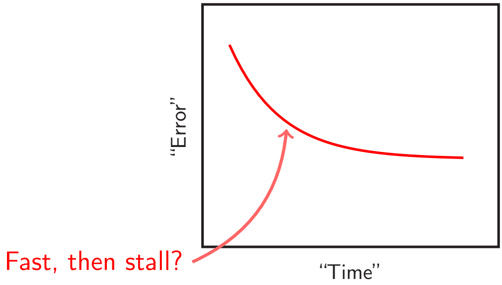
Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?



Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

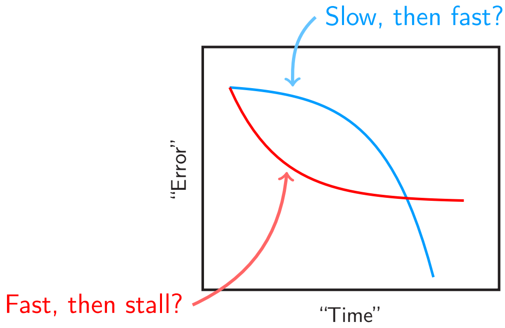
Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?



Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

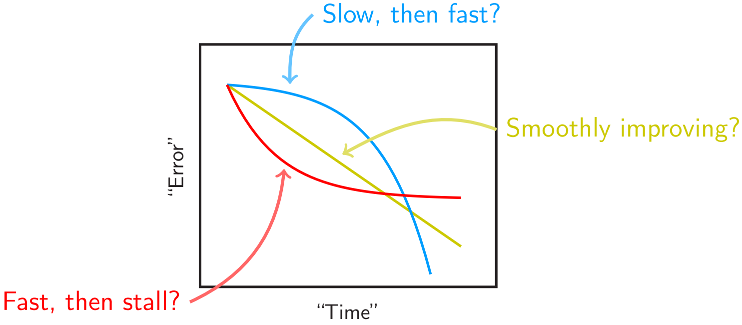
Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?



Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

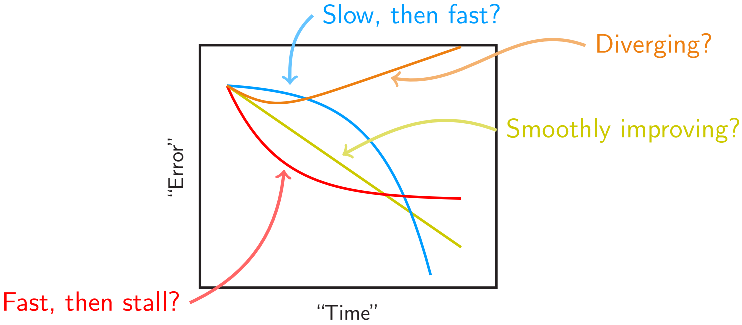
Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?



Trust in numerical algorithms is a desirable luxury.

Question: what *a priori* guarantees after N iterations?

Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?



Convergence of an algorithm? (if it works)

Convergence of an algorithm? (if it works)

Classical approaches:

Convergence of an algorithm? (if it works)

Classical approaches:

◇ “convergence” analysis (no speed),

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ “convergence” analysis (no speed),
- ◇ “asymptotical” (local) analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,
- ◇ average-case analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,
- ◇ average-case analyses,
- ◇ high-probability analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,
- ◇ average-case analyses,
- ◇ high-probability analyses,
- ◇ smoothed analyses.

A few elements of convex analysis

Extended-valued functions

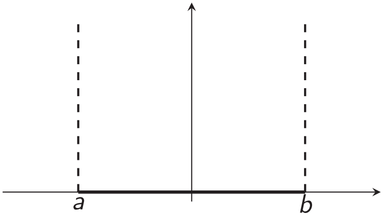
We consider **extended-valued functions** $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$.

Extended-valued functions

We consider **extended-valued functions** $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$.

Example: indicator function of interval $[a, b]$

$$i_{[a,b]}(x) = \begin{cases} 0 & \text{if } x \in [a,b] \\ +\infty & \text{otherwise.} \end{cases}$$

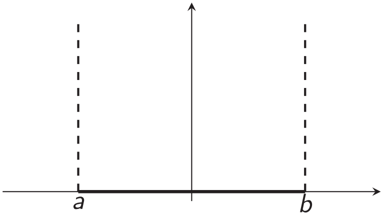


Extended-valued functions

We consider **extended-valued functions** $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$.

Example: indicator function of interval $[a, b]$

$$i_{[a,b]}(x) = \begin{cases} 0 & \text{if } x \in [a,b] \\ +\infty & \text{otherwise.} \end{cases}$$



Effective domain of $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$:

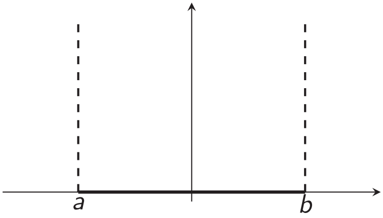
$$\operatorname{dom} f = \{x \ : \ f(x) < +\infty\}.$$

Extended-valued functions

We consider **extended-valued functions** $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$.

Example: indicator function of interval $[a, b]$

$$i_{[a,b]}(x) = \begin{cases} 0 & \text{if } x \in [a, b] \\ +\infty & \text{otherwise.} \end{cases}$$



Effective domain of $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$:

$$\operatorname{dom} f = \{x \ : \ f(x) < +\infty\}.$$

Note: so f might encode *implicit constraints*.

Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta) f(y),$$

$\forall \theta \in [0, 1]$.

Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y),$$

$\forall \theta \in [0, 1]$.

Examples:

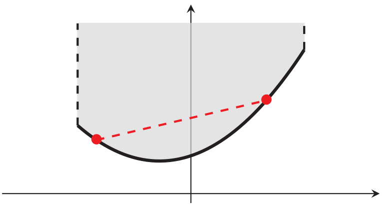
Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y),$$

$\forall \theta \in [0, 1]$.

Examples:



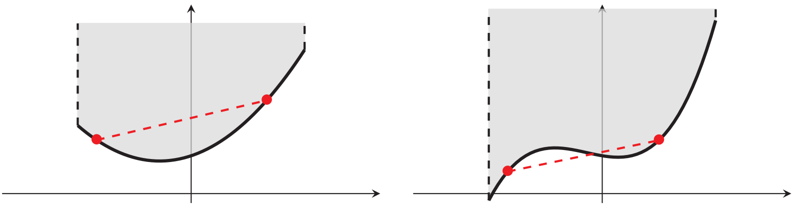
Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y),$$

$\forall \theta \in [0,1]$.

Examples:



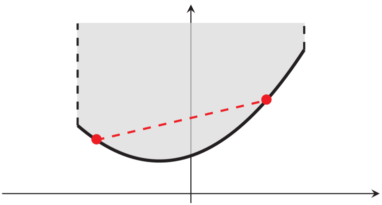
Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

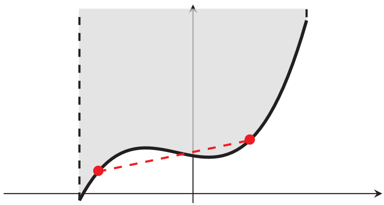
$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y),$$

$\forall \theta \in [0, 1]$.

Examples:



convex function



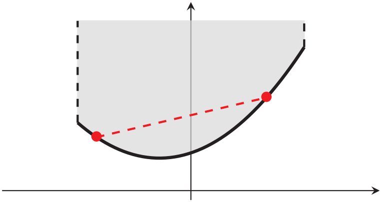
Convex functions

Graph is below line connecting any pairs $(x, f(x))$ and $(y, f(y))$:

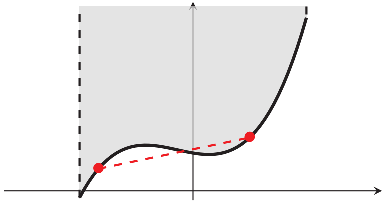
$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta) f(y),$$

$\forall \theta \in [0, 1]$.

Examples:

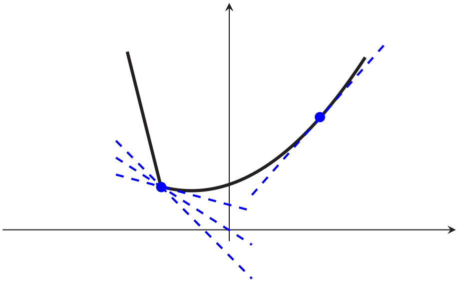


convex function

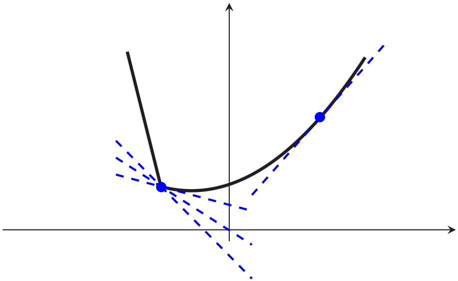


non-convex function.

Subgradients and subdifferentials

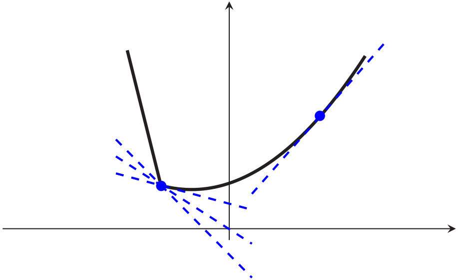


Subgradients and subdifferentials



Notation:

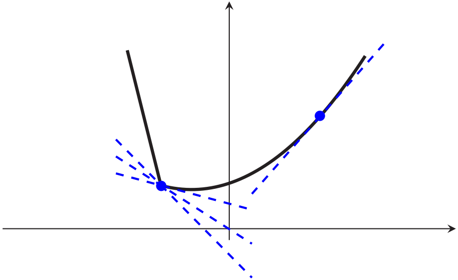
Subgradients and subdifferentials



Notation:

- **subdifferential**: $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ (power set),

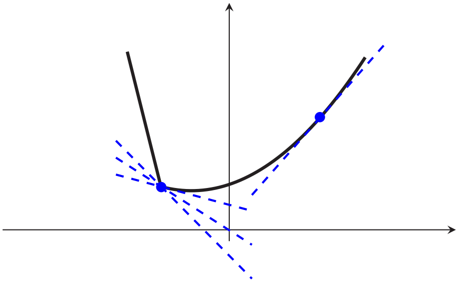
Subgradients and subdifferentials



Notation:

- ◇ **subdifferential**: $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ (power set),
- ◇ subdifferential at x:
  $\partial f(x) = \{g \ : \ f(y) \geqslant f(x) + g^T(y - x) \ \ \forall y \in \mathbb{R}^n\}$,

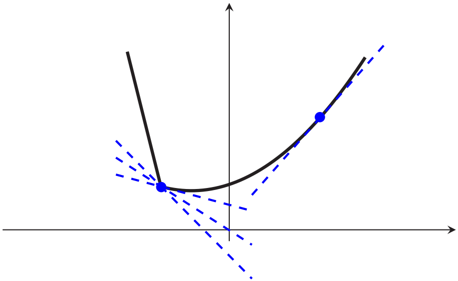
Subgradients and subdifferentials



Notation:

- **subdifferential**: $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ (power set),
- subdifferential at x:
  $\partial f(x) = \{g \ : \ f(y) \geqslant f(x) + g^T(y - x) \ \ \forall y \in \mathbb{R}^n\}$,
- any $g \in \partial f(x)$ is a **subgradient** of f at x.

Subgradients and subdifferentials

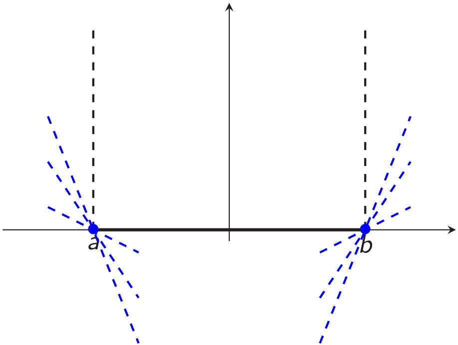


Notation:

- ◇ **subdifferential**: $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ (power set),
- ◇ subdifferential at x:
  $\partial f(x) = \{g \ : \ f(y) \geqslant f(x) + g^T(y - x) \ \ \forall y \in \mathbb{R}^n\}$,
- ◇ any $g \in \partial f(x)$ is a **subgradient** of f at x.

Essentially: (closed) convex functions have subgradients in relint(dom f).

Subgradients and subdifferentials



Notation:

- **subdifferential**: $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ (power set),
- subdifferential at x:
  $\partial f(x) = \{g \ : \ f(y) \geqslant f(x) + g^T(y - x) \ \ \forall y \in \mathbb{R}^n\}$,
- any $g \in \partial f(x)$ is a **subgradient** of f at x.

Essentially: (closed) convex functions have subgradients in $\mathrm{relint}(\mathrm{dom}\ f)$.

Optimality conditions

Convex optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Optimality condition (Fermat's rule): $x_\star$ optimal iff $0 \in \partial f(x_\star)$.

Optimality conditions

Convex optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Optimality condition (Fermat's rule): $x_\star$ optimal iff $0 \in \partial f(x_\star)$.

- Proof: x minimizes f if and only if

$$f(y) \geqslant f(x) = f(x) + 0^T(y - x) \text{ for all } y \in \mathbb{R}^n.$$

Optimality conditions

Convex optimization problem:

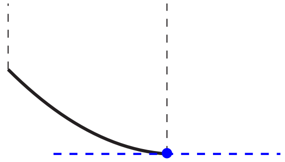
$$\min_{x \in \mathbb{R}^n} f(x)$$

Optimality condition (Fermat's rule): $x_\star$ optimal iff $0 \in \partial f(x_\star)$.

- Proof: x minimizes f if and only if

$$f(y) \geqslant f(x) = f(x) + 0^T(y - x) \text{ for all } y \in \mathbb{R}^n.$$

- Example: several subgradients at solution, including 0:



Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

- ◇ Under some conditions (e.g., "constraint qualifications"):

$$\partial(f + h)(x) = \partial f(x) + \partial h(x).$$

Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

- ◇ Under some conditions (e.g., "constraint qualifications"):

$$\partial(f + h)(x) = \partial f(x) + \partial h(x).$$

- ◇ We will search (Fermat's rule) for $x_\star$:

$$0 \in \partial(f + h)(x_\star) = \partial f(x_\star) + \partial h(x_\star),$$

where equality follows from constraint qualifications (e.g., Slater's).

Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

- Under some conditions (e.g., "constraint qualifications"):

$$\partial(f + h)(x) = \partial f(x) + \partial h(x).$$

- We will search (Fermat's rule) for $x_\star$:

$$0 \in \partial(f + h)(x_\star) = \partial f(x_\star) + \partial h(x_\star),$$

  where equality follows from constraint qualifications (e.g., Slater's).

This is strongly related to the usual KKT optimality conditions

Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

- Under some conditions (e.g., "constraint qualifications"):

$$\partial(f + h)(x) = \partial f(x) + \partial h(x).$$

- We will search (Fermat's rule) for $x_\star$:

$$0 \in \partial(f + h)(x_\star) = \partial f(x_\star) + \partial h(x_\star),$$

  where equality follows from constraint qualifications (e.g., Slater's).

This is strongly related to the usual KKT optimality conditions

- in fact: it is exactly the same (alternate terminology)

Optimality conditions (II)

Convex optimization problem (take II):

$$\min_{x \in \mathbb{R}^n} f(x) + h(x)$$

with both f and h convex (closed, non-empty effective domains).

- ◇ Under some conditions (e.g., "constraint qualifications"):

$$\partial(f + h)(x) = \partial f(x) + \partial h(x).$$

- ◇ We will search (Fermat's rule) for $x_\star$:

$$0 \in \partial(f + h)(x_\star) = \partial f(x_\star) + \partial h(x_\star),$$

  where equality follows from constraint qualifications (e.g., Slater's).

This is strongly related to the usual KKT optimality conditions

- ◇ in fact: it is exactly the same (alternate terminology)
- ◇ allows for simple manipulations.

A few approaches to constrained (convex) optimization

Splitting methods

No free lunch: no "general" method works well for all problems.

Splitting methods

No free lunch: no “general” method works well for all problems.

In numerical optimization, key is usually:

Splitting methods

No free lunch: no "general" method works well for all problems.

In numerical optimization, key is usually:

- ◇ clear identification of my specifications (timings, accuracy, etc.)?

Splitting methods

No free lunch: no "general" method works well for all problems.

In numerical optimization, key is usually:

- ◇ clear identification of my specifications (timings, accuracy, etc.)?
- ◇ divide problem into "cheap"/"simple" pieces...

Splitting methods

No free lunch: no “general” method works well for all problems.

In numerical optimization, key is usually:

- ◇ clear identification of my specifications (timings, accuracy, etc.)?
- ◇ divide problem into “cheap”/“simple” pieces...

Current library of numerical optimization algorithms:

Splitting methods

No free lunch: no "general" method works well for all problems.

In numerical optimization, key is usually:

- ◇ clear identification of my specifications (timings, accuracy, etc.)?
- ◇ divide problem into "cheap"/"simple" pieces...

Current library of numerical optimization algorithms:

- ◇ clearly a jungle,

Splitting methods

No free lunch: no “general” method works well for all problems.

In numerical optimization, key is usually:

- ◇ clear identification of my specifications (timings, accuracy, etc.)?
- ◇ divide problem into “cheap”/“simple” pieces...

Current library of numerical optimization algorithms:

- ◇ clearly a jungle,
- ◇ still, a few key “algorithmic templates” emerged.

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{ f(x) : \; x \in \mathcal{D} \}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?
- ◇ can I perform linear optimization on $\mathcal{D}$?

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?
- ◇ can I perform linear optimization on $\mathcal{D}$?
- ◇ can I compute gradients of f? Hessian?

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?
- ◇ can I perform linear optimization on $\mathcal{D}$?
- ◇ can I compute gradients of f? Hessian?
- ◇ stochastic approximations to f?

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?
- ◇ can I perform linear optimization on $\mathcal{D}$?
- ◇ can I compute gradients of f? Hessian?
- ◇ stochastic approximations to f?
- ◇ coordinate-wise optimization is easy?

Splitting methods: a few examples

A few examples for the problem:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

with f a (closed, proper) convex function and $\mathcal{D}$ a convex set.

Typical splitting strategies:

- ◇ can I project on $\mathcal{D}$?
- ◇ can I perform linear optimization on $\mathcal{D}$?
- ◇ can I compute gradients of f? Hessian?
- ◇ stochastic approximations to f?
- ◇ coordinate-wise optimization is easy?

(choices also depends on the targets, e.g., accuracy).

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x\in\mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\} .$$

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{ f(x) : \; x \in \mathcal{D} \}.$$

Assumptions:

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}.$$

Assumptions:

- I can access gradients of f,

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}.$$

Assumptions:

- I can access gradients of f,
- I can project on $\mathcal{D}$.

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}.$$

Assumptions:

- ◇ I can access gradients of f,
- ◇ I can project on $\mathcal{D}$.

Iterate: $x_{k+1} = \operatorname*{argmin}_{x \in \mathcal{D}} \left\{ f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{2}{\gamma} \|x - x_k\|_2^2 \right\}.$

Example: Projected gradient descent (I)

Differentiable (closed, proper) convex function f and convex set $\mathcal{D}$:

$$\min_{x\in\mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}.$$

Assumptions:

- ◇ I can access gradients of f,
- ◇ I can project on $\mathcal{D}$.

Iterate: $x_{k+1} = \operatorname*{argmin}_{x\in\mathcal{D}} \left\{ f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{2}{\gamma}\|x - x_k\|_2^2 \right\}.$

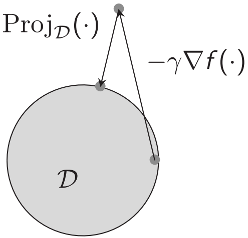
Equivalently:

$$x_{k+1} = \operatorname*{argmin}_{x\in\mathcal{D}} \left\{ \|x - (x_k - \gamma\nabla f(x_k))\|_2^2 \right\}.$$

so $x_{k+1} = \mathrm{Proj}_{\mathcal{D}}(x_k - \gamma\nabla f(x_k))$.

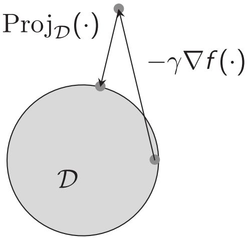
Example: Projected gradient descent (II)

$$x_{k+1} = \mathrm{Proj}_{\mathcal{D}}\left(x_k - \gamma \nabla f(x_k)\right)$$



Example: Projected gradient descent (II)

$x_{k+1} = \mathrm{Proj}_{\mathcal{D}}\left(x_k - \gamma \nabla f(x_k)\right)$



Guarantees when f convex with L-Lipschitz gradient and $\gamma \in (0, 2/L)$.
For instance, when $\gamma = 1/L$:

$$f(x_N) - f_\star \leqslant \frac{L\|x_0 - x_\star\|_2^2}{2N}.$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ◇ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ◇ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ◇ base transformations:

$$0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ◇ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ◇ base transformations:

$$0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) \Leftrightarrow 0 \in -\nabla f(x) - \partial i_{\mathcal{D}}(x)$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \left\{ \begin{array}{ll} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{array} \right.$.

Fixed-point viewpoint:

- ⋄ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ⋄ base transformations:

$$\begin{aligned} 0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) &\Leftrightarrow 0 \in -\nabla f(x) - \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x \in x - \gamma \nabla f(x) - \gamma \partial i_{\mathcal{D}}(x) \end{aligned}$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ⋄ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ⋄ base transformations:

$$\begin{aligned} 0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) &\Leftrightarrow 0 \in -\nabla f(x) - \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x \in x - \gamma \nabla f(x) - \gamma \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x + \gamma \partial i_{\mathcal{D}}(x) \ni x - \gamma \nabla f(x) \end{aligned}$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x\in\mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ⋄ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ⋄ base transformations:

$$\begin{aligned} 0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) &\Leftrightarrow 0 \in -\nabla f(x) - \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x \in x - \gamma\nabla f(x) - \gamma\partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x + \gamma\partial i_{\mathcal{D}}(x) \ni x - \gamma\nabla f(x) \\ &\Leftrightarrow x = (I + \gamma\partial i_{\mathcal{D}})^{-1}(x - \gamma\nabla f(x)) \end{aligned}$$

Example: Projected gradient descent (III)

Smooth convex function f and (closed) convex set $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} f(x) + i_{\mathcal{D}}(x),$$

with (convex) indicator function of set $\mathcal{D}$: $i_{\mathcal{D}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D} \\ +\infty & \text{otherwise.} \end{cases}$.

Fixed-point viewpoint:

- ◇ we want to solve $0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$,
- ◇ base transformations:

$$\begin{aligned} 0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) &\Leftrightarrow 0 \in -\nabla f(x) - \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x \in x - \gamma \nabla f(x) - \gamma \partial i_{\mathcal{D}}(x) \\ &\Leftrightarrow x + \gamma \partial i_{\mathcal{D}}(x) \ni x - \gamma \nabla f(x) \\ &\Leftrightarrow x = (I + \gamma \partial i_{\mathcal{D}})^{-1}(x - \gamma \nabla f(x)) \\ &\Leftrightarrow x = \mathrm{Proj}_{\mathcal{D}}(x - \gamma \nabla f(x)) \end{aligned}$$

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

Assumptions:

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

Assumptions:

- ◇ I can access gradients of f,

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \; x \in \mathcal{D}\}$$

Assumptions:

- $\diamond$ I can access gradients of f,
- $\diamond$ I can perform linear optimization on $\mathcal{D}$.

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

Assumptions:

- ⋄ I can access gradients of f,
- ⋄ I can perform linear optimization on $\mathcal{D}$.

Iterate:

$$s_k = \operatorname*{argmin}_{s \in \mathcal{D}} \left\{ f(x_k) + \nabla f(x_k)^T (s - x_k) \right\}$$

$$x_k = (1 - \lambda_k) x_{k-1} + \lambda_k s_k.$$

Example: Frank-Wolfe / conditional gradient (I)

Differentiable (closed, proper) convex f and compact convex $\mathcal{D}$:

$$\min_{x \in \mathbb{R}^n} \{f(x) : \ x \in \mathcal{D}\}$$

Assumptions:

- ◇ I can access gradients of f,
- ◇ I can perform linear optimization on $\mathcal{D}$.

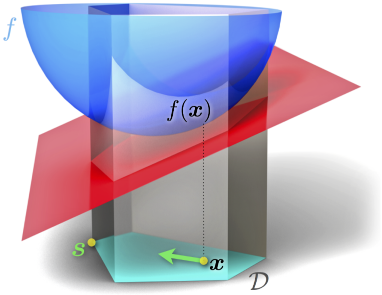
Iterate:

$$s_k = \operatorname*{argmin}_{s \in \mathcal{D}} \left\{ f(x_k) + \nabla f(x_k)^T (s - x_k) \right\}$$

$$x_k = (1 - \lambda_k) x_{k-1} + \lambda_k s_k.$$

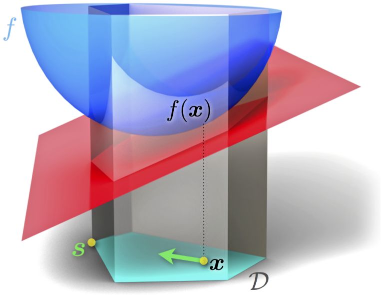
Typical choice for $\lambda_k = \frac{2}{k+1}$.

Example: Frank-Wolfe / conditional gradient (II)



Picture from M. Jaggi (2013): "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization."

Example: Frank-Wolfe / conditional gradient (II)

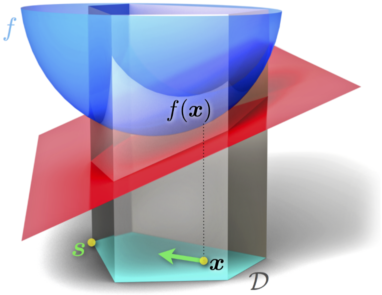


Picture from M. Jaggi (2013): "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization."

Guarantees when f convex with L-Lipschitz gradient and $\mathrm{Diam}(\mathcal{D}) < \infty$:

$$f(x_N) - f_\star \leqslant \frac{L\mathrm{Diam}(\mathcal{D})^2}{N+2}.$$

Example: Frank-Wolfe / conditional gradient (II)



Picture from M. Jaggi (2013): "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization."

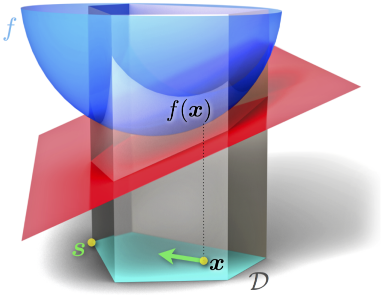
Guarantees when f convex with L-Lipschitz gradient and $\mathrm{Diam}(\mathcal{D}) < \infty$:

$$f(x_N) - f_\star \leqslant \frac{L\mathrm{Diam}(\mathcal{D})^2}{N+2}.$$

Similarly, can be seen as a fixed-point algorithm:

$$0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x)$$

Example: Frank-Wolfe / conditional gradient (II)



Picture from M. Jaggi (2013): "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization."

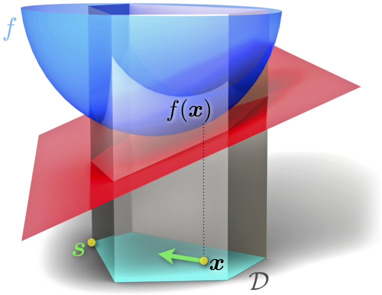
Guarantees when f convex with L-Lipschitz gradient and $\mathrm{Diam}(\mathcal{D}) < \infty$:

$$f(x_N) - f_\star \leqslant \frac{L\mathrm{Diam}(\mathcal{D})^2}{N+2}.$$

Similarly, can be seen as a fixed-point algorithm:

$$0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) \Leftrightarrow x \in \partial i_{\mathcal{D}}^{-1}(-\nabla f(x))$$

Example: Frank-Wolfe / conditional gradient (II)



Picture from M. Jaggi (2013): "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization."

Guarantees when f convex with L-Lipschitz gradient and $\mathrm{Diam}(\mathcal{D}) < \infty$:

$$f(x_N) - f_\star \leqslant \frac{L\mathrm{Diam}(\mathcal{D})^2}{N+2}.$$

Similarly, can be seen as a fixed-point algorithm:

$$\begin{aligned} 0 \in \nabla f(x) + \partial i_{\mathcal{D}}(x) &\Leftrightarrow x \in \partial i_{\mathcal{D}}^{-1}(-\nabla f(x)) \\ &\Leftrightarrow x \in (1-\lambda)\partial i_{\mathcal{D}}^{-1}(-\nabla f(x)) + \lambda x. \end{aligned}$$

Governing umbrella: fixed-point iterations

Splitting methods:

Governing umbrella: fixed-point iterations

Splitting methods:

- reformulate optimality conditions as fixed points,
  - using the operations that are "easy"
  - ex: can I invert a gradient? can I project? etc.

Governing umbrella: fixed-point iterations

Splitting methods:

- ◇ reformulate optimality conditions as fixed points,
  - using the operations that are "easy"
  - ex: can I invert a gradient? can I project? etc.
- ◇ check/hope that the fixed-point iteration converges,

Governing umbrella: fixed-point iterations

Splitting methods:

- reformulate optimality conditions as fixed points,
  - using the operations that are "easy"
  - ex: can I invert a gradient? can I project? etc.
- check/hope that the fixed-point iteration converges,
- a few other standard schemes and many variations around them
  - Douglas-Rachford Splitting, ADMM, Chambolle-Pock algorithm, three-operator (Davis-Yin) splitting...

Governing umbrella: fixed-point iterations

Splitting methods:

- ◇ reformulate optimality conditions as fixed points,
  - using the operations that are "easy"
  - ex: can I invert a gradient? can I project? etc.
- ◇ check/hope that the fixed-point iteration converges,
- ◇ a few other standard schemes and many variations around them
  - Douglas-Rachford Splitting, ADMM, Chambolle-Pock algorithm, three-operator (Davis-Yin) splitting...
- ◇ most of those strategies apply to more general situations
  - that include primal-dual optimality conditions.

The proximal-point method

Continuous-time perspective

Minimization of a convex function f:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Continuous-time perspective

Minimization of a convex function f:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Continuous-time interpretation of the minimization procedure:

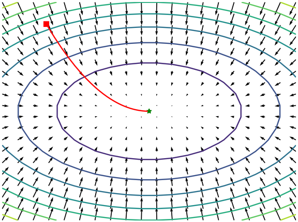
Continuous-time perspective

Minimization of a convex function f:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Continuous-time interpretation of the minimization procedure:

- ◇ "gradient flow": $\frac{d}{dt}x(t) = -\nabla f(x(t))$.



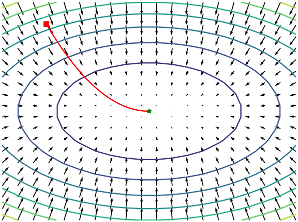
Continuous-time perspective

Minimization of a convex function f:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Continuous-time interpretation of the minimization procedure:

- ◇ "gradient flow": $\frac{d}{dt}x(t) = -\nabla f(x(t))$.



- ◇ Explicit Euler approx.: $x_{t+\Delta t} = x_t - \Delta t \nabla f(x_t)$ (gradient descent),

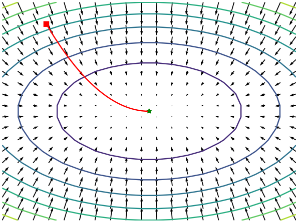
Continuous-time perspective

Minimization of a convex function f:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Continuous-time interpretation of the minimization procedure:

- ◇ "gradient flow": $\frac{d}{dt} x(t) = -\nabla f(x(t))$.



- ◇ Explicit Euler approx.: $x_{t+\Delta t} = x_t - \Delta t \nabla f(x_t)$ (gradient descent),
- ◇ Implicit Euler approx.: $x_{t+\Delta t} = x_t - \Delta t \nabla f(x_{t+\Delta t})$ (proximal point).

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Optimality condition: search x such that $0 \in \partial f(x)$.

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- ◇ Optimality condition: search x such that $0 \in \partial f(x)$.
- ◇ A fixed-point reformulation:

$$x \in x - \gamma \partial f(x)$$

  for some $\gamma > 0$.

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- ⋄ Optimality condition: search x such that $0 \in \partial f(x)$.
- ⋄ A fixed-point reformulation:

$$x \in x - \gamma \partial f(x)$$

for some $\gamma > 0$.

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- ⋄ Optimality condition: search x such that $0 \in \partial f(x)$.
- ⋄ A fixed-point reformulation:

$$x \in x - \gamma \partial f(x)$$

  for some $\gamma > 0$.
- ⋄ Proximal point: $x_{k+1} \in x_k - \gamma \partial f(x_{k+1})$.

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Optimality condition: search x such that $0 \in \partial f(x)$.
- A fixed-point reformulation:

$$x \in x - \gamma \partial f(x)$$

  for some $\gamma > 0$.
- Proximal point: $x_{k+1} \in x_k - \gamma \partial f(x_{k+1})$. Equivalently:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ f(x) + \frac{1}{2\gamma} \|x - x_k\|^2 \right\},$$

Proximal-point algorithm

Minimize a (closed, proper) convex functions:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Optimality condition: search x such that $0 \in \partial f(x)$.
- A fixed-point reformulation:

$$x \in x - \gamma \partial f(x)$$

  for some $\gamma > 0$.
- Proximal point: $x_{k+1} \in x_k - \gamma \partial f(x_{k+1})$. Equivalently:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ f(x) + \tfrac{1}{2\gamma} \|x - x_k\|^2 \right\},$$

- guaranteed:

$$f(x_k) - f(x_\star) \leqslant \frac{\|x_0 - x_\star\|^2}{4 \sum_{i=0}^{k-1} \gamma_i}.$$

Proximal-point splittings

Proximal-point operation:

- ◇ has a very long history

Proximal-point splittings

Proximal-point operation:

- has a very long history[5,6,7]

[5] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[6] Minty (1962). "Monotone (nonlinear) operators in Hilbert space."

[7] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

Proximal-point splittings

Proximal-point operation:

- ◇ has a very long history[5,6,7]
- ◇ are at the center of many "first-order" algorithms,

[5] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[6] Minty (1962). "Monotone (nonlinear) operators in Hilbert space."

[7] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

Proximal-point splittings

Proximal-point operation:

- ◇ has a very long history[5,6,7]
- ◇ are at the center of many "first-order" algorithms,[8,9,10]

[5] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[6] Minty (1962). "Monotone (nonlinear) operators in Hilbert space."

[7] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[8] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[9] Ryu, Boyd. (2016). "Primer on monotone operator methods."

[10] Condat (2022). "Proximal Splitting Algorithms for Convex Optimization: A Tour of Recent Advances, with New Twists."

Proximal-point splittings

Proximal-point operation:

- ◇ has a very long history[5,6,7]
- ◇ are at the center of many "first-order" algorithms,[8,9,10]
- ◇ Many examples of proximal operators with closed forms:

  `http://proximity-operator.net/`,

[5] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[6] Minty (1962). "Monotone (nonlinear) operators in Hilbert space."

[7] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[8] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[9] Ryu, Boyd. (2016). "Primer on monotone operator methods."

[10] Condat (2022). "Proximal Splitting Algorithms for Convex Optimization: A Tour of Recent Advances, with New Twists."

Proximal-point splittings

Proximal-point operation:

- ◇ has a very long history[5,6,7]
- ◇ are at the center of many "first-order" algorithms,[8,9,10]
- ◇ Many examples of proximal operators with closed forms:

  `http://proximity-operator.net/`,
- ◇ many results on "approximated" proximal-point too.

[5] Moreau (1962). "Fonctions convexes duales et points proximaux dans un espace hilbertien."

[6] Minty (1962). "Monotone (nonlinear) operators in Hilbert space."

[7] Rockafellar (1976). "Augmented Lagrangians and applications of the proximal point algorithm in convex programming."

[8] Combettes, Pesquet (2011). "Proximal splitting methods in signal processing."

[9] Ryu, Boyd. (2016). "Primer on monotone operator methods."

[10] Condat (2022). "Proximal Splitting Algorithms for Convex Optimization: A Tour of Recent Advances, with New Twists."

Proximal-point for convex QPs (I)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

with the proximal-point algorithm:

Proximal-point for convex QPs (I)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_{x} \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

with the proximal-point algorithm:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ f(x) + \frac{1}{2\gamma_k} \|x - x_k\|^2 \right\}.$$

Proximal-point for convex QPs (I)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \; : \; Ax \leqslant b \right\}.$$

with the proximal-point algorithm:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ f(x) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 \right\}.$$

Bad news:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ \tfrac{1}{2} x^T Q x + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 \; : \; Ax \leqslant b \right\}$$

is just as hard as the original problem.

Proximal-point for QPs (II)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\} .$$

Proximal-point for QPs (II)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

Introduce $L(x, \lambda) = \frac{1}{2} x^T Q x + q^T x + \lambda^T (Ax - b)$

Proximal-point for QPs (II)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

Introduce $L(x, \lambda) = \frac{1}{2} x^T Q x + q^T x + \lambda^T (Ax - b)$ and define dual ($\lambda \geqslant 0$)

$$d(\lambda) = \min_x \ L(x, \lambda).$$

Proximal-point for QPs (II)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

Introduce $L(x, \lambda) = \frac{1}{2} x^T Q x + q^T x + \lambda^T (Ax - b)$ and define dual ($\lambda \geqslant 0$)

$$d(\lambda) = \min_x \ L(x, \lambda).$$

$d(\lambda)$ is concave so we can apply the proximal-point on the dual:

$$\lambda_{k+1} = \operatorname*{argmax}_{\lambda \geqslant 0} \left\{ d(\lambda) - \tfrac{1}{2\gamma_k} \|\lambda - \lambda_k\|^2 \right\}.$$

Proximal-point for QPs (II)

Minimize a quadratic function (with $Q \succcurlyeq 0$) under linear constraints:

$$\min_x \left\{ \tfrac{1}{2} x^T Q x \ : \ Ax \leqslant b \right\}.$$

Introduce $L(x, \lambda) = \frac{1}{2} x^T Q x + q^T x + \lambda^T (Ax - b)$ and define dual ($\lambda \geqslant 0$)

$$d(\lambda) = \min_x \ L(x, \lambda).$$

$d(\lambda)$ is concave so we can apply the proximal-point on the dual:

$$\lambda_{k+1} = \operatorname*{argmax}_{\lambda \geqslant 0} \left\{ d(\lambda) - \tfrac{1}{2\gamma_k} \|\lambda - \lambda_k\|^2 \right\}.$$

Explicitly max. over λ (not over x) yields the **method of mulipliers**:

$$x_{k+1} \in \operatorname*{argmin}_x L\left(x, [\lambda_k - \gamma_k (Ax - b)]_+\right)$$

$$\lambda_{k+1} = [\lambda_k - \gamma_k (Ax_{k+1} - b)]_+$$

(problem in x is piecewise quadratic, but convex and unconstrained).

Proximal-point for QPs (III)

Proximal-point also applies to the saddle-point formulation:

$$\max_{\lambda \geqslant 0} \min_{x} \; L(x, \lambda)$$

which is convex-convave, yielding the **proximal method of multipliers**:

Proximal-point for QPs (III)

Proximal-point also applies to the saddle-point formulation:

$$\max_{\lambda \geqslant 0} \min_{x} \; L(x, \lambda)$$

which is convex-convave, yielding the **proximal method of multipliers**:

$$(\lambda_{k+1}, x_{k+1}) = \operatorname*{argmax}_{\lambda \geqslant 0} \operatorname*{argmin}_{x} \left\{ L(x, \lambda) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 - \tfrac{1}{2\beta_k} \|\lambda - \lambda_k\|^2 \right\}.$$

Proximal-point for QPs (III)

Proximal-point also applies to the saddle-point formulation:

$$\max_{\lambda \geqslant 0} \min_{x} \; L(x, \lambda)$$

which is convex-convave, yielding the **proximal method of multipliers**:

$$(\lambda_{k+1}, x_{k+1}) = \operatorname*{argmax}_{\lambda \geqslant 0} \operatorname*{argmin}_{x} \left\{ L(x, \lambda) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 - \tfrac{1}{2\beta_k} \|\lambda - \lambda_k\|^2 \right\}.$$

In practice, we need to:

- ◇ approximate

$$(\lambda_{k+1}, x_{k+1}) \approx_{\epsilon} \operatorname*{argmax}_{\lambda \geqslant 0} \operatorname*{argmin}_{x} \left\{ L(x, \lambda) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 - \tfrac{1}{2\beta_k} \|\lambda - \lambda_k\|^2 \right\},$$

Proximal-point for QPs (III)

Proximal-point also applies to the saddle-point formulation:

$$\max_{\lambda \geqslant 0} \min_{x} \; L(x, \lambda)$$

which is convex-convave, yielding the **proximal method of multipliers**:

$$(\lambda_{k+1}, x_{k+1}) = \operatorname*{argmax}_{\lambda \geqslant 0} \operatorname*{argmin}_{x} \left\{ L(x, \lambda) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 - \tfrac{1}{2\beta_k} \|\lambda - \lambda_k\|^2 \right\}.$$

In practice, we need to:

- ◇ approximate

$$(\lambda_{k+1}, x_{k+1}) \approx_\epsilon \operatorname*{argmax}_{\lambda \geqslant 0} \operatorname*{argmin}_{x} \left\{ L(x, \lambda) + \tfrac{1}{2\gamma_k} \|x - x_k\|^2 - \tfrac{1}{2\beta_k} \|\lambda - \lambda_k\|^2 \right\},$$

- ◇ choose appropriate step size policies (β_k and γ_k) trading-off:
  - – large values: less iterations,
  - – small values: inner problem is simpler.

Antoine Bambade

Sarah Kazdadi

Fabian Schramm

Justin Carpentier

ProxSuite

THE ADVANCED PROXIMAL OPTIMIZATION TOOLBOX

License BSD 2-Clause | docs online | CI - Linux/OSX/Windows - Conda passing | pypi package 0.3.6 | Anaconda.org 0.3.6

ProxSuite is a collection of open-source, numerically robust, precise and efficient numerical solvers (e.g., LPs, QPs, etc.) rooted in revisited primal-dual proximal algorithms. Through **ProxSuite**, we aim to offer the community scalable optimizers that can deal with dense, sparse or matrix-free problems. While the first targeted application is Robotics, **ProxSuite** can be used in other contexts without limits.

ProxSuite is actively developped and supported by the Willow and Sierra research groups, joint research teams between Inria, École Normale Supérieure de Paris and Centre National de la Recherche Scientifique localized in France.

ProxSuite is already integrated into:

- CVXPY modeling language for convex optimization problems,
- CasADi's symbolic framework for numerical optimization in general and optimal control. ProxQP is available in CasADi as plugin to solve quadratic programs,
- TSID: robotic software for efficient robot inverse dynamics with contacts and based on Pinocchio.

We are ready to integrate **ProxSuite** within other optimization ecosystems.

About

The Advanced Proximal Optimization Toolbox

cpp robotics optimization linear-programming proximal-algorithms quadratic-programming eigen3

Readme

BSD-2-Clause license

Cite this repository

Activity

288 stars

14 watching

40 forks

Report repository

Releases 35

ProxSuite 0.6.1 Latest

last month

+ 34 releases

A priori working guarantees?

for numerical optimization algorithms

Convergence of an algorithm? (if it works)

Convergence of an algorithm? (if it works)

Classical approaches:

Convergence of an algorithm? (if it works)

Classical approaches:

◇ "convergence" analysis (no speed),

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,
- ◇ average-case analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ⋄ "convergence" analysis (no speed),
- ⋄ "asymptotical" (local) analyses,
- ⋄ worst-case (global) analyses,
- ⋄ average-case analyses,
- ⋄ high-probability analyses,

Convergence of an algorithm? (if it works)

Classical approaches:

- ◇ "convergence" analysis (no speed),
- ◇ "asymptotical" (local) analyses,
- ◇ worst-case (global) analyses,
- ◇ average-case analyses,
- ◇ high-probability analyses,
- ◇ smoothed analyses.

Here: classical worst-case framework

Worst-case complexity analyses

Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

Worst-case complexity analyses

Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

How to ensure an algorithm to "**always**" work with prescribed guarantees?

Worst-case complexity analyses

Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

How to ensure an algorithm to "**always**" work with prescribed guarantees?

- ◇ "if it works on the **worst** $f \in \mathcal{F}$, it works on all $f \in \mathcal{F}$."

Worst-case complexity analyses

Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

How to ensure an algorithm to "**always**" work with prescribed guarantees?

- "if it works on the **worst** $f \in \mathcal{F}$, it works on all $f \in \mathcal{F}$."

Philosophically, what we target:

Worst-case complexity analyses

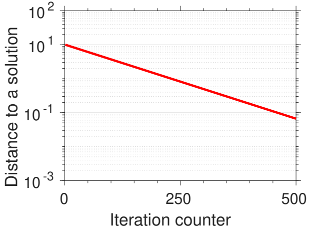
Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

How to ensure an algorithm to "**always**" work with prescribed guarantees?

- ◇ "if it works on the **worst** $f \in \mathcal{F}$, it works on all $f \in \mathcal{F}$."

Philosophically, what we target:

Good tuning of γ_k



— Worst-case guarantee

Worst-case complexity analyses

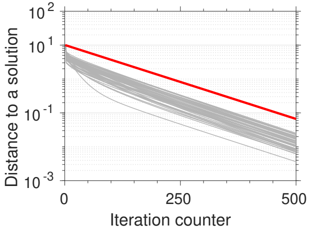
Requires **assumptions** on f (formally: $f \in \mathcal{F}$ for a certain $\mathcal{F}$).

How to ensure an algorithm to "**always**" work with prescribed guarantees?

◇ "if it works on the **worst** $f \in \mathcal{F}$, it works on all $f \in \mathcal{F}$."

Philosophically, what we target:

Good tuning of γ_k



— Worst-case guarantee — Experiments

(Here: convex quadratic min., gradient descent with random init.)

Worst-case complexity analyses

Worst-case complexity analyses

Key points:

Worst-case complexity analyses

Key points:

- worst-case convergence $\Rightarrow$ we can **trust** method as black-boxes,

Worst-case complexity analyses

Key points:

- worst-case convergence $\Rightarrow$ we can **trust** method as black-boxes,
- worst-case guarantees $\Rightarrow$ guide for **method tuning**.

Worst-case complexity analyses

Key points:

- ◇ worst-case convergence ⇒ we can **trust** method as black-boxes,
- ◇ worst-case guarantees ⇒ guide for **method tuning**.

This is a whole field of study, with many results.[11,12,13]

[11] Polyak (1964). Some methods of speeding up the convergence of iteration methods.

[12] Nemirovskii, Yudin (1983). Problem complexity and method efficiency in optimization.

[13] Nesterov (2018). Lectures on convex optimization.

Worst-case complexity analyses

Key points:

- ◇ worst-case convergence ⇒ we can **trust** method as black-boxes,
- ◇ worst-case guarantees ⇒ guide for **method tuning**.

This is a whole field of study, with many results.[11,12,13]

A few limitations of the traditional viewpoint:

- ◇ **conservative** nature,

[11] Polyak (1964). Some methods of speeding up the convergence of iteration methods.
[12] Nemirovskii, Yudin (1983). Problem complexity and method efficiency in optimization.
[13] Nesterov (2018). Lectures on convex optimization.

Worst-case complexity analyses

Key points:

- ◇ worst-case convergence ⇒ we can **trust** method as black-boxes,
- ◇ worst-case guarantees ⇒ guide for **method tuning**.

This is a whole field of study, with many results.[11,12,13]

A few limitations of the traditional viewpoint:

- ◇ **conservative** nature,
- ◇ **technical**, error-prone,

[11] Polyak (1964). Some methods of speeding up the convergence of iteration methods.

[12] Nemirovskii, Yudin (1983). Problem complexity and method efficiency in optimization.

[13] Nesterov (2018). Lectures on convex optimization.

Worst-case complexity analyses

Key points:

- ⋄ worst-case convergence $\Rightarrow$ we can **trust** method as black-boxes,
- ⋄ worst-case guarantees $\Rightarrow$ guide for **method tuning**.

This is a whole field of study, with many results.[11,12,13]

A few limitations of the traditional viewpoint:

- ⋄ **conservative** nature,
- ⋄ **technical**, error-prone,
- ⋄ **lack global insights**.

[11] Polyak (1964). Some methods of speeding up the convergence of iteration methods.
[12] Nemirovskii, Yudin (1983). Problem complexity and method efficiency in optimization.
[13] Nesterov (2018). Lectures on convex optimization.

Global insight of worst-case analyses?

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x \in \mathbb{R}^d} f(x).$$

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x \in \mathbb{R}^d} f(x).$$

(Gradient method) We decide to use: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x\in\mathbb{R}^d} f(x).$$

(Gradient method) We decide to use: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$

Question: what *a priori* guarantees after N iterations?

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x \in \mathbb{R}^d} f(x).$$

(Gradient method) We decide to use: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$

Question: what *a priori* guarantees after N iterations?

Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x\in\mathbb{R}^d} f(x).$$

(Gradient method) We decide to use: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$

Question: what *a priori* guarantees after N iterations?

Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?

Alternatively: how fast does the ratio $\frac{\text{“error at iteration } N\text{”}}{\text{“initial error”}}$ decrease with N?

Example: analysis of a gradient method

Find $x_\star \in \mathbb{R}^d$ such that

$$f(x_\star) = \min_{x\in\mathbb{R}^d} f(x).$$

(Gradient method) We decide to use: $x_{k+1} = x_k - \gamma_k \nabla f(x_k)$

Question: what *a priori* guarantees after N iterations?

Examples: what about $f(x_N) - f(x_\star)$, $\|\nabla f(x_N)\|$, $\|x_N - x_\star\|$?

Alternatively: how fast does the ratio $\frac{\text{“error at iteration } N\text{”}}{\text{“initial error”}}$ decrease with N?

Such guarantees reachable only by assuming something on f.

Typical assumptions

Nontrivial guarantees only by assuming something on class of problems!

Typical assumptions

Nontrivial guarantees only by assuming something on class of problems!

Many standard (documented) classes of functions.

Typical assumptions

Nontrivial guarantees only by assuming something on class of problems!

Many standard (documented) classes of functions.

Among many others: diff. function f is commonly assumed to be (for all $x, y \in \mathbb{R}^d$):

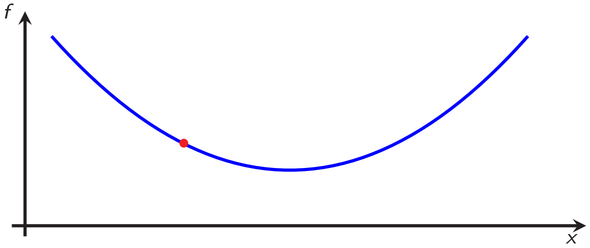
- ◇ L-smooth: $f(x) \leqslant f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2$,
- ◇ convex: $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle$,
- ◇ μ-strongly convex: $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle + \frac{\mu}{2}\|x - y\|^2$.

Smooth strongly convex functions

Consider a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, f is (μ-strongly) convex and L-smooth iff $\forall x, y \in \mathbb{R}^d$ we have:

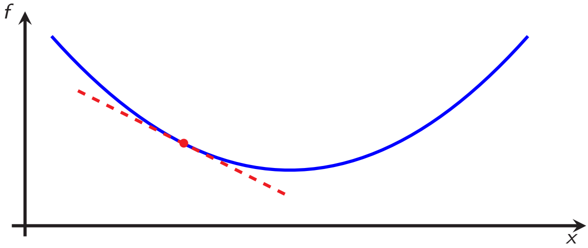
Smooth strongly convex functions

Consider a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, f is (μ-strongly) convex and L-smooth iff $\forall x, y \in \mathbb{R}^d$ we have:



Smooth strongly convex functions

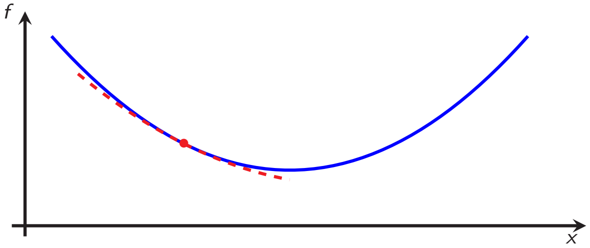
Consider a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, f is (μ-strongly) convex and L-smooth iff $\forall x, y \in \mathbb{R}^d$ we have:



(1) (Convexity) $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle$,

Smooth strongly convex functions

Consider a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, f is (μ-strongly) convex and L-smooth iff $\forall x, y \in \mathbb{R}^d$ we have:

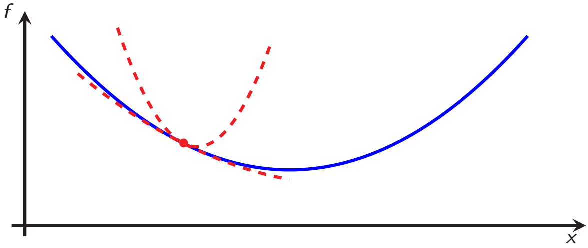


(1) (Convexity) $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle$,

(1b) (μ-strong convexity) $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle + \frac{\mu}{2}\|x - y\|^2$,

Smooth strongly convex functions

Consider a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, f is (μ-strongly) convex and L-smooth iff $\forall x, y \in \mathbb{R}^d$ we have:



(1) (Convexity) $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle$,

(1b) (μ-strong convexity) $f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle + \frac{\mu}{2}\|x - y\|^2$,

(2) (L-smoothness) $f(x) \leqslant f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2$.

Convergence rate of a gradient step

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,
- $x_\star = \underset{x}{\operatorname{argmin}}\ f(x)$?

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- ◇ L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- ◇ x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,
- ◇ $x_\star = \underset{x}{\text{argmin}}\ f(x)$?

Computing τ?

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,
- $x_\star = \operatorname{argmin}_x f(x)$?

Computing τ?

$$\begin{aligned} \tau = \max_{f, x_0, x_1, x_\star} \; & \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2} & \\ \text{s.t.} \; & f \in \mathcal{F}_{\mu,L} & \text{Functional class} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) & \text{Algorithm} \\ & \nabla f(x_\star) = 0 & \text{Optimality of } x_\star \end{aligned}$$

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,
- $x_\star = \underset{x}{\operatorname{argmin}}\ f(x)$?

Computing τ?

$$\begin{aligned} \tau = \max_{f, x_0, x_1, x_\star} \ & \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2} & \\ \text{s.t. } & f \in \mathcal{F}_{\mu,L} & \text{Functional class} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) & \text{Algorithm} \\ & \nabla f(x_\star) = 0 & \text{Optimality of } x_\star \end{aligned}$$

<u>Variables</u>: f, x_0, x_1, $x_\star$; <u>parameters</u>: μ, L, γ_0.

Convergence rate of a gradient step

Toy example: What is the smallest τ such that:

$$\|x_1 - x_\star\|^2 \leqslant \tau \|x_0 - x_\star\|^2,$$

for all

- L-smooth and μ-strongly convex function f (notation $f \in \mathcal{F}_{\mu,L}$),
- x_0, and x_1 generated by gradient step $x_1 = x_0 - \gamma_0 \nabla f(x_0)$,
- $x_\star = \underset{x}{\operatorname{argmin}}\ f(x)$?

Computing τ?

$$\begin{aligned} \tau = \max_{f, x_0, x_1, x_\star} \ & \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2} \\ \text{s.t. } & f \in \mathcal{F}_{\mu,L} && \text{Functional class} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) && \text{Algorithm} \\ & \nabla f(x_\star) = 0 && \text{Optimality of } x_\star \end{aligned}$$

Variables: f, x_0, x_1, $x_\star$; parameters: μ, L, γ_0.

Problem can be reformulated as semidefinite program (SDP).

Sampled version

Sampled version

◇ Performance estimation problem:

$$\begin{array}{rl}\displaystyle\max_{f, x_0, x_1, x_\star} & \dfrac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\ \text{subject to} & f \text{ is } L\text{-smooth and } \mu\text{-strongly convex,} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) \\ & \nabla f(x_\star) = 0. \end{array}$$

Sampled version

- Performance estimation problem:

$$\begin{aligned} \max_{f, x_0, x_1, x_\star} \quad & \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\ \text{subject to} \quad & f \text{ is } L\text{-smooth and } \mu\text{-strongly convex,} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) \\ & \nabla f(x_\star) = 0. \end{aligned}$$

- Variables: f, x_0, x_1, $x_\star$.

Sampled version

- Performance estimation problem:

$$\max_{f, x_0, x_1, x_\star} \quad \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2}$$
$$\text{subject to} \quad f \text{ is } L\text{-smooth and } \mu\text{-strongly convex,}$$
$$x_1 = x_0 - \gamma_0 \nabla f(x_0)$$
$$\nabla f(x_\star) = 0.$$

- Variables: f, x_0, x_1, $x_\star$.
- Sampled version: f is only used at x_0 and $x_\star$ (no need to sample other points)

Sampled version

- Performance estimation problem:

$$
\begin{aligned}
\max_{f, x_0, x_1, x_\star} \quad & \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\
\text{subject to} \quad & f \text{ is } L\text{-smooth and } \mu\text{-strongly convex,} \\
& x_1 = x_0 - \gamma_0 \nabla f(x_0) \\
& \nabla f(x_\star) = 0.
\end{aligned}
$$

- Variables: f, x_0, x_1, $x_\star$.
- Sampled version: f is only used at x_0 and $x_\star$ (no need to sample other points)

$$
\begin{aligned}
\max_{\substack{x_0, x_1, x_\star \\ g_0, g_\star \\ f_0, f_\star}} \quad & \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\
\text{subject to} \quad & \exists f \in \mathcal{F}_{\mu,L} \text{ such that } \begin{cases} f_i = f(x_i) & i = 0, \star \\ g_i = \nabla f(x_i) & i = 0, \star \end{cases} \\
& x_1 = x_0 - \gamma_0 g_0 \\
& g_\star = 0.
\end{aligned}
$$

Sampled version

- Performance estimation problem:

$$\begin{aligned} \max_{f, x_0, x_1, x_\star} \quad & \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\ \text{subject to} \quad & f \text{ is } L\text{-smooth and } \mu\text{-strongly convex,} \\ & x_1 = x_0 - \gamma_0 \nabla f(x_0) \\ & \nabla f(x_\star) = 0. \end{aligned}$$

- Variables: f, x_0, x_1, $x_\star$.
- Sampled version: f is only used at x_0 and $x_\star$ (no need to sample other points)

$$\begin{aligned} \max_{\substack{x_0, x_1, x_\star \\ g_0, g_\star \\ f_0, f_\star}} \quad & \frac{\|x_1 - x_0\|^2}{\|x_0 - x_\star\|^2} \\ \text{subject to} \quad & \exists f \in \mathcal{F}_{\mu,L} \text{ such that } \begin{cases} f_i = f(x_i) & i = 0, \star \\ g_i = \nabla f(x_i) & i = 0, \star \end{cases} \\ & x_1 = x_0 - \gamma_0 g_0 \\ & g_\star = 0. \end{aligned}$$

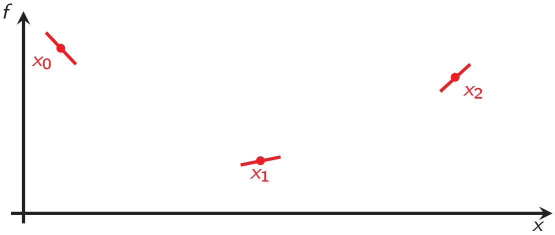
- Variables: x_0, x_1, $x_\star$, g_0, $g_\star$, f_0, $f_\star$.

Smooth strongly convex interpolation (or extension)

Consider an index set S, and its associated values $\{(x_i, g_i, f_i)\}_{i \in S}$ with coordinates x_i, (sub)gradients g_i and function values f_i.

Smooth strongly convex interpolation (or extension)

Consider an index set S, and its associated values $\{(x_i, g_i, f_i)\}_{i \in S}$ with coordinates x_i, (sub)gradients g_i and function values f_i.

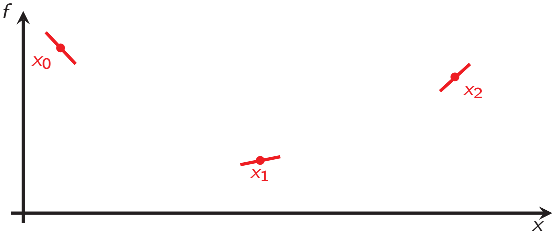


? Possible to find $f \in \mathcal{F}_{\mu,L}$ such that

$$f(x_i) = f_i, \quad \text{and} \quad g_i = \nabla f(x_i), \qquad \forall i \in S.$$

Smooth strongly convex interpolation (or extension)

Consider an index set S, and its associated values $\{(x_i, g_i, f_i)\}_{i\in S}$ with coordinates x_i, (sub)gradients g_i and function values f_i.



? Possible to find $f \in \mathcal{F}_{\mu,L}$ such that

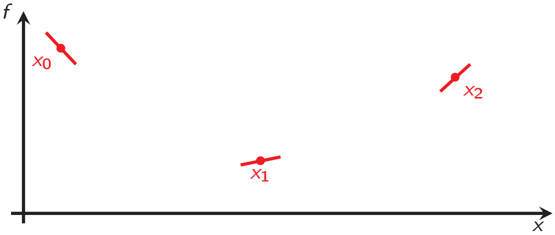
$$f(x_i) = f_i, \quad \text{and} \quad g_i = \nabla f(x_i), \qquad \forall i \in S.$$

- Necessary and sufficient condition: $\forall i, j \in S$

$$f_i \geqslant f_j + \langle g_j, x_i - x_j\rangle + \tfrac{1}{2L}\|g_i - g_j\|^2 + \tfrac{\mu}{2(1-\mu/L)}\|x_i - x_j - \tfrac{1}{L}(g_i - g_j)\|^2.$$

Smooth strongly convex interpolation (or extension)

Consider an index set S, and its associated values $\{(x_i, g_i, f_i)\}_{i\in S}$ with coordinates x_i, (sub)gradients g_i and function values f_i.



? Possible to find $f \in \mathcal{F}_{\mu,L}$ such that

$$f(x_i) = f_i, \quad \text{and} \quad g_i = \nabla f(x_i), \qquad \forall i \in S.$$

- Necessary and sufficient condition: $\forall i, j \in S$

$$f_i \geqslant f_j + \langle g_j, x_i - x_j\rangle + \tfrac{1}{2L}\left\|g_i - g_j\right\|^2 + \tfrac{\mu}{2(1-\mu/L)}\left\|x_i - x_j - \tfrac{1}{L}(g_i - g_j)\right\|^2.$$

- Simpler example: pick $\mu = 0$ and $L = \infty$ (just convexity):

$$f_i \geqslant f_j + \langle g_j, x_i - x_j\rangle.$$

Replace constraints

Replace constraints

◇ Interpolation conditions allow removing red constraints

$$\max_{\substack{x_0, x_1, x_\star \\ g_0, g_\star \\ f_0, f_\star}} \quad \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2}$$

$$\text{subject to} \quad \exists f \in \mathcal{F}_{\mu,L} \text{ such that } \begin{cases} f_i = f(x_i) & i = 0, \star \\ g_i = \nabla f(x_i) & i = 0, \star \end{cases}$$

$$x_1 = x_0 - \gamma_0 g_0$$

$$g_\star = 0,$$

Replace constraints

◇ Interpolation conditions allow removing red constraints

$$\max_{\substack{x_0, x_1, x_\star \\ g_0, g_\star \\ f_0, f_\star}} \quad \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2}$$

$$\text{subject to} \quad \exists f \in \mathcal{F}_{\mu,L} \text{ such that } \begin{cases} f_i = f(x_i) & i = 0, \star \\ g_i = \nabla f(x_i) & i = 0, \star \end{cases}$$

$$x_1 = x_0 - \gamma_0 g_0$$

$$g_\star = 0,$$

◇ replacing them by

$$f_\star \geqslant f_0 + \langle g_0, x_\star - x_0 \rangle + \tfrac{1}{2L}\|g_\star - g_0\|^2 + \tfrac{\mu}{2(1-\mu/L)}\left\|x_\star - x_0 - \tfrac{1}{L}(g_\star - g_0)\right\|^2$$

$$f_0 \geqslant f_\star + \langle g_\star, x_0 - x_\star \rangle + \tfrac{1}{2L}\|g_0 - g_\star\|^2 + \tfrac{\mu}{2(1-\mu/L)}\left\|x_0 - x_\star - \tfrac{1}{L}(g_0 - g_\star)\right\|^2.$$

Replace constraints

◇ Interpolation conditions allow removing red constraints

$$\max_{\substack{x_0, x_1, x_\star \\ g_0, g_\star \\ f_0, f_\star}} \quad \frac{\|x_1 - x_\star\|^2}{\|x_0 - x_\star\|^2}$$

$$\text{subject to} \quad \exists f \in \mathcal{F}_{\mu,L} \text{ such that } \begin{cases} f_i = f(x_i) & i = 0, \star \\ g_i = \nabla f(x_i) & i = 0, \star \end{cases}$$

$$x_1 = x_0 - \gamma_0 g_0$$

$$g_\star = 0,$$

◇ replacing them by

$$f_\star \geqslant f_0 + \langle g_0, x_\star - x_0\rangle + \tfrac{1}{2L}\|g_\star - g_0\|^2 + \tfrac{\mu}{2(1-\mu/L)}\left\|x_\star - x_0 - \tfrac{1}{L}(g_\star - g_0)\right\|^2$$

$$f_0 \geqslant f_\star + \langle g_\star, x_0 - x_\star\rangle + \tfrac{1}{2L}\|g_0 - g_\star\|^2 + \tfrac{\mu}{2(1-\mu/L)}\left\|x_0 - x_\star - \tfrac{1}{L}(g_0 - g_\star)\right\|^2.$$

◇ Same optimal value (no relaxation); but still non-convex quadratic problem.

Semidefinite lifting

Semidefinite lifting

- Using the new variables $G \succcurlyeq 0$ and F

$$G = \begin{bmatrix} \|x_0 - x_\star\|^2 & \langle g_0, x_0 - x_\star \rangle \\ \langle g_0, x_0 - x_\star \rangle & \|g_0\|^2 \end{bmatrix}, \quad F = f_0 - f_\star,$$

Semidefinite lifting

◇ Using the new variables $G \succcurlyeq 0$ and F

$$G = \begin{bmatrix} \|x_0 - x_\star\|^2 & \langle g_0, x_0 - x_\star\rangle \\ \langle g_0, x_0 - x_\star\rangle & \|g_0\|^2 \end{bmatrix}, \quad F = f_0 - f_\star,$$

◇ previous problem can be reformulated as a 2×2 SDP

$$\begin{aligned} \max_{G,\,F} \quad & G_{1,1} + \gamma_0^2 G_{2,2} - 2\gamma_0 G_{1,2} \\ \text{subject to} \quad & F + \tfrac{L\mu}{2(L-\mu)} G_{1,1} + \tfrac{1}{2(L-\mu)} G_{2,2} - \tfrac{L}{L-\mu} G_{1,2} \leqslant 0 \\ & -F + \tfrac{L\mu}{2(L-\mu)} G_{1,1} + \tfrac{1}{2(L-\mu)} G_{2,2} - \tfrac{\mu}{L-\mu} G_{1,2} \leqslant 0 \\ & G_{1,1} = 1 \\ & G \succcurlyeq 0 \end{aligned}$$

Semidefinite lifting

◇ Using the new variables $G \succcurlyeq 0$ and F

$$G = \begin{bmatrix} \|x_0 - x_\star\|^2 & \langle g_0, x_0 - x_\star\rangle \\ \langle g_0, x_0 - x_\star\rangle & \|g_0\|^2 \end{bmatrix}, \quad F = f_0 - f_\star,$$

◇ previous problem can be reformulated as a 2×2 SDP

$$\begin{aligned} \max_{G,\,F} \quad & G_{1,1} + \gamma_0^2 G_{2,2} - 2\gamma_0 G_{1,2} \\ \text{subject to} \quad & F + \tfrac{L\mu}{2(L-\mu)} G_{1,1} + \tfrac{1}{2(L-\mu)} G_{2,2} - \tfrac{L}{L-\mu} G_{1,2} \leqslant 0 \\ & -F + \tfrac{L\mu}{2(L-\mu)} G_{1,1} + \tfrac{1}{2(L-\mu)} G_{2,2} - \tfrac{\mu}{L-\mu} G_{1,2} \leqslant 0 \\ & G_{1,1} = 1 \\ & G \succcurlyeq 0 \end{aligned}$$

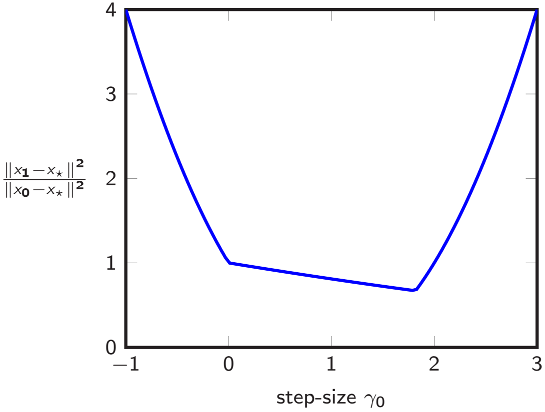
(using an an homogeneity argument and substituting x_1 and $g_\star$).

Solving the SDP...

Fix $L = 1$, $\mu = .1$ and solve the SDP for a few values of γ_0.

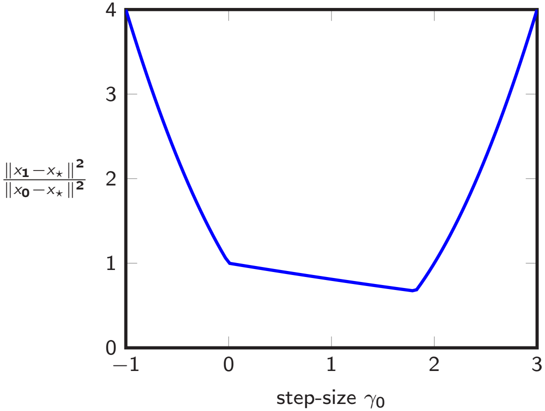
Solving the SDP...

Fix $L = 1$, $\mu = .1$ and solve the SDP for a few values of γ_0.



Solving the SDP...

Fix $L = 1$, $\mu = .1$ and solve the SDP for a few values of γ_0.



⋄ Observation: numerics match $\max\{(1-\gamma_0 L)^2, (1-\gamma_0\mu)^2\}$.

Performance estimation & gradient descent

- ⋄ We can compute for the smallest $\tau(\gamma_0)$ such that

$$\|x_1 - x_\star\|^2 \leqslant \tau(\gamma_0)\|x_0 - x_\star\|^2$$

is satisfied for all $x_0 \in \mathbb{R}^d$, $d \in \mathbb{N}$, $f \in \mathcal{F}_{\mu,L}$, and $x_1 = x_0 - \gamma_0 \nabla f(x_0)$.

Performance estimation & gradient descent

- We can compute for the smallest $\tau(\gamma_0)$ such that

$$\|x_1 - x_\star\|^2 \leqslant \tau(\gamma_0)\|x_0 - x_\star\|^2$$

  is satisfied for all $x_0 \in \mathbb{R}^d$, $d \in \mathbb{N}$, $f \in \mathcal{F}_{\mu,L}$, and $x_1 = x_0 - \gamma_0 \nabla f(x_0)$.

- Feasible points to the previous SDP correspond to lower bounds on $\tau(\gamma_0)$.

Performance estimation & gradient descent

◇ We can compute for the smallest $\tau(\gamma_0)$ such that

$$\|x_1 - x_\star\|^2 \leqslant \tau(\gamma_0)\|x_0 - x_\star\|^2$$

is satisfied for all $x_0 \in \mathbb{R}^d$, $d \in \mathbb{N}$, $f \in \mathcal{F}_{\mu,L}$, and $x_1 = x_0 - \gamma_0 \nabla f(x_0)$.

◇ Feasible points to the previous SDP correspond to lower bounds on $\tau(\gamma_0)$.

◇ Feasible points to dual SDP correspond to upper bounds on $\tau(\gamma_0)$.

Performance estimation & gradient descent

◇ We can compute for the smallest $\tau(\gamma_0)$ such that

$$\|x_1 - x_\star\|^2 \leqslant \tau(\gamma_0)\|x_0 - x_\star\|^2$$

is satisfied for all $x_0 \in \mathbb{R}^d$, $d \in \mathbb{N}$, $f \in \mathcal{F}_{\mu,L}$, and $x_1 = x_0 - \gamma_0 \nabla f(x_0)$.

◇ Feasible points to the previous SDP correspond to lower bounds on $\tau(\gamma_0)$.

◇ Feasible points to dual SDP correspond to upper bounds on $\tau(\gamma_0)$.

◇ Want to know more?

- `https://francisbach.com/computer-aided-analyses/`
- Toolboxes (next slides).

Software


François


Julien


Céline

Baptiste


Aymeric

- ◇ Performance Estimation Toolbox (PESTO) in Matlab, 2017.
- ◇ Performance Estimation in Python (PEPit), 2022.

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

Accelerated Gradient Method

Input: f L-smooth and convex, $x_0 = y_0 \in \mathbb{R}^d$.

For $i = 0 : N - 1$

$$x_{i+1} = y_i - \frac{1}{L} \nabla f(y_i)$$

$$y_{i+1} = x_{i+1} + \frac{i-1}{i+2}(x_{i+1} - x_i)$$

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

Accelerated Gradient Method

Input: f L-smooth and convex, $x_0 = y_0 \in \mathbb{R}^d$.

For $i = 0 : N - 1$

$$x_{i+1} = y_i - \frac{1}{L} \nabla f(y_i)$$
$$y_{i+1} = x_{i+1} + \frac{i-1}{i+2}(x_{i+1} - x_i)$$

What if inexact gradient used instead? Relative inaccuracy model:

$$\|\tilde{d}_f(y_i) - \nabla f(y_i)\| \leqslant \varepsilon \|\nabla f(y_i)\|.$$

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

Accelerated Gradient Method

Input: f L-smooth and convex, $x_0 = y_0 \in \mathbb{R}^d$.

For $i = 0 : N-1$

$$x_{i+1} = y_i - \frac{1}{L}\nabla f(y_i)$$
$$y_{i+1} = x_{i+1} + \frac{i-1}{i+2}(x_{i+1} - x_i)$$

What if inexact gradient used instead? Relative inaccuracy model:

$$\|\tilde{\mathrm{d}}_{\mathrm{f}}(\mathrm{y_i}) - \nabla f(y_i)\| \leqslant \varepsilon \|\nabla f(y_i)\|.$$

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

Accelerated Gradient Method

Input: f L-smooth and convex, $x_0 = y_0 \in \mathbb{R}^d$.

For $i = 0 : N-1$

$$x_{i+1} = y_i - \frac{1}{L}\tilde{\mathrm{d}}_\mathrm{f}(\mathrm{y_i})$$

$$y_{i+1} = x_{i+1} + \frac{i-1}{i+2}(x_{i+1} - x_i)$$

What if inexact gradient used instead? Relative inaccuracy model:

$$\|\tilde{\mathrm{d}}_\mathrm{f}(\mathrm{y_i}) - \nabla f(y_i)\| \leqslant \varepsilon \|\nabla f(y_i)\|.$$

PESTO example: an inexact accelerated gradient method

Minimize L-smooth convex function $f(x)$:

$$\min_{x \in \mathbb{R}^d} f(x).$$

Accelerated Gradient Method
Input: f L-smooth and convex, $x_0 = y_0 \in \mathbb{R}^d$.   For $i = 0 : N-1$ $$x_{i+1} = y_i - \frac{1}{L}\tilde{\mathrm{d}}_{\mathrm{f}}(\mathrm{y_i})$$ $$y_{i+1} = x_{i+1} + \frac{i-1}{i+2}(x_{i+1} - x_i)$$

What if inexact gradient used instead? Relative inaccuracy model:

$$\|\tilde{\mathrm{d}}_{\mathrm{f}}(\mathrm{y_i}) - \nabla f(y_i)\| \leqslant \varepsilon \|\nabla f(y_i)\|.$$

What guarantees of type

$$\frac{f(x_N) - f_\star}{\|x_0 - x_\star\|^2} \leqslant \tau(N, L)?$$

Next slide: compute $\tau(N, L)$ numerically using SDP.

PESTO example: an inexact accelerated gradient method

```
% (0) Initialize an empty PEP
P = pep();

% (1) Set up the objective function
param.mu = 0;      % strong convexity parameter
param.L  = 1;      % Smoothness parameter

F=P.DeclareFunction('SmoothStronglyConvex',param); % F is the objective function

% (2) Set up the starting point and initial condition
x0       = P.StartingPoint();        % x0 is some starting point
[xs, fs] = F.OptimalPoint();         % xs is an optimal point, and fs=F(xs)
P.InitialCondition((x0-xs)^2 <= 1);  % Add an initial condition ||x0-xs||^2<= 1

% (3) Algorithm
N = 7; % number of iterations

x    = cell(N+1,1); % we store the iterates in a cell for convenience
x{1} = x0;
y    = x0;
eps  = .1;
for i = 1:N
    d      = inexactsubgradient(y, F, eps);
    x{i+1} = y - 1/param.L * d;
    y      = x{i+1} + (i-1)/(i+2) * (x{i+1} - x{i});
end

% (4) Set up the performance measure
[g, f] = F.oracle(x{N+1});    % g=grad F(x), f=F(x)
P.PerformanceMetric(f - fs); % Worst-case evaluated as F(x)-F(xs)

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double(f - fs)   % worst-case objective function accuracy
```

PESTO example: an inexact accelerated gradient method

```
% (0) Initialize an empty PEP
P = pep();

% (1) Set up the objective function
param.mu = 0;      % strong convexity parameter
param.L  = 1;      % Smoothness parameter

F=P.DeclareFunction('SmoothStronglyConvex',param); % F is the objective function

% (2) Set up the starting point and initial condition
x0       = P.StartingPoint();        % x0 is some starting point
[xs, fs] = F.OptimalPoint();         % xs is an optimal point, and fs=F(xs)
```

```
x{1} = x0;
y    = x0;
eps  = .1;
for i = 1:N
    d       = inexactsubgradient(y, F, eps);
    x{i+1} = y - 1/param.L * d;
    y       = x{i+1} + (i-1)/(i+2) * (x{i+1} - x{i});
end
```

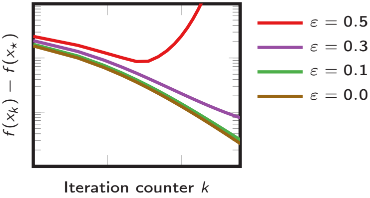
```
    y      = x{i+1} + (i-1)/(i+2) * (x{i+1} - x{i});
end

% (4) Set up the performance measure
[g, f] = F.oracle(x{N+1});    % g=grad F(x), f=F(x)
P.PerformanceMetric(f - fs); % Worst-case evaluated as F(x)-F(xs)

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double(f - fs)   % worst-case objective function accuracy
```

PESTO example: an inexact accelerated gradient method

```
% (0) Initialize an empty PEP
P = pep();

% (1) Set up the objective function
param.mu = 0;      % strong convexity parameter
param.L  = 1;      % Smoothness parameter

F=P.DeclareFunction('SmoothStronglyConvex',param); % F is the objective function

% (2) Set up the starting point and initial condition
x0       = P.StartingPoint();        % x0 is some starting point
[xs, fs] = F.OptimalPoint();         % xs is an optimal point, and fs=F(xs)
```

```
x{1} = x0;
y    = x0;
eps  = .1;
for i = 1:N
    d       = inexactsubgradient(y, F, eps);
    x{i+1} = y - 1/param.L * d;
    y       = x{i+1} + (i-1)/(i+2) * (x{i+1} - x{i});
end
```

```
    y       = x{i+1} + (i-1)/(i+2) * (x{i+1} - x{i});
end

% (4) Set up the performance measure
[g, f] = F.oracle(x{N+1});    % g=grad F(x), f=F(x)
P.PerformanceMetric(f - fs); % Worst-case evaluated as F(x)-F(xs)

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double(f - fs)   % worst-case objective function accuracy
```



PESTO example: Douglas-Rachford splitting

Minimize sum of two convex (cpp) functions

$$\min_{x \in \mathbb{R}^d} f(x) + h(x).$$

Douglas-Rachford Splitting

Input: f, h convex (cpp) functions, $w_0 \in \mathbb{R}^d$.

For $i = 0 : N-1$

$$\begin{aligned}
x_{i+1} &= \mathrm{argmin}_{x \in \mathbb{R}^d} \{\gamma h(x) + \tfrac{1}{2}\|x - w_i\|^2\} \\
y_{i+1} &= \mathrm{argmin}_{y \in \mathbb{R}^d} \{\gamma f(y) + \tfrac{1}{2}\|y - 2x_{i+1} + w_i\|^2\} \\
w_{i+1} &= w_i + \tfrac{1}{2}(y_{i+1} - x_{i+1}).
\end{aligned}$$

Next slide: compute convergence rates when f is strongly convex and h is smooth.

PESTO example: Douglas-Rachford splitting

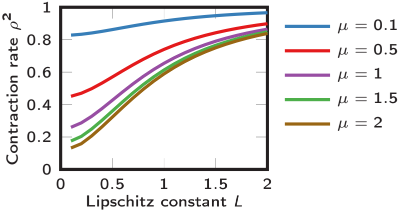
```matlab
% (0) Initialize an empty PEP
P=pep();

N = 1;
% (1) Set up the class of monotone inclusions
paramA.L  =  1; paramA.mu = 0; % A is 1-Lipschitz and 0-strongly monotone
paramB.mu = .1;                % B is .1-strongly monotone

A = P.DeclareFunction('LipschitzStronglyMonotone',paramA);
B = P.DeclareFunction('StronglyMonotone',paramB);

w  = cell(N+1,1);   wp = cell(N+1,1);
x  = cell(N,1);     xp = cell(N,1);
y  = cell(N,1);     yp = cell(N,1);

% (2) Set up the starting points
w{1}    = P.StartingPoint(); wp{1}    = P.StartingPoint();
P.InitialCondition((w{1}-wp{1})^2<=1);

% (3) Algorithm
lambda = 1.3;       % step size (in the resolvents)
theta  = .9;         % overrelaxation

for k = 1 : N
    x{k}    = proximal_step(w{k},B,lambda);
    y{k}    = proximal_step(2*x{k}-w{k},A,lambda);
    w{k+1}  = w{k}-theta*(x{k}-y{k});

    xp{k}   = proximal_step(wp{k},B,lambda);
    yp{k}   = proximal_step(2*xp{k}-wp{k},A,lambda);
    wp{k+1}   = wp{k}-theta*(xp{k}-yp{k});
end

% (4) Set up the performance measure: ||z0-z1||^2
P.PerformanceMetric((w{k+1}-wp{k+1})^2);

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double((w{k+1}-wp{k+1})^2)   % worst-case contraction factor
```

PESTO example: Douglas-Rachford splitting

```
% (0) Initialize an empty PEP
P=pep();

N = 1;
% (1) Set up the class of monotone inclusions
paramA.L  =  1; paramA.mu = 0; % A is 1-Lipschitz and 0-strongly monotone
paramB.mu = .1;                % B is .1-strongly monotone

A = P.DeclareFunction('LipschitzStronglyMonotone',paramA);
B = P.DeclareFunction('StronglyMonotone',paramB);

w  = cell(N+1,1);   wp = cell(N+1,1);
x  = cell(N,1);     xp = cell(N,1);
y  = cell(N,1);     yp = cell(N,1);

% (2) Set up the starting points
w{1}    = P.StartingPoint(); wp{1}    = P.StartingPoint();
P.InitialCondition((w{1}-wp{1})^2<=1);

% (3) Algorithm
lambda = 1.3;       % step size (in the resolvents)
theta  = .9;         % overrelaxation

x{k}     = proximal_step(w{k},B,lambda);
y{k}     = proximal_step(2*x{k}-w{k},A,lambda);
w{k+1}   = w{k}-theta*(x{k}-y{k});
    xp{k}    = proximal_step(wp{k},B,lambda);
    yp{k}    = proximal_step(2*xp{k}-wp{k},A,lambda);
    wp{k+1}    = wp{k}-theta*(xp{k}-yp{k});
end

% (4) Set up the performance measure: ||z0-z1||^2
P.PerformanceMetric((w{k+1}-wp{k+1})^2);

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double((w{k+1}-wp{k+1})^2)   % worst-case contraction factor
```

PESTO example: Douglas-Rachford splitting

```
% (0) Initialize an empty PEP
P=pep();

N = 1;
% (1) Set up the class of monotone inclusions
paramA.L  =  1; paramA.mu = 0; % A is 1-Lipschitz and 0-strongly monotone
paramB.mu = .1;                % B is .1-strongly monotone

A = P.DeclareFunction('LipschitzStronglyMonotone',paramA);
B = P.DeclareFunction('StronglyMonotone',paramB);

w  = cell(N+1,1);   wp = cell(N+1,1);
x  = cell(N,1);     xp = cell(N,1);
y  = cell(N,1);     yp = cell(N,1);

% (2) Set up the starting points
w{1}    = P.StartingPoint(); wp{1}    = P.StartingPoint();
P.InitialCondition((w{1}-wp{1})^2<=1);

% (3) Algorithm
lambda = 1.3;       % step size (in the resolvents)
theta  = .9;         % overrelaxation

x{k}     = proximal_step(w{k},B,lambda);
y{k}     = proximal_step(2*x{k}-w{k},A,lambda);
w{k+1}   = w{k}-theta*(x{k}-y{k});
    xp{k}    = proximal_step(wp{k},B,lambda);
    yp{k}    = proximal_step(2*xp{k}-wp{k},A,lambda);
    wp{k+1}    = wp{k}-theta*(xp{k}-yp{k});
end

% (4) Set up the performance measure: ||z0-z1||^2
P.PerformanceMetric((w{k+1}-wp{k+1})^2);

% (5) Solve the PEP
P.solve()

% (6) Evaluate the output
double((w{k+1}-wp{k+1})^2)   % worst-case contraction factor
```



How general is this?

Includes... but not limited to

How general is this?

Includes... but not limited to

- ◇ subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,

How general is this?

Includes... but not limited to

- ◇ subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,
- ◇ projected and proximal variants, accelerated/momentum versions,
- ◇ steepest descent, greedy/conjugate gradient methods,
- ◇ Frank-Wolfe/conditional gradients,

How general is this?

Includes... but not limited to

- ◇ subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,
- ◇ projected and proximal variants, accelerated/momentum versions,
- ◇ steepest descent, greedy/conjugate gradient methods,
- ◇ Frank-Wolfe/conditional gradients,
- ◇ Douglas-Rachford (ADMM), other operator splitting schemes,
- ◇ Krasnoselskii-Mann and Halpern fixed-point iterations,

How general is this?

Includes... but not limited to

- ◇ subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,
- ◇ projected and proximal variants, accelerated/momentum versions,
- ◇ steepest descent, greedy/conjugate gradient methods,
- ◇ Frank-Wolfe/conditional gradients,
- ◇ Douglas-Rachford (ADMM), other operator splitting schemes,
- ◇ Krasnoselskii-Mann and Halpern fixed-point iterations,
- ◇ inexact versions of all the above,
- ◇ stochastic versions: SGD, SAG, SAGA and variants.

How general is this?

Includes... but not limited to

- ◇ subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,
- ◇ projected and proximal variants, accelerated/momentum versions,
- ◇ steepest descent, greedy/conjugate gradient methods,
- ◇ Frank-Wolfe/conditional gradients,
- ◇ Douglas-Rachford (ADMM), other operator splitting schemes,
- ◇ Krasnoselskii-Mann and Halpern fixed-point iterations,
- ◇ inexact versions of all the above,
- ◇ stochastic versions: SGD, SAG, SAGA and variants.

Toolboxes contain most of the recent PEP-related advances (including techniques by other groups) available. Clean updated references & examples in user manual.

How general is this?

Includes... but not limited to

- subgradient, gradient, heavy-ball, fast gradient, optimized gradient methods,
- projected and proximal variants, accelerated/momentum versions,
- steepest descent, greedy/conjugate gradient methods,
- Frank-Wolfe/conditional gradients,
- Douglas-Rachford (ADMM), other operator splitting schemes,
- Krasnoselskii-Mann and Halpern fixed-point iterations,
- inexact versions of all the above,
- stochastic versions: SGD, SAG, SAGA and variants.

Toolboxes contain most of the recent PEP-related advances (including techniques by other groups) available. Clean updated references & examples in user manual.

First ideas in this line of research coined by Drori and Teboulle (2014).

PEPit: Performance Estimation in Python

Tests passing | codecov 90% | docs passing | pypi package 0.2.1 | downloads 12k | license MIT

This open source Python library provides a generic way to use PEP framework in Python. Performance estimation problems were introduced in 2014 by **Yoel Drori** and **Marc Teboulle**, see [1]. PEPit is mainly based on the formalism and developments from [2, 3] by a subset of the authors of this toolbox. A friendly informal introduction to this formalism is available in this blog post and a corresponding Matlab library is presented in [4] (PESTO).

Website and documentation of PEPit: https://pepit.readthedocs.io/

Source Code (MIT): https://github.com/PerformanceEstimation/PEPit

Using and citing the toolbox

This code comes jointly with the following `reference`:

```
B. Goujaud, C. Moucer, F. Glineur, J. Hendrickx, A. Taylor, A. Dieuleveut (2022).
"PEPit: computer-assisted worst-case analyses of first-order optimization methods in Python."
```

When using the toolbox in a project, please refer to this note via this Bibtex entry:

```
@article{pepit2022,
  title={{PEPit}: computer-assisted worst-case analyses of first-order optimization methods in {P}yt
  author={Goujaud, Baptiste and Moucer, C\'eline and Glineur, Fran\c{c}ois and Hendrickx, Julien and
  journal={arXiv preprint arXiv:2201.04040},
  year={2022}
}
```

About

PEPit is a package enabling computer-assisted worst-case analyses of first-order optimization methods.

pepit.readthedocs.io/en/latest/

python | optimization | semidefinite-programming | worst-case-analyses | first-order-methods | performance-estimation-problems

Readme

MIT license

Activity

68 stars

4 watching

6 forks

Report repository

Releases 5

0.2.1 Latest
on Dec 17, 2022

+ 4 releases

Important inspiration & reference:

- ⋄ Drori, and Teboulle ('14). "Performance of first-order methods for smooth convex minimization: a novel approach."

Important inspiration & reference:

◇ Drori, and Teboulle ('14). "Performance of first-order methods for smooth convex minimization: a novel approach."

Second part of the presentation:

◇ T., Hendrickx, Glineur ('17). "Smooth strongly convex interpolation and exact worst-case performance of first-order methods."

◇ T., Hendrickx, Glineur ('17). "Exact worst-case performance of first-order methods for composite convex optimization."

◇ T., Hendrickx, Glineur ('17). "Performance estimation toolbox (PESTO): Automated worst-case analysis of first-order optimization methods."

◇ Goujaud, Moucer, et al. ('22). "PEPit: computer-assisted worst-case analyses of first-order optimization methods in Python."

Important inspiration & reference:

- ⋄ Drori, and Teboulle ('14). "Performance of first-order methods for smooth convex minimization: a novel approach."

Second part of the presentation:

- ⋄ T., Hendrickx, Glineur ('17). "Smooth strongly convex interpolation and exact worst-case performance of first-order methods."
- ⋄ T., Hendrickx, Glineur ('17). "Exact worst-case performance of first-order methods for composite convex optimization."
- ⋄ T., Hendrickx, Glineur ('17). "Performance estimation toolbox (PESTO): Automated worst-case analysis of first-order optimization methods."
- ⋄ Goujaud, Moucer, et al. ('22). "PEPit: computer-assisted worst-case analyses of first-order optimization methods in Python."

Designing algorithms with PEPs:

- ⋄ Drori, T ('20). "Efficient first-order methods for convex minimization: a constructive approach."
- ⋄ Drori, T ('22). "On the oracle complexity of smooth strongly convex minimization."
- ⋄ T, Drori ('23). "An optimal gradient method for smooth strongly convex minimization."

Informal introduction: `https://francisbach.com/computer-aided-analyses/`.

Concluding remarks

Optimization algorithms: currently a wild jungle.

- ◇ still: certain guiding principles & main driving algorithms,
- ◇ guarantees $\Rightarrow$ trust and black box,
- ◇ ... but somewhat behind schedule.

Concluding remarks

Optimization algorithms: currently a wild jungle.

- ◇ still: certain guiding principles & main driving algorithms,
- ◇ guarantees $\Rightarrow$ trust and black box,
- ◇ ... but somewhat behind schedule.

Performance estimation's philosophy

Concluding remarks

Optimization algorithms: currently a wild jungle.

- ◇ still: certain guiding principles & main driving algorithms,
- ◇ guarantees $\Rightarrow$ trust and black box,
- ◇ ... but somewhat behind schedule.

Performance estimation's philosophy

- ◇ numerically allows obtaining tight bounds (rigorous baselines),
  - fast prototyping
  - worth checking before trying to prove a method works.

Concluding remarks

Optimization algorithms: currently a wild jungle.

- ◇ still: certain guiding principles & main driving algorithms,
- ◇ guarantees ⇒ trust and black box,
- ◇ ... but somewhat behind schedule.

Performance estimation's philosophy

- ◇ numerically allows obtaining tight bounds (rigorous baselines),
  - – fast prototyping
  - – worth checking before trying to prove a method works.
- ◇ algebraic insights into proofs: principled approach.

Concluding remarks

Optimization algorithms: currently a wild jungle.

- ◇ still: certain guiding principles & main driving algorithms,
- ◇ guarantees $\Rightarrow$ trust and black box,
- ◇ ... but somewhat behind schedule.

Performance estimation's philosophy

- ◇ numerically allows obtaining tight bounds (rigorous baselines),
  - – fast prototyping
  - – worth checking before trying to prove a method works.
- ◇ algebraic insights into proofs: principled approach.
- ◇ validation & benchmark tool for proofs (also for reviews ☺).

Thanks! Questions?

PerformanceEstimation/Performance-Estimation-Toolbox on Github

PerformanceEstimation/PEPit on Github