Lecture 12: Linearly constrained nonlinear optimization

Michael Patriksson

24 February 2005

Feasible-direction methods

- Consider the problem to find

$$f^* = \text{minimum } f(\boldsymbol{x}), \tag{1a}$$

$$\text{subject to } \boldsymbol{x} \in X, \tag{1b}$$

  $X \subseteq \mathbb{R}^n$ nonempty, closed and convex; $f : \mathbb{R}^n \to \mathbb{R}$ is C^1 on X

- Most methods for (1) manipulate the constraints defining X; in some cases even such that the sequence $\{\boldsymbol{x}_k\}$ is infeasible until convergence. Why?

- Consider a constraint “$g_i(\boldsymbol{x}) \leq b_i$,” where g_i is nonlinear

- Checking whether $\boldsymbol{p}$ is a feasible direction at $\boldsymbol{x}$, or what the maximum feasible step from $\boldsymbol{x}$ in the direction of $\boldsymbol{p}$ is, is very difficult

- For which step length $\alpha > 0$ does it happen that $g_i(\boldsymbol{x} + \alpha\boldsymbol{p}) = b_i$? This is a nonlinear equation in α!

- Assuming that X is polyhedral, these problems are not present

Feasible-direction descent methods

Step 0. Determine a *starting point* $\boldsymbol{x}_0 \in \mathbb{R}^n$ such that $\boldsymbol{x}_0 \in X$. Set $k := 0$.

Step 1. Determine a *search direction* $\boldsymbol{p}_k \in \mathbb{R}^n$ such that $\boldsymbol{p}_k$ is a feasible descent direction.

Step 2. Determine a *step length* $\alpha_k > 0$ such that $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) < f(\boldsymbol{x}_k)$ and $\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \in X$.

Step 3. Let $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$.

Step 4. If a *termination criterion* is fulfilled, then stop! Otherwise, let $k := k + 1$ and go to Step 1.

Notes

- Similar form as the general method for unconstrained optimization
- Just as *local* as methods for unconstrained optimization
- Search directions typically based on the approximation of f—a relaxation
- Search direction often of the form $\boldsymbol{p}_k = \boldsymbol{y}_k - \boldsymbol{x}_k$, where $\boldsymbol{y}_k \in X$ solves the approximate problem
- Line searches similar; note the maximum step
- Termination criteria and descent based on first-order optimality and/or fixed-point theory ($\boldsymbol{p}_k \approx \mathbf{0}^n$)

LP-based algorithm, I: The Frank–Wolfe method

- The Frank–Wolfe method is based on a first-order approximation of f around the iterate $\boldsymbol{x}_k$. This means that the relaxed problems are LPs, which can then be solved by using the Simplex method.

- Remember the first-order optimality condition: *If $\boldsymbol{x}^* \in X$ is a local minimum of f on X then*

$$\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0, \qquad \boldsymbol{x} \in X,$$

holds.

- Remember also the following equivalent statement:

$$\underset{\boldsymbol{x} \in X}{\text{minimum}} \; \nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) = 0$$

- Follows that if, given an iterate $\boldsymbol{x}_k \in X$,

$$\operatorname*{minimum}_{\boldsymbol{y} \in X} \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_k) < 0,$$

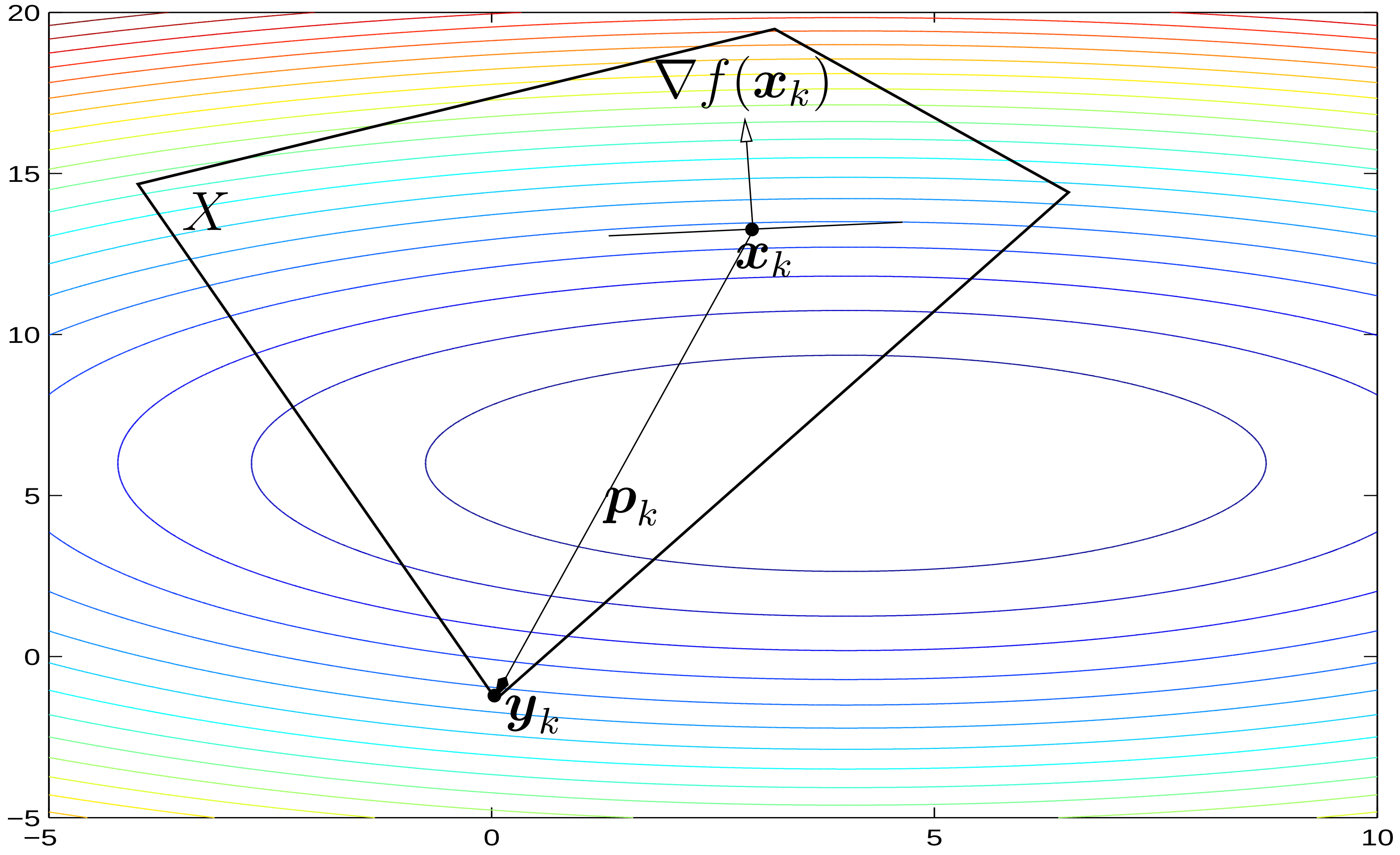
  and $\boldsymbol{y}_k$ is a solution to this LP problem, then the direction of $\boldsymbol{p}_k := \boldsymbol{y}_k - \boldsymbol{x}_k$ is a feasible descent direction with respect to f at $\boldsymbol{x}$.

- Search direction towards an extreme point [one that is optimal in the LP over X with costs $\boldsymbol{c} = \nabla f(\boldsymbol{x}_k)$]

- This is the basis of the Frank–Wolfe algorithm

- We assume that X is bounded in order to ensure that the LP always has a finite solution. The algorithm can be extended to allow for unbounded polyhedra.
- The search directions then are either towards an extreme point (finite solution to LP) or in the direction of an extreme ray of X (unbounded solution to LP).
- Both cases identified in the Simplex method

The search-direction problem



Algorithm description, Frank–Wolfe

Step 0. Find $\boldsymbol{x}_0 \in X$ (for example any extreme point in X). Set $k := 0$.

Step 1. Find a solution $\boldsymbol{y}_k$ to the problem to

$$\underset{\boldsymbol{y} \in X}{\text{minimize}}\ z_k(\boldsymbol{y}) := \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_k). \tag{2}$$

Let $\boldsymbol{p}_k := \boldsymbol{y}_k - \boldsymbol{x}_k$ be the search direction.

Step 2. Approximately solve the problem to minimize $f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k)$ over $\alpha \in [0, 1]$. Let α_k be the step length.

Step 3. Let $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$.

Step 4. If, for example, $z_k(\boldsymbol{y}_k)$ or α_k is close to zero, then terminate! Otherwise, let $k := k + 1$ and go to Step 1.

Convergence

- *Suppose* $X \subset \mathbb{R}^n$ *nonempty polytope;* f *in* C^1 *on* X
- *In Step 2 of the Frank–Wolfe algorithm, we either use an exact line search or the Armijo step length rule.*
- *Then: the sequence* $\{\boldsymbol{x}_k\}$ *is bounded and every limit point (at least one exists) is stationary;*
- $\{f(\boldsymbol{x}_k)\}$ *is descending, and therefore has a limit;*
- *the sequence* $\{z_k(\boldsymbol{y}_k)\} \to 0$.
- *If* f *is convex on* X*, then every limit point is globally optimal.*
- *Proof:*

The convex case: Lower bounds

- Remember the following characterization of convex functions in C^1 on X: *f is convex on X* $\Longleftrightarrow$

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}), \quad \boldsymbol{x}, \boldsymbol{y} \in X$$

- Suppose f is convex on X. Then, $f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k) \leq f^*$ (lower bound, LBD), and $f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k) = f^*$ if and only if $\boldsymbol{x}_k$ is globally optimal. *A relaxation—cf. the Relaxation Theorem!*

- Utilize the lower bound as follows: we know that $f^* \in [f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k), f(\boldsymbol{x}_k)]$. Store the best LBD, and check in Step 4 whether $[f(\boldsymbol{x}_k) - \text{LBD}]/|\text{LBD}|$ is small, and if so terminate.

Notes

- Frank–Wolfe uses linear approximations—works best for almost linear problems
- For highly nonlinear problems, the approximation is bad—the optimal solution may be far from an extreme point. (Compare Steepest descent!)
- In order to find a near-optimum requires many iterations—the algorithm is slow.
- Another reason is that the information generated (the extreme points) are forgotten. If we keep the linear subproblems, we can do much better by storing and utilizing this information.

LP-based algorithm, II: Simplicial decomposition

- Remember the Representation Theorem (special case for polytopes): *Let* $P = \{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b};\ \boldsymbol{x} \geq \mathbf{0}^n \}$, *be nonempty and bounded, and* $V = \{\boldsymbol{v}^1, \dots, \boldsymbol{v}^K\}$ *be the set of extreme points of* P*. Every* $\boldsymbol{x} \in P$ *can be represented as a convex combination of the points in* V*, that is,*

$$\boldsymbol{x} = \sum_{i=1}^{K} \alpha_i \boldsymbol{v}^i,$$

for some $\alpha_1, \dots, \alpha_k \geq 0$ *such that* $\sum_{i=1}^{K} \alpha_i = 1$.

- The idea behind the Simplicial decomposition method is to generate the extreme points $\boldsymbol{v}^i$ which can be used to describe an optimal solution $\boldsymbol{x}^*$, that is, the vectors $\boldsymbol{v}^i$ with positive weights α_i in

$$\boldsymbol{x}^* = \sum_{i=1}^{K} \alpha_i \boldsymbol{v}^i.$$

- The process is still iterative: we generate a “working set” $\mathcal{P}_k$ of indices i, optimize the function f over the convex hull of the known points, and check for stationarity and/or generate a new extreme point.

Algorithm description, Simplicial decomposition

Step 0. Find $\boldsymbol{x}_0 \in X$, for example any extreme point in X. Set $k := 0$. Let $\mathcal{P}_0 := \emptyset$.

Step 1. Let $\boldsymbol{y}_k$ be a solution to the LP problem (2). Let $\mathcal{P}_{k+1} := \mathcal{P}_k \cup \{k\}$.

Step 2. Let $(\mu_k, \boldsymbol{\nu}_{k+1})$ be an approximate solution to the *restricted master problem* (RMP) to

$$\begin{aligned} \underset{(\mu,\boldsymbol{\nu})}{\text{minimize}} \quad & f\left(\mu \boldsymbol{x}_k + \sum_{i \in \mathcal{P}_{k+1}} \nu_i \boldsymbol{y}^i\right), && \text{(3a)} \\ \text{subject to} \quad & \mu + \sum_{i \in \mathcal{P}_{k+1}} \nu_i = 1, && \text{(3b)} \\ & \mu, \nu_i \geq 0, \qquad i \in \mathcal{P}_{k+1}. && \text{(3c)} \end{aligned}$$

Step 3. Let $\boldsymbol{x}_{k+1} := \mu_{k+1}\boldsymbol{x}_k + \sum_{i \in \mathcal{P}_{k+1}} (\boldsymbol{\nu}_{k+1})_i \boldsymbol{y}^i$.

Step 4. If, for example, $z_k(\boldsymbol{y}_k)$ is close to zero, or if $\mathcal{P}_{k+1} = \mathcal{P}_k$, then terminate! Otherwise, let $k := k + 1$ and go to Step 1.

- This basic algorithm keeps all information generated, and adds one new extreme point in every iteration
- An alternative is to drop columns (vectors $\boldsymbol{y}^i$) that have received a zero weight, or to keep only a maximum number of vectors. (Stated in the book.)
- Special case: maximum number of vectors kept $= 1 \Longrightarrow$ the Frank–Wolfe algorithm!
- We obviously improve the Frank–Wolfe algorithm by utilizing more information

Convergence

- It does at least as well as the Frank–Wolfe algorithm: line segment $[\boldsymbol{x}_k, \boldsymbol{y}_k]$ feasible in RMP
- Convergence finite if the (RMPs) are solved exactly, and the maximum number of vectors kept is at least as many as are needed to span $\boldsymbol{x}^*$
- Much more efficient than the Frank–Wolfe algorithm in practice (cf. the above FW example!)
- We can solve the RMPs efficiently, since the constraints are simple

An illustration of FW vs. SD

- A large-scale nonlinear network flow problem which is used to estimate traffic flows in cities. New Section 4.6.3!
- The model is over the small city of Sioux Falls in North Dakota, USA, whose representation has 24 nodes, 76 links, and 528 pairs of origin and destination.
- Three algorithms for the RMPs were tested—a Newton method and two gradient projection methods (see the next section). A MATLAB implementation.
- Remarkable difference—The Frank–Wolfe method suffers from very small steps being taken.

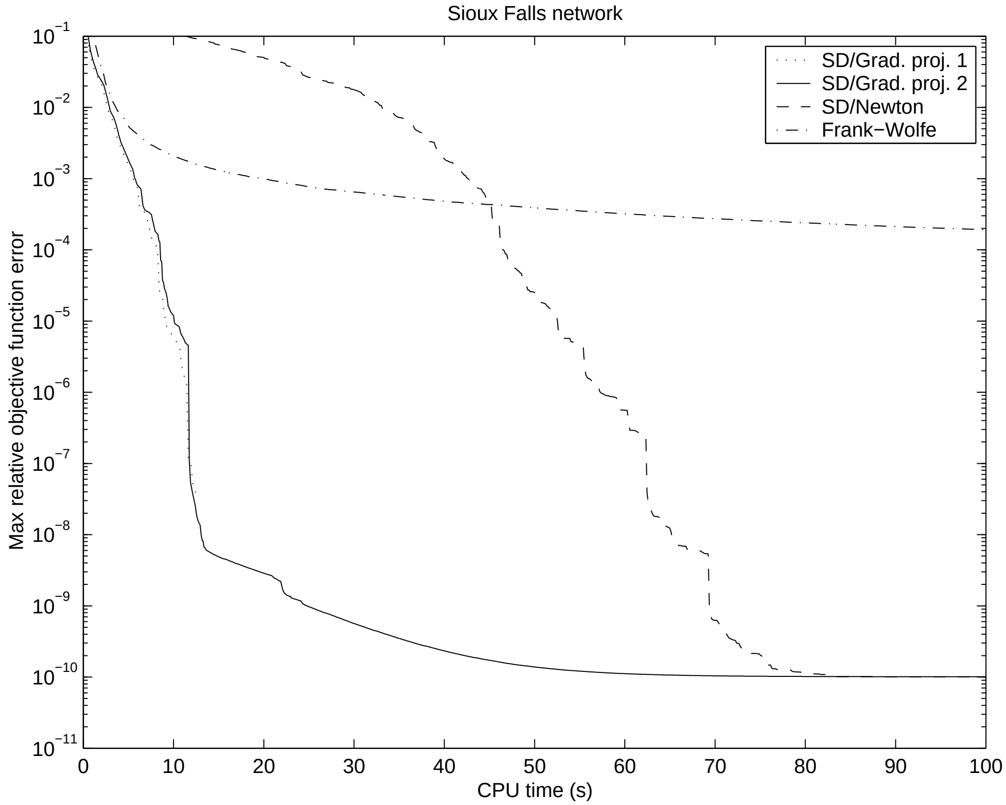
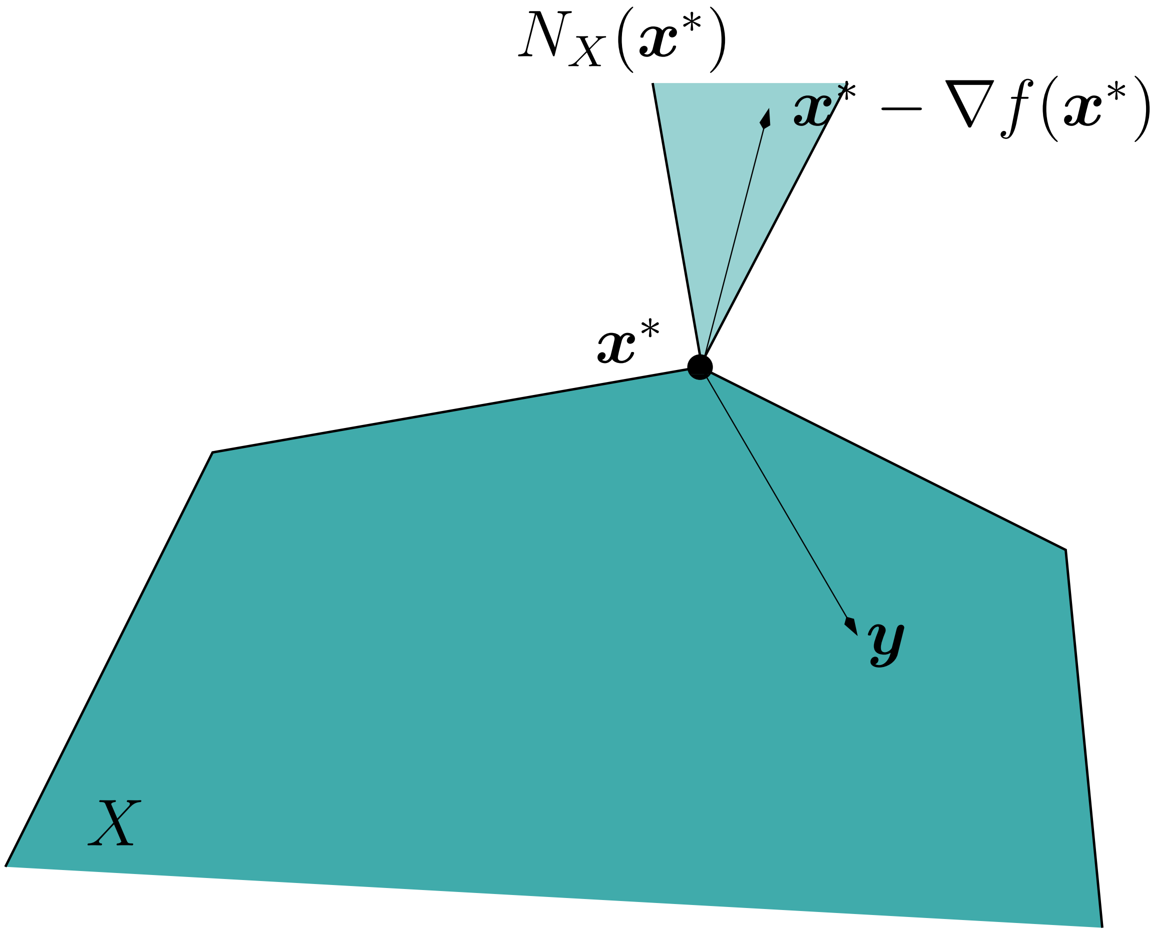


Figure 1: The performance of SD vs. FW on the Sioux Falls network

QP-based algorithm: The gradient projection algorithm

- The gradient projection algorithm is based on the projection characterization of a stationary point: $\boldsymbol{x}^* \in X$ *is a stationary point if and only if, for any* $\alpha > 0$,
$$\boldsymbol{x}^* = \mathrm{Proj}_X[\boldsymbol{x}^* - \alpha \nabla f(\boldsymbol{x}^*)]$$

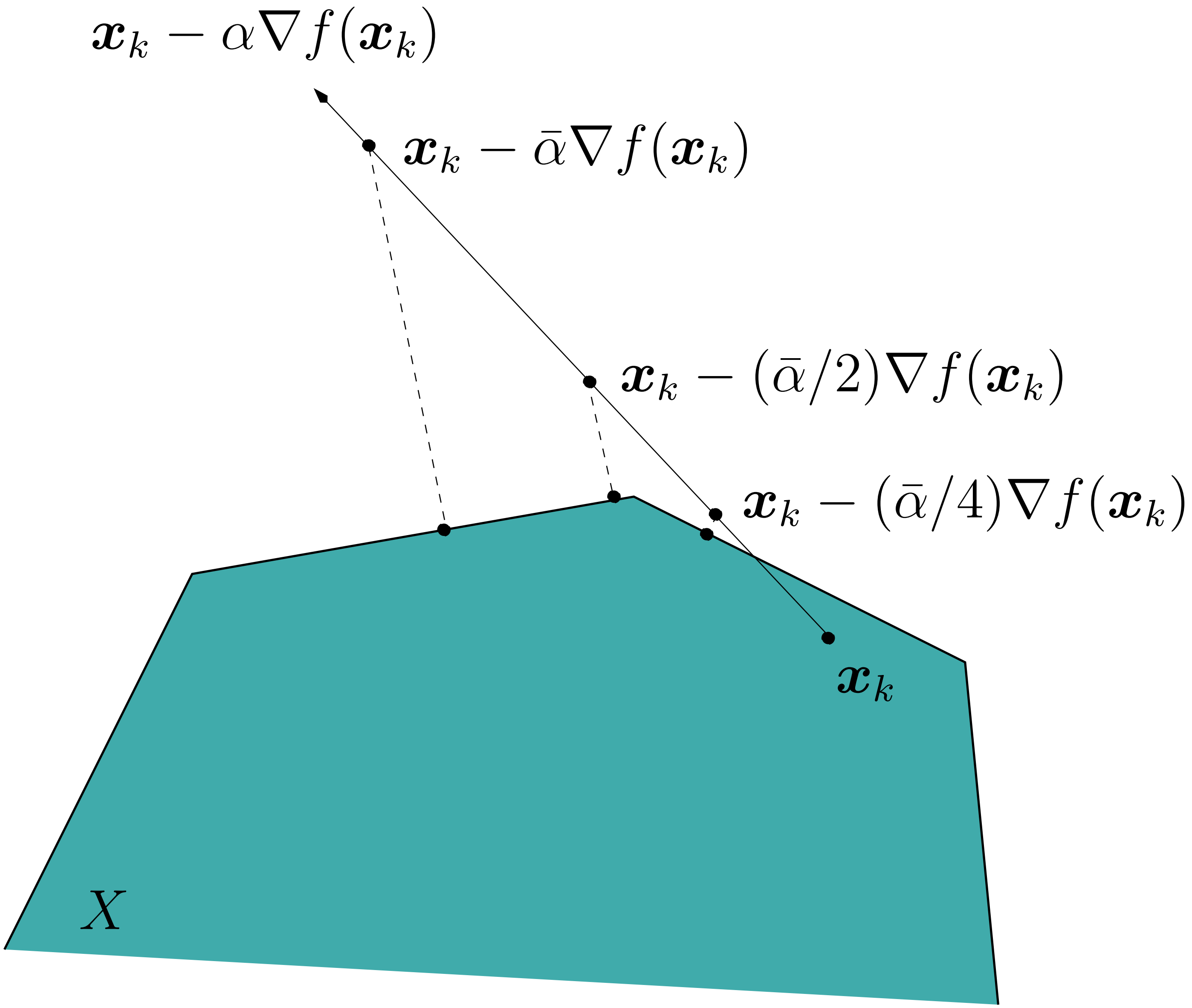

$N_X(\boldsymbol{x}^*)$
$\boldsymbol{x}^* - \nabla f(\boldsymbol{x}^*)$
$\boldsymbol{x}^*$
$\boldsymbol{y}$
X

- Let $\boldsymbol{p} := \operatorname{Proj}_X[\boldsymbol{x} - \alpha\nabla f(\boldsymbol{x})] - \boldsymbol{x}$, for any $\alpha > 0$. Then, if and only if $\boldsymbol{x}$ is non-stationary, $\boldsymbol{p}$ is a feasible descent direction of f at $\boldsymbol{x}$

- The gradient projection algorithm is normally stated such that the line search is done over the *projection arc*, that is, we find a step length α_k for which

$$\boldsymbol{x}_{k+1} := \operatorname{Proj}_X[\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)], \qquad k = 1, \dots . \tag{4}$$

has a good objective value. Use the Armijo rule to determine α_k:


$\boldsymbol{x}_k - \alpha\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - \bar{\alpha}\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - (\bar{\alpha}/2)\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - (\bar{\alpha}/4)\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k$
X

Convergence, I

- $X \subseteq \mathbb{R}^n$ *nonempty, closed, convex;* $f \in C^1$ *on* X;
- *for the starting point* $\boldsymbol{x}_0 \in X$ *it holds that the level set* $\operatorname{lev}_f(f(\boldsymbol{x}_0))$ *intersected with* X *is bounded.*
- *In the algorithm* (4), *the step length* α_k *is given by the Armijo step length rule along the projection arc.*
- *Then: the sequence* $\{\boldsymbol{x}_k\}$ *is bounded;*
- *every limit point of* $\{\boldsymbol{x}_k\}$ *is stationary;*
- $\{f(\boldsymbol{x}_k)\}$ *descending, lower bounded, hence convergent.*
- Gradient projection becomes steepest descent with Armijo line search when $X = \mathbb{R}^n$!
- Convergence arguments similar to steepest descent one

Convergence, II

- *$X \subseteq \mathbb{R}^n$ nonempty, closed, convex;*
- *$f \in C^1$ on X; f convex;*
- *an optimal solution $\boldsymbol{x}^*$ exists.*
- *In the algorithm (4), the step length α_k is given by the Armijo step length rule along the projection arc.*
- *Then: the sequence $\{\boldsymbol{x}_k\}$ converges to an optimal solution.*
- Note: with $X = \mathbb{R}^n \Longrightarrow$ convergence of steepest descent!