Introduction to Optimization

Vincent Leclère (CERMICS, ENPC)

Autumn 2026

Why should I bother to learn this stuff?

- For most realistic engineering problems, there is no formula for the optimal solution: a computer must approximate it.
- Different problems require different numerical methods, with different compromises between speed and guarantees.
- Knowing the basic principles helps you understand what a solver is doing, what its stopping message means, and whether you should trust the result.
- Gradient methods are among the basic building blocks of modern numerical optimization.

Contents

An optimization problem

When we consider an optimization problem

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & f(x) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

- We cannot expect to solve it analytically (*i.e.*with an explicit formula of the parameters) or even exactly (*i.e.*finding a global minimum) in general.
- We generally strive to find an approximate solution x with a finite number of operations.

➥ This is the goal of optimization algorithms.

- In addition, we sometimes want:
  - a lower bound on the optimal value (to evaluate the quality of our solution);
  - a dual optimal solution, for sensitivity analysis;
  - additional information on the problem (e.g. active constraints at the solution).

An optimization problem

When we consider an optimization problem

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & f(x) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

- We cannot expect to solve it analytically (*i.e.*with an explicit formula of the parameters) or even exactly (*i.e.*finding a global minimum) in general.
- We generally strive to find an approximate solution x with a finite number of operations.

➥ This is the goal of optimization algorithms.

- In addition, we sometimes want:
  - a lower bound on the optimal value (to evaluate the quality of our solution);
  - a dual optimal solution, for sensitivity analysis;
  - additional information on the problem (e.g. active constraints at the solution).

An optimization algorithm

An optimization algorithm specifies how a candidate solution is produced from the problem data.

- Input: problem data, an initial point and numerical parameters;
- Iteration: a well-defined update rule producing a sequence of candidate points;
- Stopping test: a rule deciding when the current approximation is sufficient;
- Output: an approximate solution and, when available, certificates or diagnostic information.

In practice, the procedure should be implementable on a computer. For theoretical analysis, one may nevertheless consider idealized steps (for instance, an exact subproblem or an oracle) that would themselves need to be approximated in an implementation.

An optimization algorithm

An optimization algorithm specifies how a candidate solution is produced from the problem data.

- Input: problem data, an initial point and numerical parameters;
- Iteration: a well-defined update rule producing a sequence of candidate points;
- Stopping test: a rule deciding when the current approximation is sufficient;
- Output: an approximate solution and, when available, certificates or diagnostic information.

In practice, the procedure should be implementable on a computer. For theoretical analysis, one may nevertheless consider idealized steps (for instance, an exact subproblem or an oracle) that would themselves need to be approximated in an implementation.

Heuristics vs exact methods

Two main categories of (very numerous) optimization algorithms:

- Heuristics: try to find a good solution in a reasonable time, generally without guarantee of optimality.
  - when you want a quick solution
  - when the problem is very difficult (typically combinatorial optimization)
  - when you do not care about guarantees
  - when you want a good initial solution for an exact method
- Examples: local search, genetic algorithms, simulated annealing, ...

- Exact methods: come with a guarantee of convergence to an optimal solution (or an ε-optimal one)
  - when you need an ε-optimal solution
  - when you want to offer guarantees of your solution
  - when the problem is of a specific class (e.g. convex unconstrained optimization, linear programming, ...)
- Examples: gradient descent, simplex method, interior point methods, ...

Heuristics vs exact methods

Two main categories of (very numerous) optimization algorithms:

- Heuristics: try to find a good solution in a reasonable time, generally without guarantee of optimality.
  - when you want a quick solution
  - when the problem is very difficult (typically combinatorial optimization)
  - when you do not care about guarantees
  - when you want a good initial solution for an exact method
- Examples: local search, genetic algorithms, simulated annealing, ...

- Exact methods: come with a guarantee of convergence to an optimal solution (or an ε-optimal one)
  - when you need an ε-optimal solution
  - when you want to offer guarantees of your solution
  - when the problem is of a specific class (e.g. convex unconstrained optimization, linear programming, ...)
- Examples: gradient descent, simplex method, interior point methods, ...

Contents

Descent methods (unconstrained case)

Descent methods are a large class of optimization algorithms for unconstrained optimization problems.

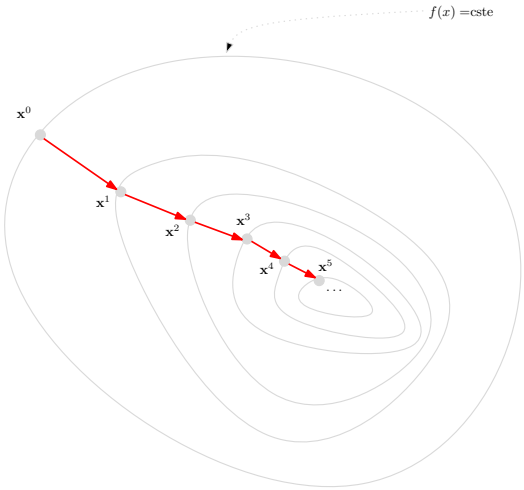
$$v^\sharp = \min_{x \in \mathbb{R}^n} \quad f(x).$$

A *descent direction algorithm* is an algorithm that constructs a sequence of points $(x^{(k)})_{k \in \mathbb{N}}$, that are recursively defined with:

$$x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)}$$

where

- $x^{(0)}$ is the initial point,
- $d^{(k)} \in \mathbb{R}^n$ is the descent direction,
- $t^{(k)}$ is the step length.



Descent methods (unconstrained case)

Descent methods are a large class of optimization algorithms for unconstrained optimization problems.

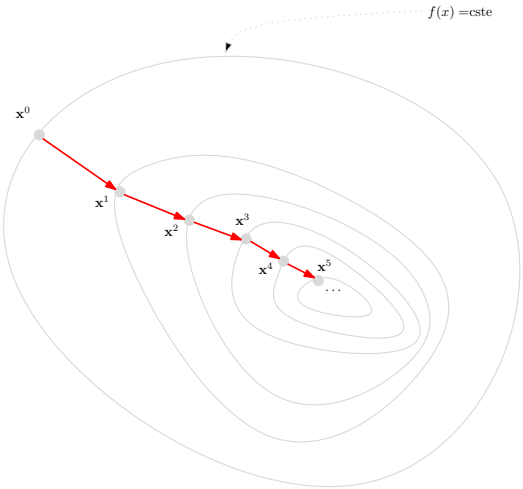
$$v^\sharp = \min_{x \in \mathbb{R}^n} \quad f(x).$$

A *descent direction algorithm* is an algorithm that constructs a sequence of points $(x^{(k)})_{k \in \mathbb{N}}$, that are recursively defined with:

$$x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)}$$

where

- $x^{(0)}$ is the initial point,
- $d^{(k)} \in \mathbb{R}^n$ is the descent direction,
- $t^{(k)}$ is the step length.



Descent methods (unconstrained case)

Descent methods are a large class of optimization algorithms for unconstrained optimization problems.

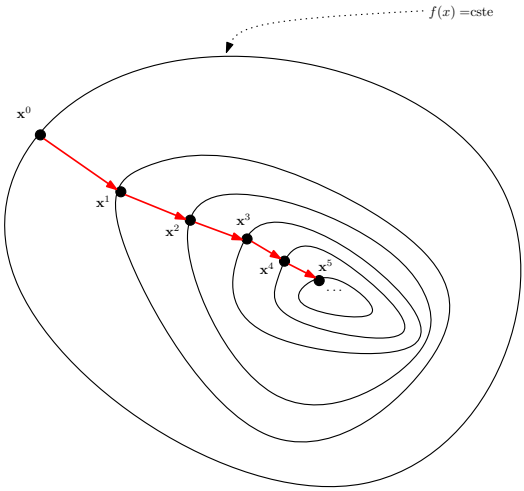
$$v^\sharp = \min_{x \in \mathbb{R}^n} \quad f(x).$$

A *descent direction algorithm* is an algorithm that constructs a sequence of points $(x^{(k)})_{k \in \mathbb{N}}$, that are recursively defined with:

$$x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)}$$

where

- $x^{(0)}$ is the initial point,
- $d^{(k)} \in \mathbb{R}^n$ is the descent direction,
- $t^{(k)}$ is the step length.



Descent direction

For an iterative algorithm of the form

$$x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)},$$

to be a descent method, the direction $d^{(k)}$ must be a descent direction, *i.e.* $d^{(k)}$ must satisfy[1]

$$\nabla f(x^{(k)})^\top d^{(k)} < 0.$$

Indeed, if $d^{(k)}$ is a descent direction, then for $t^{(k)}$ small enough,

$$f(x^{(k)} + t^{(k)} d^{(k)}) = f(x^{(k)}) + t^{(k)} \nabla f(x^{(k)}) \cdot d^{(k)} + o(t^{(k)}) < f(x^{(k)}).$$

[1] Sometimes we can have a non-strict inequality, but this is a technicality.

Descent direction

For an iterative algorithm of the form

$$x^{(k+1)} = x^{(k)} + t^{(k)} d^{(k)},$$

to be a descent method, the direction $d^{(k)}$ must be a descent direction, *i.e.* $d^{(k)}$ must satisfy[1]

$$\nabla f(x^{(k)})^\top d^{(k)} < 0.$$

Indeed, if $d^{(k)}$ is a descent direction, then for $t^{(k)}$ small enough,

$$f(x^{(k)} + t^{(k)} d^{(k)}) = f(x^{(k)}) + t^{(k)} \nabla f(x^{(k)}) \cdot d^{(k)} + o(t^{(k)}) < f(x^{(k)}).$$

[1] Sometimes we can have a non-strict inequality, but this is a technicality.

Step-size choice

The step-size $t^{(k)}$ can be:

- fixed $t^{(k)} = t^{(0)}$,
  - too small and it will take forever
  - too large and it won't converge
- exact line search $t^{(k)} \in \arg\min_{\tau \geq 0} f(x^{(k)} + \tau d^{(k)})$,
  - computing it requires solving an unidimensional problem
  - might not be worth the computation
- a backtracking choice[2]: start with a large step and decrease it until a sufficient decrease of the function is observed.

[2] Many other alternatives exist

Stopping test

To complete the algorithm, we need a stopping test. Typical choices include:

- an optimality certificate, for instance a certified gap

$$f(x^{(k)}) - LB_k \leq \varepsilon,$$

  where LB_k is a lower bound on the optimal value;

- stationarity or progress diagnostics, for instance

$$\|\nabla f(x^{(k)})\| \leq \varepsilon \qquad \text{or} \qquad \|x^{(k+1)} - x^{(k)}\| \leq \varepsilon;$$

- a resource limit: maximum number of iterations, running time, ...

The latter tests are useful stopping criteria, but are not in general certificates of ε-optimality without additional assumptions.

Stopping test

To complete the algorithm, we need a stopping test. Typical choices include:

- an optimality certificate, for instance a certified gap

$$f(x^{(k)}) - LB_k \leq \varepsilon,$$

  where LB_k is a lower bound on the optimal value;

- stationarity or progress diagnostics, for instance

$$\|\nabla f(x^{(k)})\| \leq \varepsilon \qquad \text{or} \qquad \|x^{(k+1)} - x^{(k)}\| \leq \varepsilon;$$

- a resource limit: maximum number of iterations, running time, ...

The latter tests are useful stopping criteria, but are not in general certificates of ε-optimality without additional assumptions.

Stopping test

To complete the algorithm, we need a stopping test. Typical choices include:

- an optimality certificate, for instance a certified gap

$$f(x^{(k)}) - LB_k \leq \varepsilon,$$

  where LB_k is a lower bound on the optimal value;

- stationarity or progress diagnostics, for instance

$$\|\nabla f(x^{(k)})\| \leq \varepsilon \qquad \text{or} \qquad \|x^{(k+1)} - x^{(k)}\| \leq \varepsilon;$$

- a resource limit: maximum number of iterations, running time, ...

The latter tests are useful stopping criteria, but are not in general certificates of ε-optimality without additional assumptions.

Contents

Gradient descent

- The gradient descent algorithm is a first-order descent direction algorithm with $d^{(k)} = -\nabla f(x^{(k)})$.
- That is, with an initial point x_0, we have

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)}).$$

- The three step-size choices (fixed, exact line search and backtracking) lead to variations of the algorithm.

- This algorithm is slow, but robust in the sense that it often ends up converging.
- Most implementations of advanced algorithms have fail-safe procedures that default to a gradient step when something goes wrong for numerical reasons.
- It is the basis of the stochastic-gradient algorithm, which is used (in advanced form) to train ML models.

Gradient descent

- The gradient descent algorithm is a first-order descent direction algorithm with $d^{(k)} = -\nabla f(x^{(k)})$.
- That is, with an initial point x_0, we have

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)}).$$

- The three step-size choices (fixed, exact line search and backtracking) lead to variations of the algorithm.

- This algorithm is slow, but robust in the sense that it often ends up converging.
- Most implementations of advanced algorithms have fail-safe procedures that default to a gradient step when something goes wrong for numerical reasons.
- It is the basis of the stochastic-gradient algorithm, which is used (in advanced form) to train ML models.

Convergence results

Assume that f is convex with L-Lipschitz gradient, *i.e.*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

Let $x^\sharp$ be a minimizer of f, $v^\sharp = f(x^\sharp)$, and consider

$$x^{(k+1)} = x^{(k)} - t\nabla f(x^{(k)}).$$

Theorem (Convergence — convex case)

For a fixed step size $0 < t \leq 1/L$,

$$f(x^{(k)}) - v^\sharp \leq \frac{\|x^{(0)} - x^\sharp\|^2}{2tk} = O(1/k).$$

Theorem (Convergence — strongly convex case)

Assume in addition that f *is* α*-strongly convex. With the fixed step* $t = 1/L$,

$$f(x^{(k)}) - v^\sharp \leq \left(1 - \frac{\alpha}{L}\right)^k (f(x^{(0)}) - v^\sharp).$$

Convergence results

Assume that f is convex with L-Lipschitz gradient, *i.e.*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

Let $x^\sharp$ be a minimizer of f, $v^\sharp = f(x^\sharp)$, and consider

$$x^{(k+1)} = x^{(k)} - t\nabla f(x^{(k)}).$$

Theorem (Convergence — convex case)

For a fixed step size $0 < t \leq 1/L$,

$$f(x^{(k)}) - v^\sharp \leq \frac{\|x^{(0)} - x^\sharp\|^2}{2tk} = O(1/k).$$

Theorem (Convergence — strongly convex case)

Assume in addition that f *is* α*-strongly convex. With the fixed step* $t = 1/L$,

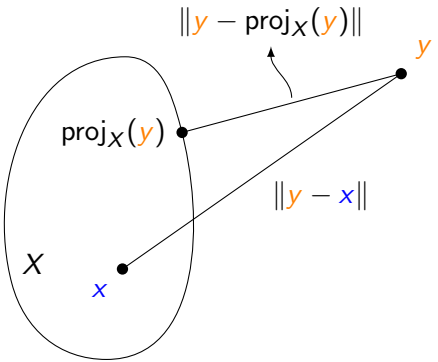
$$f(x^{(k)}) - v^\sharp \leq \left(1 - \frac{\alpha}{L}\right)^k (f(x^{(0)}) - v^\sharp).$$

Contents

Projection on a convex set

- The projection of a point $y \in \mathbb{R}^n$ onto a closed convex set $X \subseteq \mathbb{R}^n$ is defined as the (unique) solution of

$$\text{proj}_X(y) = \arg\min_{x \in X} \|x - y\|^2.$$



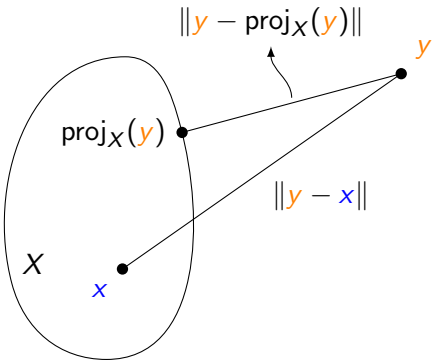
Projection on a convex set

- The projection of a point $y \in \mathbb{R}^n$ onto a closed convex set $X \subseteq \mathbb{R}^n$ is defined as the (unique) solution of

$$\text{proj}_X(y) = \arg\min_{x \in X} \|x - y\|^2.$$

- proj_X is contracting:

$$\|\text{proj}_X(y) - \text{proj}_X(z)\| \leq \|y - z\|.$$



Projection on a convex set

- The projection of a point $y \in \mathbb{R}^n$ onto a closed convex set $X \subseteq \mathbb{R}^n$ is defined as the (unique) solution of

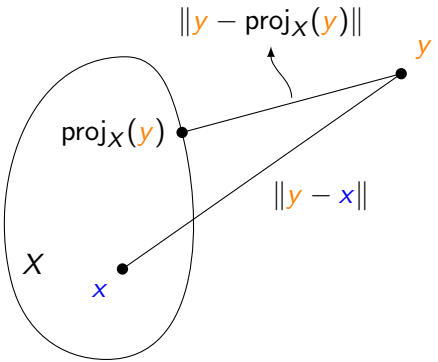
$$\text{proj}_X(y) = \arg\min_{x \in X} \|x - y\|^2.$$

- proj_X is contracting:

$$\|\text{proj}_X(y) - \text{proj}_X(z)\| \leq \|y - z\|.$$

- $\bar{x} = \text{proj}_X(y)$ if and only if $\bar{x} \in X$ and for all $x \in X$ we have

$$(y - \bar{x})^\top (x - \bar{x}) \leq 0.$$



Projected gradient descent

The projected gradient descent algorithm is a first-order method for constrained convex optimization problems of the form

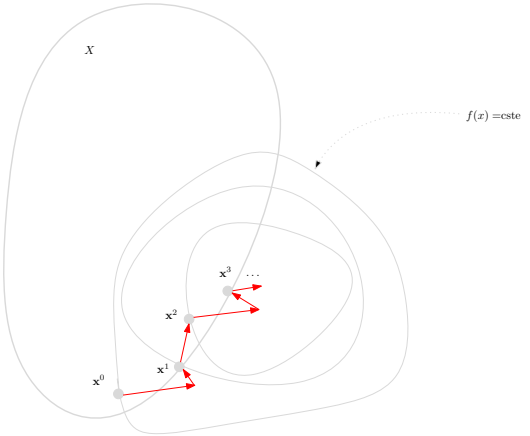
$$v^\sharp = \min_{x \in X} \quad f(x).$$

with X a convex set.

It constructs a sequence of points $(x^{(k)})_{k \in \mathbb{N}}$, that are recursively defined with:

$$x^{(k+1/2)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)})$$

$$x^{(k+1)} = \text{proj}_X \left(x^{(k+1/2)} \right)$$



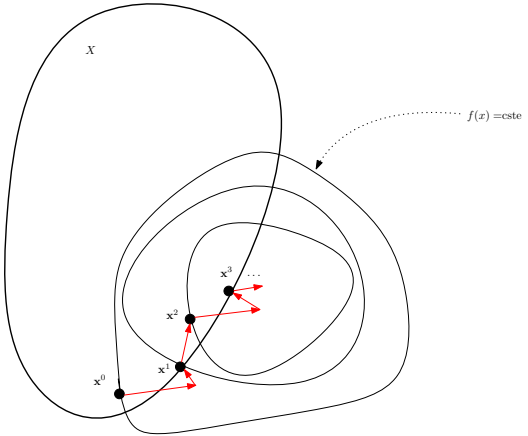
Projected gradient descent

The projected gradient descent algorithm is a first-order method for constrained convex optimization problems of the form

$$v^\sharp = \min_{x \in X} \quad f(x).$$

with X a convex set.
It constructs a sequence of points $(x^{(k)})_{k \in \mathbb{N}}$, that are recursively defined with:

$$x^{(k+1/2)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)})$$
$$x^{(k+1)} = \text{proj}_X \left(x^{(k+1/2)} \right)$$



Projected gradient descent: convergence theorem

Theorem (Convergence — projected gradient descent)

Let f be convex with L-Lipschitz gradient, and let X be a nonempty closed convex set. Assume that $\min_{x \in X} f(x)$ admits a minimizer $x^\sharp$.
For a fixed step size $0 < t \leq 1/L$,

$$f(x^{(k)}) - v^\sharp \leq \frac{\|x^{(0)} - x^\sharp\|^2}{2t\,k} = O(1/k).$$

Projected gradient descent: important warning

- The projected gradient descent algorithm allows to solve constrained optimization problems with a simple modification of the gradient descent algorithm.
- However, it requires to be able to compute efficiently the projection onto the set X.
- This is possible for simple sets (e.g. box constraints, Euclidean ball, half-space, ...), but not for general sets (e.g. polyhedra).

➥ For a general convex set X, the projection onto X requires to solve a quadratic optimization problem with constraints, which is (in general) as difficult as the original problem.

➥ Therefore, the projected gradient descent algorithm is only used for simple sets X.

Contents

Linear programming

A linear program is an optimization problem with a linear objective and linear constraints. In inequality form:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given data.

- The objective function is linear: $f(x) = c^\top x$.
- The feasible set is a polyhedron: an intersection of finitely many half-spaces.
- Equalities and sign constraints can be represented by linear inequalities, so this form is general up to reformulation.
- Linear programs can be solved efficiently with dedicated algorithms (simplex, interior-point methods, ...).

Linear programming

A linear program is an optimization problem with a linear objective and linear constraints. In inequality form:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given data.

- The objective function is linear: $f(x) = c^\top x$.
- The feasible set is a polyhedron: an intersection of finitely many half-spaces.
- Equalities and sign constraints can be represented by linear inequalities, so this form is general up to reformulation.
- Linear programs can be solved efficiently with dedicated algorithms (simplex, interior-point methods, ...).

Linear programming as convex optimization

A linear program is a convex optimization problem with a linear objective function and a polyhedral feasible set.

The objective $f(x) = c^\top x$ is convex and differentiable, with $\nabla f(x) = c$.

The feasible set

$$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is a closed convex set.

Therefore the theory developed so far applies. In particular:

- affine constraints are qualified at every feasible point;
- KKT conditions characterize optimality;
- strong duality holds[3];
- the dual of a linear program is itself a linear program.

[3] Provided at least one of the primal or dual problems is feasible.

Linear programming as convex optimization

A linear program is a convex optimization problem with a linear objective function and a polyhedral feasible set.

The objective $f(x) = c^\top x$ is convex and differentiable, with $\nabla f(x) = c$.

The feasible set

$$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is a closed convex set.

Therefore the theory developed so far applies. In particular:

- affine constraints are qualified at every feasible point;
- KKT conditions characterize optimality;
- strong duality holds[3];
- the dual of a linear program is itself a linear program.

[3] Provided at least one of the primal or dual problems is feasible.

Linear programming as convex optimization

A linear program is a convex optimization problem with a linear objective function and a polyhedral feasible set.

The objective $f(x) = c^\top x$ is convex and differentiable, with $\nabla f(x) = c$.

The feasible set

$$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is a closed convex set.

Therefore the theory developed so far applies. In particular:

- affine constraints are qualified at every feasible point;
- KKT conditions characterize optimality;
- strong duality holds[3];
- the dual of a linear program is itself a linear program.

[3] Provided at least one of the primal or dual problems is feasible.

Linear programming as convex optimization

A linear program is a convex optimization problem with a linear objective function and a polyhedral feasible set.

The objective $f(x) = c^\top x$ is convex and differentiable, with $\nabla f(x) = c$.

The feasible set

$$X = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is a closed convex set.

Therefore the theory developed so far applies. In particular:

- affine constraints are qualified at every feasible point;
- KKT conditions characterize optimality;
- strong duality holds[3];
- the dual of a linear program is itself a linear program.

[3] Provided at least one of the primal or dual problems is feasible.

Linear programs: inequality and standard forms

Two useful forms are:

- Inequality form:

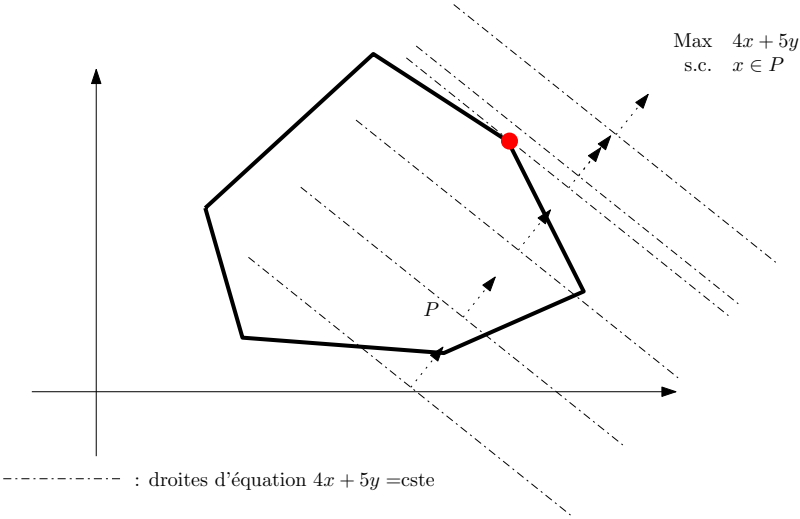
$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

- Standard form (convenient for the simplex method):

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Min}} \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Up to adding variables and constraints, any linear program can be rewritten in either form.
Exercise. Show how to convert one form into the other.

Geometric interpretation



Key geometric fact

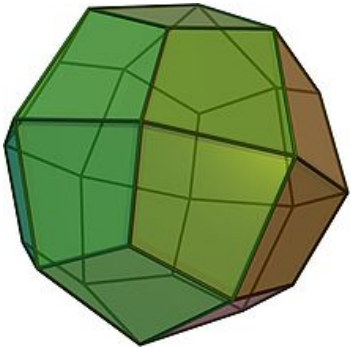
Contents

Definitions

- A half-space is a set of the form $\{x \in \mathbb{R}^n \mid a^\top x \leq b\}$, with $a \in \mathbb{R}^n$, $b \in \mathbb{R}$.
- A polyhedron is an intersection of a finite number of half-spaces.

➥ A polyhedron is a convex set.

- A polytope is a bounded polyhedron.
- A face of a polyhedron P is the intersection of P with a supporting hyperplane.
- A vertex (or extreme point) is a face of dimension 0.
- An edge is a face of dimension 1.

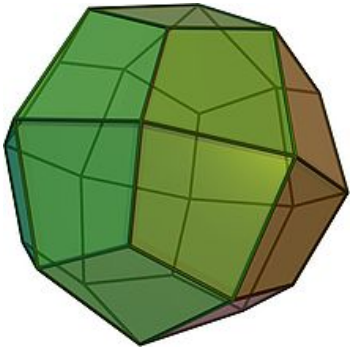


Definitions

- A half-space is a set of the form $\{x \in \mathbb{R}^n \mid a^\top x \leq b\}$, with $a \in \mathbb{R}^n$, $b \in \mathbb{R}$.
- A polyhedron is an intersection of a finite number of half-spaces.

➥ A polyhedron is a convex set.

- A polytope is a bounded polyhedron.
- A face of a polyhedron P is the intersection of P with a supporting hyperplane.
- A vertex (or extreme point) is a face of dimension 0.
- An edge is a face of dimension 1.



Representation of polyhedra

A polyhedron can be represented in two ways:

- H-representation: as an intersection of half-spaces, *i.e.*

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}.$$

- V-representation: as a convex hull of points and directions, *i.e.*

$$P = \left\{ \sum_{i=1}^{k} \lambda_i v_i + \sum_{j=1}^{m} \mu_j d_j \;\middle|\; \lambda \in [0,1]^k,\ \sum_{i=1}^{k} \lambda_i = 1,\ \mu \geq 0 \right\},$$
$$= \operatorname{conv}\{v_1, \ldots, v_k\} + \operatorname{cone}\{d_1, \ldots, d_m\}.$$

The two representations are mathematically equivalent, but conversion can be computationally expensive and may require exponentially many vertices or facets.

Representation of polyhedra

A polyhedron can be represented in two ways:

- H-representation: as an intersection of half-spaces, *i.e.*

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}.$$

- V-representation: as a convex hull of points and directions, *i.e.*

$$P = \left\{ \sum_{i=1}^{k} \lambda_i v_i + \sum_{j=1}^{m} \mu_j d_j \;\middle|\; \lambda \in [0,1]^k,\ \sum_{i=1}^{k} \lambda_i = 1,\ \mu \geq 0 \right\},$$
$$= \operatorname{conv}\{v_1, \ldots, v_k\} + \operatorname{cone}\{d_1, \ldots, d_m\}.$$

The two representations are mathematically equivalent, but conversion can be computationally expensive and may require exponentially many vertices or facets.

What you should know

- The main ingredients of an optimization algorithm: initialization, iteration, stopping test, output.
- The distinction between heuristics and exact methods.
- The principle of gradient descent and the role of the step size.
- The principle of projected gradient descent for constrained problems.

What you should be able to do

- Perform a few iterations of gradient descent on a simple problem.
- Compute a projection onto a simple convex set.
- Perform a few iterations of projected gradient descent.
- Interpret a stopping criterion and distinguish it from a guarantee of optimality.