ORF522 – Linear and Nonlinear Optimization

23. The role of optimization

Bartolomeo Stellato — Fall 2022

Difference of convex programming

Difference of convex programming

$$\begin{array}{ll} \text{minimize} & \boxed{f_0(x) - g_0(x)} \\ \text{subject to} & \boxed{f_i(x) - g_i(x)} \leq 0, \quad i = 1, \ldots, m \end{array}$$

Difference of convex functions

where f_i and g_i are convex

Very powerful

it can represent any twice differentiable function

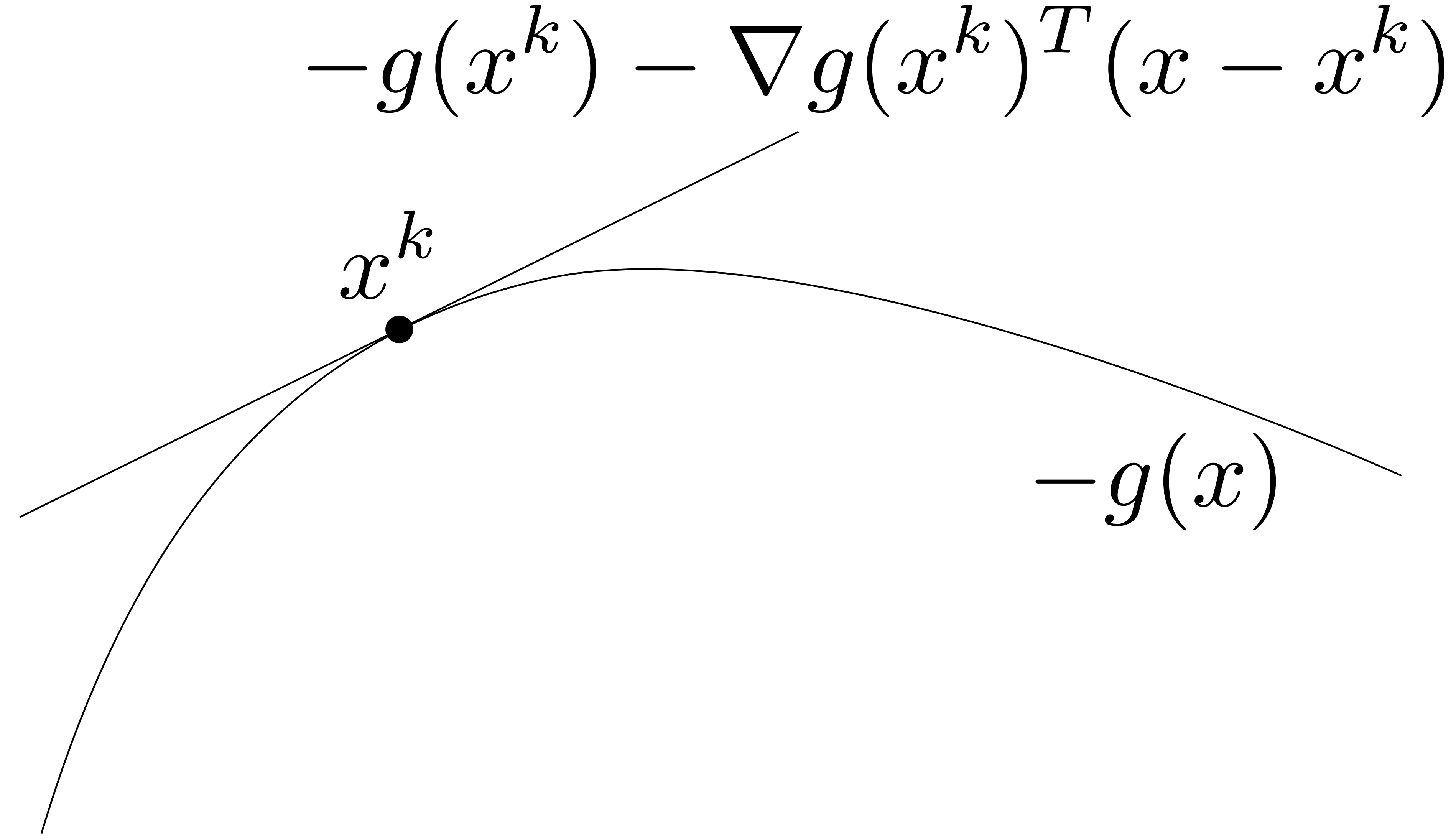
Hard

nonconvex problem unless g_i are affine

[On Functions Representable As A Difference Of Convex Functions, Hartman]

Difference of convex programming

Convexification



Convexify $f(x) - g(x)$

$$f(x) - \hat{g}(x) = f(x) - g(x^k) - \nabla g(x^k)^T(x - x^k)$$

$$\Downarrow$$

$$f(x) - g(x) \leq f(x) - \hat{g}(x)$$

Remarks

- True objective better than convexified objective
- True feasible set contains convexified feasible set $\longrightarrow$ **No trust region needed**

Difference of convex programming

Iterations

Convex-concave procedure

1. Convexify: form $\hat{g}_i(x) = g_i(x^k) + \nabla g_i(x^k)^T(x - x^k)$ for $i = 0, \dots, m$
2. Solve to obtain x^{k+1}

$$\begin{array}{ll} \text{minimize} & f_0(x) - \hat{g}_0(x) \\ \text{subject to} & f_i(x) - \hat{g}_i(x) \leq 0 \end{array}$$

Remarks

It always converges to a stationary point (it might be a maximum)

[Variations and extension of the convex–concave procedure, Lipp, Boyd]

Path planning example

Find shortest path connecting a and b in $\mathbf{R}^d$

Avoid circles centered at c_j with radius r_j with $j = 1, \ldots, m$

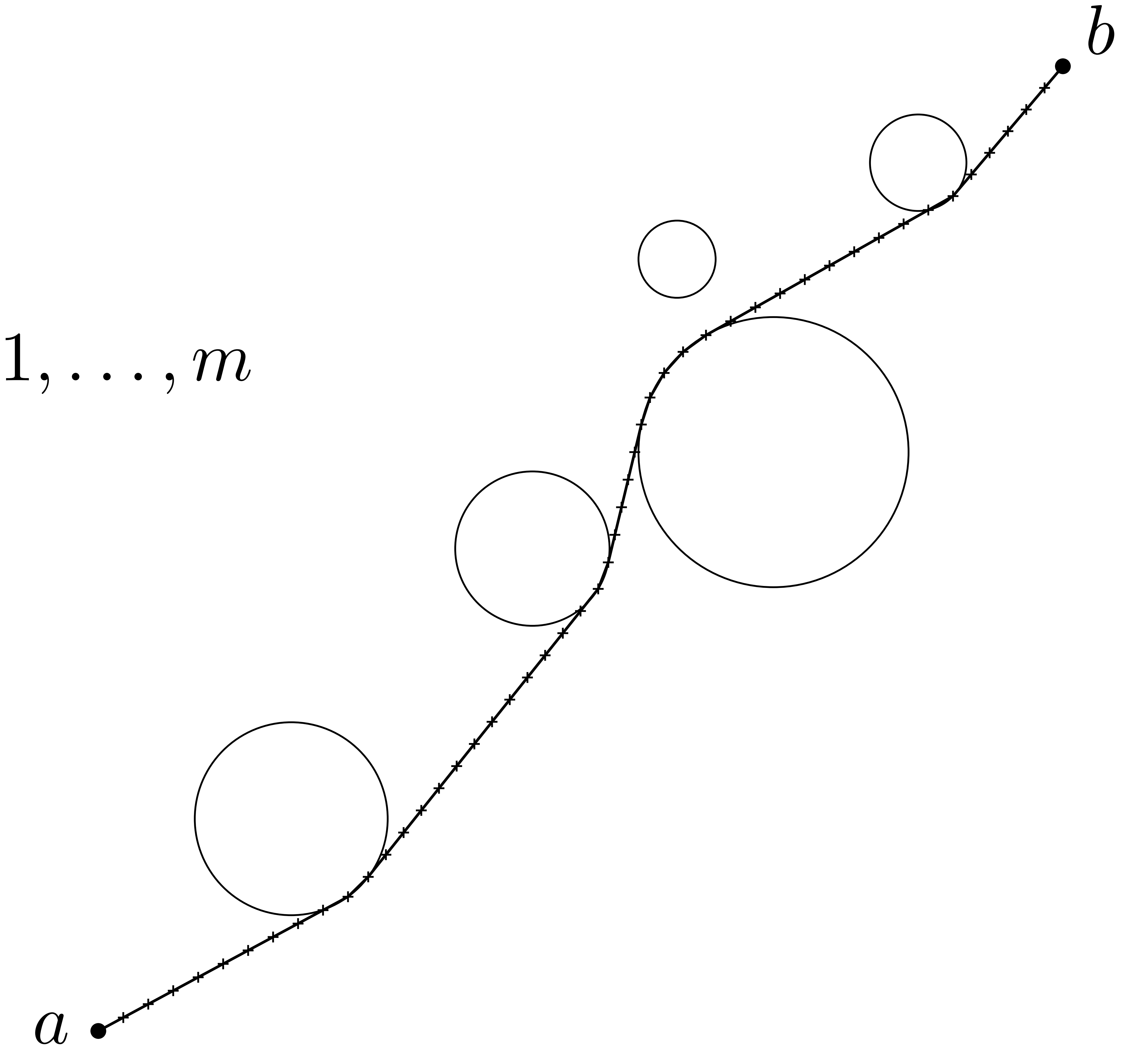
$$
\begin{array}{ll}
\text{minimize} & L \\
\text{subject to} & x_0 = a, \quad x_n = b \\
& \|x_i - x_{i-1}\|_2 \leq L/n, \quad i = 1, \ldots, n \\
& \|x_i - c_j\|_2 \geq r_j, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m
\end{array}
$$

path lengths — $\|x_i - x_{i-1}\|_2 \leq L/n$

obstacle constraints (not convex) — $\|x_i - c_j\|_2 \geq r_j$

Path planning example

$$\begin{array}{ll} \text{minimize} & L \\ \text{subject to} & x_0 = a, \quad x_n = b \\ & \|x_i - x_{i-1}\|_2 \le L/n, \quad i = 1, \ldots, n \\ & \|x_i - c_j\|_2 \ge r_j, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m \end{array}$$



Dimension: $d = 2$ Steps: $n = 50$

It converges in 26 iterations (convex problems)

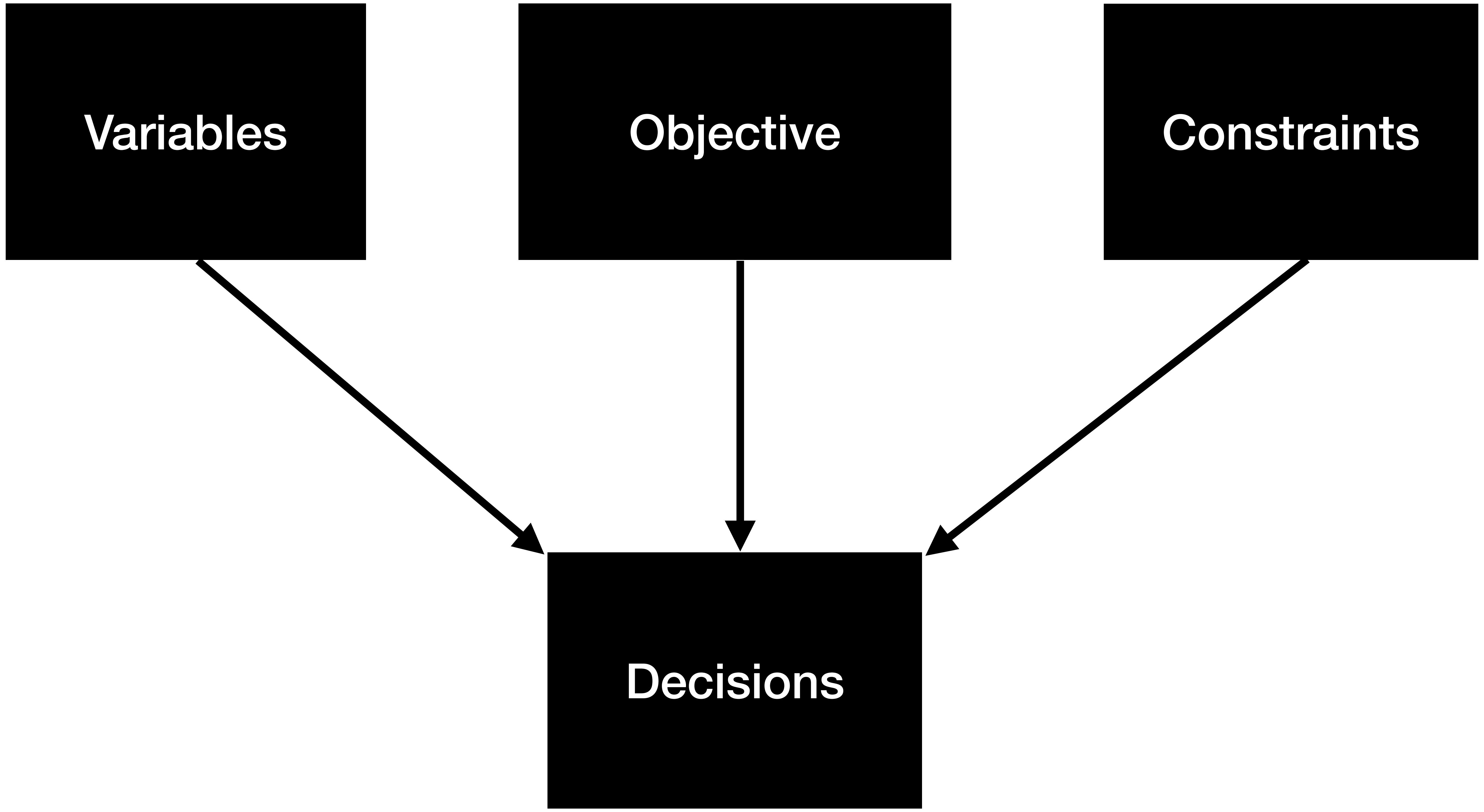
[Disciplined Convex-Concave Programming, Shen, Diamond, Gu, Boyd]

Today's lecture
The role of optimization

- Geometry of optimization problems
- Solving optimization problems
- What's left out there?
- The role of optimization

Basic use of optimization

Optimal decisions



Mathematical language

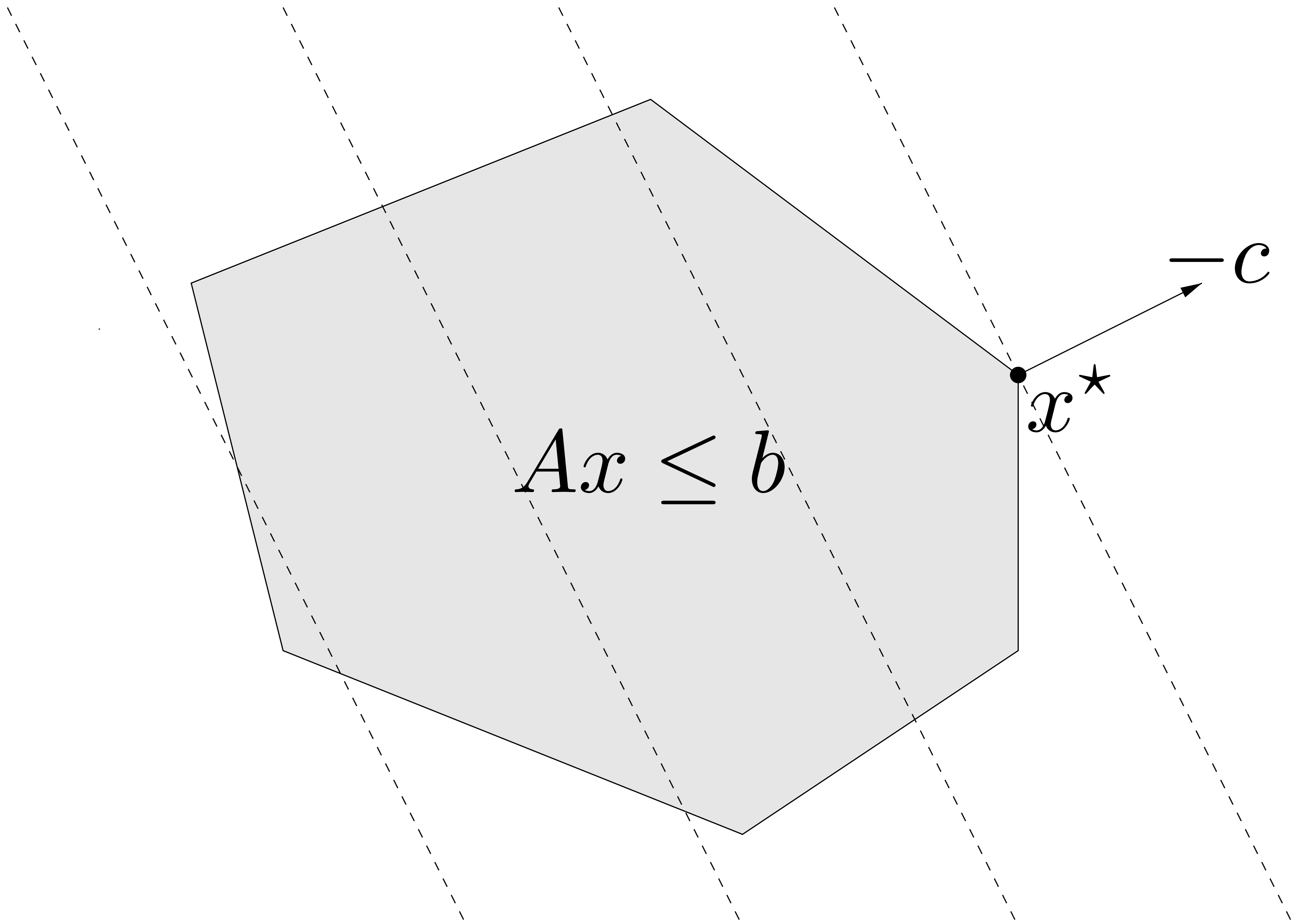
The algorithm computes them for you

Most optimization problems cannot be solved

Geometry of optimization problems

Linear optimization

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b \end{array}$$

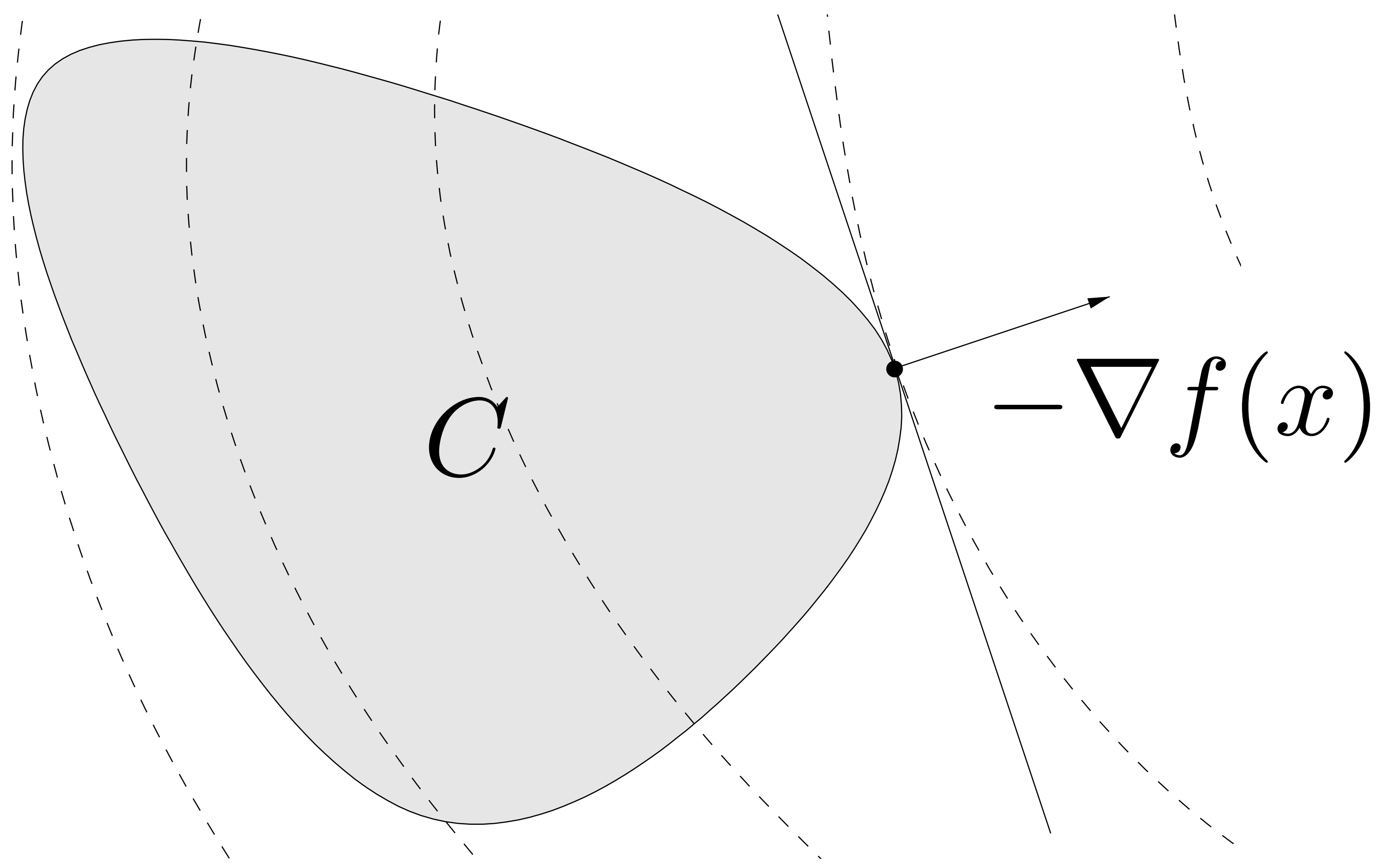


Optimal point properties

- Extreme points are optimal
- Need to search only between extreme points

Nonlinear optimization

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in C \end{array}$$

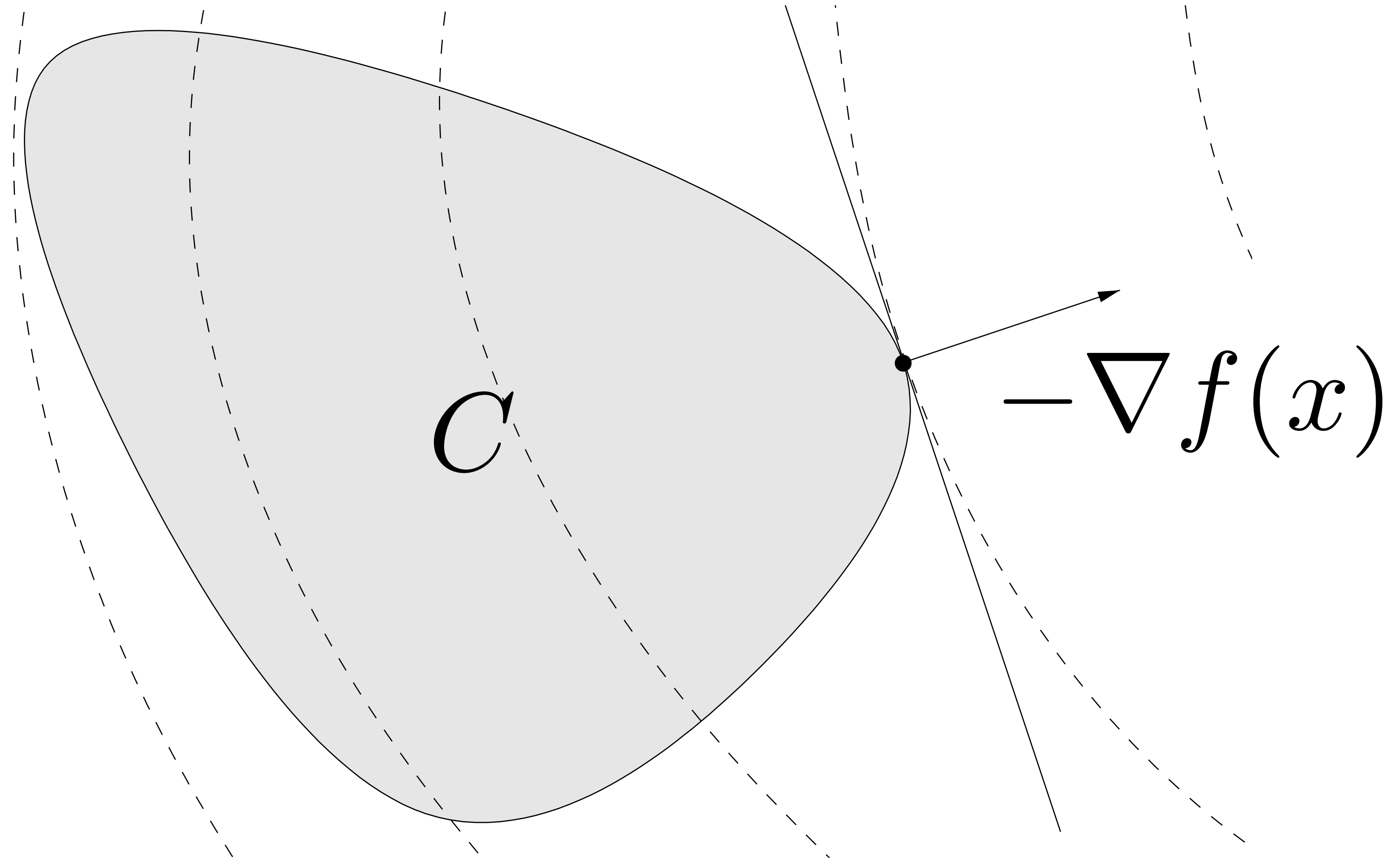


Optimal point properties

- Any feasible point could be optimal
- Can have many locally optimal points

Fermat's optimality conditions

minimize $\quad f(x)$
subject to $\quad x \in C$



Stationarity conditions

$$0 \in \partial f(x) + \mathcal{I}_C(x)$$

$\longrightarrow$

Differentiable f convex C

$$-\nabla f(x) \in \mathcal{N}_C(x)$$

Properties

- *Convex optimization* (necessary and sufficient)
- *Nonconvex optimization* (necessary)

KKT optimality conditions

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \end{array}$$

$$\nabla f(x^\star) + \sum_{i=1}^{m} y_i^\star \nabla g_i(x^\star) = 0$$ **stationarity**

$$y^\star \geq 0$$ **dual feasibility**

$$g_i(x^\star) \leq 0, \quad i = 1, \ldots, m$$ **primal feasibility**

$$y_i^\star g_i(x^\star) = 0, \quad i = 1, \ldots, m$$ **complementary slackness**

Remarks

- Require Slater's conditions or constraint qualifications (LICQ)
- Can be derived from Fermat's optimality
- Necessary and sufficient for convex problems
- Only necessary for nonconvex problems

In practice

$\downarrow$

Search for KKT points

Certifying optimality

Dual function

$$g(y)$$

$\longrightarrow$

Properties

- Lower bound: $g(y) \leq f(x), \quad \forall x, y$
- Always -convex (concave)

Strong duality

$$g(y^\star) = f(x^\star)$$

- Linear optimization (unless primal and dual infeasible)
- Convex optimization (if Slater's condition holds)

Optimality gap

- Convex optimization without strong duality
- Nonconvex optimization

$\longrightarrow$

It provides a suboptimality certificate

Solving optimization problems

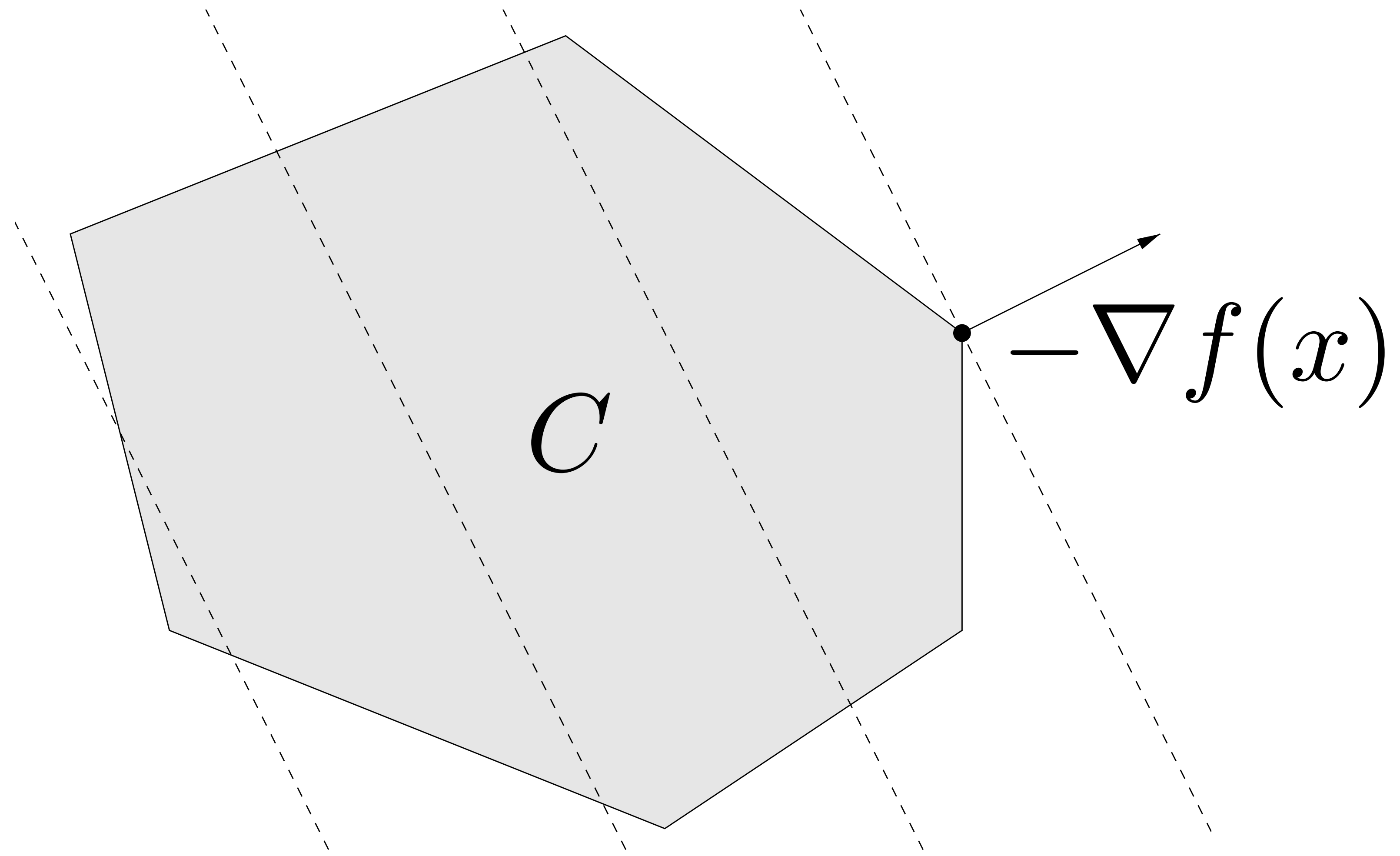
Classical vs modern view

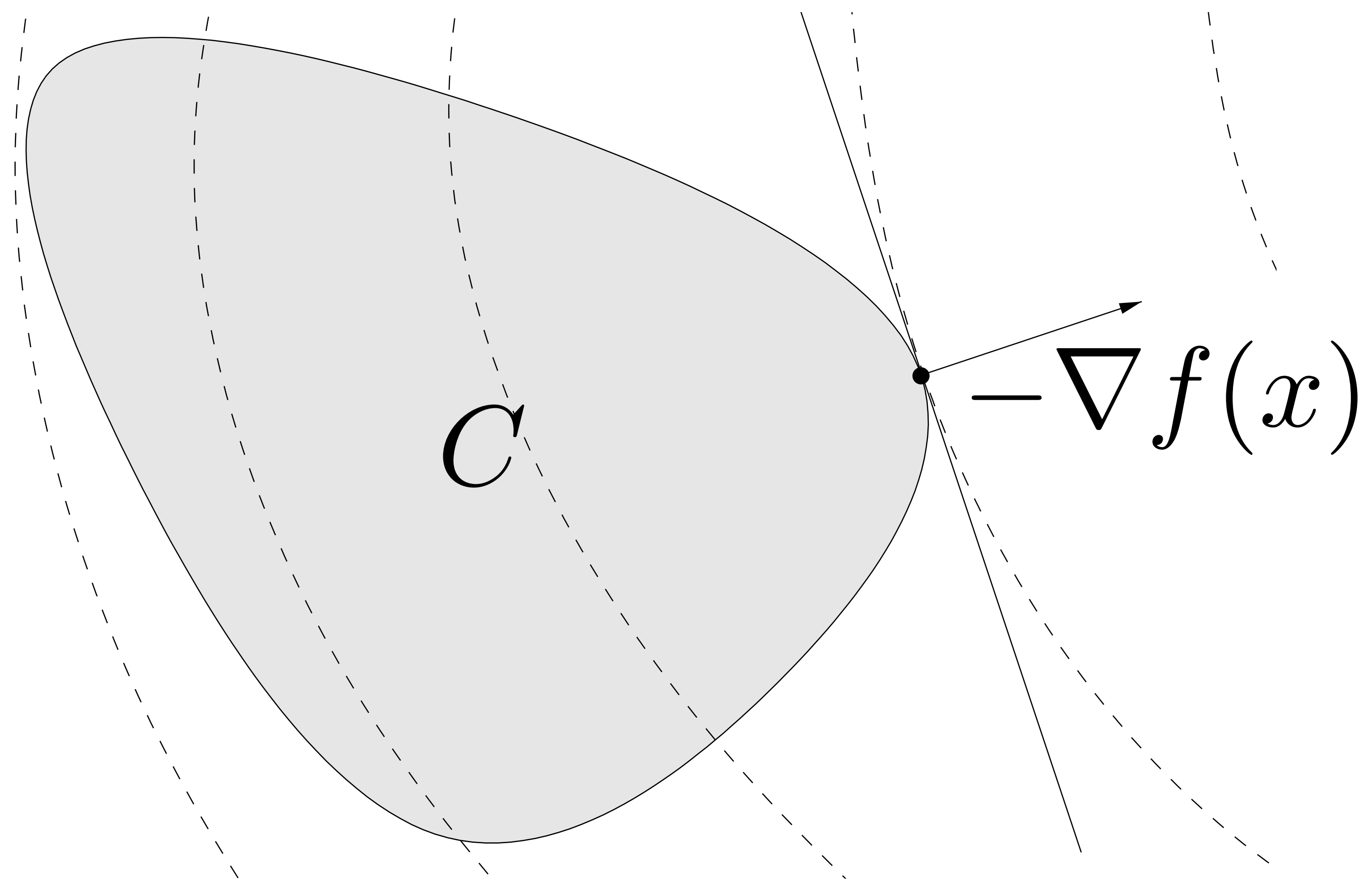
Classical view

- **Linear optimization** (zero curvature) is easy
- **Nonlinear optimization** (nonzero curvature) is hard

Correct view

- **Convex optimization** (nonnegative curvature) is easy
- **Nonconvex optimization** (negative curvature) is hard





The classical view is wrong

Numerical linear algebra

The core of optimization algorithms is linear systems solution

$$Ax = b$$

Direct method

1. Factor $A = A_1 A_2 \dots A_k$ in “simple” matrices ($O(n^3)$)
2. Compute $x = A_k^{-1} \dots A_1^{-1} b$ by solving k “easy” linear systems ($O(n^2)$)

Main benefit
factorization can be reused
with different right-hand sides b

You **never** invert A

Solving convex problems

Simplex methods

- Tailored to LPs
- Exponential worst-case performance
- Up to 10,000 variables

Cheap iterations
(rank-1 updates)

Second-order methods
(e.g., interior-point)

- Up to ~10,000 variables
- Polynomial worst-case complexity

Expensive iterations
(matrix factorizations)

First-order methods

- Up to 1B variables
- Several convergence rates
- Sensitive to data scaling

Cheap iterations
(matrix prefactored)

Convex optimization solvers

Remarks

- **No babysitting**/initialization required
- Very **reliable** and **efficient**
- Can solve problems in **milliseconds** on embedded platforms
- **Simplex** and **interior-point** solvers are **almost a technology**
- **First-order** methods are more **sensitive to data scaling** but work in **huge dimensions**

First-order methods for large-scale convex optimization

Per-iteration cost

Number of iterations

- Gradient/subgradient method
- Forward-backward splitting (proximal algorithms)
- Accelerated forward-backward splitting
- Douglas-Rachford splitting (ADMM)
- Interior-point methods (not covered for convex)

Large-scale systems

- start with feasible method with cheapest per-iteration cost
- if too many iterations, transverse down the list

Methods for nonconvex optimization

Convex optimization algorithms: global and typically **fast**

Nonconvex optimization algorithms: must give up one, global or fast

- **Local methods: fast** but **not global**
  Need not find a global (or even feasible) solution.
  They cannot certify global optimality because
  KKT conditions are not sufficient.

  $\longrightarrow$ **Heuristics**

- **Global methods: global** but often **slow**
  They find a global solution and certify it.

  $\longrightarrow$ **Global methods**

What’s left out there?

What we did not cover in nonlinear optimization

Second-order methods: High accuracy on small/medium-scale data

- Newton's method
- Quasi-Newton (BFGS, L-BFGS)
- Interior-point methods for nonlinear optimization (IPOPT)

Stochastic gradient methods

- Stochastic gradient descent
- Variance reduction methods
- Deep learning optimizers

⟶ Covered in ELE539: Optimization for Machine Learning

Optimization in data science

- Compressed sensing
- Low-rank matrix recovery
- Many more…

⟶ (was!) covered in ELE520: Mathematics of Data Science

What we did not cover in convex optimization?

More in details on convex analysis

Conic optimization

- Second-order cone programming
- Semidefinite programming
- Sum-of-squares optimization

$\longrightarrow$ Covered in ORF523: Convex and Conic Optimization

Convex relaxations of NP-hard problems

The role of optimization

Optimization as a surrogate for real goal

Very often, optimization is not the actual goal

The goal usually comes from practical
implementation (new data, real dynamics, etc.)

Real goal is usually encoded (approximated) in cost/constraints

Optimization problems are just models

"All models are wrong, some are useful."

— George Box

Implications

- Problem formulation does not need to be "accurate"
- Objective function and constraints "guide" the optimizer
- The model includes parameters to tune

We often do not need to solve most problems to extreme accuracy

Portfolio

Optimization problem

$$
\begin{array}{ll}
\text{maximize} & \mu^T x - \gamma x^T \Sigma x \\
\text{subject to} & \mathbf{1}^T x = 1 \\
& x \geq 0
\end{array}
$$

Goal

Optimize backtesting performance

Uncertain returns

p_t random variable:
mean μ, covariance Σ

$\longrightarrow$

Backtesting performance

(sum over all past realizations)

- Total returns
- Cumulative risk (quadratic term)

Control

Optimization problem
(control policy)

$$\begin{aligned} \phi(\bar{x}) = \quad & \underset{u}{\text{argmin}} \quad && \sum_{t=0}^{T-1} \ell(x_t, u_t) \\ & \text{subject to} && x_{t+1} = f(x_t, u_t) \\ &&& x_0 = \bar{x} \\ &&& x_t \in \mathcal{X}, \quad u_t \in \mathcal{U} \end{aligned}$$

Goal:
Optimize closed-loop performance

Real dynamics

$x_{t+1} = f(x_t, u_t, w_t)$
w_t uncertainty

Control input

$u_t = \phi(x_t)$

Closed-loop performance

$$J = \sum_{t=0}^{\infty} \ell(x_t, u_t)$$

Quadcopter control

Low accuracy works well

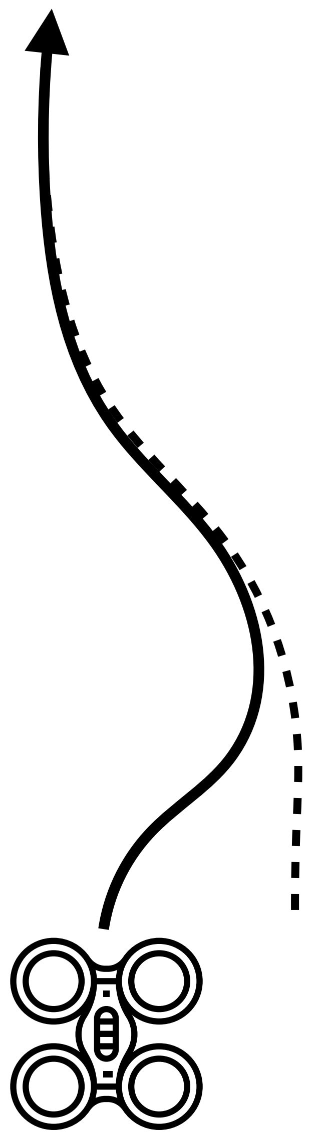
Quadcopter example

Linearized dynamics $x_{t+1} = Ax_t + Bu_t + w_t$

$x_t \in \mathbf{R}^{12}, \quad u_t \in \mathbf{R}^4$

Input and state constraints

$x_t \in [\underline{x}, \overline{x}], \quad u_t \in [\underline{u}, \overline{u}]$



Goal: track trajectory minimize $\sum_t \|x_t - x_t^{\text{des}}\|_2^2 + \gamma \|u_t\|_2^2$

Closed loop simulation

Simulated dynamics $x_{t+1} = Ax_t + Bu_t + w_t$

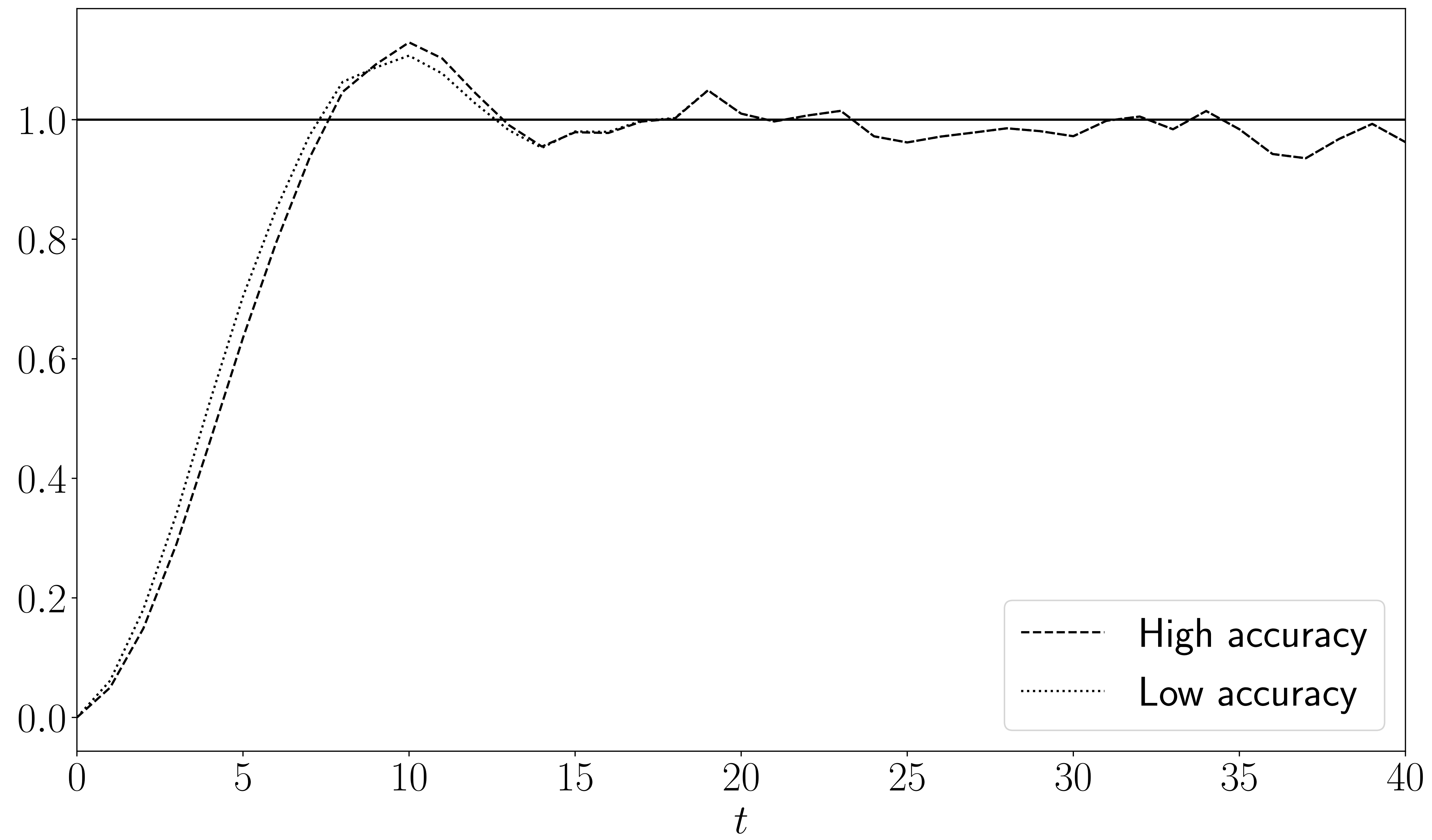
random variable
(nonlinearities,
disturbances, etc.)

Quadcopter control

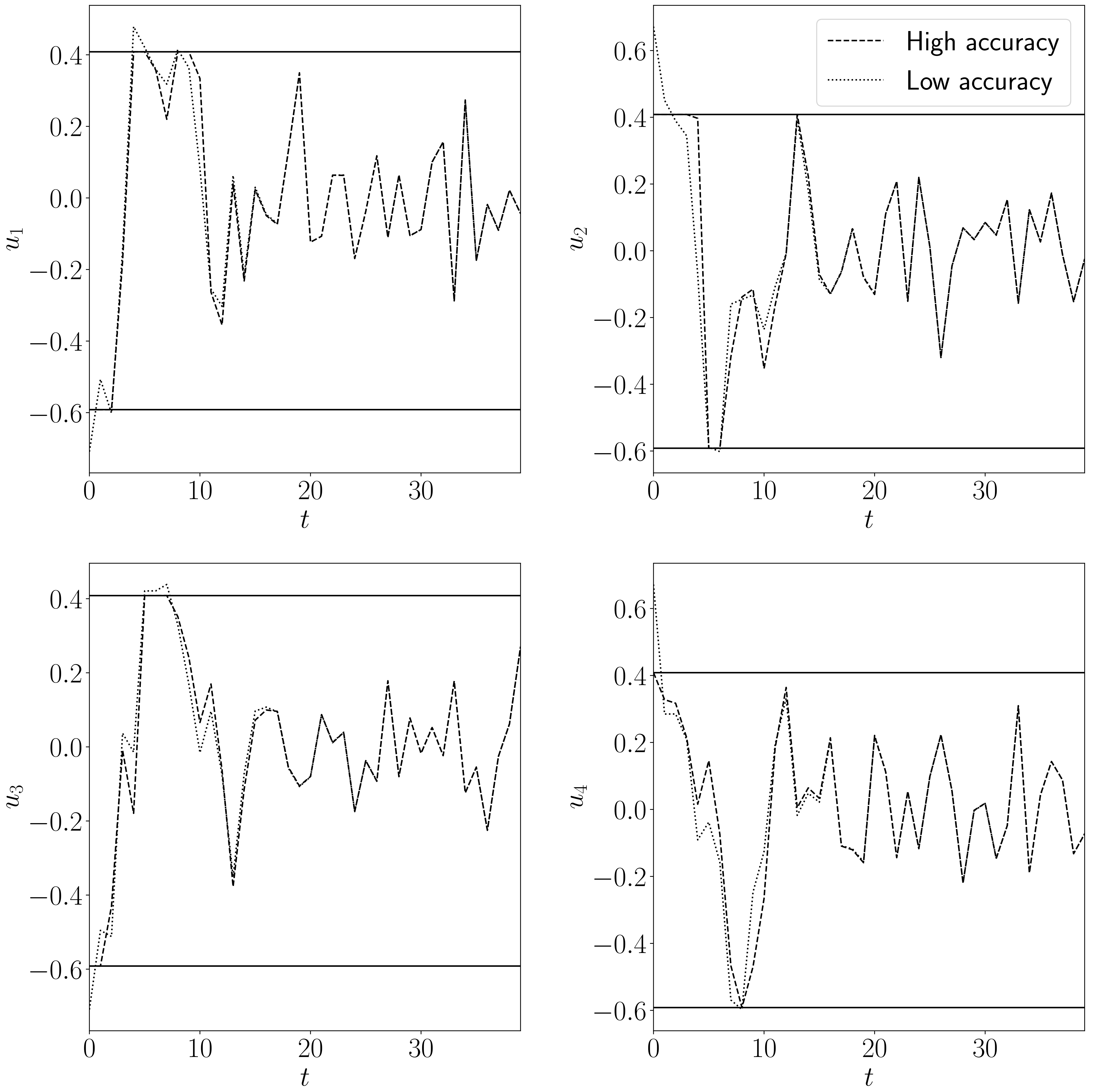
Closed-loop behavior with OSQP solver

- Low accuracy: $\epsilon = 0.1$
- High accuracy: $\epsilon = 0.0004$

Altitude reference tracking



Control effort



Model fitting

Training data

$$\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N} \longrightarrow$$

Optimization problem

$$\underset{w}{\text{minimize}} \quad f_{\text{train}}(w) = \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \ell(y_i, h_w(x_i))$$

Goal
Optimize test performance

Test data
(unknown)

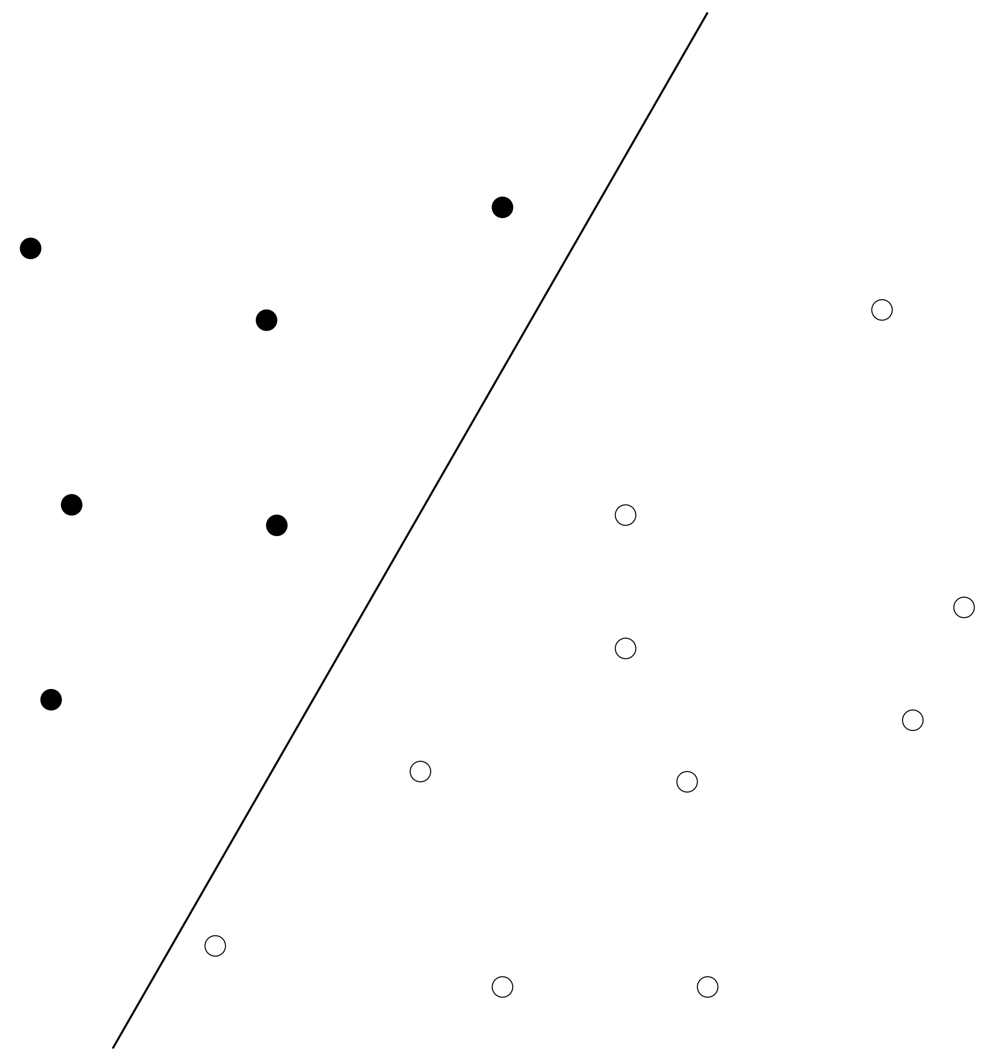
$$\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{N} \longrightarrow$$

Test performance

$$f_{\text{test}}(w) = \sum_{(x_i, y_i) \in \mathcal{D}_{\text{test}}} \ell(y_i, h_w(x_i))$$

Model fitting

Support vector machine (linear classification)



Given a set of points $\{v_1, \ldots, v_N\}$ with binary labels $s_i \in \{-1, 1\}$

Find hyperplane that strictly separates the tho classes

$$\begin{aligned} a^T v_i + b > 0 \quad \text{if} \quad s_i = 1 \\ a^T v_i + b < 0 \quad \text{if} \quad s_i = -1 \end{aligned} \xrightarrow{\text{(homogeneous)}} \begin{array}{c}\text{Equivalent to} \\ s_i \nu_i^T x \geq 1\end{array} \qquad \begin{array}{c}\nu_i = (v_i, 1) \\ x = (a, b)\end{array}$$

minimize $\quad \sum_{i=1}^N \max\{0, 1 - s_i \nu_i^T x\} + \gamma/2\|x\|_2^2$

quadratic term
(interpretation:
maximum margin)

Consensus SVM

Operator splitting form

$$\begin{array}{ll} \text{minimize} & \overbrace{\sum_{i=1}^{N} \max\{0, 1 - s_i \nu_i^T x\}}^{f} + \overbrace{\gamma/2\|z\|_2^2}^{g} \\ \text{subject to} & x = z \end{array}$$

split across workers j
with samples $\mathcal{D}_j$

Worker loss

$$f_j(x) = \sum_{i \in \mathcal{D}_j} \max\{0, 1 - s_i \nu_i^T x\}$$

Distributed model fitting ADMM

$$x_j^{k+1} = \mathbf{prox}_{\lambda f_j}(z^k - u_j^k)$$ **Local SVM**

$$z^{k+1} = \frac{N/\lambda}{1/\gamma + N/\lambda}(\bar{x}^{k+1} + \bar{u}^{k+1})$$ **Averaging**

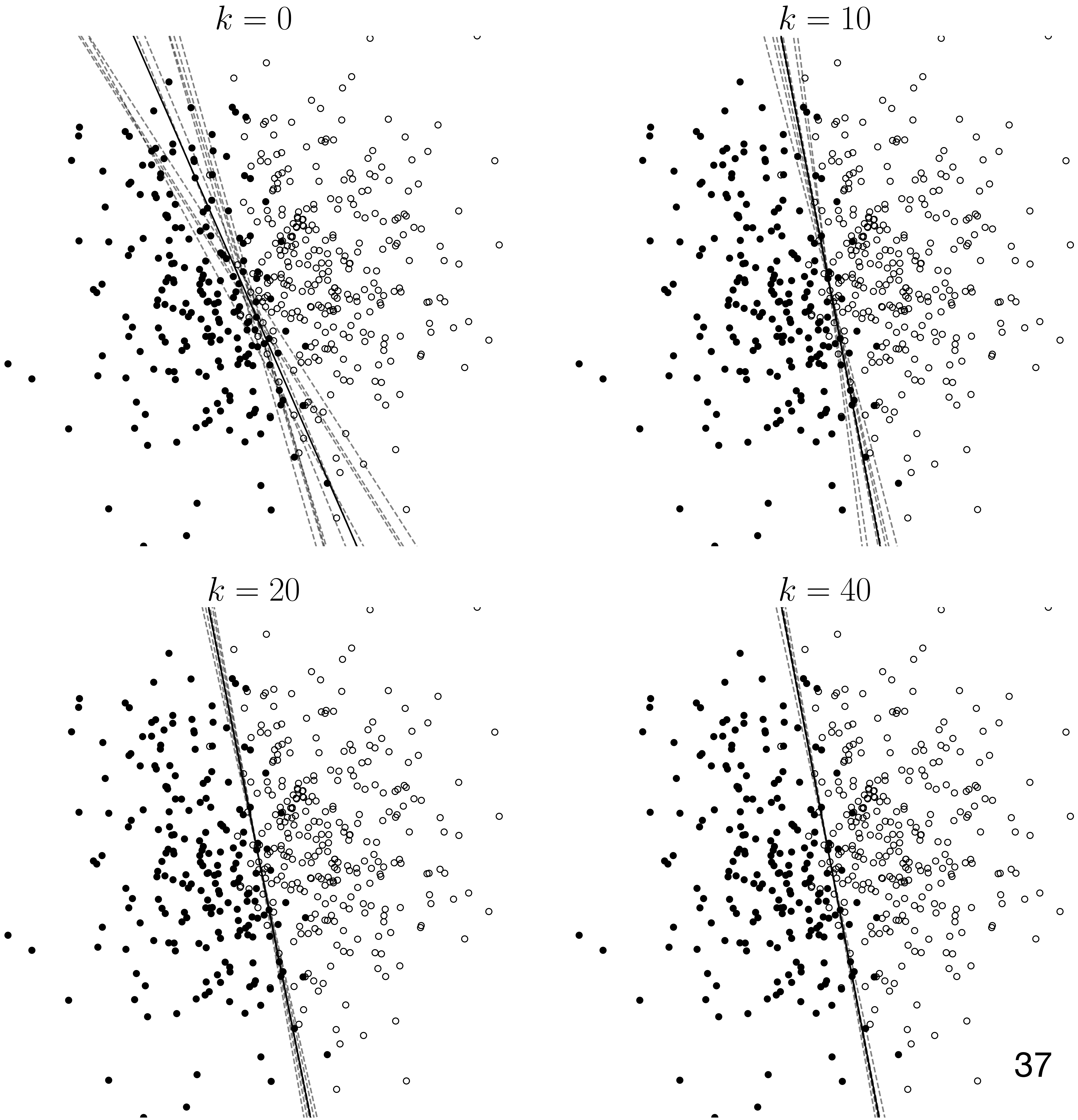
$$u_j^{k+1} = u_j^k + x_j^{k+1} - z^{k+1}$$ **Local update**

Consensus SVM

Linear classification

Dashed lines are local workers' hyperplanes

Optimal consensus hyperplane on test set after ~10 iterations



Conclusions

In ORF522, we learned to:

- **Model decision-making problems** across different disciplines as mathematical optimization problems.
- **Apply the most appropriate optimization tools** when faced with a concrete problem.
- **Implement** optimization algorithms and **prove** their **convergence.**
- **Understand** the limitations of optimization

Optimization cannot solve all our problems

It is just a mathematical model

But it can help us making better decisions

Thank you!

Bartolomeo Stellato