



NTNU

Norwegian University of
Science and Technology

TTK4135 – Optimization and Control

Spring 2021

Lecturer: Lars Imsland

Teaching Assistant: Joakim R. Andersen

6 Student Assistants

Learning Objectives

- Optimization – important concepts and theory
- Formulating an engineering problem into a mathematical optimization problem (modeling for optimization)
- Solving an optimization problem numerically
 - choosing the right algorithm for your problem,
 - use of (the right) optimization software,
 - some implementation of algorithmsfor a few important classes of optimization problems
- Applications in control engineering – model predictive control

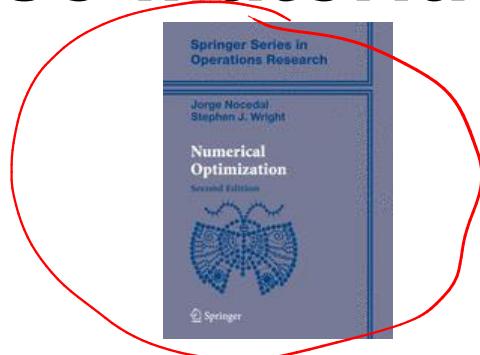
Numerical optimization is an incredibly versatile tool across most engineering domains

Course Information: General

- Description:
 - All course information is provided through Blackboard
 - Course description: <http://www.itk.ntnu.no/emner/ttk4135>
- Assignments and assessment (More information on Blackboard):
 - Exercises: 7 of 10 assignments must be approved
 - No extra assignments will be given, deadlines are absolute
 - Pay attention and make sure your delivered assignments are approved
 - Do not copy (kok)!
 - Helicopter lab: must be approved
 - Evaluation weighted 26% towards grade
 - In addition to deliver (group) reports, you have to give feedback on other groups' reports (**new**)
 - Matlab assessments (**new**)
 - 6 Matlab assessments, each counts 4% towards grade (pass/no pass)
 - Final exam (digital home exam)
 - Evaluation weighted 50% towards grade

Course Information: Course Material

- Lectures: Online on Blackboard
 - Will not cover the full curriculum in lectures
 - Will focus on difficult parts and build intuition
 - Will be recorded, and PDF made available afterwards
 - Online lecturing is new for lecturer
 - Constructive feedback welcome!
 - Ask questions in chat or use “raise hand”
 - If he is not recording, remind him!
- Course Material:
 - Numerical Optimization, J. Nocedal and S. J Wright, 2nd ed., Springer (ISBN-10: 0-387-30303-0 or ISBN-13: 987-0387-30303-1). Download [here](#) from campus or through VPN.
 - Errata on Blackboard
 - Note on Merging Optimization and Control, B. Foss and T. A. Heirung (Blackboard)
 - Note on Matrix Calculus, T. A. Heirung (Blackboard)



Course Information: Practical

- Grading
 - Final exam: 50%
 - Lab report (helicopter): 26% (group work)
 - Matlab assessments: 24% (6 individual tasks)
- Timetable
 - Lectures: Tuesday 08:15 - 10:00 on Blackboard
 - Assignment Sessions: Friday 08:15 - 10:00 on Blackboard
 - Assignment Sessions: Monday 16:15 - 17:00 on Blackboard
- Exam: May 28, 09:00 – 13:00 (?)
- Reference group!
- Video lectures from 2014: <https://mediasite.ntnu.no/Mediasite/Catalog/catalogs/ttk4135-v14>

Expected Background

- Linear algebra and real analysis
 - Quick recap next time (Also: Note on Blackboard + Exercise 0)
- Some numerical analysis (Newton's method)
- Basic control theory:
 - TTK4105 Control engineering
 - Advantage: TTK4115 Linear system theory

Tentative Lecture schedule

	TTK4135 Plan for Spring 2021					
Week no.	Lectures Tuesday 08:15-10:00 Online	Lectures Friday 08:15-10:00 Online	Helicopter project	Exercise out (Mon 15:00)	Help session Monday 16:15-17:00 Online	Exercise in (Wed 23:59)
2	Lecture 1 Introduction on optimization - N&W Ch.1	Lecture 2 Optimality conditions - N&W Ch. 12.1-12.2		0: Matrix Calculus, 1: KKT		
3	Lecture 3 Optimality conditons and linear algebra - N&W Ch.12.3, 12.5 (12.8, 12.9)	Lecture 4 Linear Programming - N&W Ch.13.1-13.5		2: LP	0, 1, 2	
4	Lecture 5 Linear Programming - N&W Ch.13.1-13.5	Lecture 6 Quadratic programming - N&W Ch.15.3-15.5, 16.1-2,4-5		3: LPQP	2, 3	0, 1
5	Lecture 7 Quadratic Programming - N&W Ch.15.3-15.5, 16.1-2,4-5	Lecture 8 Open loop dynamic optimization - MPC note Ch.3-3.2	Helicopter Lab week	4: QP	3, 4	2
6	Lecture 9 Model predictive control - MPC note Ch.3.3-4.2.1	Lecture 10 Model predictive control - MPC note Ch.4.2.2-4.3.1	Helicopter Lab week	5: OLMPC	4, 5	3
7	Lecture 11 Linear quadratic control - MPC note Ch.4.3.2-4.4	Lecture 12 Linear quadratic control - MPC note repetition and 4.6	Helicopter Lab week	6: MPCLQR	5	4
8	Lecture 13 Unconstrained optimization - N&W Ch.2.1-2.2	No lecture	Helicopter Lab week		5, 6	

- Updated schedule will be available on Blackboard



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 1

Optimization: What and Why?

Spring 2021

Lecturer: Lars Imsland

Purpose of Lecture

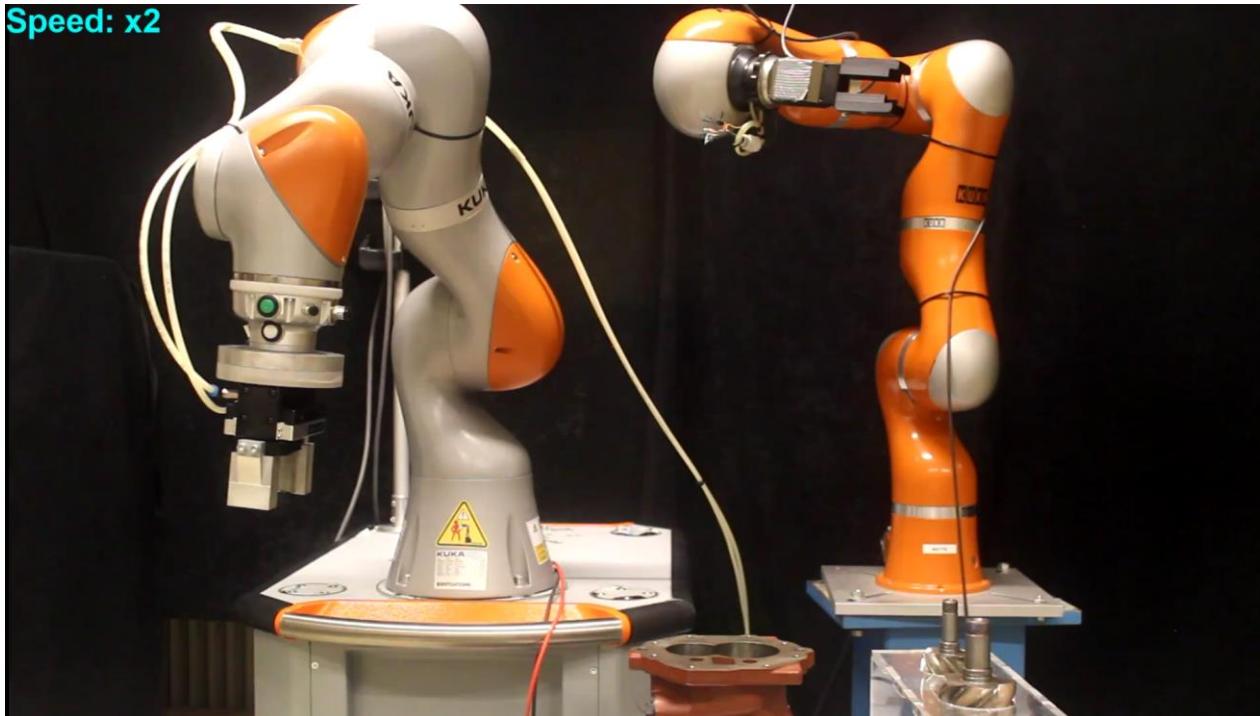
- Brief Timeline & Motivation
- Formulation of optimization problems, classes of optimization problems
- Definition of important terms
 - Convexity and non-convexity
 - Global vs. local solution
 - Constrained vs. unconstrained problems
 - Feasible set

Reference: Chapter 1 Nocedal & Wright

Brief Timeline

~1600 BC	Ancient Geometry: Babylonian method for solving $x^2 + bx = c$
~300 BC	Ancient Geometry: Euclid's minimal distance between point and line
~200s	Iterative approaches: Han Dynasty methods for solving $\sum_{i=0}^3 a_i x^i = 0$
~900s	Modern algebra and arithmetics: Muhammad Al-Khwarizmi ("Algorismi") gives various root solving methods
1600s	Basis of Calculus of Variations: Newton's Body of minimal resistance, Bernoulli's Brachistochrone problem
1700s	Calculus of Variations and combinatorial optimization: Maupertius' Principle of Least Action, Samuel König's optimal honeycomb
1800s	First "Optimization algorithms": Hamilton-Jacobi Equation, Extreme Value Theorem, Rolle's Theorem, Cauchy's Gradient Descent
1900-1957	Rigorous theory and applications: Minkowski's Convex Sets, Hancock's Theory of Minima and Maxima, Kantorevich's Linear Optimization Problems, Dantzig's Simplex method, Neumann and Morgenstern's Dynamic Programming, Karush-Kuhn-Tucker's Optimality Conditions, Bellman's Optimality principle, Pontryagin's Maximum Principle
1950+	Optimization is applied to economics, agriculture, space travel, social media, robots, manufacturing, art and everything in between

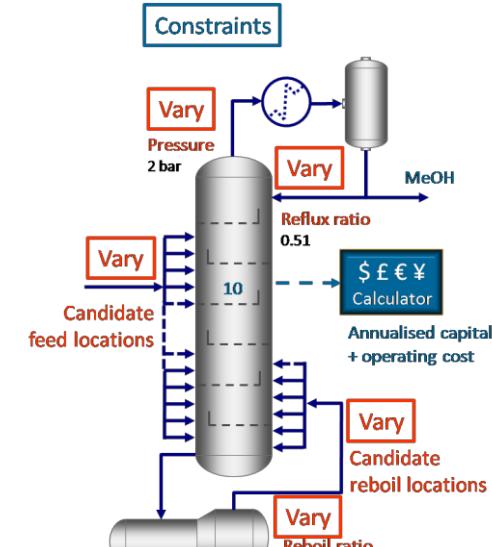
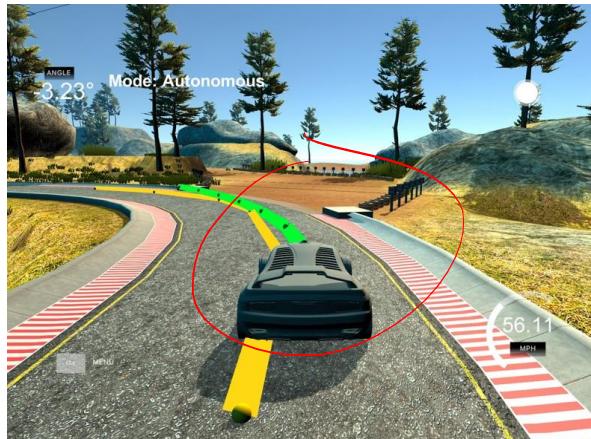
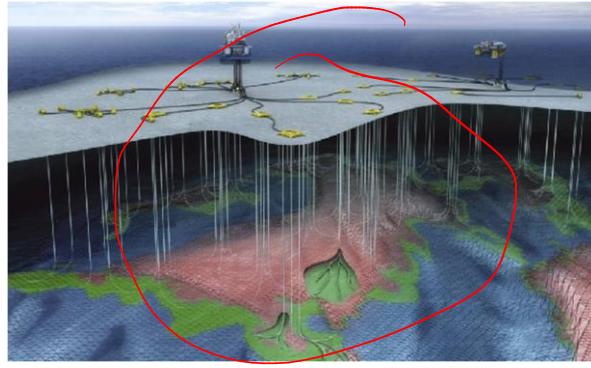
Modern Applications: Motion Control



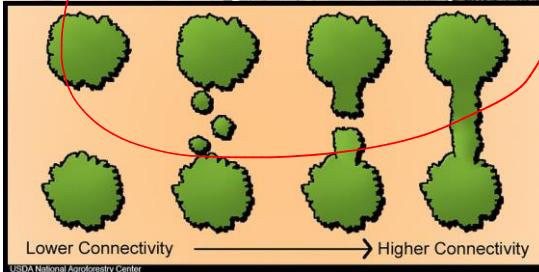
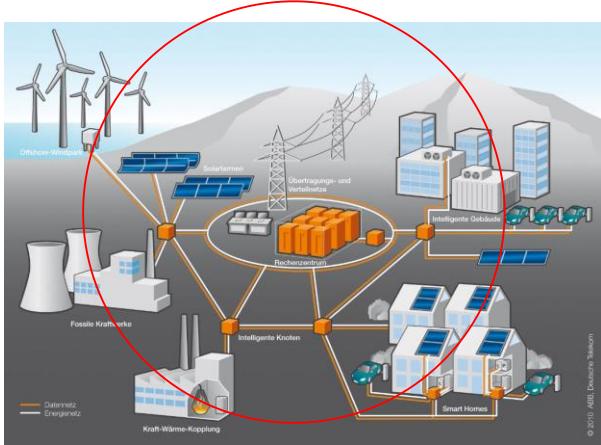
LBR iiwa avoiding collision with static LBR 4+ while performing assembly

Source: Y. Pane, M. H. Arbo, E. Aertbeliën, W. Decré, "A System Architecture for CAD-Based Robotic Assembly with Sensor-Based Skills", T-ASE 2019

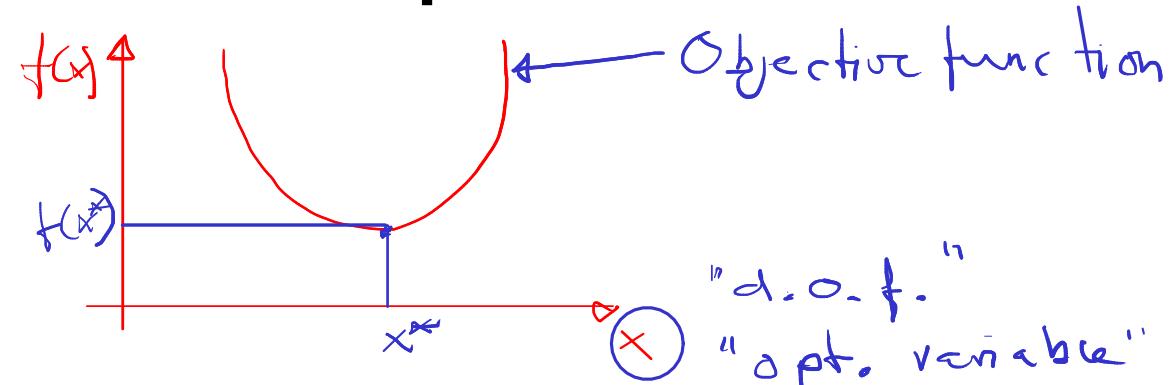
Control Applications: Model Predictive Control for all domains



Lots of other applications



What is optimization?



What characterizes an optimum?

Necessary conditions for optimality of x^* :

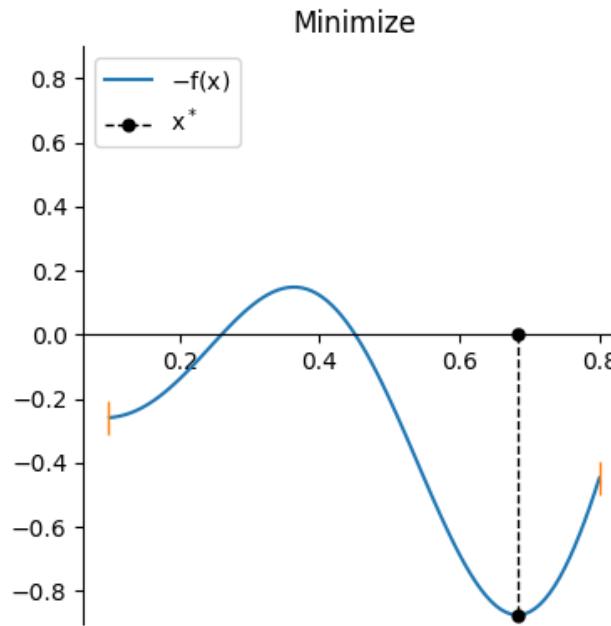
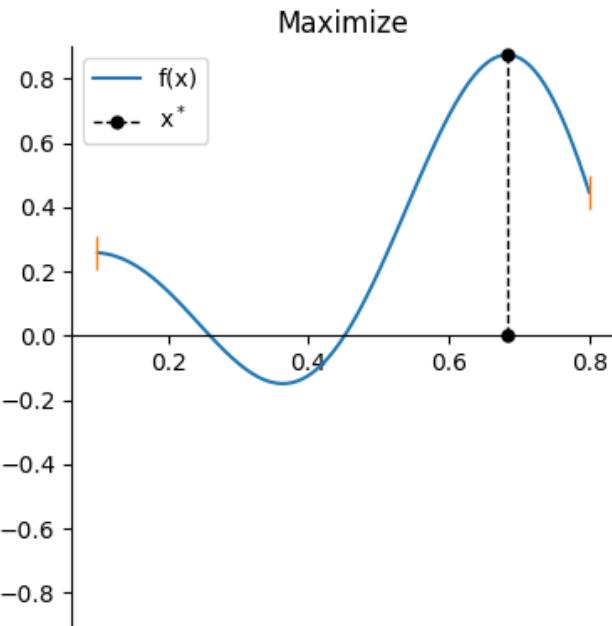
1. order : $f'(x^*) = 0$

2. order : $f''(x^*) \geq 0$

Unconstrained opt.

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

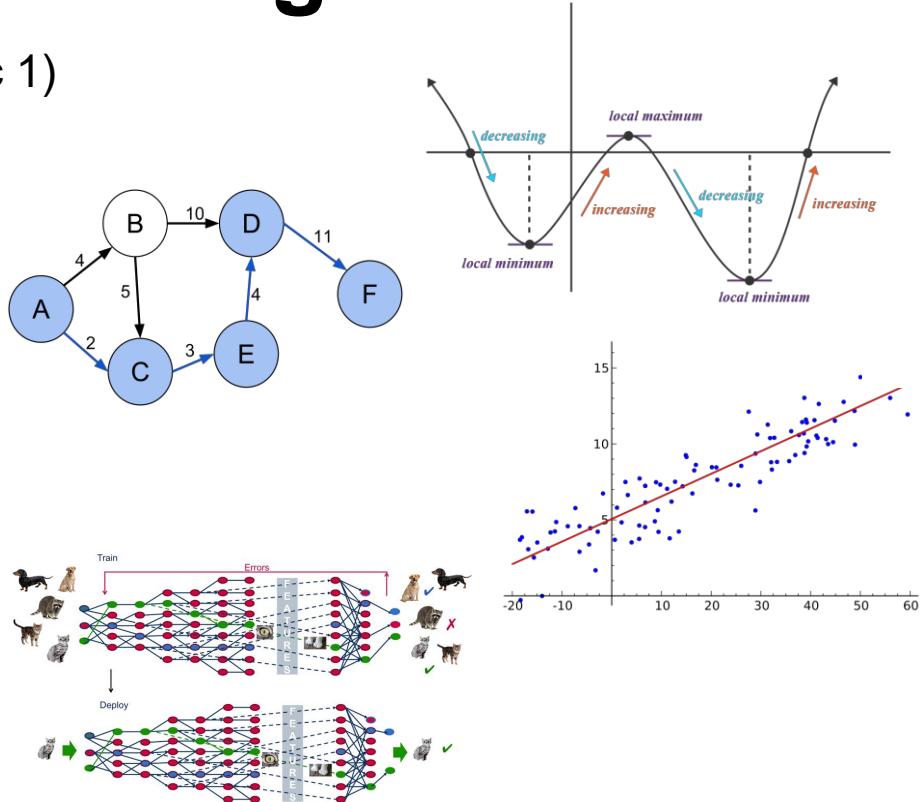
Minimization or Maximization?



Convention this course: Minimization!

Optimization – A recurring friend

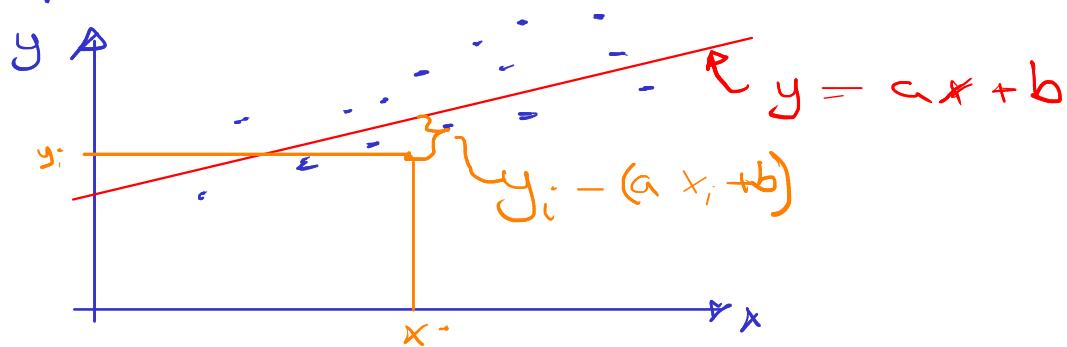
- Finding max and min of a function (Calc 1)
- The Lagrange Method (Calc 2)
- Algorithms course (Shortest path, dynamic programming, max flow, travelling salesman, etc)
- Statistics (Least-squares, data fitting)
- Machine Learning (Gradient descent)
- (And many applications in control...)



Example optimization problem: Least Squares

x_i : Temperature day i
 y_i : # ice creams sold day i

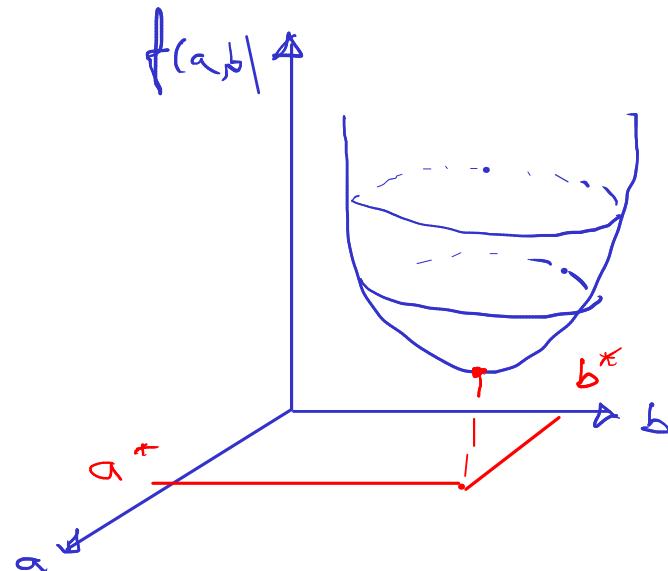
Previous data



Find a, b :

$$\min_{a, b} \sum_{i=1}^n (y_i - ax_i - b)^2$$

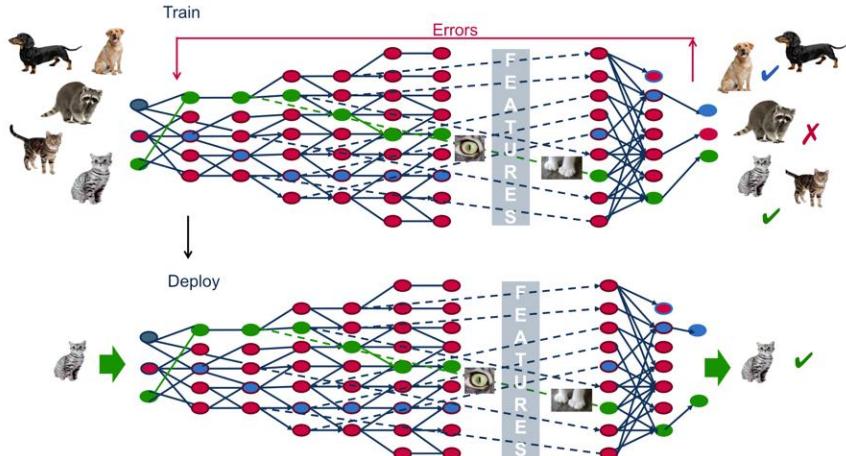
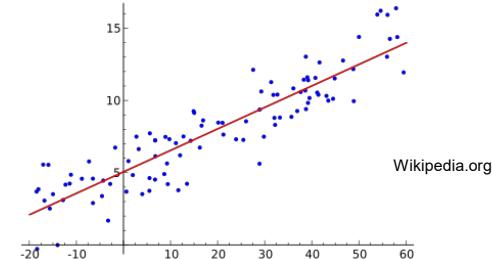
$f(a, b)$



Predict for $y = a^* x + b^*$

Example: Machine Learning

- Learn, and make predictions, from data
- Linear regression is the most basic ML algorithm, solved using optimization
 - “Least squares”, Ch. 10, N&W
- In a similar fashion: ML, neural networks, deep learning etc. are “trained” using “gradient descent” algorithms
 - Topic of Ch. 2-10, N&W



Constrained optimization problems

Unc constr. opt.

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

Constrained opt.

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

$$\text{s.t. } c_i(x) = 0, i \in \mathcal{E}$$

$$c_i(x) \geq 0, i \in \mathcal{I}$$

Feasible set:

$$\Sigma = \{x \in \mathbb{R}^n \mid c_i(x) = 0, i \in \mathcal{E} \text{ and } c_i(x) \geq 0, i \in \mathcal{I}\}$$

General Optimization Problem

- Example:

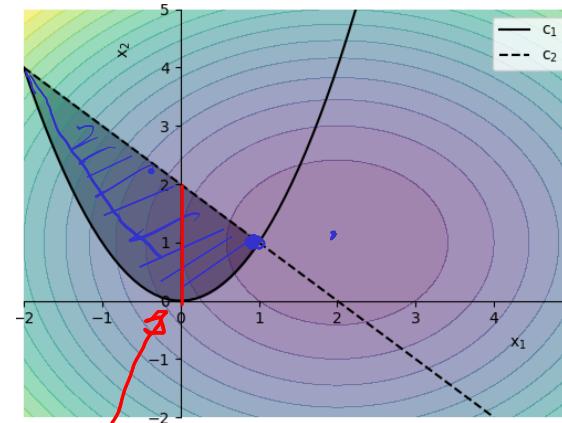
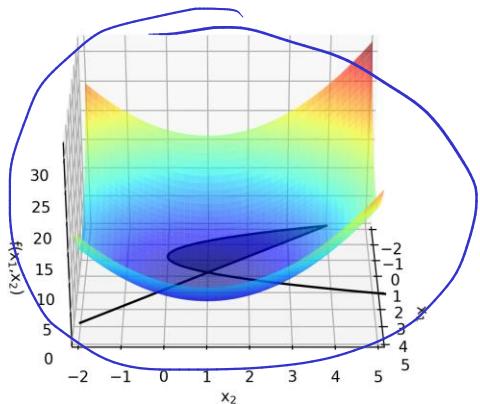
$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to}$$

$$c_i(x) = 0, \quad i \in \mathcal{E};$$
$$c_i(x) \geq 0, \quad i \in \mathcal{I}.$$

$\{\mathcal{E}, \mathcal{I}\}$

$$\min (x_1 - 2)^2 + (x_2 - 1)^2$$

$$\text{subject to} \quad x_1^2 - x_2 \leq 0, \quad \leftarrow c_1$$
$$x_1 + x_2 \leq 2. \quad \leftarrow c_2$$



- What if we add equality-constraint $x_1 = 0$?

Linear Programming

$$\min c^T x$$

x

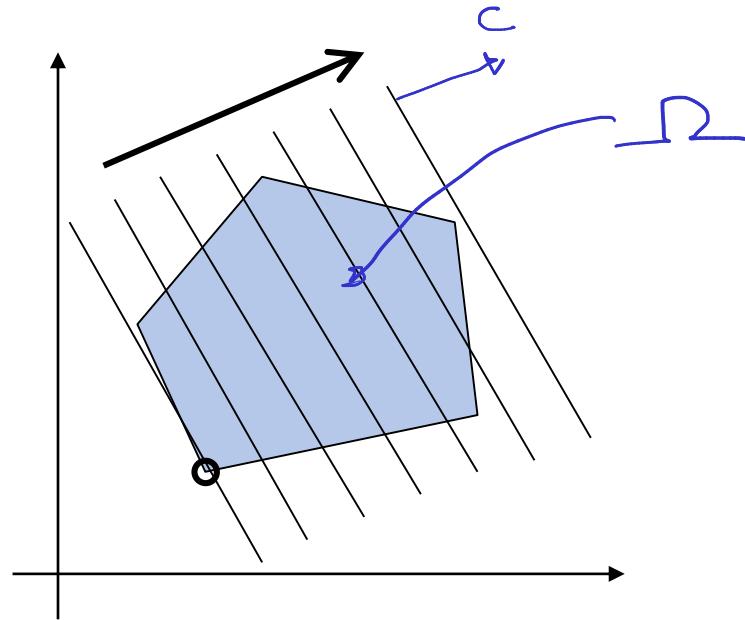
s.t.

$$a_i^T x = b_i, i \in \mathcal{E}$$

$$a_i^T x \geq b_i, i \in \mathcal{I}$$

- (-)

$$\boxed{\min c^T x \text{ s.t. } A x \leq b \\ x \geq 0}$$

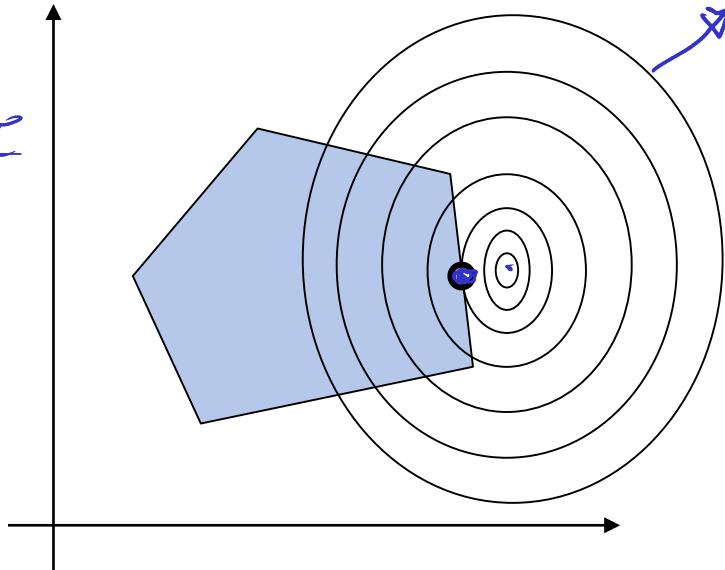


Quadratic Programming

$$\min_x \quad q(x) = \frac{1}{2} x^T G x + d^T x$$

$$\text{s.t. } a_i^T x = b_i, i \in \mathcal{E}$$

$$a_i^T x \geq b_i, i \in \mathcal{I}$$



LP Example: Farming

- A farmer wants to grow apples (A) and bananas (B)
- He has a field of size 100 000 m²
- Growing 1 tonne of A requires an area of 4 000 m², growing 1 tonne of B requires an area of 3 000 m²
- A requires 60 kg fertilizer per tonne grown, B requires 80 kg fertilizer per tonne grown
- The profit for A is 7000 per tonne (including fertilizer cost), the profit for B is 6000 per tonne (including fertilizer cost)
- The farmer can legally use up to 2000 kg of fertilizer
- The farmer wants to maximize his profits



Formulating a LP optimization problem

x_1 : # tonnes A

x_2 : # tonnes B

Objective $7000 \cdot x_1 + 6000 \cdot x_2$

constraints

$$\begin{aligned} 4000x_1 + 3000x_2 &\leq 100000 \\ 60x_1 + 80x_2 &\leq 2000 \end{aligned}$$

$$\min_{x_1, x_2} -7000x_1 - 6000x_2$$

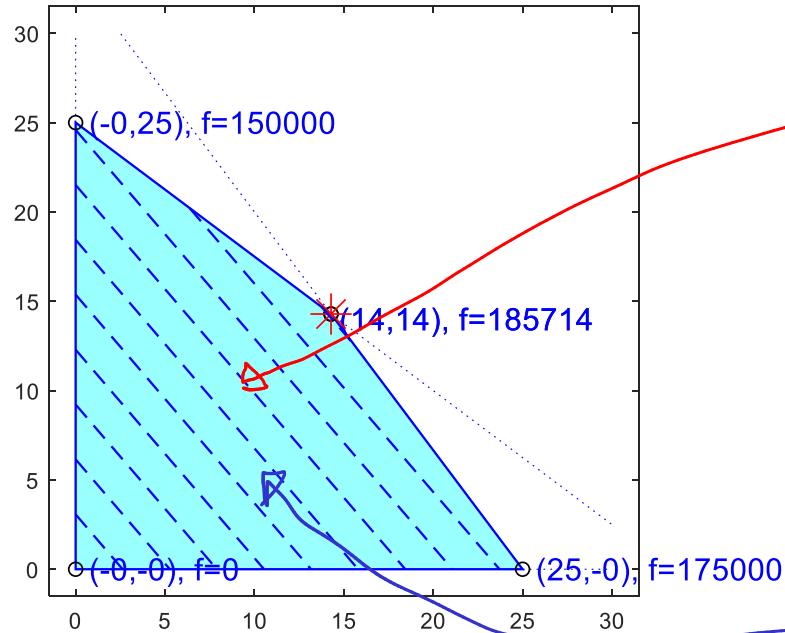
s.t.

$$x_1 \geq 0$$

$$x_2 \geq 0$$

$$\begin{cases} \min c^T x \text{ s.t. } Ax \leq b, x \geq 0 \\ c = [-7000, -6000] \\ A = \begin{bmatrix} 4000 & 3000 \\ 60 & 80 \end{bmatrix}, b = \begin{bmatrix} 100000 \\ 2000 \end{bmatrix} \end{cases}$$

Farming Example: Geometric Interpretation and Solution



$$\begin{aligned} & \max_{x_1, x_2} && 7000x_1 + 6000x_2 \\ \text{subject to: } & && 4000x_1 + 3000x_2 \leq 100000 \\ & && 60x_1 + 80x_2 \leq 2000 \\ & && x_1 \geq 0 \\ & && x_2 \geq 0 \end{aligned}$$



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 2

Optimality Conditions for Constrained Optimization: KKT Conditions

Lecturer: Lars Imsland

Purpose of Lecture

- Optimization problems and Convexity
- Motivating Examples for KKT Conditions
- KKT Conditions

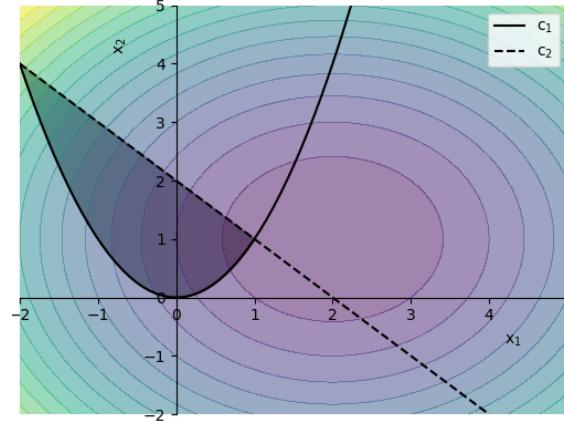
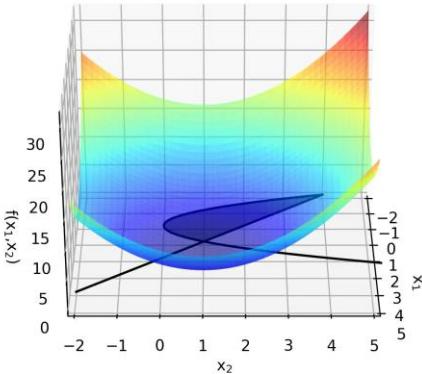
Reference: Chapter 12.1, 12.2 in N&W

General Optimization Problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned}$$

- Example:

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \quad \text{subject to} \quad \begin{aligned} x_1^2 - x_2 &\leq 0, \\ x_1 + x_2 &\leq 2. \end{aligned}$$



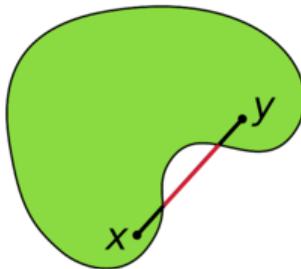
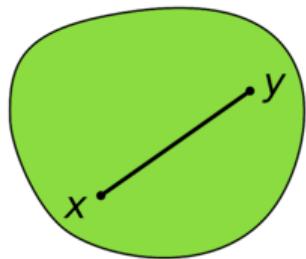
- What if we add equality-constraint $x_1 = 0$?

Definitions: Feasible Set and Solutions

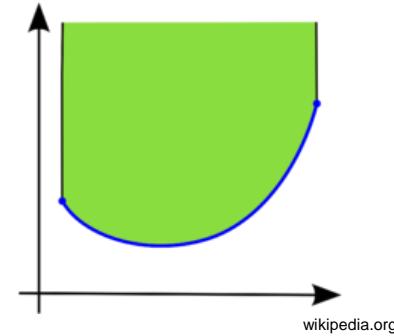
$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases} \quad (\mathbf{P})$$

- Feasible set: $\Omega = \{x \in \mathbb{R}^n \mid c_i(x) = 0, i \in \mathcal{E}; c_i(x) \geq 0, i \in \mathcal{I}\}$
- A vector x^* is a *global solution* to (P) if $x^* \in \Omega$ and $f(x) \geq f(x^*)$ for $x \in \Omega$.
- A vector x^* is a *local solution* to (P) if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that $f(x) \geq f(x^*)$ for $x \in \mathcal{N} \cap \Omega$.
- A vector x^* is a *strict local solution* to (P) if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that $f(x) > f(x^*)$ for $x \in \mathcal{N} \cap \Omega$ with $x \neq x^*$.

Convexity: An important property



If the line segment between any two points within a **set** is inside the set, the set is **convex**.



A **function** is **convex** if the epigraph is a convex set.

- A convex optimization problem: Both $f(x)$ and the feasible set convex
- Convex optimization problems are preferable!
 - For convex optimization problems, **every local minimum is also a global minimum**. **Sufficient to search for a local minimum!** Which is much easier than searching for global minimum.
 - For many convex optimization problems, it is easy to find derivatives, exploit structure, etc. making them efficient to solve.
 - They typically have “guaranteed complexity”.

Convexity: Conditions

- When is an optimization problem convex?

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

- Conditions for a convex optimization problem:

- $f(x)$ is a convex function:

$$\forall x, y \in \Omega, \forall \alpha \in [0, 1] : f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

- The **feasible set** $\Omega = \{x \in \mathbb{R}^n | c_i(x) = 0, i \in \mathcal{E}, c_i(x) \geq 0, i \in \mathcal{I}\}$ is convex:

$$\forall x, y \in \Omega, \forall \alpha \in [0, 1] : \alpha x + (1 - \alpha)y \in \Omega$$

- When is the feasible set convex?

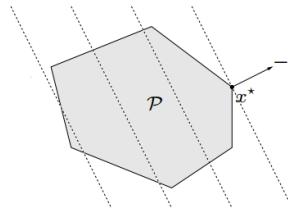
- $c_i(x), i \in \mathcal{E}$ are linear
 - $c_i(x), i \in \mathcal{I}$ are concave

Convex problems: Any local solution is global

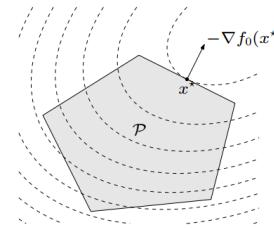
Types of Constrained Optimization Problems

- Linear programming
 - Convex problem
 - Feasible set polyhedron
- Quadratic programming
 - Convex problem if $P \geq 0$
 - Feasible set polyhedron
- Nonlinear programming
 - In general non-convex!

$$\begin{aligned} & \min c^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

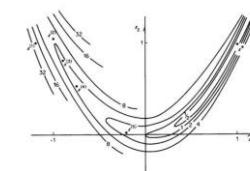


$$\begin{aligned} & \min \frac{1}{2} x^T Px + q^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$



$$\begin{aligned} & \min f(x) \\ \text{subject to } & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



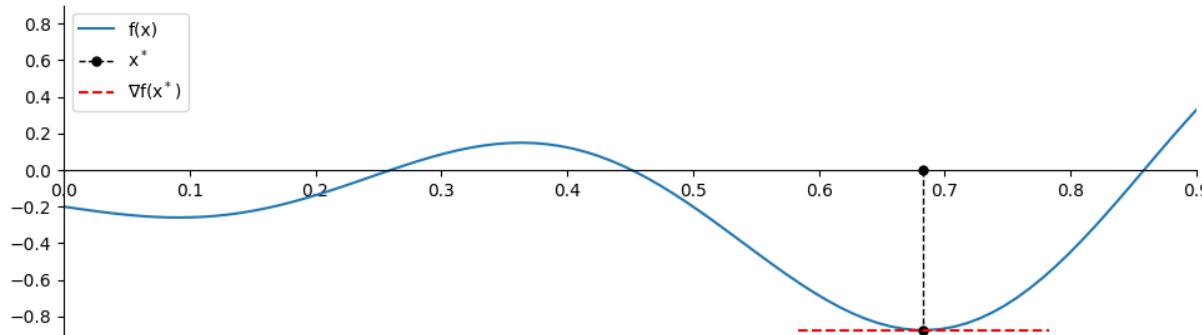
$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{subject to } & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

Necessary Conditions for Unconstrained Optimization

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

Theorem 2.2 (First-Order Necessary Conditions).

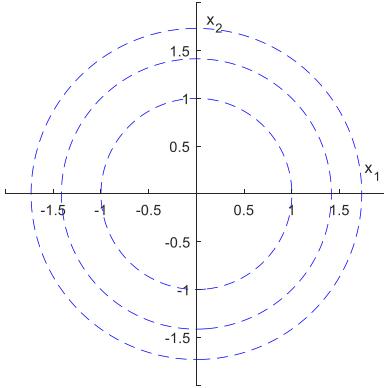
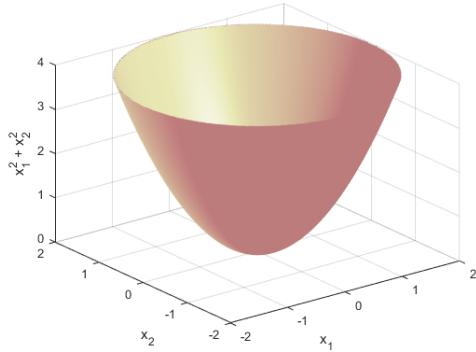
If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^* , then $\nabla f(x^*) = 0$.



- What about constrained problems?

Contours/level curves, gradients and directions

$$f(x_1, x_2) = x_1^2 + x_2^2$$



Contours/level curves, gradients and directions

Necessary conditions for optimality

KKT Conditions

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned}$$

- Lagrangian

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

- Note: One Lagrangian multiplier λ_i for each constraint
- Necessary conditions for x^* to be a solution (under some mild regularity conditions):

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

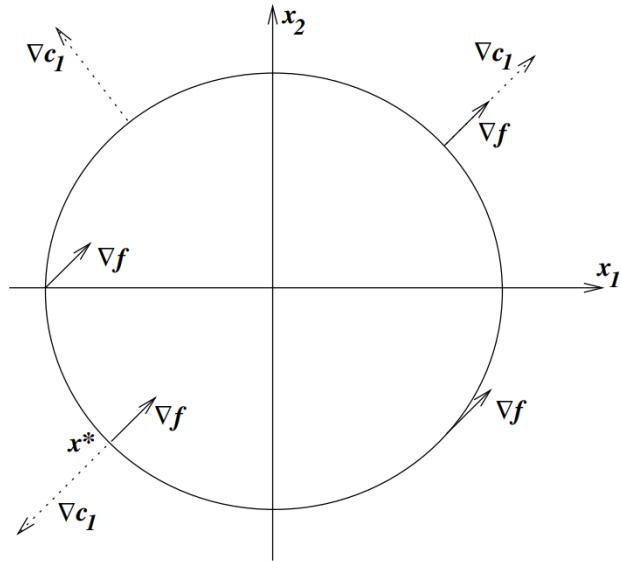
- These are called the *KKT conditions*

We will not prove KKT, but study 3 motivating cases (Ex. 12.1-12.3 in N&W)

Looking for points where there are no descent directions...
...as these are potential local solutions

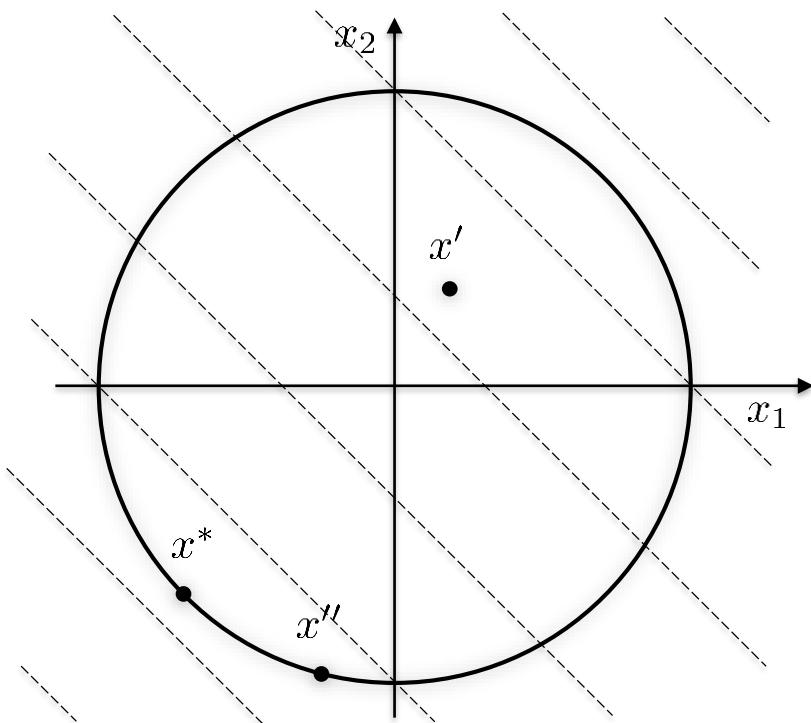
Case I: Equality constraint (Example 12.1)

$$\min x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 2 = 0$$



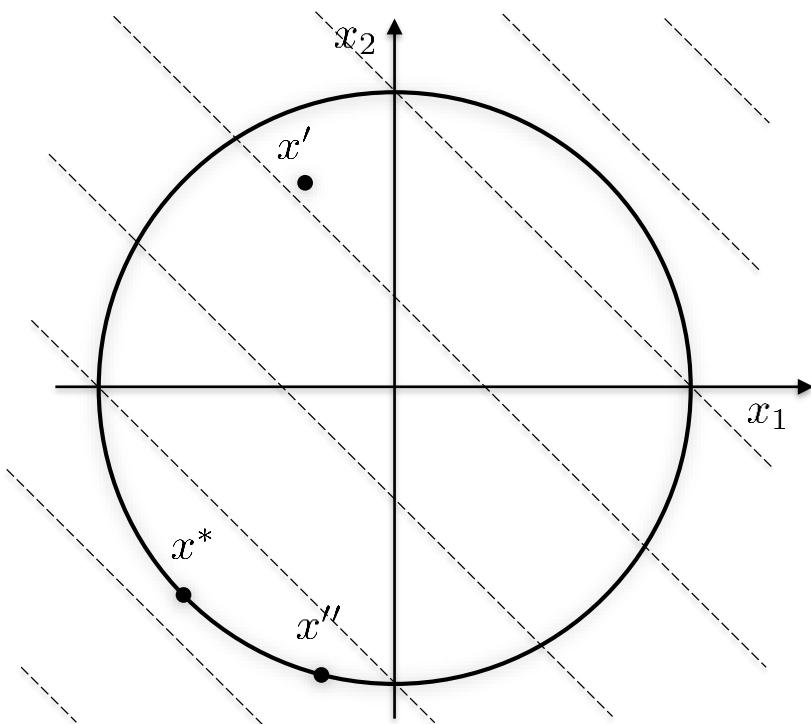
Case II: Inequality constraint (Example 12.2)

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0$$



Case II: Inequality constraint (Example 12.2)

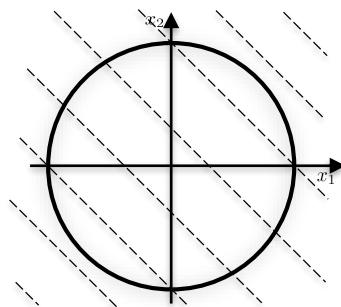
$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0$$



Active Set

The active set $\mathcal{A}(x)$ at any feasible point x consists of the equality constraint indices from \mathcal{E} together with the indices of the inequality constraints i for which $c_i(x) = 0$. That is,

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}$$



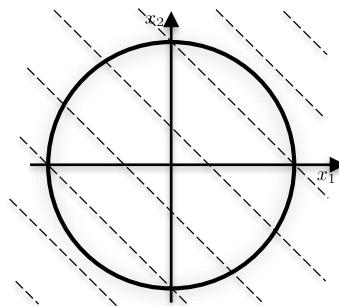
$$\min x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 2 = 0$$

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0$$

Set of Feasible Directions

Given a feasible point x and the active constraint set $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(x)$ is

$$\mathcal{F}(x) = \left\{ d \mid \begin{array}{ll} d^\top \nabla c_i(x) = 0, & \text{for all } i \in \mathcal{E}, \\ d^\top \nabla c_i(x) \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\}$$

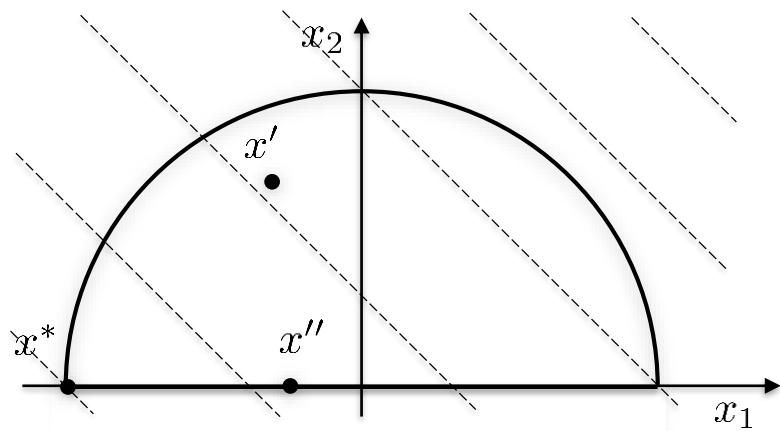


$$\min x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 2 = 0$$

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0$$

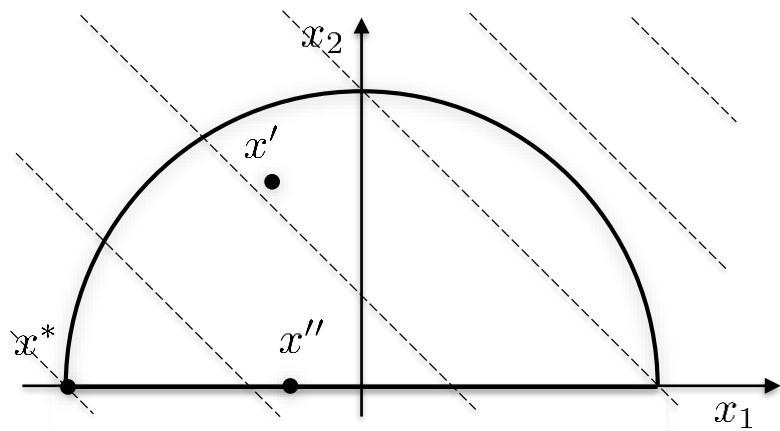
Case III: Two inequality constraints (Example 12.3)

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0$$

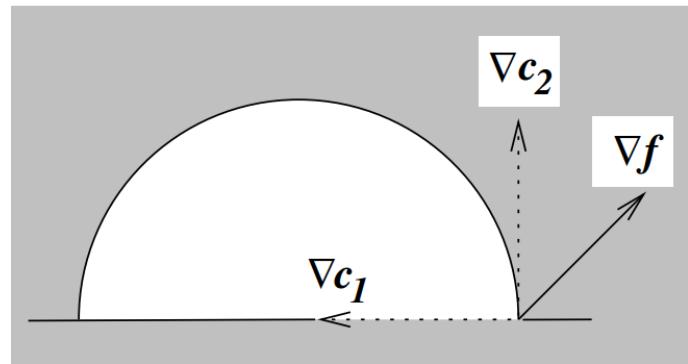
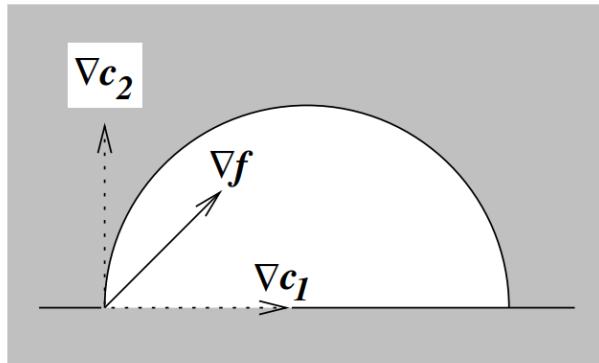


Case III: Two inequality constraints (Example 12.3)

$$\min x_1 + x_2 \quad \text{s.t.} \quad 2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0$$



Case III: Two inequality constraints (Example 12.3)



$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}, \end{cases} \quad (12.1)$$

Theorem 12.1 (First-Order Necessary Conditions).

Suppose that x^* is a local solution of (12.1), that the functions f and c_i in (12.1) are continuously differentiable, and that the LICQ holds at x^* . Then there is a Lagrange multiplier vector λ^* , with components λ_i^* , $i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*)

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \quad (12.34a)$$

$$c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E}, \quad (12.34b)$$

$$c_i(x^*) \geq 0, \quad \text{for all } i \in \mathcal{I}, \quad (12.34c)$$

$$\lambda_i^* \geq 0, \quad \text{for all } i \in \mathcal{I}, \quad (12.34d)$$

$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all } i \in \mathcal{E} \cup \mathcal{I}. \quad (12.34e)$$

Linear Independence Constraint Qualification (LICQ)

Given the point x and the active set $\mathcal{A}(x)$, we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$ is linearly independent.

Why are KKT-conditions so important?

- KKT conditions can be used to solve nonlinear programming problems, but only for *very simple* problems
- But: Most algorithms for constrained optimization look for candidate solutions that fulfill KKT conditions
 - These are iterative algorithms that stop when KKT conditions fulfilled
- And also:
 - When faced with an optimization problem that you don't know how to handle, write down the optimality conditions
 - Often you can learn about a problem by examining the properties of its optimal solutions
- And finally:
 - The Lagrange multipliers tell you the 'hidden cost' of constraints

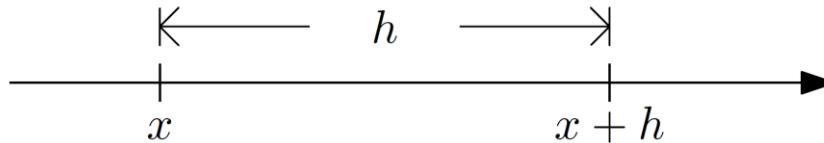
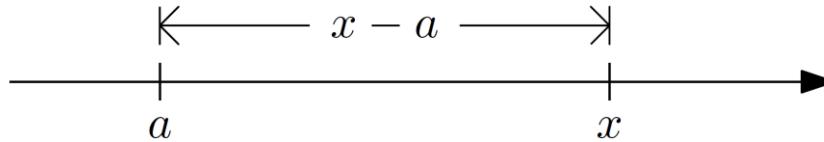
Taylor Expansions

- From calculus?

$$f(x) = f(a) + (x - a)f'(a) + \frac{(x - a)^2}{2}f''(a) + O(\|x - a\|^3)$$

- This course:

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + O(\|h\|^3)$$



$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$

KKT-conditions (First-order necessary conditions):
If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

$$\begin{aligned}\nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}.\end{aligned}$$



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 3

Optimality Conditions for Constrained Optimization (KKT & 2nd order)

Lecturer: Lars Imsland

Purpose of Lecture

- Repetition of definitions:
 - Gradient, Hessian
 - Feasible Set
 - Local vs Global Optima
- Conditions for optimality
 - **KKT conditions** (1st order, necessary conditions)
 - Examples
 - Constraint qualifications
 - 2nd order conditions (necessary and sufficient)
- Reference: Chapter 12.3, 12.5 (12.8, 12.9) in N&W

Administrative

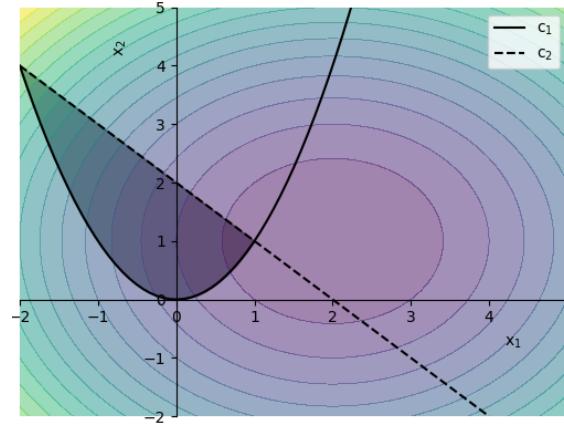
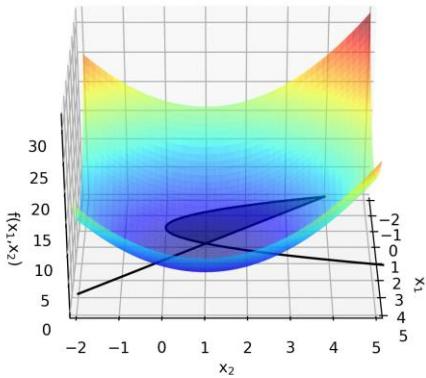
- We need more members in the reference group. Please volunteer in the chat!
- The first Matlab assessment is now active
 - Do not be intimidated by the amount of text. The task is probably simpler than you think.
 - You have unlimited attempts. You can discuss the problem with your classmates.
 - It is not obligatory, but will count 4% (...) towards your grade

General Optimization Problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned}$$

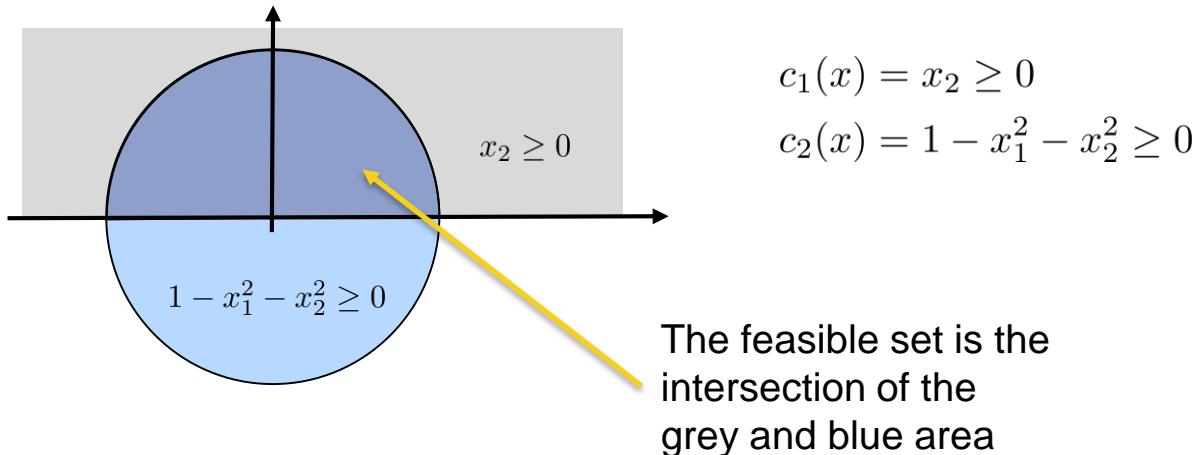
- Example:

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \quad \text{subject to} \quad \begin{aligned} x_1^2 - x_2 &\leq 0, \\ x_1 + x_2 &\leq 2. \end{aligned}$$



Feasible Set

Feasible set: Collection of all points that satisfy all constraints:



$$\text{Feasible set: } \Omega = \{x \in \mathbb{R}^n \mid c_i(x) = 0, i \in \mathcal{E}; c_i(x) \geq 0, i \in \mathcal{I}\}$$

Gradient and Hessian

- The *gradient* (or first derivative) of a function $f(x)$ of several variables is defined as

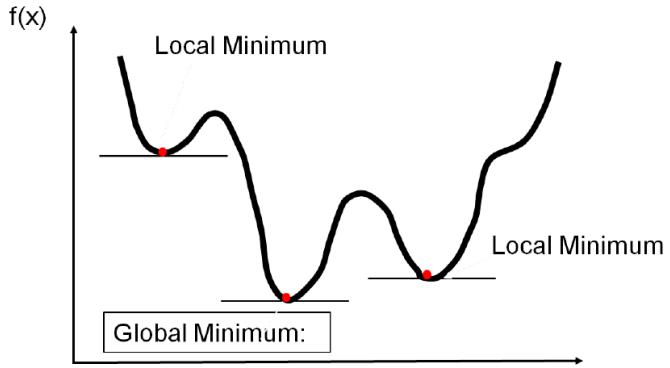
$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

- The matrix of second partial derivatives of $f(x)$ is known as the *Hessian*, and is defined as

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- We will frequently use $\nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*)$, the *Hessian of the Lagrangian*

Local and Global Optima



$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases} \quad (\mathbf{P})$$

A point x^* is a *global solution* to (\mathbf{P}) if $x^* \in \Omega$ and $f(x) \geq f(x^*)$ for $x \in \Omega$.

A point x^* is a *local solution* to (\mathbf{P}) if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} of x^* such that $f(x) \geq f(x^*)$ for $x \in \mathcal{N} \cap \Omega$.

Convex optimization problems: local solutions are global.

Unconstrained Optimality Conditions

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

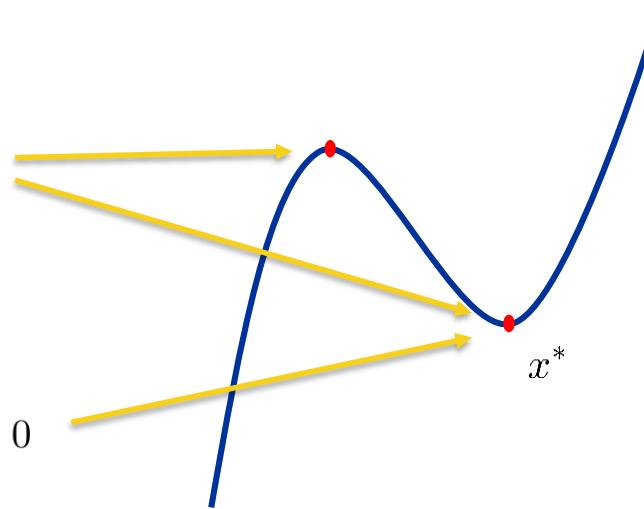
We want to test a point x^* for local optimality:

Necessary condition:

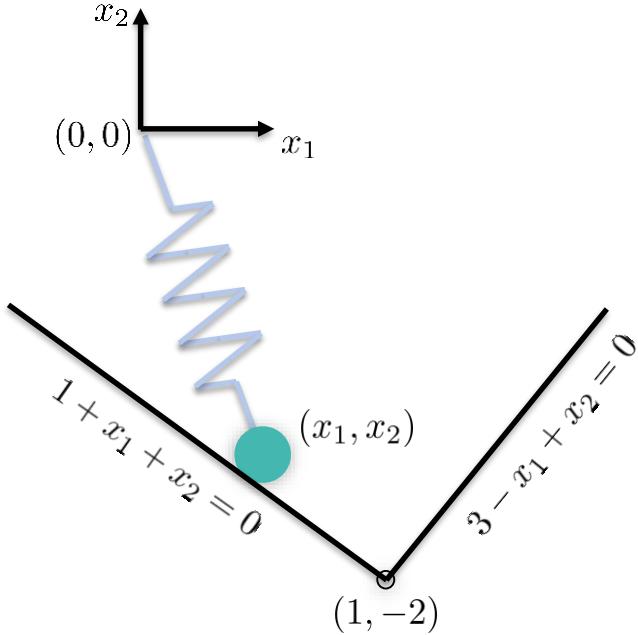
$$\nabla f(x^*) = 0 \quad (\text{stationarity})$$

Sufficient condition:

$$x^* \text{ stationary and } \nabla^2 f(x^*) > 0$$



Simple example: Ball and Spring

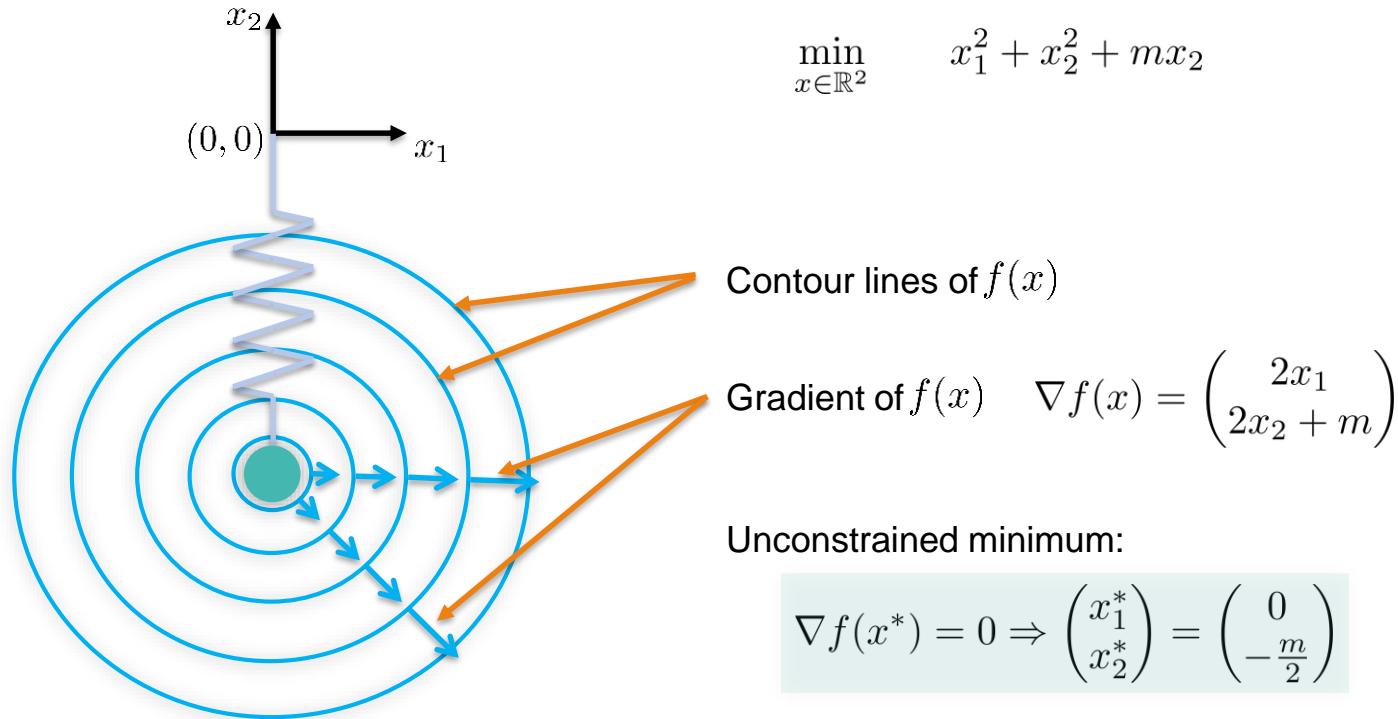


To find position at rest,
minimize potential energy!

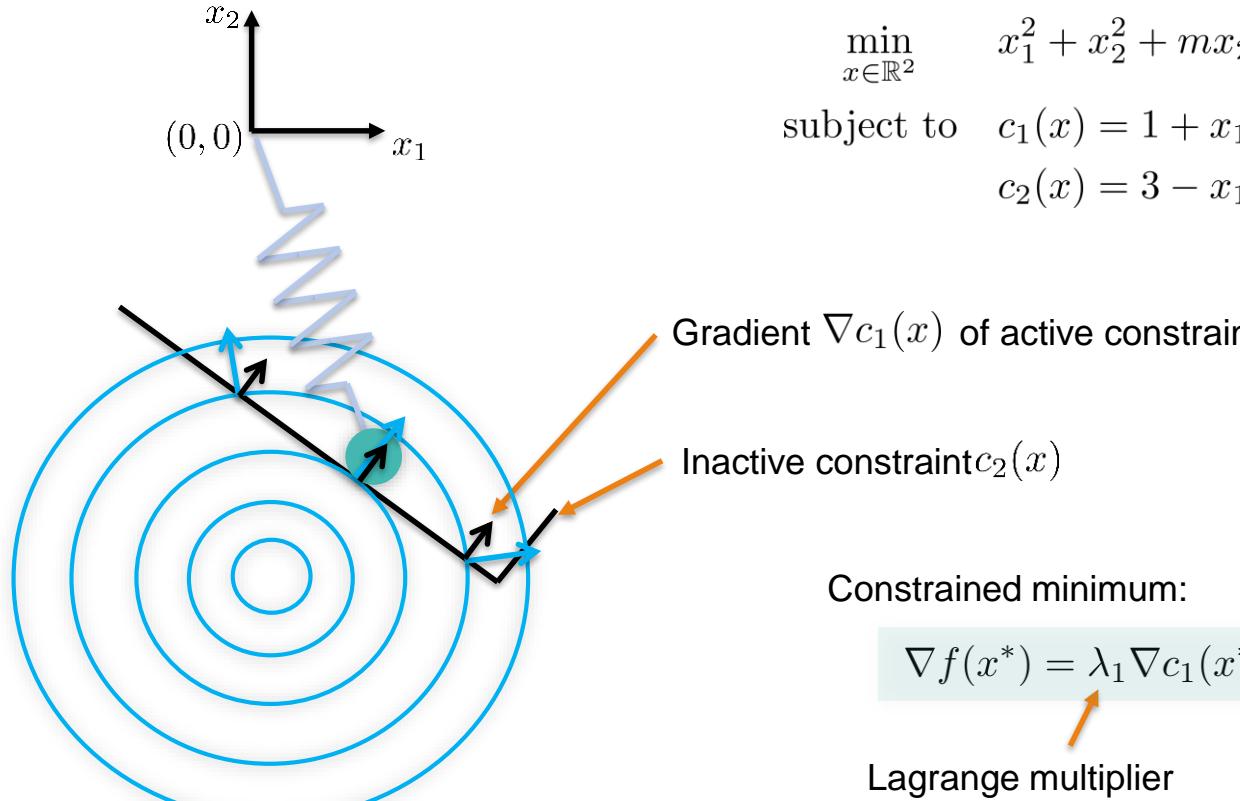
$$\min_{x \in \mathbb{R}^2} \underbrace{x_1^2 + x_2^2}_{\text{spring}} + \underbrace{mx_2}_{\text{gravity}}$$

subject to $c_1(x) = 1 + x_1 + x_2 \geq 0$
 $c_2(x) = 3 - x_1 + x_2 \geq 0$

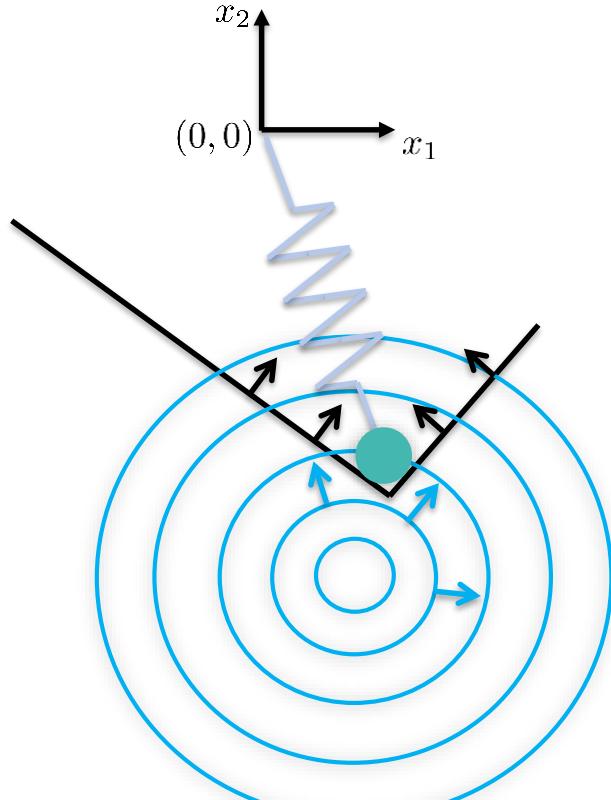
Ball and Spring: No Constraints



Ball and Spring: With one (active) constraint



Ball and Spring: With two active constraints



$$\min_{x \in \mathbb{R}^2} x_1^2 + x_2^2 + mx_2$$

subject to $c_1(x) = 1 + x_1 + x_2 \geq 0$
 $c_2(x) = 3 - x_1 + x_2 \geq 0$

Constrained minimum at “equilibrium of forces”:

$$\nabla f(x^*) = \lambda_1 \nabla c_1(x^*) + \lambda_2 \nabla c_2(x^*), \quad \lambda_1, \lambda_2 \geq 0$$

“Constraint forces”

The Lagrangian

For constrained optimization problems, introduce modification of objective function:

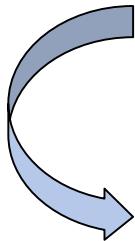
$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

- Multipliers for *equality* constraints may have both signs in a solution
- Multipliers for *inequality* constraints cannot be negative (cf. shadow prices)
- For (inequality) constraints that are *inactive*, multipliers are zero

KKT Conditions (Theorem 12.1)

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

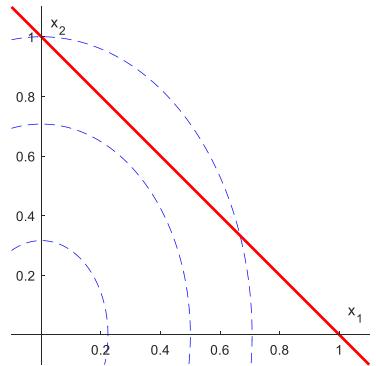
$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, && \left. \right\} \text{(primal feasibility)} \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, && \text{(dual feasibility)} \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. && \text{(complementarity condition/} \\ &&& \text{complementary slackness)} \end{aligned}$$



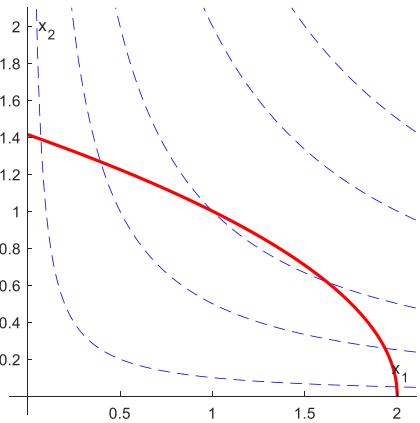
Either $\lambda_i^* = 0$ or $c_i(x^*) = 0$

(*strict* complimentarity: Only one of them is zero)

KKT Ex. 1



KKT Ex. 2



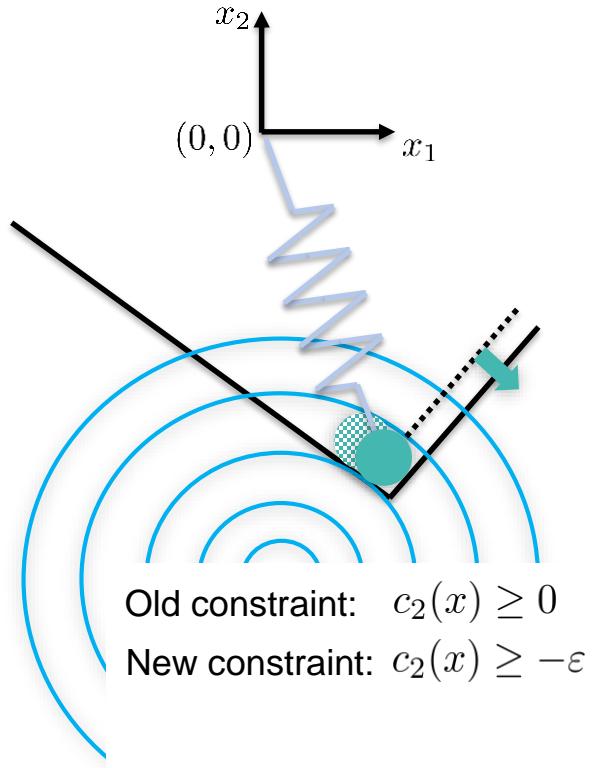
KKT Ex. 2, cont'd

KKT Ex. 2, cont'd

Solvability of KKT conditions

- KKT conditions can only be solved for very simple problems
 - The main complexity is the complementarity conditions – that is, deciding which constraints are active or not
- What is then the use of the KKT conditions?
 - Algorithms for LP and QP are constructed by searching for points that fulfill the KKT conditions
 - LPs and (some) QPs are convex – KKT are necessary *and* sufficient
 - For nonlinear programming, we use KKT to check whether a certain iterate is a *candidate* solution
 - In general KKT are *necessary* conditions!

Multipliers: “Shadow prices”



What happens if we relax a constraint?

Feasible set becomes larger, so new minimum $f(x_\varepsilon^*)$ becomes smaller.

How much would we gain?

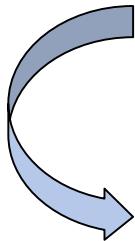
$$f(x_\varepsilon^*) \approx f(x^*) - \lambda\varepsilon$$

That is: The Lagrangian multipliers are the “hidden cost” (aka “shadow prices”) of constraints

KKT Conditions (Theorem 12.1)

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, && \left. \right\} \text{(primal feasibility)} \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, && \text{(dual feasibility)} \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. && \text{(complementarity condition/} \\ &&& \text{complementary slackness)} \end{aligned}$$

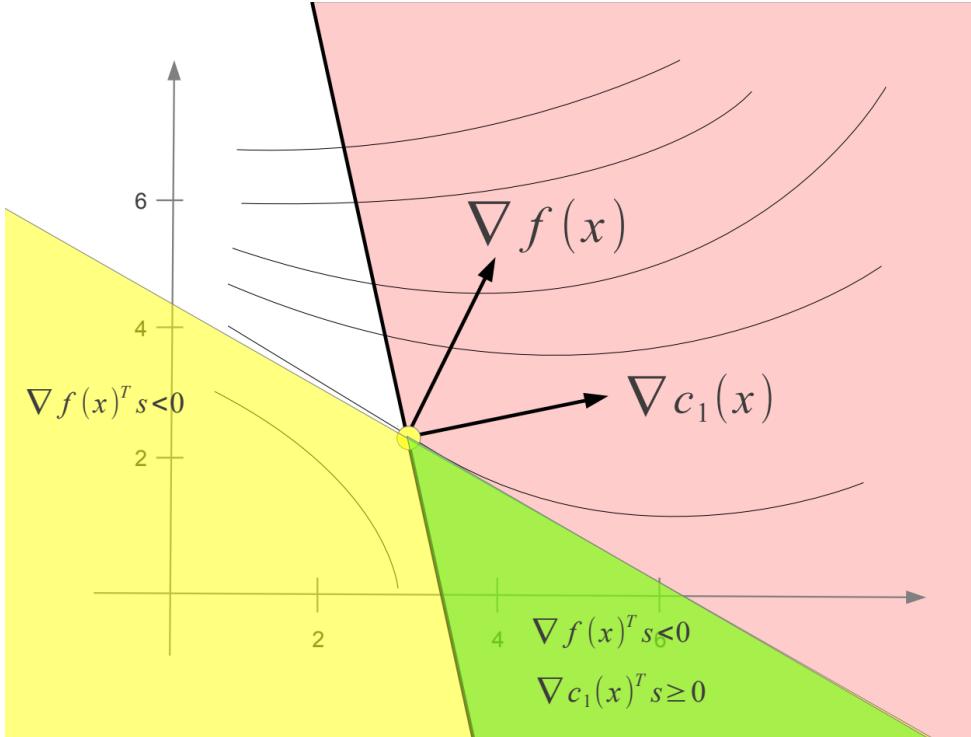


Either $\lambda_i^* = 0$ or $c_i(x^*) = 0$

(*strict* complimentarity: Only one of them is zero)

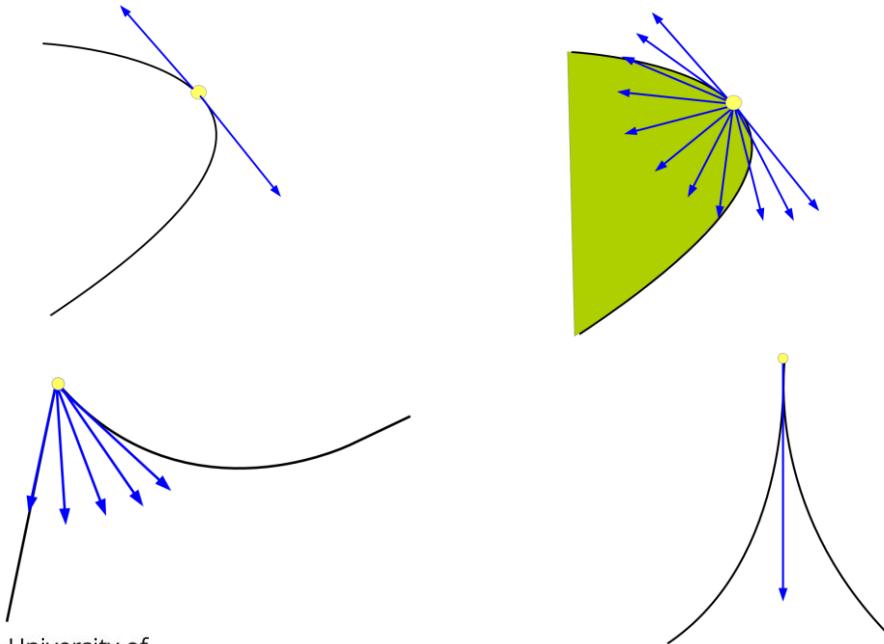
Constraint Qualifications

- Recall:



Tangent Cone

The tangent cone to a set Ω at a point $x \in \Omega$, denoted by $T_\Omega(x)$, consists of the limits of all (secant) rays which originate at x and pass through a sequence of points $p_i \in \Omega - \{x\}$ which converges to x .



Active Set

The active set $\mathcal{A}(x)$ at any feasible point x consists of the equality constraint indices from \mathcal{E} together with the indices of the inequality constraints i for which $c_i(x) = 0$. That is,

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}$$

Set of (linearized) Feasible Directions

Given a feasible point x and the active constraint set $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(x)$ is

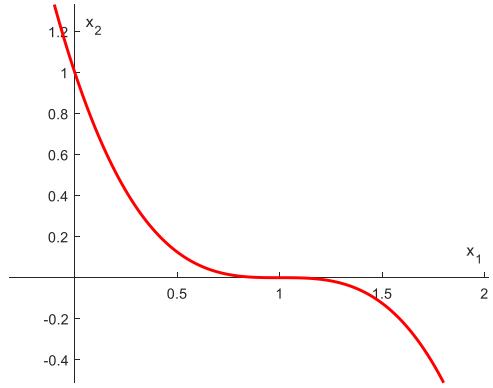
$$\mathcal{F}(x) = \left\{ d \mid \begin{array}{ll} d^\top \nabla c_i(x) = 0, & \text{for all } i \in \mathcal{E}, \\ d^\top \nabla c_i(x) \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\}$$

- Note 1: The definition of $T_\Omega(x)$ depends on the geometry of the feasible set Ω .
- Note 2: The definition of $\mathcal{F}(x)$ depends on the algebraic definition of the constraints.

Constraint Qualifications

- Constraint Qualifications are needed to rule out special cases where optimal solutions does not fulfill the KKT conditions
 - A constraint qualification is an assumption that ensures similarity of the constraint set Ω and its linearized approximation, in a neighborhood of a point x^* .
 - In other words: Constraint qualifications ensure that the linearized feasible set $\mathcal{F}(x^*)$ and the tangent cone $T_\Omega(x^*)$ are the same
- The most used Constraint Qualification is LICQ:
 - Given the point x and the active set $\mathcal{A}(x)$, we say that the *linear independence constraint qualification* (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$ is linearly independent.
 - Other constraint qualifications exists (N&W 12.6, not syllabus)
- Note: LICQ implies uniqueness of Lagrange multipliers

LICQ Ex.



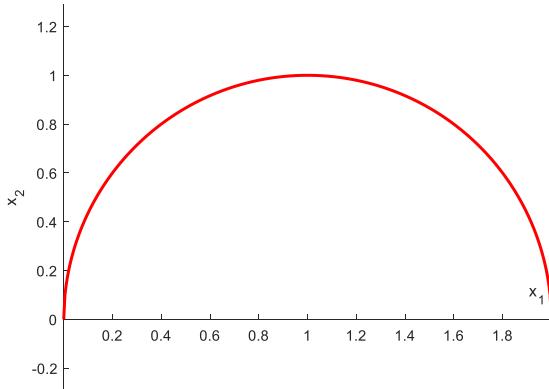
2nd Order Conditions: Critical Cone

- Say there are directions $w \in \mathcal{F}(x^*)$ that does not lead to an increase in the objective function, that is $w^\top \nabla f(x^*) = 0$, $w \neq 0$. How do we decide whether x^* is actually a minimum?
- Second-order conditions answer this by looking at the curvature (2nd derivative) in these directions
- Define the *critical cone*:

$$w \in \mathcal{C}(x^*, \lambda^*) \Leftrightarrow \begin{cases} \nabla c_i(x^*)^\top w = 0, & \forall i \in \mathcal{E}, \\ \nabla c_i(x^*)^\top w = 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^\top w \geq 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \end{cases}$$

- Note: $\mathcal{C}(x^*, \lambda^*) \subseteq \mathcal{F}(x^*)$. Difference: Inequalities with positive Lagrange multiplier treated as equalities
- $\mathcal{C}(x^*, \lambda^*)$ contains the “undecided” directions from $\mathcal{F}(x^*)$, the directions where decrease/increase cannot be decided from $\nabla f(x^*)$ alone

Critical cone Ex.



$$\min_{x \in \mathbb{R}^2} x_1$$

$$\text{s.t. } c_1(x) = x_2 \geq 0$$

$$c_2(x) = -(x_1 - 1)^2 - x_2^2 + 1 \geq 0$$

2nd Order Conditions: Necessary & Sufficient

- Second-order necessary conditions (Theorem 12.5):

Suppose that x^* is a local solution and that the LICQ condition is satisfied. Let λ^* be the Lagrange multiplier vector for which the KKT conditions are satisfied. Then

$$w^\top \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w \geq 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*)$$

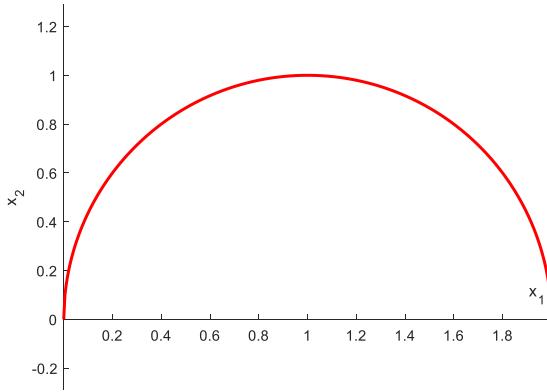
- Second-order sufficient conditions (Theorem 12.6):

Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions are satisfied. Suppose also that

$$w^\top \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w > 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*), w \neq 0$$

Then x^* is a strict local solution.

2nd order cond., Ex.

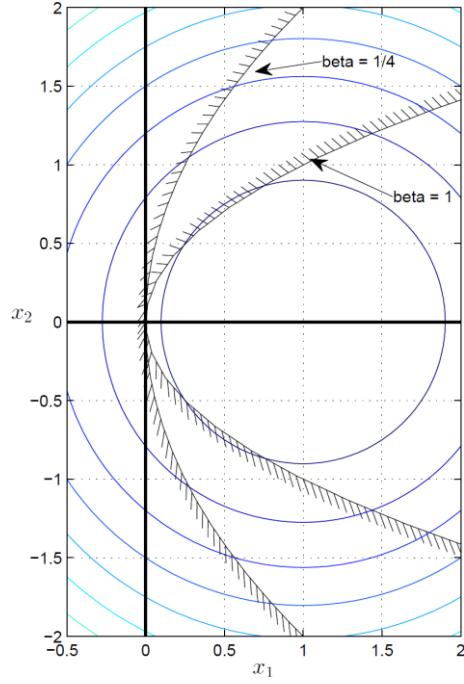


$$\min_{x \in \mathbb{R}^2} x_1$$

$$\text{s.t. } c_1(x) = x_2 \geq 0$$

$$c_2(x) = -(x_1 - 1)^2 - x_2^2 + 1 \geq 0$$

Example:



$$\min_{x \in \mathbb{R}^2} f(x) = \frac{1}{2} ((x_1 - 1)^2 + x_2^2)$$

$$\text{s.t. } c_1(x) = -x_1 + \beta x_2^2 = 0$$

Positive Definiteness

A square, symmetric matrix A is *positive definite* if the following equivalent conditions hold:

- There is a positive scalar α such that

$$x^\top Ax \geq \alpha x^\top x, \quad \text{for all } x \in \mathbb{R}^n.$$

- $x^\top Ax > 0$, for all $x \neq 0$.
- If all *eigenvalues* $\lambda_i > 0$.

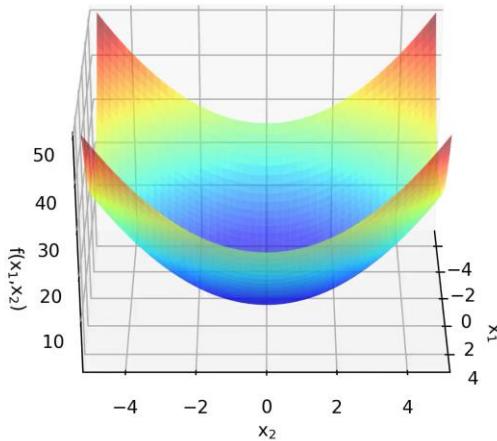
We also write $A > 0$ when A is PD.

A square matrix A is *positive semidefinite* if

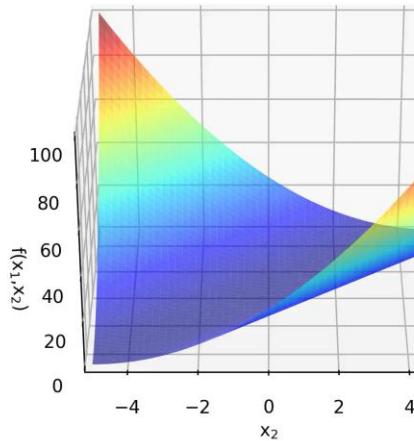
$$x^\top Ax \geq 0, \quad \text{for all } x \in \mathbb{R}^n$$

We also write $A \geq 0$ when A is PSD.

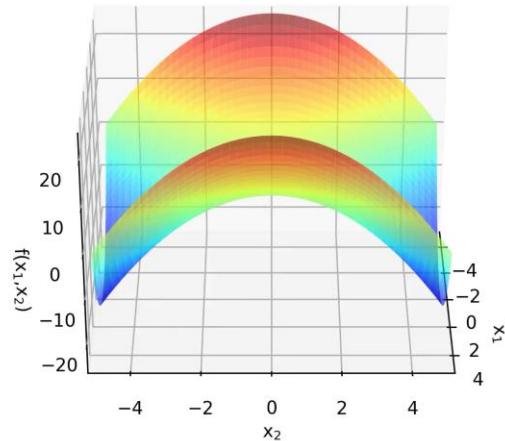
Visualizations



Positive Definite
 $x^T Px = x_1^2 + x_2^2$



Positive Semi-definite
 $x^T Px = x_1^2 + 2x_1x_2 + x_2^2$



Indefinite
 $x^T Px = x_1^2 - x_2^2$



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 4

2nd order optimality conditions

Linear programming

Lecturer: Lars Imsland

Purpose of Lecture

- Finishing optimality conditions: 2nd order
- Brief recap: linear algebra
- Linear programming (LP): formulation and standard form
- KKT conditions for LP
- Dual LP, weak & strong duality

Reference: Chapter 13.1 (12.9) in N&W (Linear algebra: App A.1)

OPTIMALITY CONDITIONS

KKT Conditions (Thm 12.1)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} f(x) & \text{subject to} \\ c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{array}$$

Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{(primal feasibility)} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array}$$

(dual feasibility) (complementarity condition/
complementary slackness)

Active Set

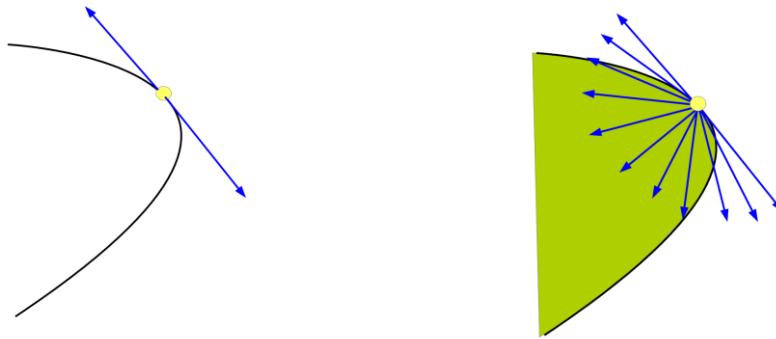
The active set $\mathcal{A}(x)$ at any feasible point x consists of the equality constraint indices from \mathcal{E} together with the indices of the inequality constraints i for which $c_i(x) = 0$. That is,

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}$$

Set of (linearized) Feasible Directions

Given a feasible point x and the active constraint set $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(x)$ is

$$\mathcal{F}(x) = \left\{ d \mid \begin{array}{ll} d^\top \nabla c_i(x) = 0, & \text{for all } i \in \mathcal{E}, \\ d^\top \nabla c_i(x) \geq 0, & \text{for all } i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\}$$



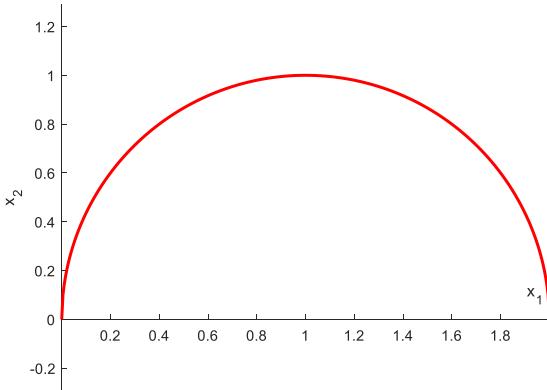
2nd Order Conditions: Critical Cone

- We have found a point x^* that fulfills KKT conditions
- Say there are directions $w \in \mathcal{F}(x^*)$ that does not lead to an increase in the objective function, that is $w^\top \nabla f(x^*) = 0$, $w \neq 0$. How do we decide whether x^* is actually a minimum?
- Second-order conditions answer this by looking at the curvature (2nd derivative) in these directions
- Define the *critical cone*:

$$w \in \mathcal{C}(x^*, \lambda^*) \Leftrightarrow \begin{cases} \nabla c_i(x^*)^\top w = 0, & \forall i \in \mathcal{E}, \\ \nabla c_i(x^*)^\top w = 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^\top w \geq 0, & \forall i \in \mathcal{A}(x^*) \cap \mathcal{I} \end{cases}$$

- Note: $\mathcal{C}(x^*, \lambda^*) \subseteq \mathcal{F}(x^*)$. Difference: Inequalities with positive Lagrange multiplier treated as equalities
- $\mathcal{C}(x^*, \lambda^*)$ contains the “undecided” directions from $\mathcal{F}(x^*)$, the directions where decrease/increase cannot be decided from $\nabla f(x^*)$ alone

Critical cone Ex.



$$\min_{x \in \mathbb{R}^2} x_1$$

$$\text{s.t. } c_1(x) = x_2 \geq 0$$

$$c_2(x) = -(x_1 - 1)^2 - x_2^2 + 1 \geq 0$$

2nd Order Conditions: Necessary & Sufficient

- Second-order necessary conditions (Theorem 12.5):

Suppose that x^* is a local solution and that the LICQ condition is satisfied. Let λ^* be the Lagrange multiplier vector for which the KKT conditions are satisfied. Then

$$w^\top \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w \geq 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*)$$

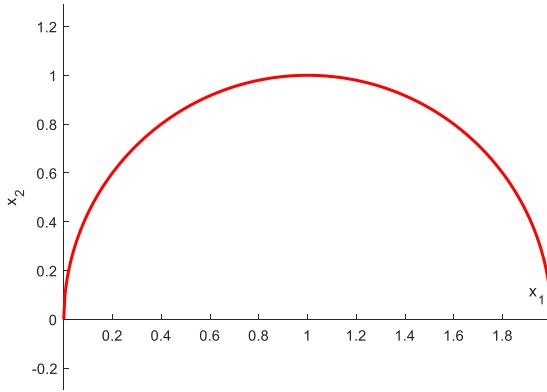
- Second-order sufficient conditions (Theorem 12.6):

Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions are satisfied. Suppose also that

$$w^\top \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w > 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*), w \neq 0$$

Then x^* is a strict local solution.

2nd order cond., Ex.



$$\min_{x \in \mathbb{R}^2} x_1$$

$$\text{s.t. } c_1(x) = x_2 \geq 0$$

$$c_2(x) = -(x_1 - 1)^2 - x_2^2 + 1 \geq 0$$

BRIEF RECAP: LINEAR ALGEBRA

Norms

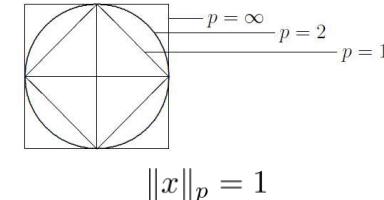
Vector norm: A mapping $\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}^+$ that satisfies

- $\|x\| = 0 \Rightarrow x = 0$
- $\|x + z\| \leq \|x\| + \|z\|$, for all $x, z \in \mathbb{R}^n$
- $\|\alpha x\| = |\alpha| \|x\|$, for all $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$



Common norms (p -norms):

- 1-norm: $\|x\|_1 = |x_1| + |x_2| + \dots + |x_n|$ (sum norm)
- 2-norm: $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$ (Euclidean norm)
- ∞ -norm: $\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$ (max norm)



Induced matrix norms, $A \in \mathbb{R}^{m \times n}$: $\|A\| := \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$

- 1-norm: $\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^m |A_{ij}|$
- 2-norm: $\|A\|_2 = \lambda_{\max}(\sqrt{A^\top A})$
- ∞ -norm: $\|A\|_\infty = \max_{i=1,\dots,m} \sum_{j=1}^n |A_{ij}|$

Other matrix norms, not induced:

- Frobenius-norm $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2}$

Useful property, induced matrix norms:

- $\|Ax\| \leq \|A\| \|x\|$

>> help norm

norm Matrix or vector norm.

norm(X,2) returns the 2-norm of X.

norm(X) is the same as norm(X,2).

norm(X,1) returns the 1-norm of X.

norm(X,Inf) returns the infinity norm of X.

norm(X,'fro') returns the Frobenius norm of X.

$$\begin{pmatrix} 4 & -1 & 2 & 0 \\ 0 & 3 & 0 & 2 \\ -2 & 1 & 5 & 1 \\ 6 & 5 & 7 & 3 \end{pmatrix} \begin{matrix} 7 \\ 5 \\ 9 \\ 1 \end{matrix} \begin{matrix} \|A\|_\infty \\ \|A\|_1 \\ \|A\|_1 \end{matrix}$$

Fundamental Theorem of Linear Algebra

A matrix $A \in \mathbb{R}^{m \times n}$ is a mapping:

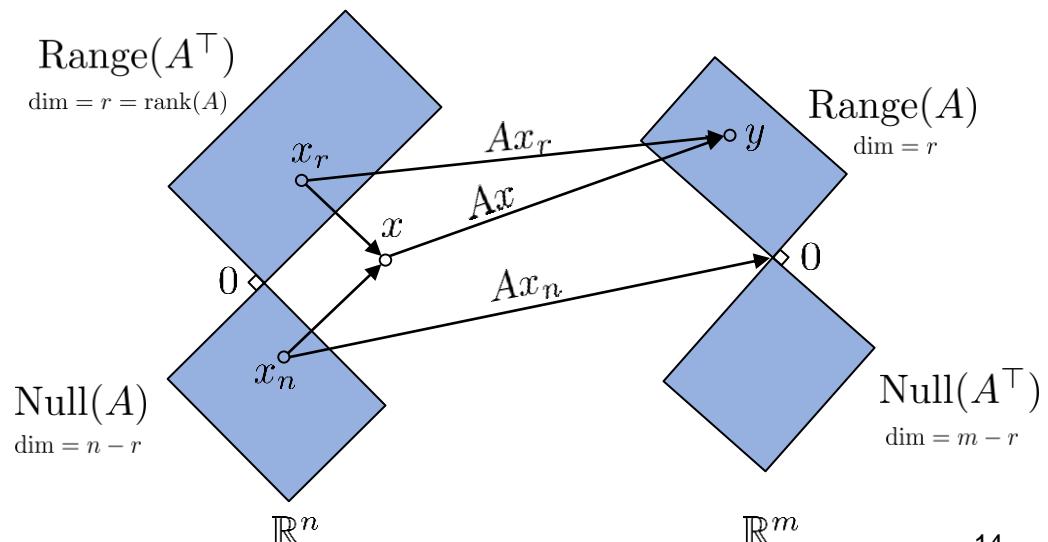


Nullspace of A : $\text{Null}(A) = \{v \in \mathbb{R}^n \mid Av = 0\}$

Rangespace (columnspace) of A : $\text{Range}(A) = \{w \in \mathbb{R}^m \mid w = Av, \text{ for some } v \in \mathbb{R}^n\}$

Fundamental theorem of linear algebra:

$$\text{Null}(A) \oplus \text{Range}(A^\top) = \mathbb{R}^n$$



Matrix Factorizations

“All” algorithms in this course involve solving linear equation systems:

$$Ax = b \quad \Rightarrow \quad x = A^{-1}b$$

```
>> help \
\ Backslash or left matrix divide.
A\b is the matrix division of A into B, which is roughly the
same as INV(A)*B , except it is computed in a different way.
If A is an N-by-N matrix and B is a column vector with N
components, or a matrix with several such columns, then
X = A\b is the solution to the equation A*X = B. A warning
message is printed if A is badly scaled or nearly singular.
```

In practice, **never** use the matrix inverse. It is inefficient and numerically unstable.

Instead, use matrix factorizations:

- General matrix A : Use LU-decomposition (*Gaussian elimination*)

$$A = LU : \quad Ax = L \underbrace{Ux}_y = b \quad \Rightarrow \quad Ly = b \quad \Rightarrow \quad Ux = y$$

– Due to triangular structure of L and U , we easily solve the two linear systems by substitution

- Symmetric positive definite matrix A : Use Cholesky decomposition

$$A = LL^\top$$

– Half the cost of LU. Solve system as for LU.
– For symmetric, indefinite matrices: Use LDL-factorization instead (book: $A =LBL^\top$)

- Generally, algorithms use permutations:

$$PA = LU, \quad PAP^\top = LL^\top$$

- Other important factorizations

– $A = QR$: Finds orthogonal (orthonormal) basis for rangespace of A and nullspace of A^\top
– Eigenvalue (spectral) decomposition, singular value decomposition

Condition Number of a Matrix (when solving $Ax = b$)

- Well-conditioned: Small perturbations give small changes in solution:

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3.00001 \\ 2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.99999 \\ 1.00001 \end{pmatrix}$$

- Ill-conditioned: Small perturbations give large changes in solution:

$$\begin{pmatrix} 1.00001 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2.00001 \\ 2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1.00001 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

- Condition number:

$$\kappa(A) = \|A\| \|A^{-1}\|$$

- A small condition number (say, 1-100) implies the matrix is well-conditioned, a large condition number (say, >10 000) implies the matrix is ill-conditioned.

The condition number (2-norm) of the above matrices are 6.9 and 400 000, respectively.

```
>> help cond  
cond Condition number with respect to inversion.  
cond(X) returns the 2-norm condition number (the ratio of the largest singular value of X to the smallest).  
Large condition numbers indicate a nearly singular matrix.
```

cond(X,P) returns the condition number of X in P-norm:
NORM(X,P) * NORM(INV(X),P).

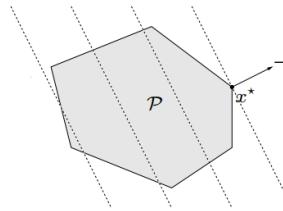
where P = 1, 2, inf, or 'fro'.

LINEAR PROGRAMMING

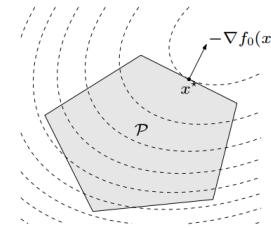
Types of Constrained Optimization Problems

- Linear programming
 - Convex problem
 - Feasible set polyhedron
- Quadratic programming
 - Convex problem if $P \geq 0$
 - Feasible set polyhedron
- Nonlinear programming
 - In general non-convex!

$$\begin{aligned} & \min c^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

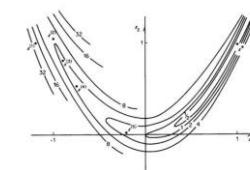


$$\begin{aligned} & \min \frac{1}{2} x^T Px + q^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$



$$\begin{aligned} & \min f(x) \\ \text{subject to } & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{subject to } & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Defense Analysts put together a list they call the “Top Ten Algorithms of the Century.”

“We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century,” Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here’s the CiSE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer’s reputation for deterministic calculation, it’s fitting that one of its earliest applications was the generation of random numbers.



In terms of widespread use, George Dantzig’s simplex method is among the most successful algorithms of all time.

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig’s algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the “real” problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of **Krylov subspace iteration methods**.

These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that A is a huge $n \times n$ matrix so that the algebraic answer $x = b/A$ is not so easy to compute.

Standard form LP:

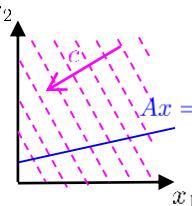
$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

LPs does not always have a solution

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

There are two sources for no solution:

1. Infeasibility: $Ax = b$ has no solution (the feasible set Ω is empty)
2. Unboundedness: There exists a sequence $x^k \in \Omega$ such that $c^\top x^k \rightarrow -\infty$

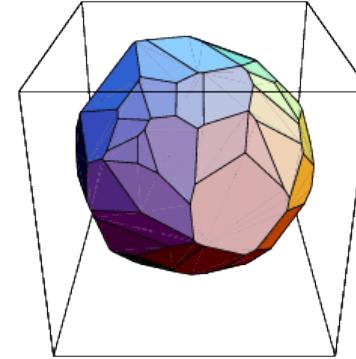


We can assume without loss of generality that $A \in \mathbb{R}^{m \times n}$, $m < n$

(If $m \geq n$, the problem is either infeasible or can be transformed to an equivalent problem with $m < n$)

Linear Programming Solutions

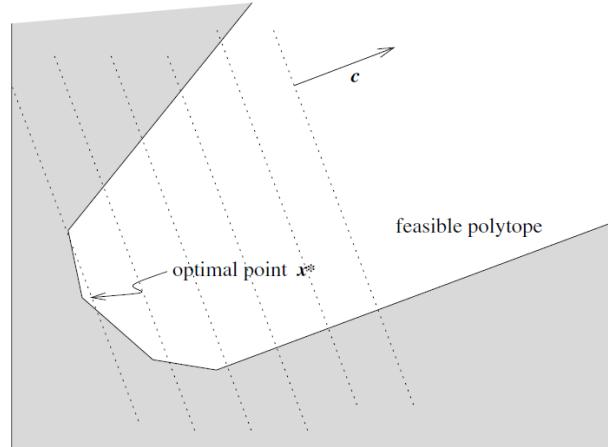
The feasible set is a polytope (a convex set with flat faces)



The objective function contours are planar

Three possible cases for solutions:

- No solutions: Feasible set is empty or problem is unbounded
- One solution: A vertex
- Infinite number of solutions: An “edge” is a solution



KKT Conditions for LPs

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Lagrangian:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

KKT-conditions:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0,$$

$$c_i(x^*) = 0, \quad \forall i \in \mathcal{E},$$

$$c_i(x^*) \geq 0, \quad \forall i \in \mathcal{I},$$

$$\lambda_i^* \geq 0, \quad \forall i \in \mathcal{I},$$

$$\lambda_i^* c_i(x^*) = 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}.$$

KKT for LPs is necessary and sufficient

The dual LP problem

$$\max_{\lambda \in \mathbb{R}^m} b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c$$

Lagrangian:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

KKT-conditions:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0,$$

$$c_i(x^*) = 0, \quad \forall i \in \mathcal{E},$$

$$c_i(x^*) \geq 0, \quad \forall i \in \mathcal{I},$$

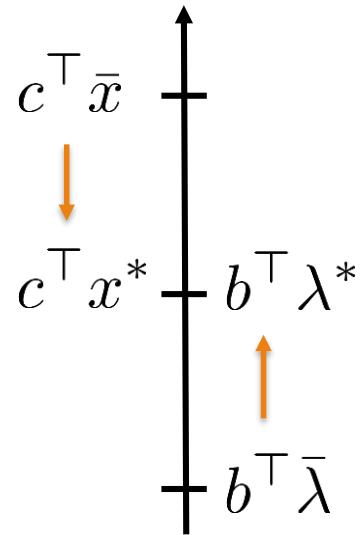
$$\lambda_i^* \geq 0, \quad \forall i \in \mathcal{I},$$

$$\lambda_i^* c_i(x^*) = 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}.$$

Weak duality

$$c^\top \bar{x} \geq c^\top x^* = b^\top \lambda^* \geq b^\top \bar{\lambda}$$

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$



$$\max_{\lambda \in \mathbb{R}^m} b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c$$

Strong duality

Theorem 13.1 Strong duality for LP

- i) If primal or dual has a finite solution, so does the other, and $c^\top x^* = b^\top \lambda^*$
- ii) If primal or dual is unbounded, the other is infeasible

Sensitivity

- Given LP in standard form:

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

- Assume optimal solution x^* , corresponding Lagrangian multiplier λ^*
- Do a small perturbation in b_i : $\tilde{b}_i = b_i + \epsilon$
- New solution fulfills

$$c^\top x_{\text{new}}^* = c^\top x^* \pm \epsilon \lambda_i^*$$

Notation: $f(x) \in C^1$ means $f(x)$ once continuously differentiable, C^2 twice, etc.

Derivatives

- Gradient and Hessian:** For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(x) \in C^2$, the gradient and Hessian are

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

- Directional derivative:** The directional derivative of $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is

$$D(f(x); p) := \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon p) - f(x)}{\epsilon}$$

Also valid when $f(x)$ is not continuously differentiable. When $f(x) \in C^1$,

$$D(f(x); p) = \nabla f(x)^\top p$$

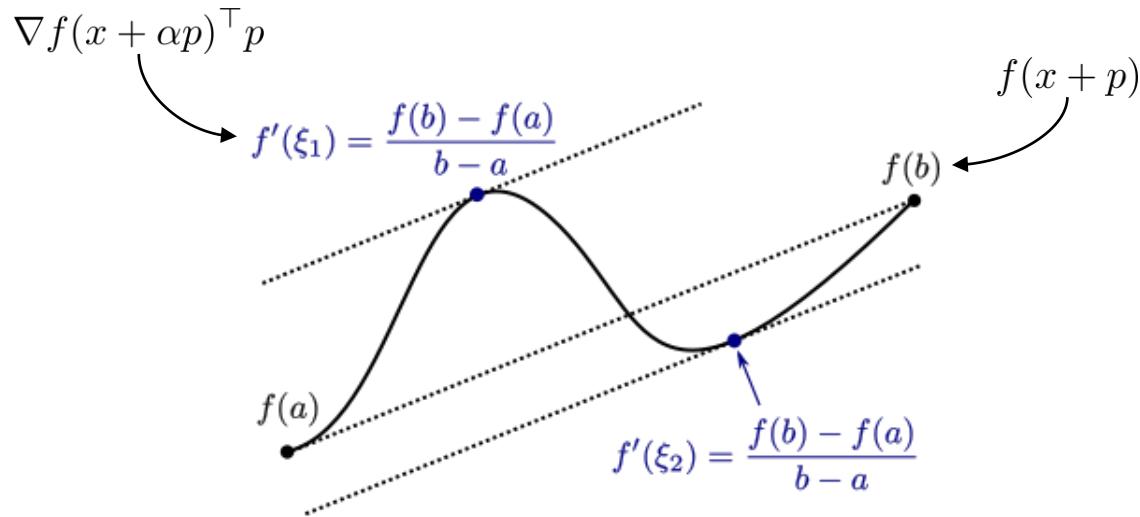
- Lipschitz continuity:** A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is Lipschitz continuous in a neighborhood \mathcal{N} , if

$$\|f(x) - f(y)\| \leq L\|x - y\|, \quad \text{for all } x, y \in \mathcal{N}$$

Mean Value Theorem

- For $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(x) \in C^1$, we have

$$f(x + p) = f(x) + \nabla f(x + \alpha p)^\top p \quad \text{for some } \alpha \in \langle 0, 1 \rangle$$



wikipedia.org

LP, Standard Form, and KKT

LP, standard form: $\min_{x \in \mathbb{R}^n} c^T x$ subject to $\begin{cases} Ax = b \\ x \geq 0 \end{cases}$

Lagrangian: $\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x$

KKT-conditions (necessary *and* sufficient for LP):

$$A^T \lambda^* + s^* = c,$$

$$Ax^* = b,$$

$$x^* \geq 0,$$

$$s^* \geq 0,$$

$$x_i^* s_i^* = 0, \quad i = 1, 2, \dots, n$$



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 5

Solving LPs – the Simplex method

Lecturer: Lars Imsland

Purpose of lecture

- Brief recap previous lecture
- The geometry of the feasible set
- Basic feasible points, “The fundamental theorem of linear programming”
- The simplex method
- Example 13.1
- Some implementation issues

Reference: N&W Ch.13.2-13.3, also 13.4-13.5

Linear programming, standard form and KKT: recap

LP: $\min_{x \in \mathbb{R}^n} c^T x$ subject to $\begin{cases} a_i x = b_i, & i \in \mathcal{E} \\ a_i x \geq b_i, & i \in \mathcal{I} \end{cases}$

LP, standard form: $\min_{x \in \mathbb{R}^n} c^T x$ subject to $\begin{cases} Ax = b \\ x \geq 0 \end{cases}$

Lagrangian: $\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x$

KKT-conditions (LPs: necessary *and* sufficient for optimality):

$$A^T \lambda^* + s^* = c,$$

$$Ax^* = b,$$

$$x^* \geq 0,$$

$$s^* \geq 0,$$

$$x_i^* s_i^* = 0, \quad i = 1, 2, \dots, n$$

Duality

Primal problem

$$\min_x \quad c^\top x$$

$$\text{s.t.} \quad Ax = b$$

$$x \geq 0$$

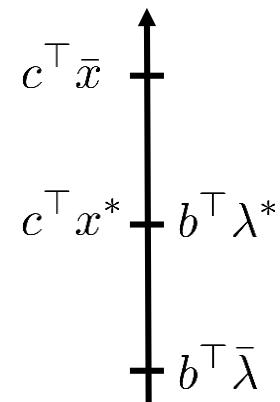
Dual problem

$$\max_{\lambda, s} \quad b^\top \lambda$$

$$\text{s.t.} \quad A^\top \lambda + s = c$$

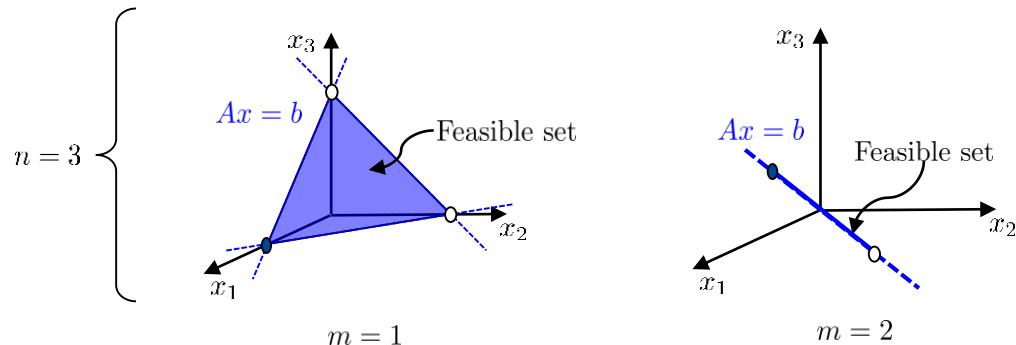
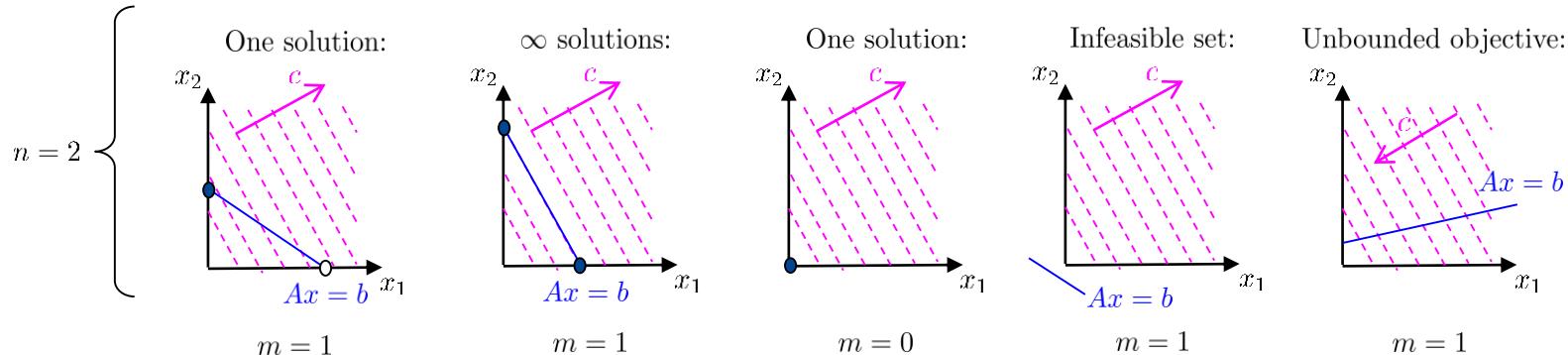
$$s \geq 0$$

- Primal and dual have same KKT conditions!
- Equal optimal value: $c^\top x^* = b^\top \lambda^*$
- Weak duality: $c^\top \bar{x} \geq c^\top x^* = b^\top \lambda^* \geq b^\top \bar{\lambda}$
- Duality gap: $c^\top \bar{x} - b^\top \bar{\lambda}$
- Strong duality (Thm 13.1):
 - i) If primal or dual has finite solution, both are equal
 - ii) If primal or dual is unbounded, the other is infeasible



LP: Geometry of the feasible set

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$



- Basic optimal point (BOP)
- Basic feasible point (BFP)
(if they exist)

In general, the BFP has at most m non-zero components

Basic feasible point (BFP)

A point x is a basic feasible point if

- x is feasible
- There is an index set $\mathcal{B}(x) \subset \{1, \dots, n\}$ such that
 - $\mathcal{B}(x)$ contains m indices
 - $i \notin \mathcal{B}(x) \Rightarrow x_i = 0$
 - $B = [A_i]_{i \in \mathcal{B}(x)}$ is non-singular
- $\mathcal{B}(x)$ is called a basis for the LP

Facts about Simplex method

The Simplex method generates iterates that are BFP, and converge to a solution if

- 1) there are BFPs, and
- 2) one of them is a solution (Basic Optimal Point, BOP)

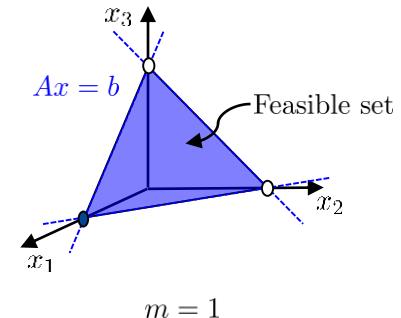
Theorem 13.2 (Fundamental theorem of Linear Programming): For standard for LP

- 1) If there is a feasible point, there is a BFP
- 2) If the LP has a solution, one solution is a BOP
- 3) If LP is feasible and bounded, there is a solution

Theorem 13.3: All vertices of the feasible polytope

$$\{x \mid Ax = b, x \geq 0\}$$

are BFPs (and all BFPs are vertices)



Facts about Simplex method, cont'd

Degeneracy: A LP is *degenerate* if there is a BFP x with $x_i = 0$ for some $i \in \mathcal{B}(x)$

Theorem 13.4: If an LP is bounded and non-degenerate, the Simplex method terminates at a BOP

Simplex definitions

LP KKT conditions (necessary&sufficient)

- Simplex method iterates BFPs until one that fulfills KKT is found.

$$A^T \lambda + s = c, \quad (\text{KKT-1})$$

$$Ax = b, \quad (\text{KKT-2})$$

$$x \geq 0, \quad (\text{KKT-3})$$

$$s \geq 0, \quad (\text{KKT-4})$$

$$x_i s_i = 0, \quad i = 1, 2, \dots, n \quad (\text{KKT-5})$$

- Each step is a move from a vertex to a neighboring vertex (*one change in the basis*), that decreases the objective

One step of Simplex-algorithm

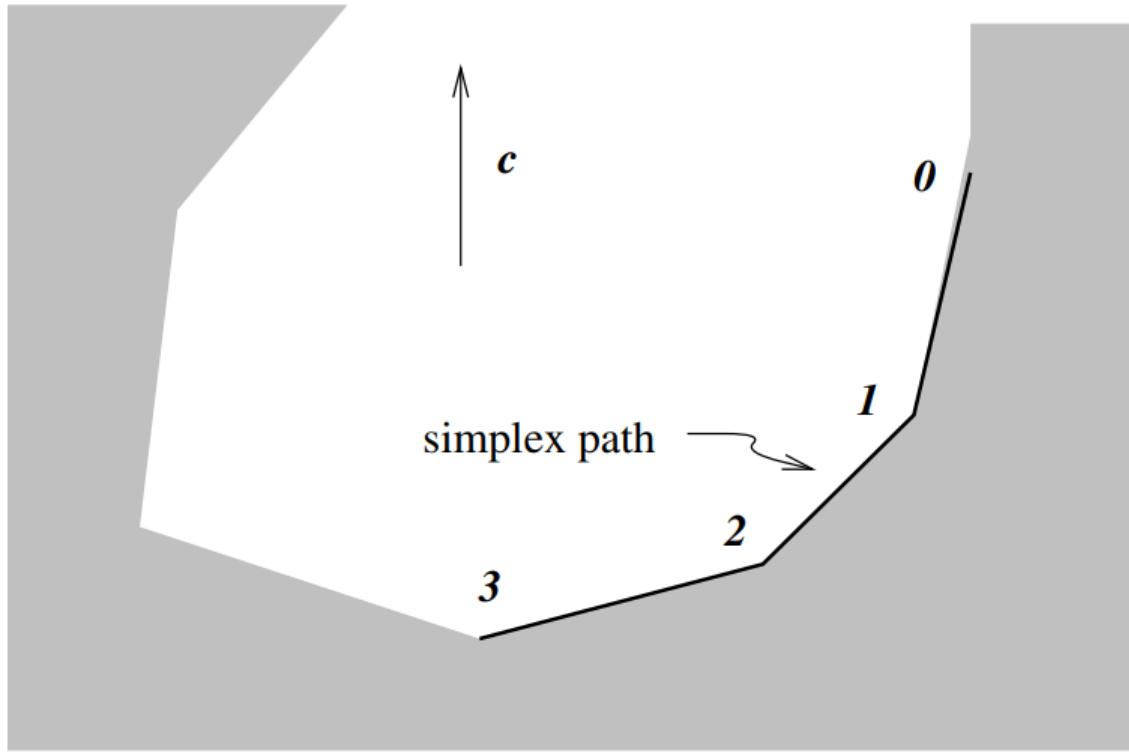
$$A^T \lambda + s = c, \quad (\text{KKT-1})$$

$$Ax = b, \quad (\text{KKT-2})$$

$$x \geq 0, \quad (\text{KKT-3})$$

$$s \geq 0, \quad (\text{KKT-4})$$

$$x_i s_i = 0, \quad i = 1, \dots, n \quad (\text{KKT-5})$$



Check KKT-conditions for BFP

- Given BFP x , and corresponding basis $\mathcal{B}(x)$. Define

$$\mathcal{N}(x) = \{1, 2, \dots, n\} \setminus \mathcal{B}(x)$$

- Partition x, s and c :

$$x_B = [x_i]_{i \in \mathcal{B}(x)} \quad x_N = [x_i]_{i \in \mathcal{N}(x)}$$

- KKT conditions

KKT-2: $Ax = Bx_B + Nx_N = Bx_B = b$ (since x is BFP)

KKT-3: $x_B = B^{-1}b \geq 0, \quad x_N = 0$ (since x is BFP)

KKT-5: $x^\top s = x_B^\top s_B + x_N^\top s_N = 0$ if we choose $s_B = 0$

KKT-1: $\begin{bmatrix} B^T \\ N^T \end{bmatrix} \lambda + \begin{bmatrix} s_B \\ s_N \end{bmatrix} = \begin{bmatrix} c_B \\ c_N \end{bmatrix} \Rightarrow \begin{cases} \lambda = B^{-T} c_B \\ s_N = c_N - N^T \lambda \end{cases}$

KKT-4: Is $s_N \geq 0$?

- If $s_N \geq 0$, then the BFP x fulfills KKT and is a solution
- If not, change basis, and try again
 - E.g. pick smallest element of s_N (index q), increase x_q along $Ax=b$ until x_p becomes zero. Move q from \mathcal{N} to \mathcal{B} , and p from \mathcal{B} to \mathcal{N} . This guarantees decrease of objective, and no “cycling” (if non-degenerate).

Ex. 13.1

Ex. 13.1, first iteration

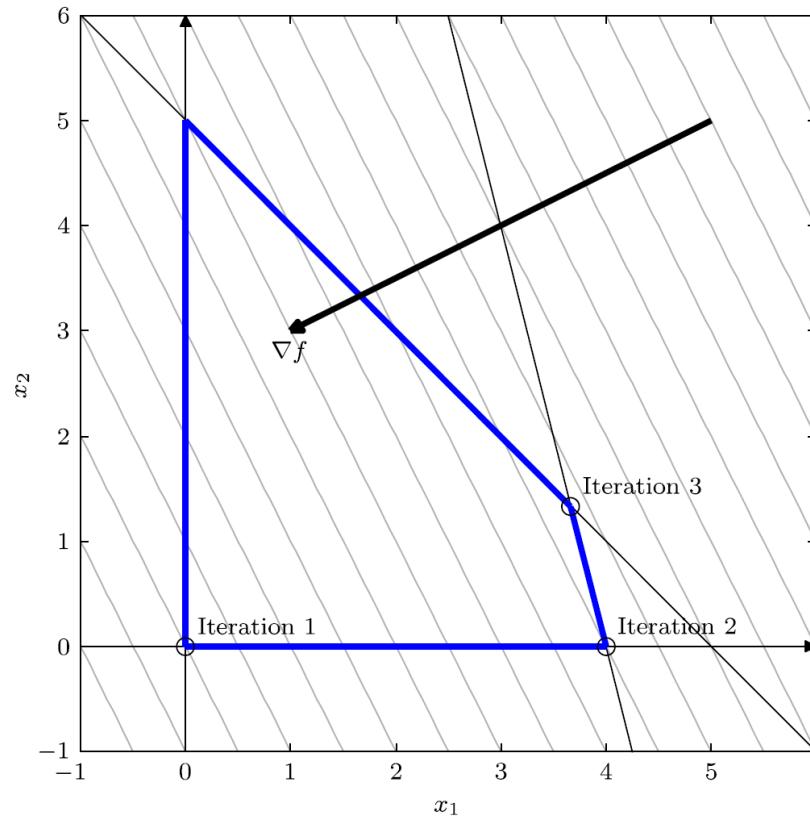
$$c^\top = \begin{pmatrix} -4 & -2 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 2 & .5 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ 8 \end{pmatrix}$$

Ex. 13.1, second iteration $c^\top = \begin{pmatrix} -4 & -2 & 0 & 0 \end{pmatrix}$, $A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 2 & .5 & 0 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 5 \\ 8 \end{pmatrix}$

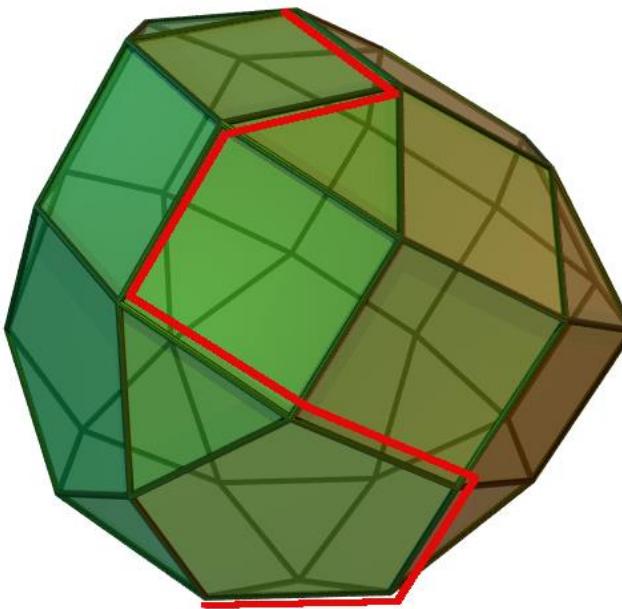
Ex. 13.1, third iteration

$$c^\top = \begin{pmatrix} -4 & -2 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 2 & .5 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ 8 \end{pmatrix}$$

Example 13.1 – figure



Simplex in 3D



wikipedia.org

Linear algebra – LU factorization

- Two linear systems must be solved in each iteration:
 - $B^\top \lambda = c_B$
 - $Bd = A_q$ (to find the direction to check when increasing x_q)
 - We also have $Bx_B = b$. Since x_B is not needed in the iterations, we don't need to solve this (apart from in the final iteration)
 - This is the major work per iteration of simplex, efficiency is important!
- B is a general, non-singular matrix
 - Guaranteed a solution to the linear systems
 - LU factorization is the appropriate method to use (same for both systems)
 - Don't use matrix inversion!
- In each step of Simplex method, one column of B is replaced:
 - Can update (“maintain”) the LU factorization of B in a smart and efficient fashion
 - No need to do a new LU factorization in each step, save time!

Other practical implementation issues (Ch. 13.5)

- Selection of “entering index” q
 - Dantzig’s rule: Select the index of the most negative element in s_N
 - Other rules have proved to be more efficient in practice
- Handling of degenerate bases/degenerate steps (when a positive x_q is not possible)
 - If no degeneracy, each step leads to decrease in objective and convergence in finite number of iterations is guaranteed (Thm 13.4)
 - Degenerate steps lead to no decrease in objective. Not necessarily a problem, but can lead to cycling (we end up in the same basis as before)
 - Practical algorithms use perturbation strategies to avoid this
- Starting the simplex method
 - We assumed an initial BFP available – but finding this is as difficult as solving the LP
 - Normally, simplex algorithms have two phases:
 - Phase I: Find BFP
 - Phase II: Solve LP
 - Phase I: Design other LP with trivial initial BFP, and whose solution is BFP for original problem

$$\min e^\top z \text{ subject to } Ax + Ez = b, \quad (x, z) \geq 0$$

$$e = (1, 1, \dots, 1)^\top, \quad E \text{ diagonal matrix with } \begin{cases} E_{jj} = 1 & \text{if } b_j \geq 0 \\ E_{jj} = -1 & \text{if } b_j < 0 \end{cases}$$

- Presolving (Ch. 13.7)
 - Reducing the size of the problem before solving, by various tricks to eliminate variables and constraints. Size reduction can be huge. Can also detect infeasibility.

Simplex – an active set method

- Complexity:
 - Typically, at most $2m$ to $3m$ iterations
 - Worst case: All vertices must be visited (exponential complexity in n)
 - Compare interior point method: Guaranteed polynomial complexity, but in practice hard to beat simplex on many problems
- Active set methods (such as simplex method):
 - Maintains explicitly an estimate of the set of inequality constraints that are active at the solution (the set \mathcal{N} for the simplex method)
 - Makes small changes to the set in each iteration (a single index in simplex)
- Next time: Active set method for QP



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 6

Quadratic Programming

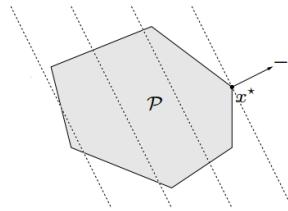
Equality-constrained QPs

Lecturer: Lars Imsland

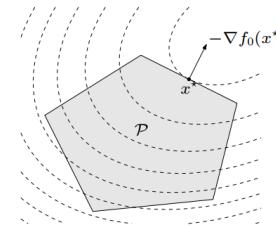
Types of Constrained Optimization Problems

- Linear programming
 - Convex problem
 - Feasible set polyhedron
- Quadratic programming
 - Convex problem if $P \geq 0$
 - Feasible set polyhedron
- Nonlinear programming
 - In general non-convex!

$$\begin{aligned} & \min c^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

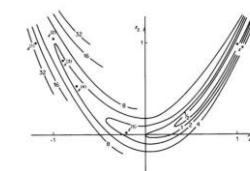


$$\begin{aligned} & \min \frac{1}{2} x^T Px + q^T x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$



$$\begin{aligned} & \min f(x) \\ \text{subject to } & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

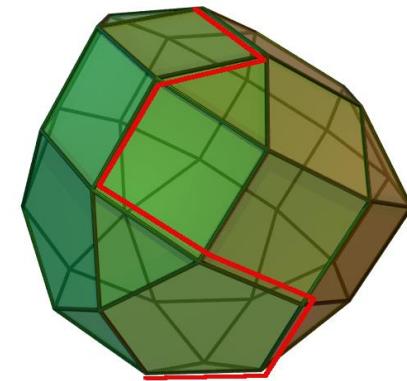


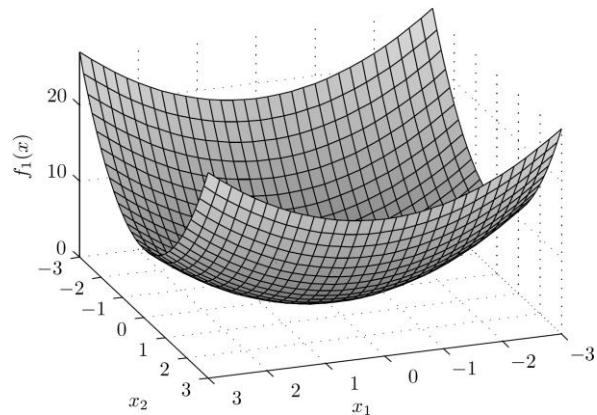
$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{subject to } & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

$$\begin{aligned}
 & \min_x \quad c^\top x \\
 \text{s.t.} \quad & Ax = b \\
 & x \geq 0
 \end{aligned}$$

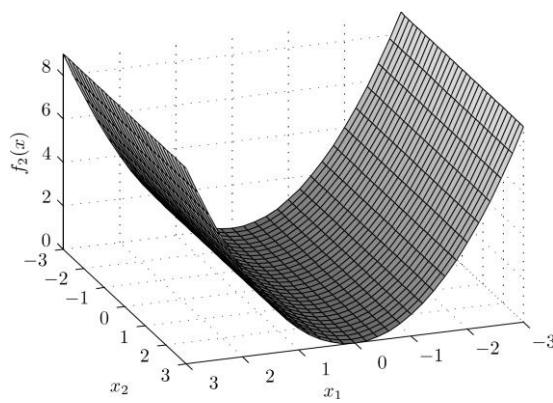
Last time: The simplex method for LP

- The Simplex algorithm
 - The feasible set of LPs are (convex) polytopes
 - LP solution is a vertex/“corner”/**BFP** of the feasible set
 - Simplex works by going from vertex to neighbouring vertex in such a manner that the objective decreases in each iteration
 - In each iteration, we solve a linear system to find which component in the **basis** (set of “not active constraints”) we should change
 - “Almost” guaranteed convergence (if LP not unbounded or infeasible)
- Complexity:
 - Typically, at most $2m$ to $3m$ iterations
 - Worst case: All vertices must be visited (exponential complexity in n)
- Active set methods (such as simplex method):
 - Maintains explicitly an estimate of the set of inequality constraints that are active at the solution (the set \mathcal{N} for the Simplex method)
 - Makes small changes to the set in each iteration (a single index in Simplex)
- Today, and next lecture: Active set method for QP

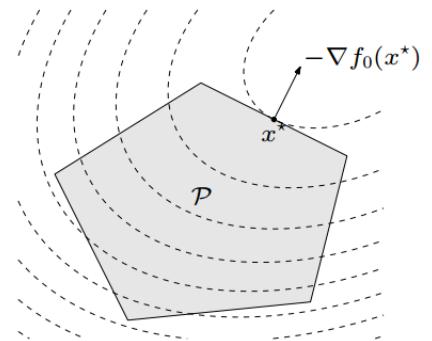




$G > 0$, strictly convex



$G \geq 0$, convex



Why are we interested in (convex) QPs?

- It is the “easiest” nonlinear programming problem
 - “easy”: efficient algorithms exist for convex QPs
- The QP is the basic building block of SQP (“sequential quadratic programming”), a common method for solving general nonlinear programs
 - Topic in end of course (N&W Ch. 18)
- QPs are often used in control, especially as solvers in Model Predictive Control
 - Topic in a few weeks
 - Also used in finance (“Portfolio optimization”), some types of Machine Learning/regression problems, control allocation, economics, ...

QP Example: Farming example with changing prices

- A farmer wants to grow apples (A) and bananas (B)
- He has a field of size 100 000 m²
- Growing 1 tonne of A requires an area of 4 000 m², growing 1 tonne of B requires an area of 3 000 m²
- A requires 60 kg fertilizer per tonne grown, B requires 80 kg fertilizer per tonne grown
- The profit for A is $7000 - 200x_1$ per tonne (including fertilizer cost), the profit for B is $6000 - 140x_2$ per tonne (including fertilizer cost)
- The farmer can legally use up to 2000 kg of fertilizer
- The farmer wants to maximize his profits



LP farming example: Geometric interpretation and solution

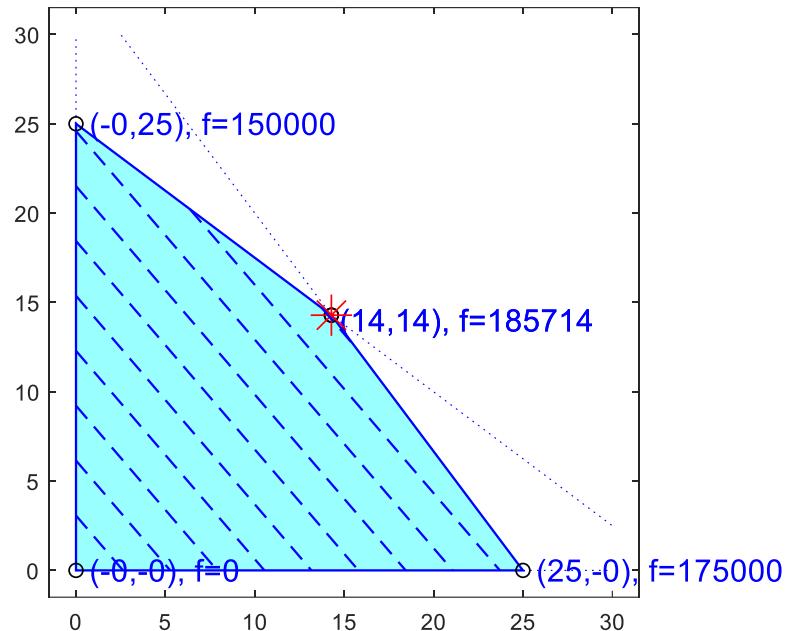
$$\max_{x_1, x_2} 7000x_1 + 6000x_2$$

$$\text{subject to: } 4000x_1 + 3000x_2 \leq 100000$$

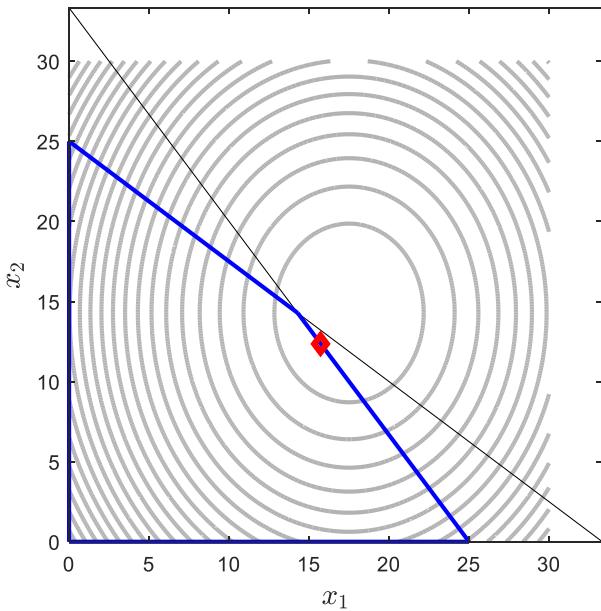
$$60x_1 + 80x_2 \leq 2000$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$



QP farming example: Geometric interpretation and solution



$$\max_{x_1, x_2} (7000 - 200x_1)x_1 + (6000 - 140x_2)x_2$$

$$\text{subject to: } 4000x_1 + 3000x_2 \leq 100000$$

$$60x_1 + 80x_2 \leq 2000$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

KKT Conditions (Thm 12.1)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} f(x) & \text{subject to} \\ c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{array}$$

Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{(primal feasibility)} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array}$$

(dual feasibility)
(complementarity condition/
complementary slackness)

Important special case: Equality-constrained QP

Nullspace

Solving the EQP

“Proof” Theorem 16.2

Example 16.2

$$\min_x \frac{1}{2} x^\top G x + c^\top x$$

subject to $Ax = b$

$$\min_{x_1, x_2, x_3} 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3$$

subject to $x_1 + x_3 = 3, \quad x_2 + x_3 = 0$

Matrices: $G = \begin{pmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{pmatrix}, \quad c = \begin{pmatrix} -8 \\ -3 \\ -3 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$

Note symmetry of G.
Always possible!

```
>> G = [6 2 1; 2 5 2; 1 2 4]; c = [-8; -3; -3]; A = [1 0 1; 0 1 1]; b = [3; 0];
>> K = [G, -A'; A, zeros(2,2)];
>> K\b[-c;b] % X = A\b is the solution to the equation A*X = B
```

ans =

2.0000
-1.0000
1.0000
3.0000
-2.0000

x^*

λ^*

Fundamental Theorem of Linear Algebra

A matrix $A \in \mathbb{R}^{m \times n}$ is a mapping:

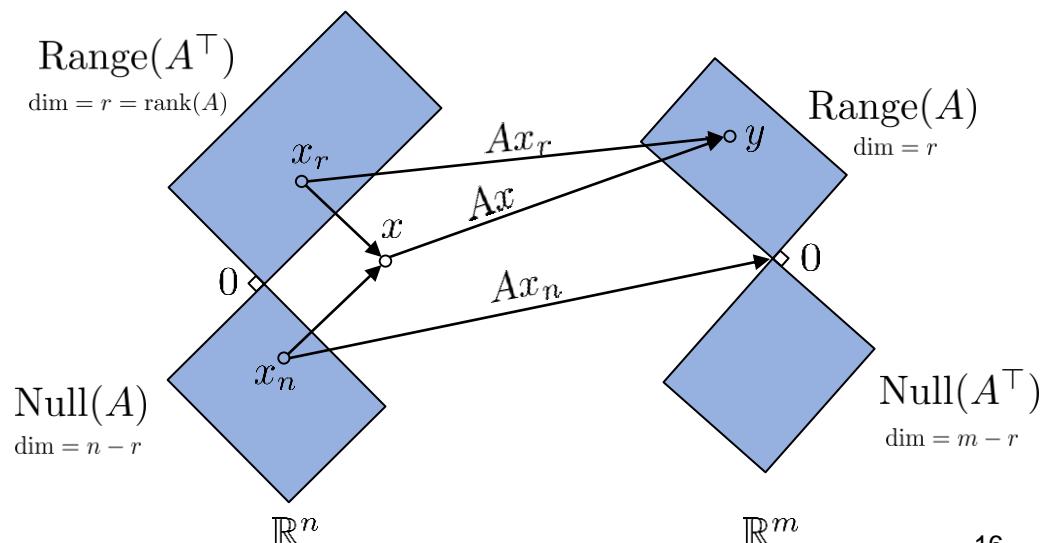


Nullspace of A : $\text{Null}(A) = \{v \in \mathbb{R}^n \mid Av = 0\}$

Rangespace (columnspace) of A : $\text{Range}(A) = \{w \in \mathbb{R}^m \mid w = Av, \text{ for some } v \in \mathbb{R}^n\}$

Fundamental theorem of linear algebra:

$$\text{Null}(A) \oplus \text{Range}(A^\top) = \mathbb{R}^n$$



Nullspace method/Elimination of variables (N&W 16.2/15.3)

Nullspace method/Elimination of variables (N&W 16.2/15.3)

Example 16.2

$$\begin{array}{ll}\min_x & \frac{1}{2}x^\top Gx + c^\top x \\ \text{subject to} & Ax = b\end{array}$$

$$\min_{x_1, x_2, x_3} 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3$$

$$\text{subject to } x_1 + x_3 = 3, \quad x_2 + x_3 = 0$$

Matrices: $G = \begin{pmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{pmatrix}, \quad c = \begin{pmatrix} -8 \\ -3 \\ -3 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$

Note symmetry of G.
Always possible!

```
>> G = [6 2 1; 2 5 2; 1 2 4]; c = [-8; -3; -3]; A = [1 0 1; 0 1 1]; b = [3; 0];
>> K = [G, -A'; A, zeros(2,2)];
>> K\[-c;b]      % X = A\b is the solution to the equation A*X = B
```

ans =

$$\begin{pmatrix} 2.0000 \\ -1.0000 \\ 1.0000 \\ 3.0000 \\ -2.0000 \end{pmatrix} \quad x^*$$

```
>> [Q,R,P] = qr(A')
```

Q =

$$\begin{pmatrix} -0.7071 & 0.4082 & -0.5774 \\ 0 & -0.8165 & -0.5774 \\ -0.7071 & -0.4082 & 0.5774 \end{pmatrix} \quad Y \quad Z$$

R =

$$\begin{pmatrix} -1.4142 & -0.7071 \\ 0 & -1.2247 \\ 0 & 0 \end{pmatrix}$$

P =

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

```
>> Z = Q(:,3);
>> Z'*G*Z
```

ans =

$$4.3333$$

Summing up: Direct solutions of KKT system (16.2)

Solution of KKT system when $Z^\top GZ > 0$

- Full space:

$$\begin{pmatrix} G & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} -p \\ \lambda^* \end{pmatrix} = \begin{pmatrix} c + Gx \\ Ax - b \end{pmatrix}$$

- Use LU (or LDL-method, since KKT-matrix is symmetric)

- Reduced space, efficient if $n-m \ll n$:

$$\begin{aligned}(AY)p_Y &= b - Ax \\ (Z^\top GZ)p_Z &= -Z^\top GYp_Y + Z^\top(c + Gx) \\ p &= Yp_Y + Zp_Z\end{aligned}$$

- Solve two much smaller systems using LU and Cholesky (both with complexity that scales with n^3)
 - Main complexity is calculating basis for nullspace. Usual method is using QR.

- Alternative to direct methods: Iterative methods (16.3)
 - For very large systems, can be parallelized



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 7

Active Set Method for Quadratic programming

Lecturer: Lars Imsland

Overview of lecture

- Quadratic programming – used for control (MPC), in finance, ...
- Recap last time – Equality-constrained QPs (EQPs)
- **Active set method for solving QPs**
 - For medium-sized problems – for large problems, interior point methods may be faster (not part of this course)
- Example 16.4

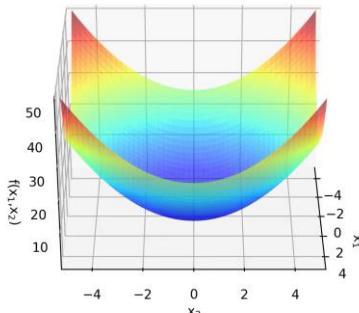
Reference: N&W Ch.15.3-15.5, **16.1-2,4-5**

Quadratic programming

Solving (convex) quadratic programs, QPs

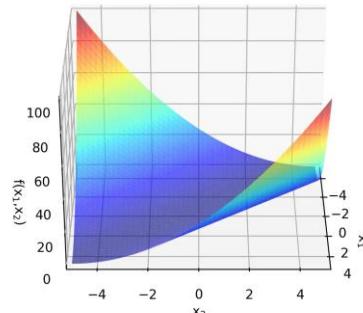
$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^\top G x + c^\top x \quad \text{subject to} \quad \begin{cases} a_i^\top x = b_i, & i \in \mathcal{E} \\ a_i^\top x \geq b_i, & i \in \mathcal{I} \end{cases}$$

- Feasible set convex (as for LPs)
- The QP is convex if $G \geq 0$ (strictly convex if $G > 0$)



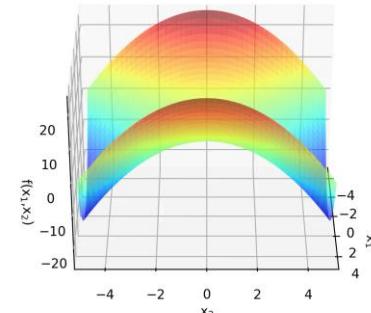
Strictly convex

$$x^\top P x = x_1^2 + x_2^2$$



Convex

$$x^\top P x = x_1^2 + 2x_1x_2 + x_2^2$$



Non-convex

$$x^\top P x = x_1^2 - x_2^2$$

Equality-constrained QP (EQP)

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2} x^\top G x + c^\top x \\ \text{subject to} \quad & Ax = b, \quad A \in \mathbb{R}^{m \times n} \end{aligned}$$

Basic assumption:
A full row rank

- KKT-conditions (KKT system, KKT matrix):

$$\begin{pmatrix} G & -A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -c \\ b \end{pmatrix} \quad \text{or, if we let } x^* = x + p, \quad \begin{pmatrix} G & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} -p \\ \lambda^* \end{pmatrix} = \begin{pmatrix} c + Gx \\ Ax - b \end{pmatrix}$$

- Solvable when $Z^\top G Z > 0$ (columns of Z basis for nullspace of A):

$$Z^\top G Z > 0 \stackrel{\text{Lemma 16.1}}{\Rightarrow} K = \begin{pmatrix} G & A^\top \\ A & 0 \end{pmatrix} \text{ non-singular} \Rightarrow \begin{pmatrix} x^* = x + p \\ \lambda^* \end{pmatrix} \text{ unique solution of KKT system}$$

Theorem 16.2 $\Rightarrow x^*$ is the unique solution to EQP

- How to solve KKT system (KKT matrix indefinite, but symmetric):

- Full-space: Symmetric indefinite (LDL) factorization: $P^\top K P = LBL^\top$
- Reduced-space: Use $Ax=b$ to eliminate m variables. Requires computation of Z , which can be costly.
Reduced space method faster than full-space when many constraints (if $n-m \ll n$).

Active set method for QPs, simplified

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^\top G x + c^\top x \quad \text{subject to} \quad \begin{cases} a_i^\top x = b_i, & i \in \mathcal{E} \\ a_i^\top x \geq b_i, & i \in \mathcal{I} \end{cases}$$

1. Make a guess of which constraints are active at the optimal solution
2. Solve corresponding EQP
3. Check KKT-conditions
 1. IF KKT OK, then finished
 2. If not, update guess of active constraints in smart way, start over

KKT Conditions (Thm 12.1)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} f(x) & \text{subject to} \\ c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{array}$$

Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{(primal feasibility)} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array}$$

(dual feasibility)
(complementarity condition/
complementary slackness)

KKT for QP

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^\top G x + c^\top x \quad \text{subject to} \quad \begin{cases} a_i^\top x = b_i, & i \in \mathcal{E} \\ a_i^\top x \geq b_i, & i \in \mathcal{I} \end{cases}$$

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

Theorem 16.4

Degeneracy

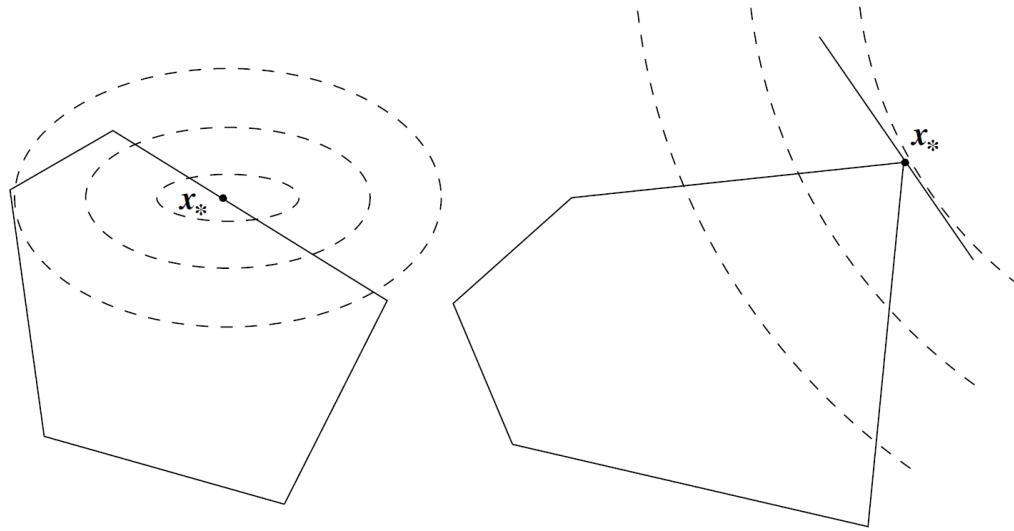


Figure 16.2 in Nocedal & Wright.

- 1) Strict complementarity does not hold
- 2) Constraints linearly dependent at solution

If active set known, QP can be solved as EQP

One step of active set method for QP

One step of active set method for QP, cont'd

General QP problem

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^\top G x + x^\top c \\ \text{s.t. } & a_i^\top x = b_i, \quad i \in \mathcal{E} \\ & a_i^\top x \geq b_i, \quad i \in \mathcal{I} \end{aligned}$$

- Lagrangian

$$\mathcal{L}(x^*, \lambda^*) = \frac{1}{2} x^* \top G x + x^* \top c - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i (a_i^\top x^* - b_i)$$

- KKT conditions

General:

$$\begin{aligned} Gx^* + c - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* a_i &= 0 \\ a_i^\top x^* &= b_i, \quad i \in \mathcal{E} \\ a_i^\top x^* &\geq b_i, \quad i \in \mathcal{I} \\ \lambda_i^* &\geq 0, \quad i \in \mathcal{I} \\ \lambda_i^* (a_i^\top x^* - b_i) &= 0, \quad i \in \mathcal{E} \cup \mathcal{I} \end{aligned}$$

Defined via active set:

$$\begin{aligned} \mathcal{A}(x^*) &= \mathcal{E} \cup \left\{ i \in \mathcal{I} \mid a_i^\top x^* = b_i \right\} \\ Gx^* + c - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* a_i &= 0 \\ a_i^\top x^* &= b_i, \quad i \in \mathcal{A}(x^*) \\ a_i^\top x^* &\geq b_i, \quad i \in \mathcal{I} \setminus \mathcal{A}(x^*) \\ \lambda_i^* &\geq 0, \quad i \in \mathcal{A}(x^*) \cap \mathcal{I} \end{aligned}$$

Active set method for convex QP

Algorithm 16.3 (Active-Set Method for Convex QP).

Compute a feasible starting point x_0 ;

Set \mathcal{W}_0 to be a subset of the active constraints at x_0 ;

for $k = 0, 1, 2, \dots$

Solve (16.39) to find p_k ;

if $p_k = 0$

 Compute Lagrange multipliers $\hat{\lambda}_i$ that satisfy (16.42),
 with $\hat{\mathcal{W}} = \mathcal{W}_k$;

if $\hat{\lambda}_i \geq 0$ for all $i \in \mathcal{W}_k \cap \mathcal{I}$

stop with solution $x^* = x_k$;

else

$j \leftarrow \arg \min_{j \in \mathcal{W}_k \cap \mathcal{I}} \hat{\lambda}_j$;

$x_{k+1} \leftarrow x_k$; $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$;

else (* $p_k \neq 0$ *)

 Compute α_k from (16.41);

$x_{k+1} \leftarrow x_k + \alpha_k p_k$;

if there are blocking constraints

 Obtain \mathcal{W}_{k+1} by adding one of the blocking
 constraints to \mathcal{W}_k ;

else

$\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$;

end (for)

$$\min_p \frac{1}{2} p^T G p + g_k^T p \quad (16.39a)$$

$$\text{subject to } a_i^T p = 0, \quad i \in \mathcal{W}_k. \quad (16.39b)$$

$$\sum_{i \in \hat{\mathcal{W}}} a_i \hat{\lambda}_i = g = G \hat{x} + c, \quad (16.42)$$

$$\alpha_k \stackrel{\text{def}}{=} \min \left(1, \min_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k} \right). \quad (16.41)$$

Example 16.4

$$\min_x q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2$$

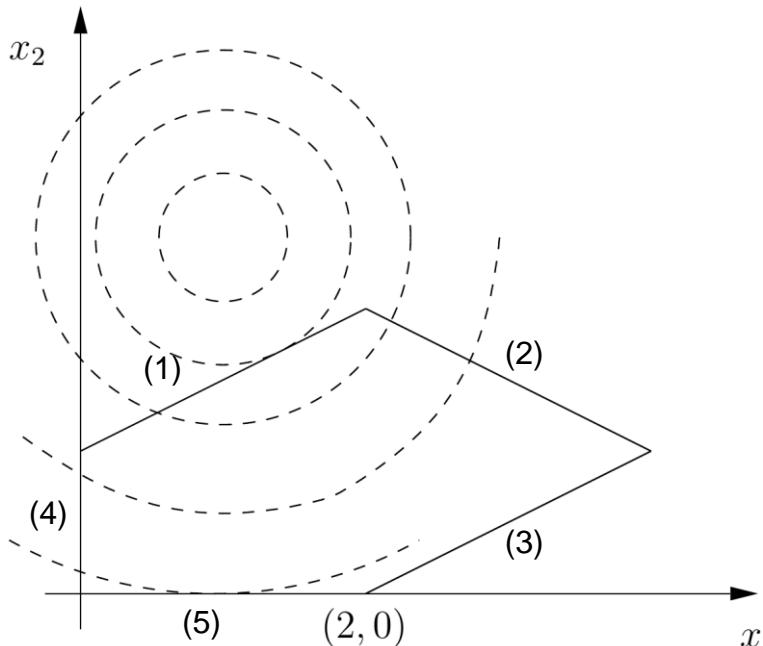
$$\text{subject to} \quad x_1 - 2x_2 + 2 \geq 0 \quad (1)$$

$$-x_1 - 2x_2 + 6 \geq 0 \quad (2)$$

$$-x_1 + 2x_2 + 2 \geq 0 \quad (3)$$

$$x_1 \geq 0 \quad (4)$$

$$x_2 \geq 0 \quad (5)$$



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

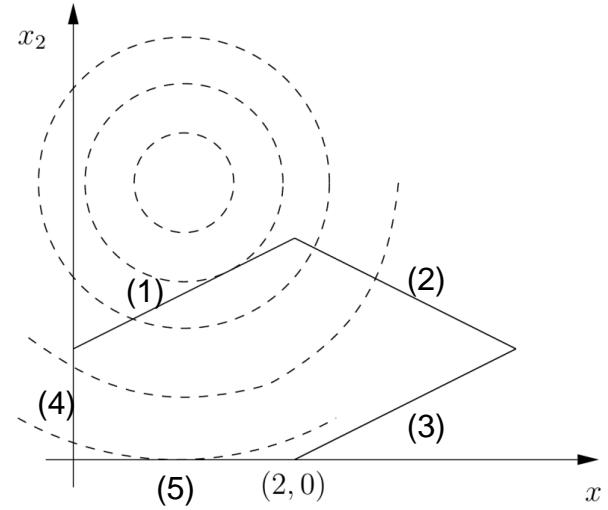
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

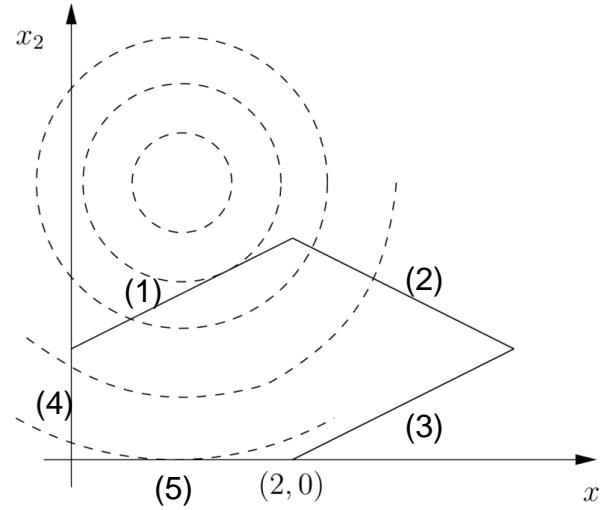
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

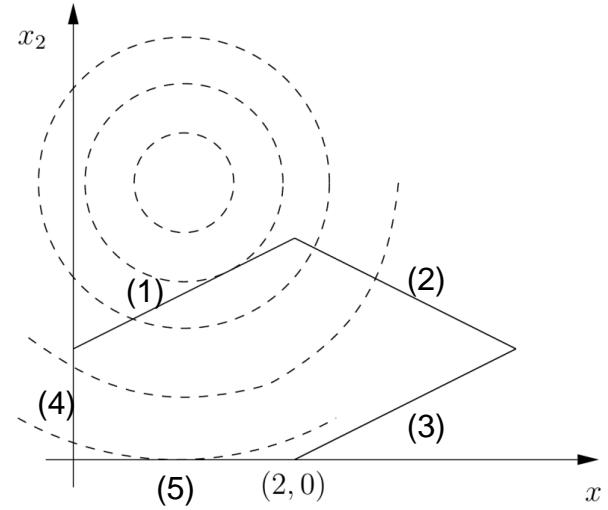
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

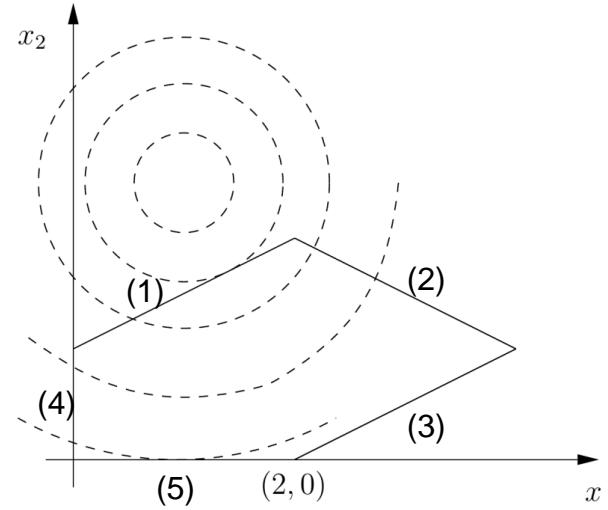
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

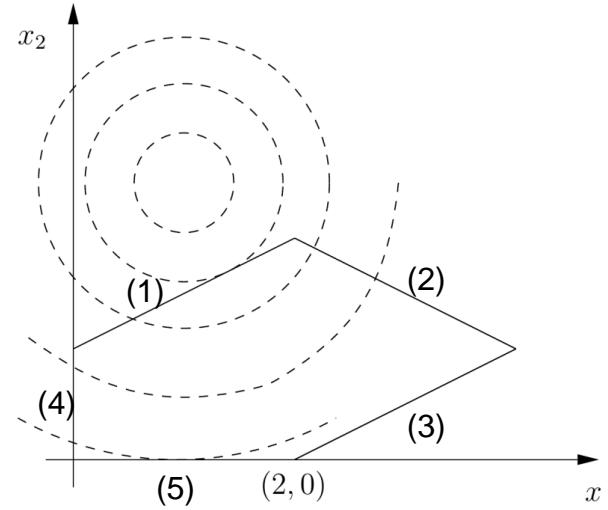
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

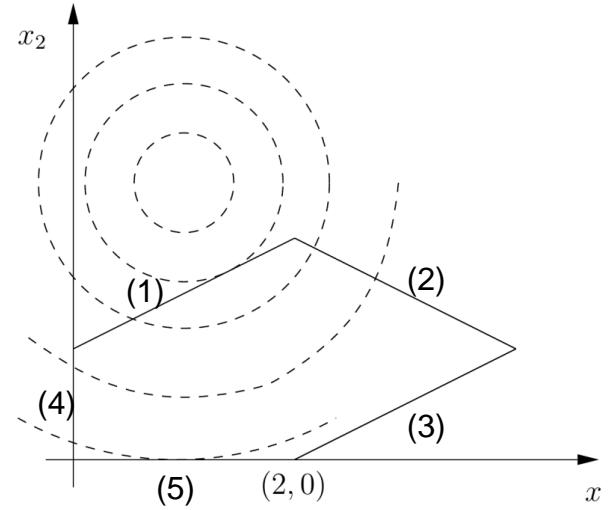
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Example 16.4

$$\min_x q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2$$

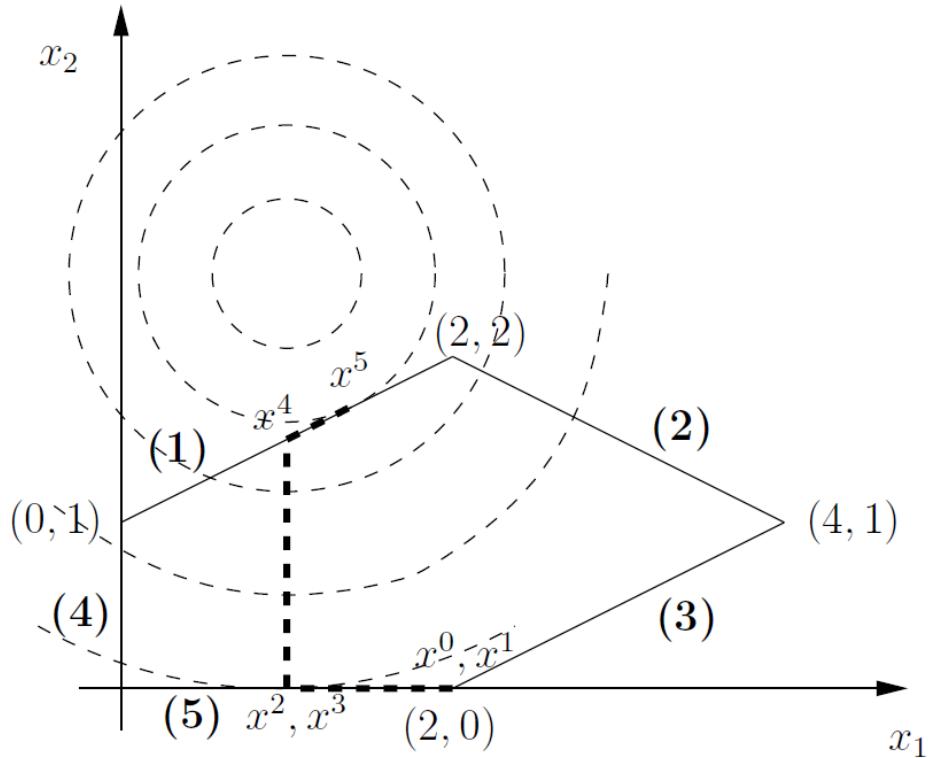
$$\text{subject to} \quad x_1 - 2x_2 + 2 \geq 0 \quad (1)$$

$$-x_1 - 2x_2 + 6 \geq 0 \quad (2)$$

$$-x_1 + 2x_2 + 2 \geq 0 \quad (3)$$

$$x_1 \geq 0 \quad (4)$$

$$x_2 \geq 0 \quad (5)$$



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

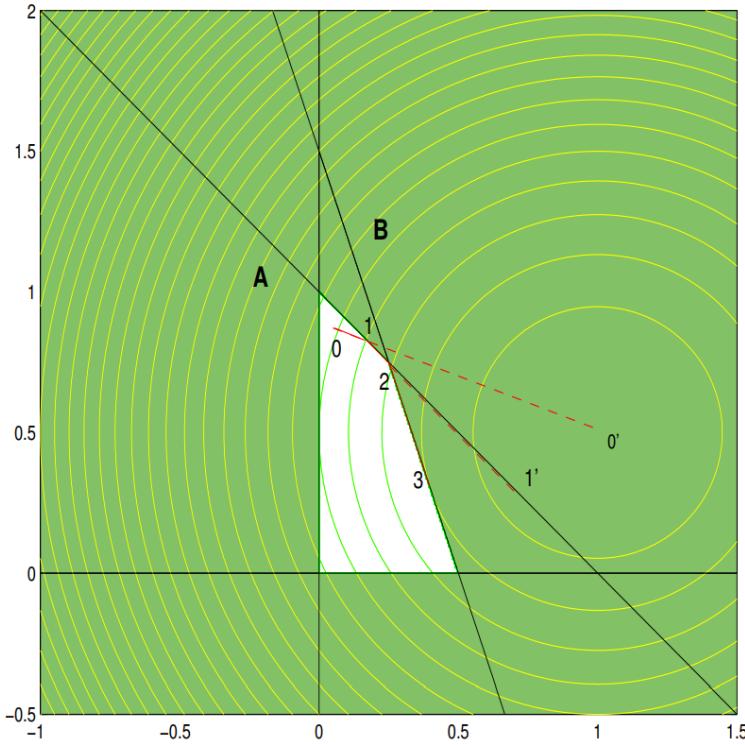
$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Another example (N. Gould)



$$\begin{aligned} & \min(x_1 - 1)^2 + (x_2 - 0.5)^2 \\ & \text{subject to } x_1 + x_2 \leq 1 \\ & \quad 3x_1 + x_2 \leq 1.5 \\ & \quad (x_1, x_2) \geq 0 \end{aligned}$$

0. Starting point
- 0'. Unconstrained minimizer
1. Encounter constraint A
- 1'. Minimizer on constraint A
2. Encounter constraint B,
move off constraint A
3. Minimizer on constraint B
= required solution

How to find feasible initial point?

- Same way as for LP:
 - Phase I: Define another LP with known feasible initial point, where solution is feasible for original LP.
 - Phase II: Solve original LP.
- Alternative method: “Big M”
 - Relax all constraints; penalize constraint violations in objective

Initialization methods

Phase 1

$$\begin{aligned} & \min_{(x,z)} e^T z \\ \text{subject to } & a_i^T x + \gamma_i z_i = b_i, \quad i \in \mathcal{E}, \\ & a_i^T x + \gamma_i z_i \geq b_i, \quad i \in \mathcal{I}, \\ & z \geq 0, \\ & e = (1, 1, \dots, 1)^T, \gamma_i = -\text{sign}(a_i^T \tilde{x} - b_i) \text{ for } i \in \mathcal{E} \\ & \gamma_i = 1 \text{ for } i \in \mathcal{I} \end{aligned}$$

- Feasible initial guess for LP problem:

$$\begin{aligned} x &= \tilde{x} \\ z_i &= |a_i^T \tilde{x} - b_i| \quad (i \in \mathcal{E}) \\ z_i &= \max(b_i - a_i^T \tilde{x}, 0) \quad (i \in \mathcal{I}) \end{aligned}$$

Big M

$$\begin{aligned} & \min_{(x,\eta)} \frac{1}{2} x^T G x + x^T c + M\eta, \\ \text{subject to } & (a_i^T x - b_i) \leq \eta, \quad i \in \mathcal{E}, \\ & -(a_i^T x - b_i) \leq \eta, \quad i \in \mathcal{E}, \\ & b_i - a_i^T x \leq \eta, \quad i \in \mathcal{I}, \\ & 0 \leq \eta, \end{aligned}$$

- Feasible initial guess for Big M: Whatever.
- η nonzero? Increase M and try again.

Concluding remarks

- Solves similar EQPs iteratively: recalculate only what's needed
- Active set method: Potentially slow, but with good initial guess will be FAST
- Alternative to Active Set: Interior Point (not curriculum)

Nonconvex QP

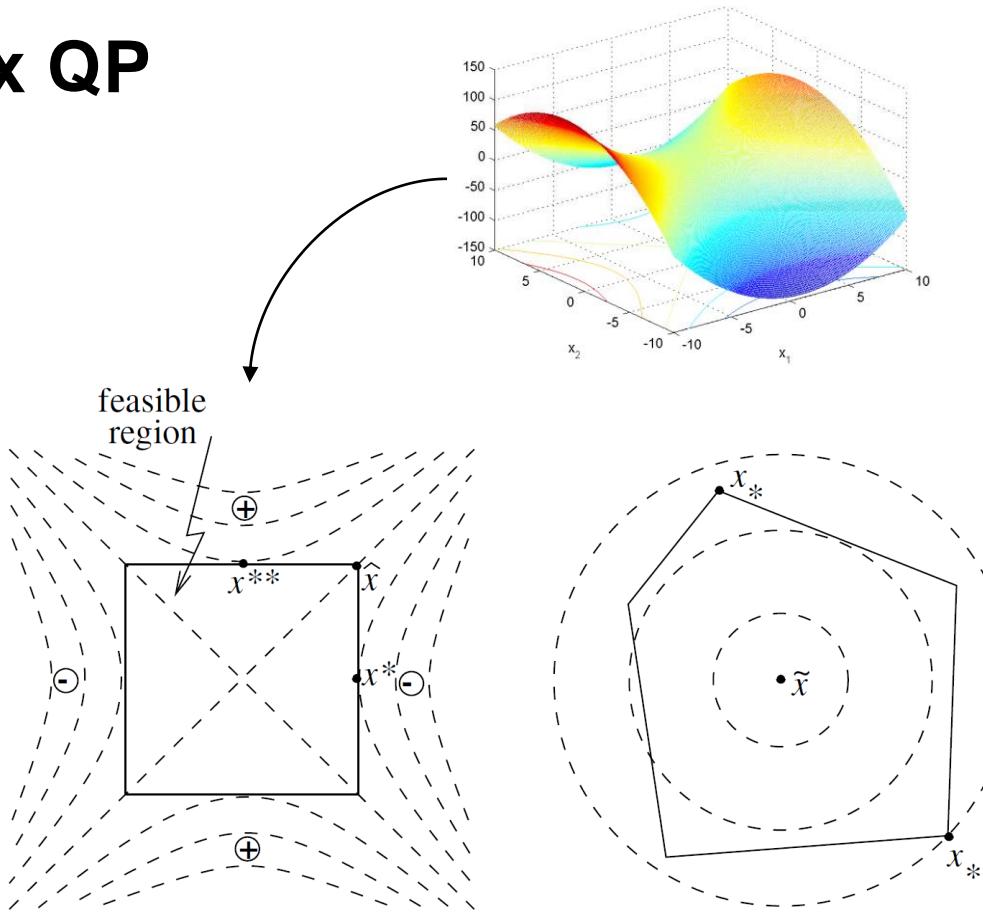


Figure 16.1 in Nocedal & Wright.



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 8

Open-Loop Dynamic optimization

Lecturer: Lars Imsland

Outline

- Static vs dynamic optimization (and “quasi-dynamic”)
- Dynamic optimization = optimization of dynamic systems
- How to construct objective function for dynamic optimization
- Batch approach vs recursive approach for solving dynamic optimization problems

Reference: F&H Ch. 3,4

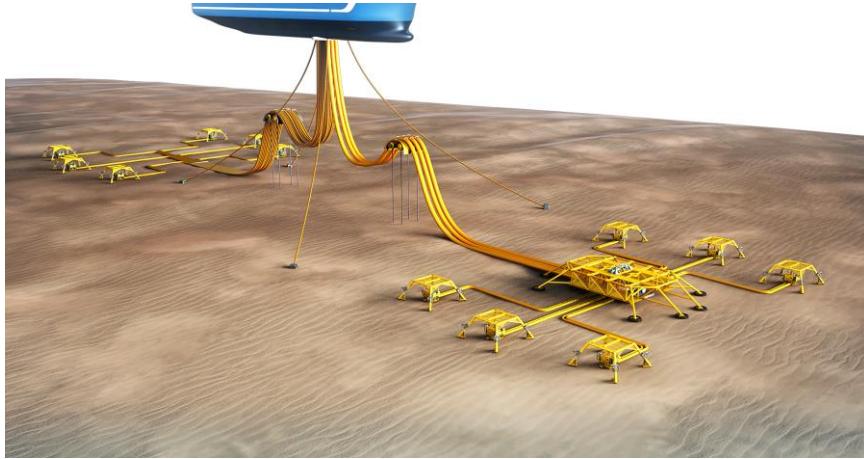
Static vs dynamic optimization

When using optimization for solving practical problems (that is, we optimize some *process*) we have two cases:

- The model of the process is time independent, resulting in **static optimization**
 - Common in finance, economic optimization, ...
 - Recall farming example
- The model of the process is time dependent, resulting in **dynamic optimization**
 - The typical case in control engineering
 - The process is a mechanical system (boat, drone, robot, ...), chemical process (e.g. chemical reactor, process plant, ...), electrical grid, ...
- F&H argues for a third option called **quazi-dynamic optimization**
 - The process is slowly time-varying, and can be assumed to be static for the purposes of optimization
 - We take care of the time-varying effects by re-solving regularly (or when the model has changed sufficiently)

Oil production

(example of quasi-dynamic optimization, Ex. 2 in F&H)



Dynamic models (in this course)

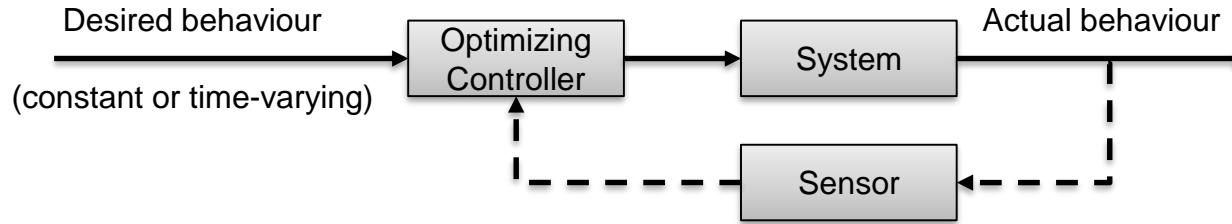
$$x_{t+1} = g(x_t, u_t) \quad (\text{nonlinear})$$

$$x_{t+1} = A_t x_t + B_t u_t \quad (\text{LTV})$$

$$x_{t+1} = Ax_t + Bu_t \quad (\text{LTI})$$

General dynamic optimization problem

Possible objectives in dynamic optimization



Typical objectives in control:

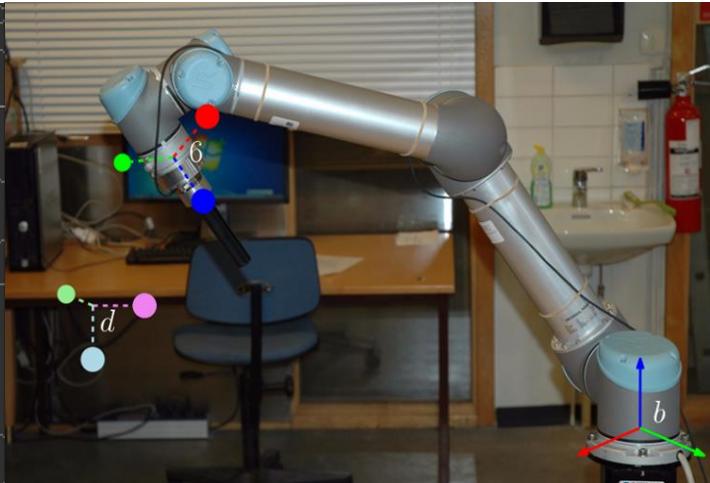
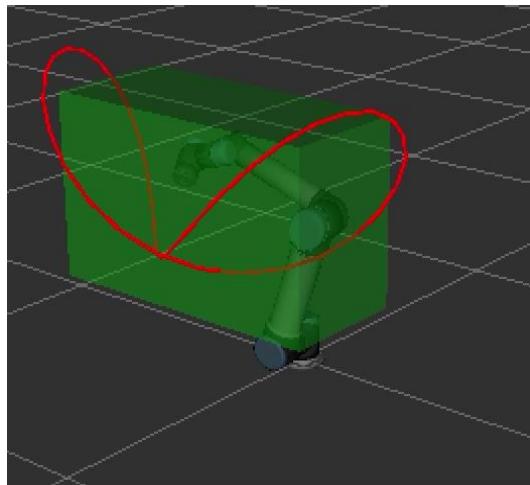
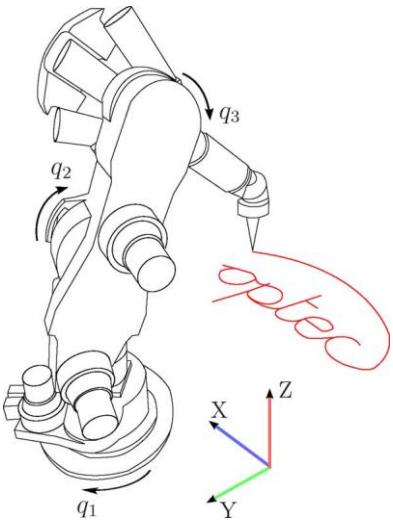
- Penalize deviations from a constant reference/setpoint (*regulation*), or deviations from a reference trajectory (*tracking*).

Other types of objectives:

- Economic objectives. Optimize economic profit: maximize production (e.g. oil), and/or minimize costs (e.g energy or raw material)
- Limit tear and wear of equipment (e.g. valves)
- Reach a specific endpoint as fast as possible
- Reach a specific endpoint, possibly avoiding obstacles

“Standard” stage costs in dynamic optimization

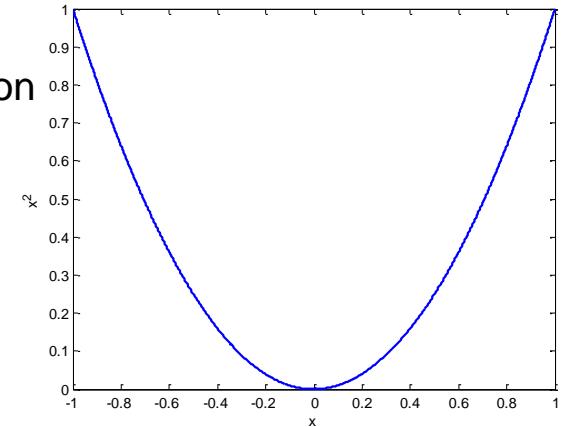
Examples tracking



Why quadratic objective?

Two main reasons:

- Because it is convenient, mathematically
 - Smooth is good, both for analysis and numerical optimization
 - Give linear gradients
- Because it is natural; the effect is often desirable
 - Tends to ignore small deviations
 - Tends to punish large deviations
- However, other types of objective functions are also used



Standard linear dynamic optimization problem

Standard linear dynamic optimization problem

Batch approach v1, “Full space”

Standard linear dynamic optimization problem

Batch approach v2, “Reduced space”

Standard linear dynamic optimization problem

Batch approach v2, “Reduced space”

Linear quadratic control: Dynamic optimization without constraints

$$\begin{aligned} \min_z \quad & \sum_{t=0}^{N-1} x_{t+1}^\top Q x_{t+1} + u_t^\top R u_t \\ \text{s.t.} \quad & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ & z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

Three approaches for implementation:

- Batch approach v1, “full space” – solve as QP
- Batch approach v2, “reduced space” – solve as QP
- Recursive approach – solve as linear state feedback

Linear Quadratic Control

Batch approach v1, “Full space” QP

- Formulate with model as equality constraints, all inputs and states as optimization variables

$$\min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1$$

$$z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top$$

$$\begin{aligned} & \min_z \quad \frac{1}{2} z^\top \begin{pmatrix} R & & & \\ & Q & & \\ & & R & \\ & & & \ddots \\ & & & & Q \end{pmatrix} z \\ & \text{s.t. } \begin{pmatrix} -B & I & & & & & & \\ & -A & -B & I & & & & \\ & & -A & -B & I & & & \\ & & & \ddots & \ddots & & & \\ & & & & -A & -B & I & \end{pmatrix} z = \begin{pmatrix} Ax_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ & z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

Linear Quadratic Control

Batch approach v2, “Reduced space” QP

- Use model to eliminate states as variables
 - Future states as function of inputs and initial state

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{pmatrix} x_0 + \begin{pmatrix} B & & & \\ AB & B & & \\ A^2 & AB & B & \\ \vdots & \vdots & \vdots & \ddots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = S^x x_0 + S^u U$$

- Insert into objective (no constraints!)
- Solution found by setting gradient equal to zero:

$$U = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = -((S^u)^\top \mathbf{Q} S^u + \mathbf{R})^{-1} (S^u)^\top \mathbf{Q} S^x x_0 = -F x_0$$

$$\begin{aligned} \min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

$$\mathbf{Q} = \begin{pmatrix} Q & & \\ & Q & \\ & & \ddots \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} R & & \\ & R & \\ & & \ddots \end{pmatrix}$$

Linear Quadratic Control

Recursive approach

$$\begin{aligned} & \min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ & z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

- By writing up the KKT-conditions, we can show (we will do this later) that the solution can be formulated as:

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$K_t = R^{-1} B^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_t = Q + A^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_N = Q$$

Comments to the three solution approaches

- All give same numerical solution
 - If problem is strictly convex (Q psd, R pd), solution is unique
- The batch approaches give an open-loop solution, the recursive approach give a closed-loop (feedback) solution
 - Means the recursive solution is more robust in implementation

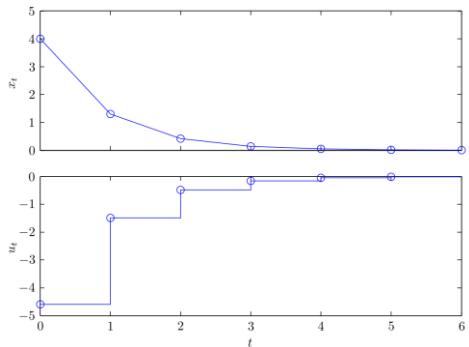
$$\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = -Fx_0 \quad \text{vs} \quad u_t = -K_t x_t$$

- Constraints on inputs and states:
 - Straightforward to add constraints as inequalities to batch approaches (both becomes convex QPs)
 - Much more difficult to add constraints to the recursive approach
- How to add feedback (and thereby robustness) to batch approaches?
 - Model predictive control!

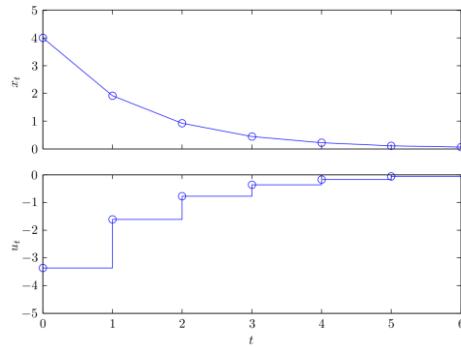
The significance of weights

$$\begin{aligned} & \min \sum_{t=0}^5 q x_{t+1}^2 + r u_t^2 \\ \text{s.t. } & x_{t+1} = 0.9x_t + 0.5u_t, \quad t = 0, \dots, N-1 \end{aligned}$$

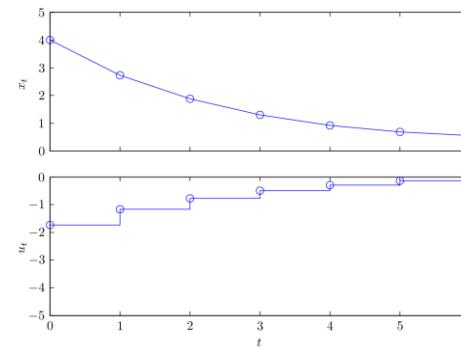
$q = 5, r = 1$



$q = 2, r = 1$



$q = 1, r = 2$



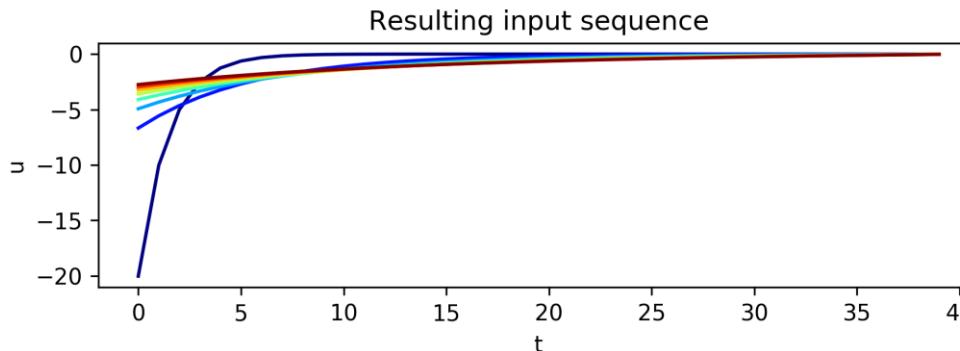
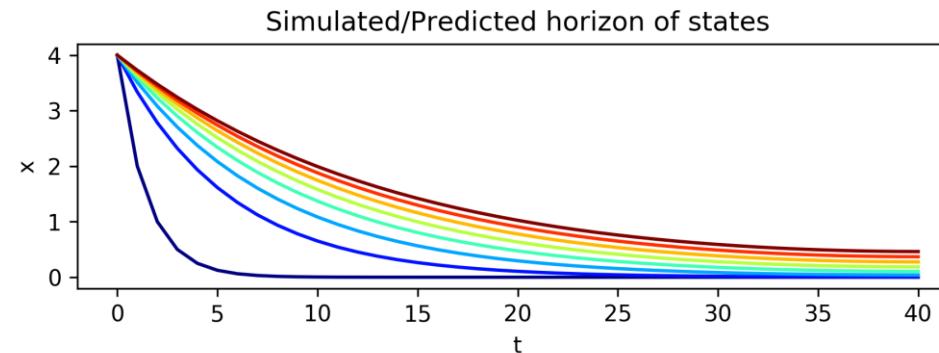
$$\sum_{t=0}^{N-1} x_{t+1}^2 = 1.9, \quad \sum_{t=0}^{N-1} u_t^2 = 23.6$$

$$\sum_{t=0}^{N-1} x_{t+1}^2 = 4.8, \quad \sum_{t=0}^{N-1} u_t^2 = 14.7$$

$$\sum_{t=0}^{N-1} x_{t+1}^2 = 14.3, \quad \sum_{t=0}^{N-1} u_t^2 = 5.3$$

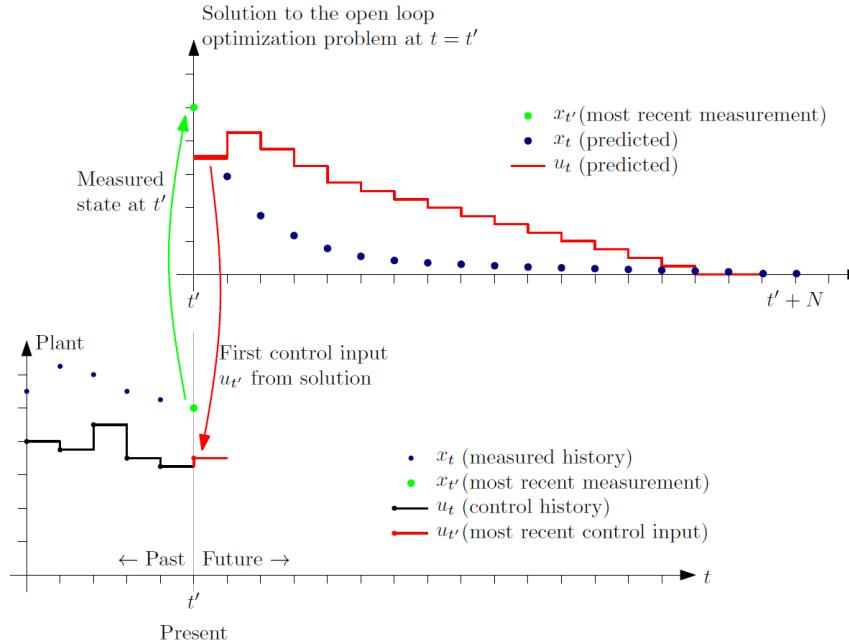
Significance of weights – Ratios

$$x_{t+1} = 1.001x_t + 0.1u_t, q = 5, r \in [0.1, \dots, 10]$$



Open loop vs closed loop

- Next time: How to use open-loop optimization for closed-loop (feedback!)
 - This is called Model Predictive Control





NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 9

Linear Quadratic Control

Lecturer: Lars Imsland

Outline

- Recap: Open-loop linear dynamic optimization problem
- Recap: Three ways of solving this
 - Two batch methods (-> QPs)
 - One recursive method
- Today: Linear Quadratic Control (= “The recursive method”)
 - Finite horizon
 - Infinite horizon

Reference: F&H Ch. 4.3-4.4

Last time: Dynamic open-loop optimization (with linear state-space model)

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + d_{x,t+1}^\top x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + d_{u,t}^\top u_t + \frac{1}{2} \Delta u_t^\top S \Delta u_t$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$-\Delta u^{\text{high}} \leq \Delta u_t \leq \Delta u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$Q_t \succeq 0, \quad t = \{1, \dots, N\}$$

$$R_t \succ 0, \quad t = \{0, \dots, N-1\}$$

where

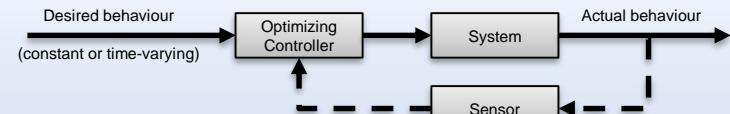
x_0 and u_{-1} is given

$$\Delta u_t := u_t - u_{t-1}$$

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

$$n = N \cdot (n_x + n_u)$$

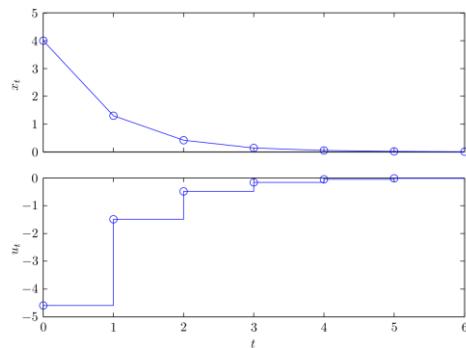
Open-loop vs closed-loop/feedback:



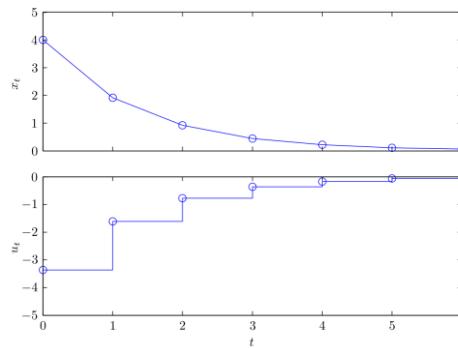
The significance of weights

$$\begin{aligned} & \min \sum_{t=0}^5 q x_{t+1}^2 + r u_t^2 \\ \text{s.t. } & x_{t+1} = 0.9x_t + 0.5u_t, \quad t = 0, \dots, 5 \end{aligned}$$

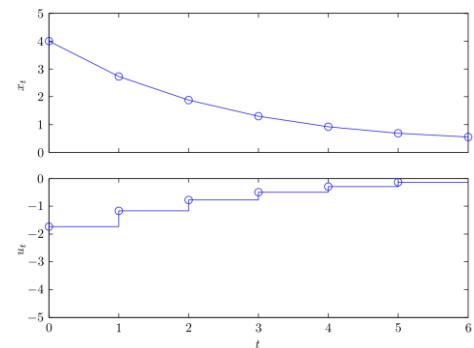
$$q = 5, r = 1$$



$$q = 2, r = 1$$



$$q = 1, r = 2$$



$$\sum_{t=0}^{N-1} x_{t+1}^2 = 1.9, \quad \sum_{t=0}^{N-1} u_t^2 = 23.6$$

$$\sum_{t=0}^{N-1} x_{t+1}^2 = 4.8, \quad \sum_{t=0}^{N-1} u_t^2 = 14.7$$

$$\sum_{t=0}^{N-1} x_{t+1}^2 = 14.3, \quad \sum_{t=0}^{N-1} u_t^2 = 5.3$$

Linear quadratic control: Dynamic optimization without constraints

$$\begin{aligned} \min_z \quad & \sum_{t=0}^{N-1} x_{t+1}^\top Q x_{t+1} + u_t^\top R u_t \\ \text{s.t.} \quad & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ & z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

Three approaches for solution:

- Batch approach v1, “full space” – solve as QP
- Batch approach v2, “reduced space” – solve as QP
- Recursive approach – solve as linear state feedback



Also work with input- and state constraints!

Only work without constraints!

Linear Quadratic Control

Batch approach v1, “Full space” QP

- Formulate with model as equality constraints, all inputs and states as optimization variables

$$\min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1$$

$$z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top$$

$$\left\{ \begin{array}{l} \min_z \frac{1}{2} z^\top \begin{pmatrix} R & & & \\ & Q & & \\ & & R & \\ & & & \ddots \\ & & & & Q \end{pmatrix} z \\ \text{s.t. } \begin{pmatrix} -B & I & & & & & & \\ & -A & -B & I & & & & \\ & & -A & -B & I & & & \\ & & & -A & -B & I & & \\ & & & & \ddots & \ddots & & \\ & & & & & -A & -B & I \end{pmatrix} z = \begin{pmatrix} Ax_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{array} \right.$$

Linear Quadratic Control

Batch approach v2, “Reduced space” QP

- Use model to eliminate states as variables
 - Future states as function of inputs and initial state

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{pmatrix} x_0 + \begin{pmatrix} B & B & B \\ AB & AB & AB \\ A^2 & A^3 & A^4 \\ \vdots & \vdots & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = S^x x_0 + S^u U$$

- Insert into objective (no constraints!)

$$\min_U \frac{1}{2} (S^x x_0 + S^u U)^\top \mathbf{Q} (S^x x_0 + S^u U) + \frac{1}{2} U^\top \mathbf{R} U$$

- Solution found by setting gradient equal to zero:

$$U = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = -((S^u)^\top \mathbf{Q} S^u + \mathbf{R})^{-1} (S^u)^\top \mathbf{Q} S^x x_0 = -F x_0$$

$$\begin{aligned} \min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

$$\mathbf{Q} = \begin{pmatrix} Q & & \\ & Q & \\ & & \ddots \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} R & & \\ & R & \\ & & \ddots \end{pmatrix}$$

Linear Quadratic Control

Recursive approach

$$\begin{aligned} & \min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\ & z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top \end{aligned}$$

- By writing up the KKT-conditions, we can show (we will do this today) that the solution can be formulated as:

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$K_t = R^{-1} B^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_t = Q + A^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_N = Q$$

Comments to the three solution approaches

- All give same numerical solution
 - If problem is strictly convex (Q psd, R pd), solution is unique
- The batch approaches give an open-loop solution, the recursive approach give a closed-loop (feedback) solution

$$\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = -Fx_0 \quad \text{vs} \quad u_t = -K_t x_t$$

- Constraints on inputs and states:
 - Easy for batch approaches (both becomes convex QPs)
 - Difficult for the recursive approach
- How to add feedback (and thereby robustness) to batch approaches?
 - Model predictive control! (Next time)

Today: The recursive solution (LQ control)

KKT Conditions (Thm 12.1)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} f(x) & \text{subject to} \\ c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{array}$$

Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{(primal feasibility)} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array}$$

(dual feasibility)
(complementarity condition/
complementary slackness)

LQR:

$$\min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

s.t. $x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1$

$$z = (u_0, x_1, u_1, \dots, u_{N-1}, x_N)^\top$$

KKT:

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

Second-order conditions

Theorem 12.6 (Second-Order Sufficient Conditions).

Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions (12.34) are satisfied. Suppose also that

$$w^T \nabla_{xx}^2 \mathcal{L}(x^*, \lambda^*) w > 0, \quad \text{for all } w \in \mathcal{C}(x^*, \lambda^*), w \neq 0. \quad (12.65)$$

Then x^* is a strict local solution for (12.1).

- Critical directions:

$$w \in \mathcal{C}(x^*, \lambda^*) \Leftrightarrow \begin{cases} \nabla c_i(x^*)^T w = 0, & \text{for all } i \in \mathcal{E}, \\ \nabla c_i(x^*)^T w = 0, & \text{for all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^T w \geq 0, & \text{for all } i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* = 0. \end{cases} \quad (12.53)$$

- The critical directions are the “allowed” directions where it is not clear from KKT-conditions whether the objective will decrease or increase

Thm 16.4: For convex QP, KKT is sufficient

- From N&W, p. 464:

For convex QP, when G is positive semidefinite, the conditions (16.37) are in fact sufficient for x^* to be a global solution, as we now prove.

KKT conditions

Theorem 16.4.

If x^* satisfies the conditions (16.37) for some $\lambda_i^*, i \in \mathcal{A}(x^*)$, and G is positive semidefinite, then x^* is a global solution of (16.1).

- That is: Since the solution of the Riccati equation implies the KKT conditions are fulfilled, Thm 16.4 means that the Riccati equation gives the global solution
 - Side-remark: It is, in fact, the *unique* global solution. If G is positive definite (implied by Q positive definite), this follows from the proof of Thm 16.4. If Q positive semidefinite, further arguments are necessary (for instance using Thm 12.6 as in the note).

- Finite horizon LQ controller

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t$$

subject to $x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \dots, N-1$

x_0 = given

$Q_t \succeq 0 \quad t = 1, \dots, N$

$R_t \succ 0 \quad t = 0, \dots, N-1$

where

$$\begin{aligned} z^\top &:= (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top) \\ n &= N \cdot (n_x + n_u) \end{aligned}$$

- State feedback solution

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$K_t = R_t^{-1} B_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, \quad t = 0, \dots, N-1$$

$$P_t = Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, \quad t = 0, \dots, N-1$$

$$P_N = Q_N$$

(discrete) Riccati equation 

Linear quadratic control (finite horizon)

- The optimal solution to LQ control is a linear, time-varying state feedback:

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$\begin{aligned} K_t &= R_t^{-1} B_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t = 0, \dots, N-1 \\ P_t &= Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t = 0, \dots, N-1 \\ P_N &= Q_N \end{aligned}$$

- Note that the gain matrix K_t is independent of the states, and can therefore be computed in advance (knowing A_t , B_t , Q_t , R_t)
- The matrix (difference) equation

$$\begin{aligned} P_t &= Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t = 0, \dots, N-1 \\ P_N &= Q_N \end{aligned}$$

is called the (discrete) *Riccati equation*

- Note that the “boundary condition” is given at the end of the horizon, and the P_t -matrices must be found iterating backwards in time

$$u_t = -K_t x_t$$

Example

where the feedback gain matrix is derived by

$$K_t = R^{-1} B^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

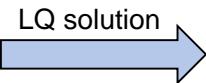
$$P_t = Q + A^\top P_{t+1} (I + BR^{-1}B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_N = Q$$

Example

$$\min \sum_{t=0}^{10} \frac{1}{2} x_{t+1}^2 + \frac{1}{2} r u_t^2$$

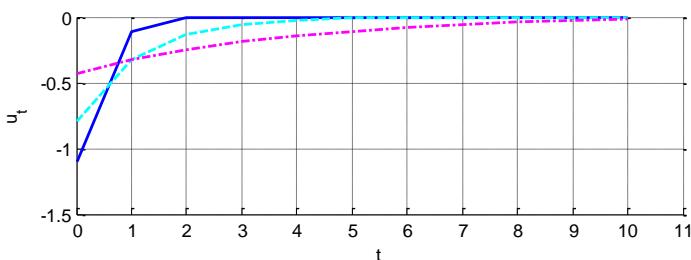
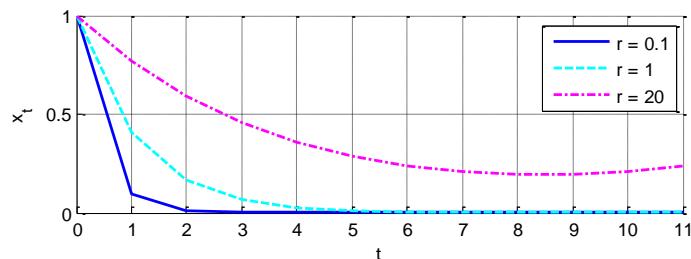
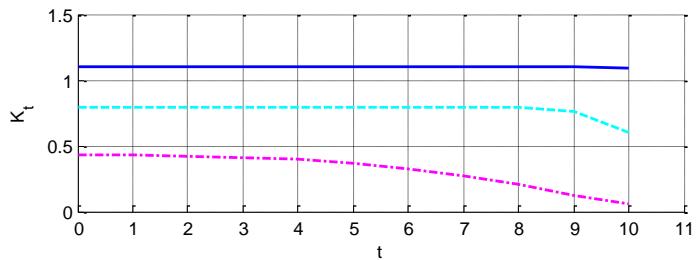
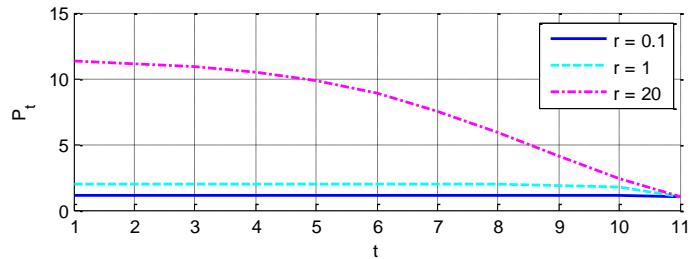
$$\text{s.t. } x_{t+1} = 1.2x_t + u_t, \quad t = 0, 1, \dots, 10$$

LQ solution 

$$P_t = 1 + \frac{1.44rP_{t+1}}{P_{t+1} + r}, \quad t = 10, \dots, 1$$

$$P_{11} = 1$$

$$K_t = 1.2 \frac{P_{t+1}}{P_{t+1} + r}, \quad t = 0, \dots, 10$$

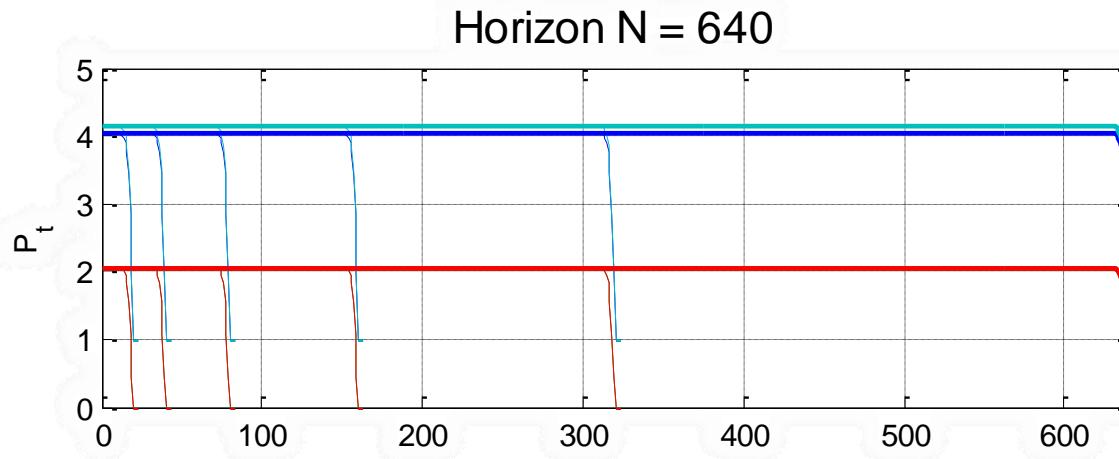


Increasing LQ horizon

$$\min \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1$$

$$A = \begin{pmatrix} 1 & 0.5 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.125 \\ 0.5 \end{pmatrix}, \quad Q = I, \quad R = 1.$$



Infinite horizon LQ control

$$\begin{aligned} \min & \sum_{t=0}^{\infty} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots \end{aligned}$$

Fact: Steady-state ($P_{t+1} = P_t$) backwards-in-time solution of Riccati equation is **infinite horizon solution**

$$u_t = -K_t x_t$$

where

$$K_t = R^{-1} B^\top P_{t+1} (I + BR^{-1} B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_t = Q + A^\top P_{t+1} (I + BR^{-1} B^\top P_{t+1})^{-1} A, \quad t = 0, \dots, N-1$$

$$P_N = Q$$



$$u_t = -K x_t$$

where

$$K = R^{-1} B^\top P (I + BR^{-1} B^\top P)^{-1} A$$

$$P = Q + A^\top P (I + BR^{-1} B^\top P)^{-1} A$$

Infinite horizon LQ control

Theorem: The solution (when one exists) to

$$\begin{aligned} \min & \sum_{t=0}^{\infty} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots \end{aligned}$$

is given by

$$u_t = -Kx_t$$

where

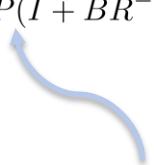
$$K = R^{-1}B^\top P(I + BR^{-1}B^\top P)^{-1}A$$

$$P = Q + A^\top P(I + BR^{-1}B^\top P)^{-1}A$$

Two central questions:

- When does a solution exist?
- When is the closed-loop stable?

(Discrete-time Algebraic Riccati Equation, DARE)



Controllability vs stabilizability

Observability vs detectability

- Stabilizable: All unstable modes are controllable
(that is: all uncontrollable modes are stable)
- Detectability: All unstable modes are observable
(that is: all unobservable modes are stable)
- Controllability implies stabilizability
- Observability implies detectability

Riccati equations

- Discrete-time Riccati equation in the note (and lecture)

$$P_t = Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, \quad P_N = Q_N$$

- However, another, equivalent, form is found in other sources:

$$P_t = Q_t + A_t^\top P_{t+1} A_t - A_t^\top P_{t+1} B_t (R_t + B_t^\top P_{t+1} B_t)^{-1} B_t^\top P_{t+1} A_t, \quad P_N = Q_N$$

- The latter is more numerically stable due to “enforced symmetry”

- The trick used to get the different formulas is the “Matrix Inversion Lemma” (a very useful Lemma in control theory, optimization, ...)

- Discrete-time Algebraic Riccati equation (DARE) in the note (and lecture)

$$P = Q + A^\top P (I + B R^{-1} B^\top P)^{-1} A$$

- Equivalent form (e.g. Matlab)

$$P = Q + A^\top P A - A^\top P B (R + B^\top P B)^{-1} B^\top P A$$

- Note: This is a quadratic equation with two solutions. The one we want is the positive definite solution (the “stabilizing” solution).

```
>> help dare  
dare Solve discrete-time algebraic Riccati equations.
```

`[X,L,G] = dare(A,B,Q,R,S,E)` computes the unique stabilizing solution `X` of the discrete-time algebraic Riccati equation



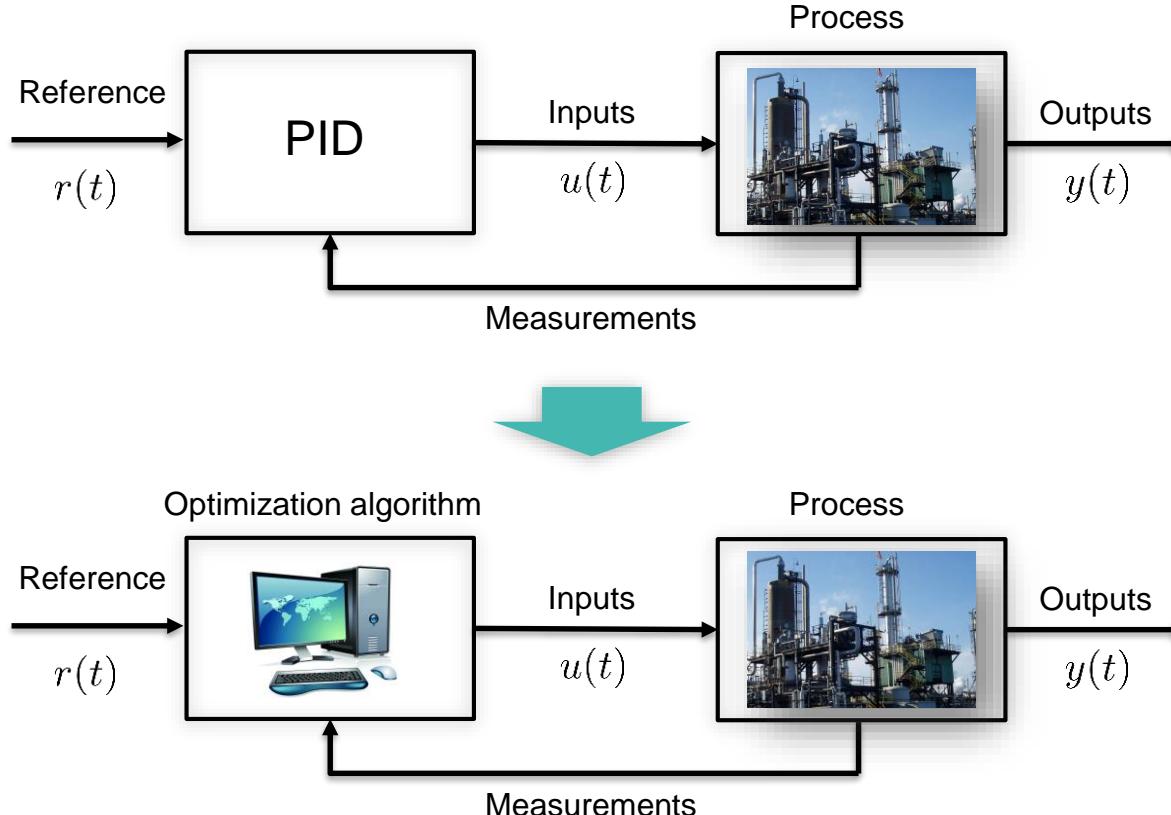
NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 10

Model Predictive Control

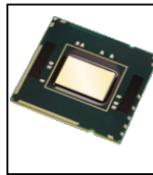
Lecturer: Lars Imsland

Model Predictive control – control based on optimization



A **model** of the process is used to compute the **control** signals (inputs) that optimize **predicted** future process behavior

MPC: Applications

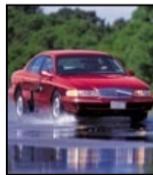


Computer control

ns



Power systems



Traction control

μs



Buildings

Seconds



Refineries

Minutes



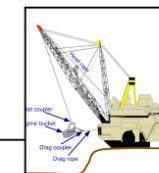
Nurse rostering

Hours



Train scheduling

Days



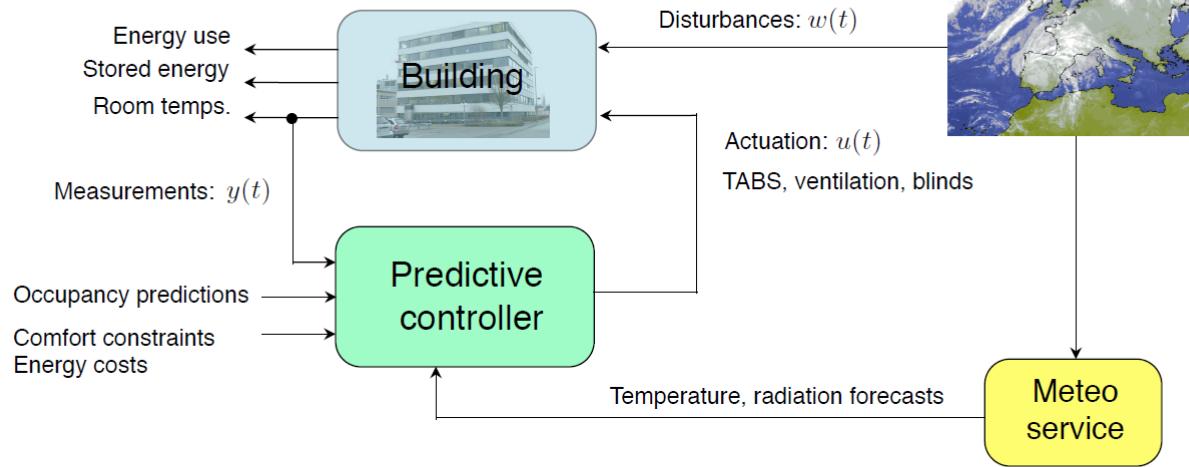
Production planning

Weeks

Zeilinger, Jones, Borrelli, Morari (ETH)

Model predictive control (MPC)

Principle of operation



$$\text{Predicted Cost} = \underset{u(t)}{\text{minimize}} \text{ Expected} \left(\sum_t^{t+N} \text{energy cost}(t) \right)$$

Minimize the predicted energy cost

$$\begin{aligned} \text{subject to } u(t) &\in \mathcal{U} && \text{Actuation within limits} \\ x(t) &\in \mathcal{X} && \text{Predicted temperatures within limits} \\ x(t+1) &= f(x(t), u(t), w(t)) && \text{Predicted dynamics of the building} \end{aligned}$$

From ETH

Open-loop optimization with linear state-space model

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + d_{x,t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + d_{u,t} u_t + \frac{1}{2} \Delta u_t^\top S \Delta u_t$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$-\Delta u^{\text{high}} \leq \Delta u_t \leq \Delta u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

QP

where

x_0 and u_{-1} is given

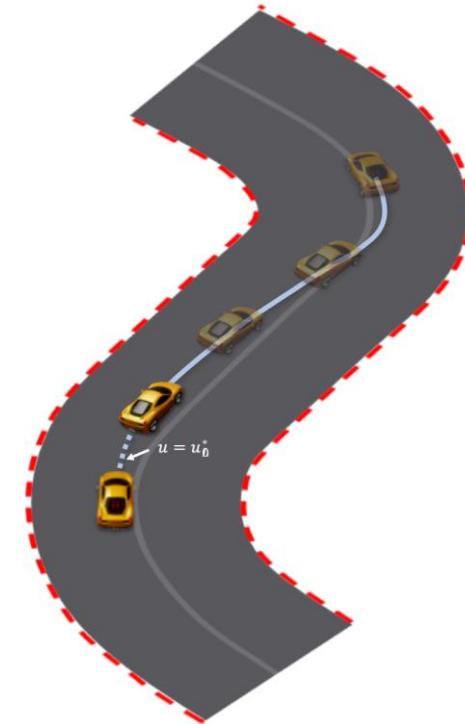
$$\Delta u_t := u_t - u_{t-1}$$

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

$$n = N \cdot (n_x + n_u)$$

$$Q_t \succeq 0 \quad t = \{1, \dots, N\}$$

$$R_t \succeq 0 \quad t = \{0, \dots, N-1\}$$



Open-loop dynamic optimization problem as QP

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

subject to

$$x_{t+1} = Ax_t + Bu_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

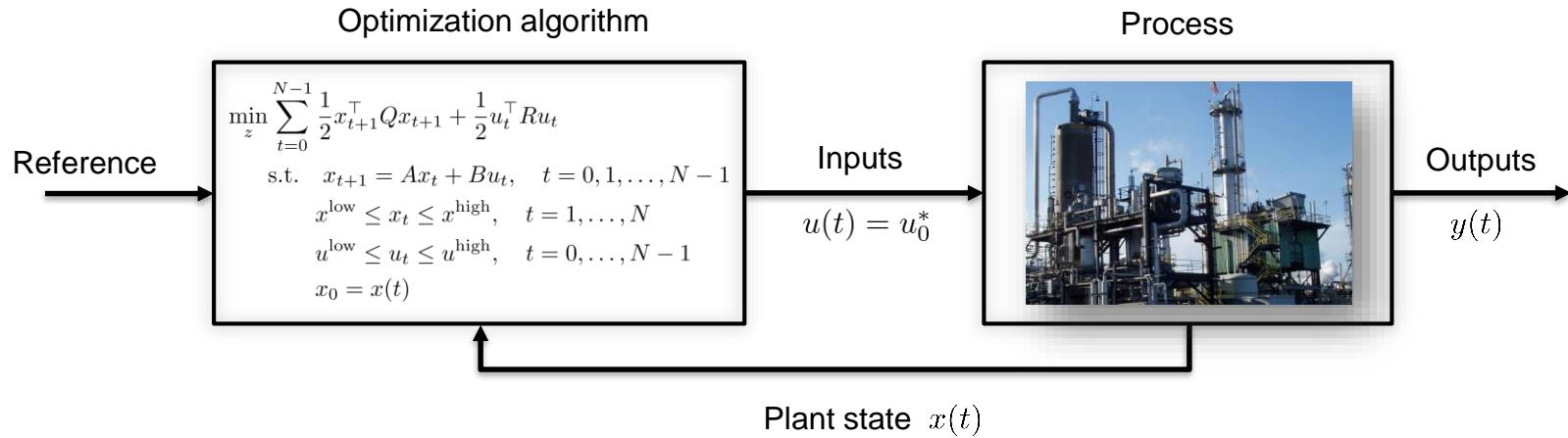
where

x_0 is given

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

$$Q \succeq 0, \quad R \succ 0$$

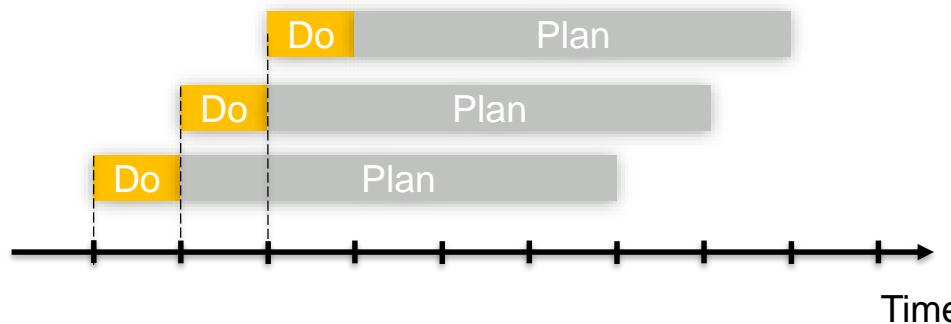
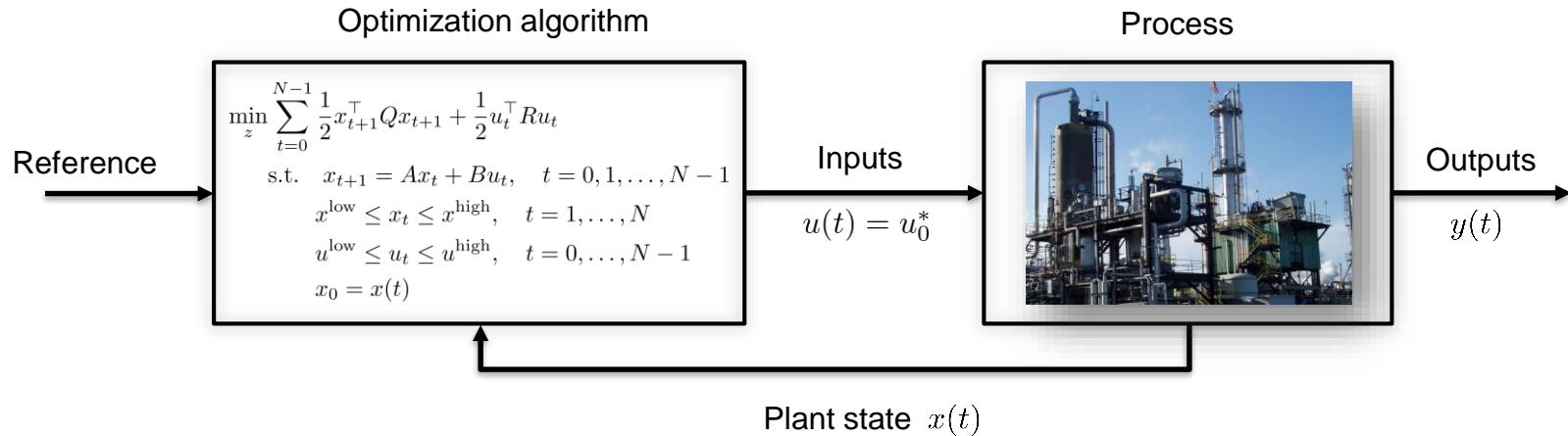
Model predictive control principle



At each sample time:

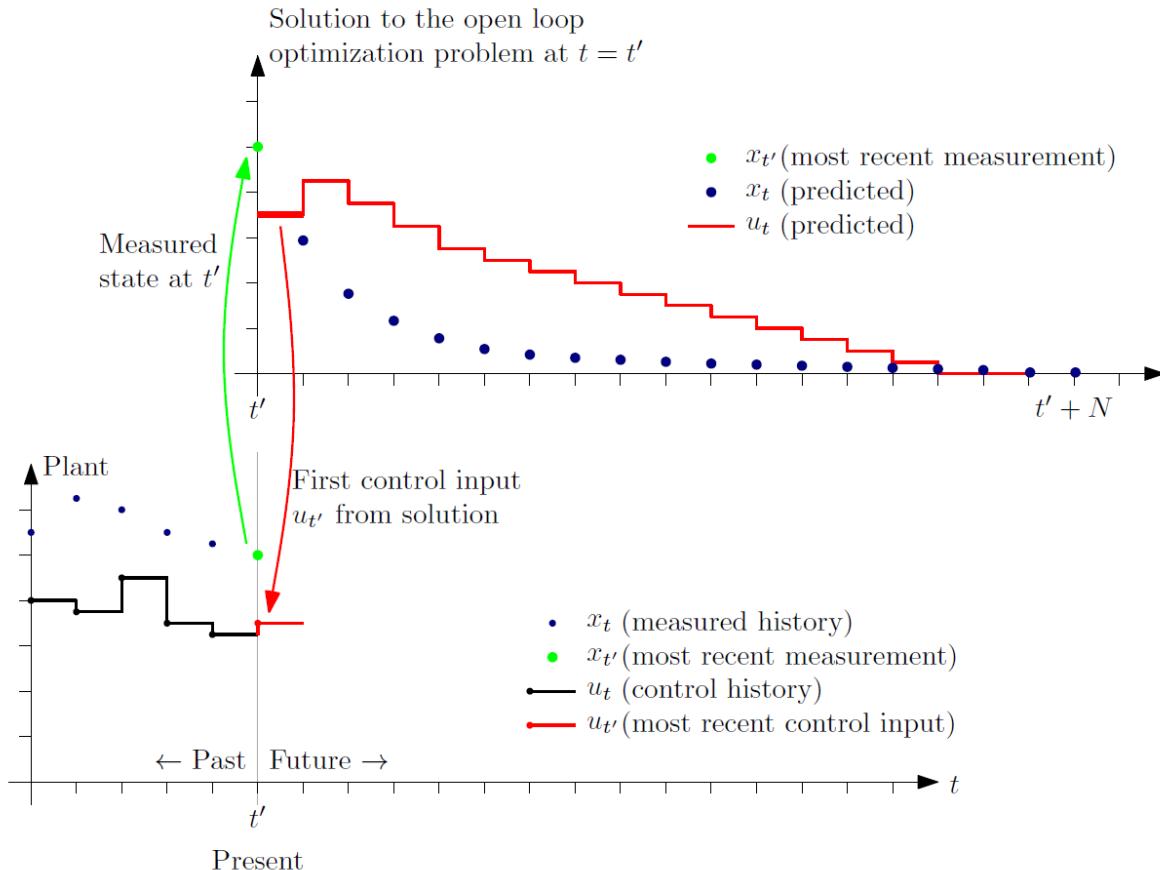
- Measure or estimate current state $x(t)$
- Find optimal input sequence $U^* = (u_0^*, u_1^*, \dots, u_{N-1}^*)$
- Implement only the first element of sequence: $u(t) = u_0^*$

Model predictive control principle



Why? This introduces feedback!

Model predictive control principle



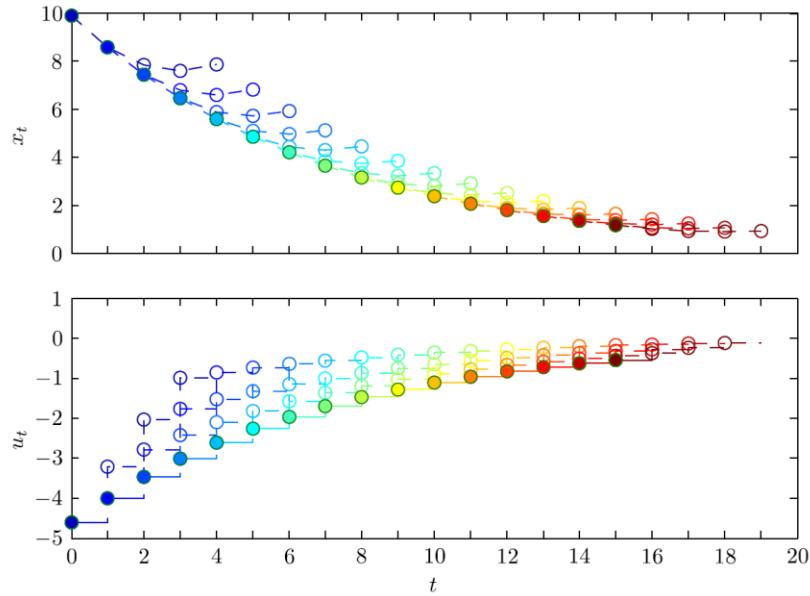
(MPC animation)

Feasibility (or: is there a solution to the QP?)

Open-loop vs closed-loop trajectories

$$\min \sum_{t=0}^4 x_{t+1}^2 + 4 u_t^2$$

$$\text{s.t. } x_{t+1} = 1.2x_t + 0.5u_t, \quad t = 0, \dots, 4$$



- Closed-loop trajectories different from open-loop (optimized) trajectories!
- It is the closed-loop trajectories that must be analyzed for feasibility and stability.

Example: Is MPC always stable?

Design MPC for $x_{t+1} = 1.2x_t + u_t$, no constraints, $N = 2$

Example: Is MPC always stable?

Design MPC for $x_{t+1} = 1.2x_t + u_t$, no constraints, $N = 2$

MPC optimality implies stability?

$$\min \sum_{t=0}^1 x_{t+1}^2 + r u_t^2$$

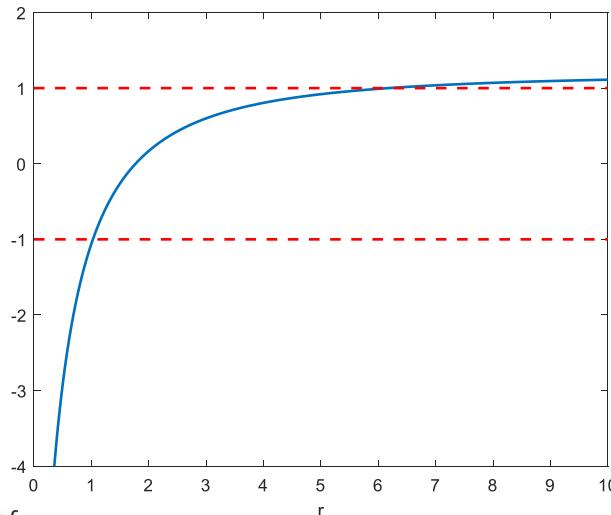
$$\text{s.t. } x_{t+1} = 1.2x_t + u_t, \quad t = 0, 1$$

MPC solution

$$u_t = -\frac{1.2 + 2.64r}{1 + 3.2r + r^2} x_t$$

MPC closed loop

$$x_{t+1} = \underbrace{\left(1.2 - \frac{1.2 + 2.64r}{1 + 3.2r + r^2}\right)}_{\text{MPC closed loop}} x_t$$



MPC and stability

Nominal vs robust stability

- “Nominal stability”: Stability when optimization model = plant model
 - No “model-plant mismatch”, no disturbances
- “Robust stability”: Stability when optimization model \neq plant model
 - “Model-plant mismatch” and/or disturbances (more difficult to analyze)

Requirements for nominal stability:

- Stabilizability ((A,B) stabilizable)
- Detectability ((A,D) detectable)
 - D is a matrix such that $Q = D^T D$ (that is, “ D is matrix square root of Q ”)
 - Detectability: No modes can grow to infinity without being “visible” through Q
- But more is needed to guarantee stability...

How to achieve nominal stability?

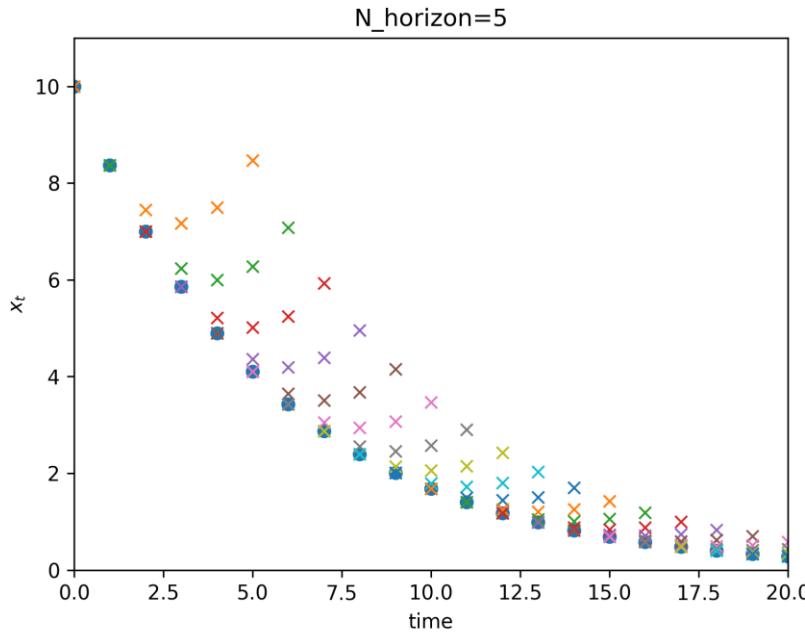
- Choose prediction horizon equal to infinity ($N = \infty$)
 - Usually not possible
- For given N , design Q and R such that MPC is stable (cf. example)
 - Difficult in general!
- Change the optimization problem such that
 - The new problem gives a finite upper bound of infinite horizon problem cost
 - The constraints is guaranteed to hold after the prediction horizon

$$\begin{aligned}
 & \min_z \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\
 \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\
 & x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = 1, \dots, N \\
 & u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = 0, \dots, N-1
 \end{aligned}$$

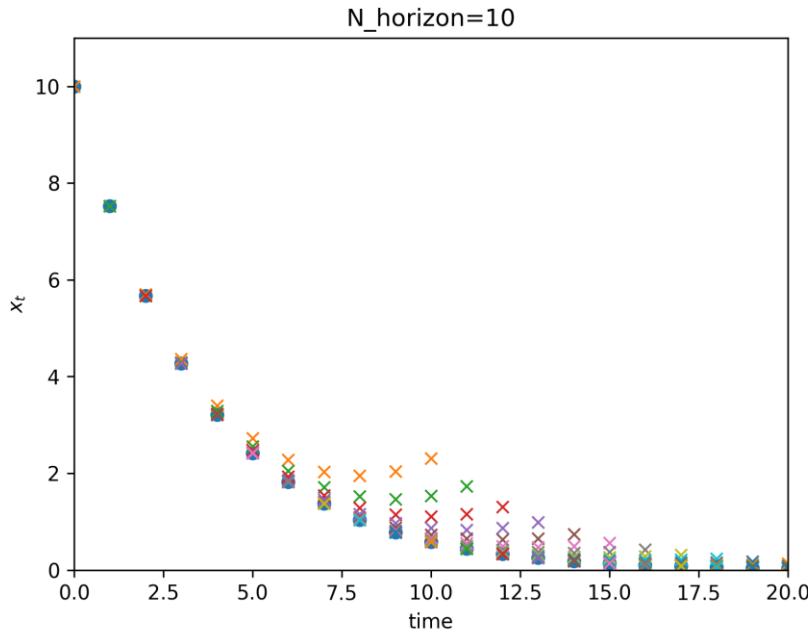
$$\begin{aligned}
 & \min_z \sum_{t=0}^{N-1} \left(\frac{1}{2} x_t^\top Q x_t + \frac{1}{2} u_t^\top R u_t \right) + \frac{1}{2} \cancel{x_N^\top P x_N} \quad \longleftarrow \text{Terminal cost} \\
 \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \\
 & x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = 1, \dots, N \\
 & u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = 0, \dots, N-1 \\
 & \cancel{x_N \in \mathcal{S}} \quad \longleftarrow \text{Terminal constraint}
 \end{aligned}$$

- Fairly straightforward to do in theory, but not always done
- Typically, in practice: Choose N “large”
 - Stability guaranteed for N large enough, but difficult/conservative to compute this limit
 - So what is “large enough” in practice? Rule of thumb: longer than dominating dynamics

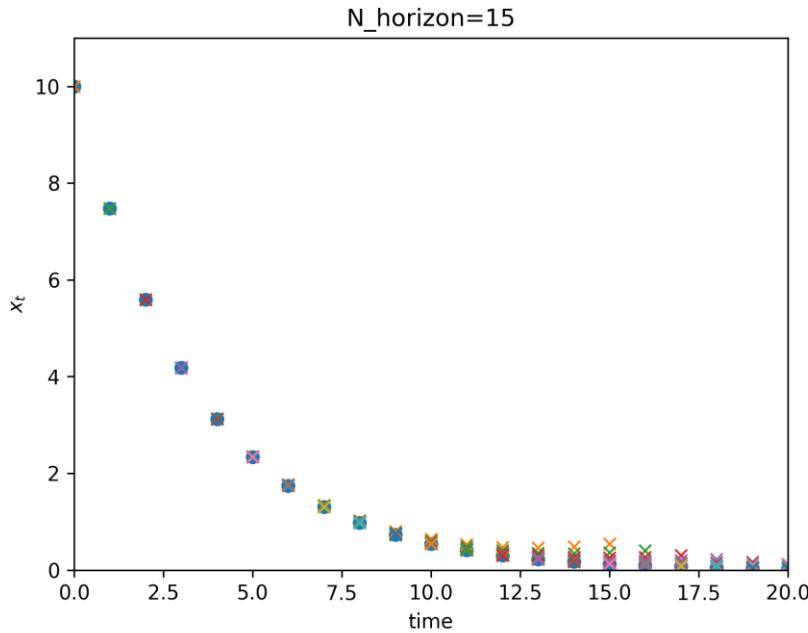
Open-Loop vs Closed-Loop: $N = 5$



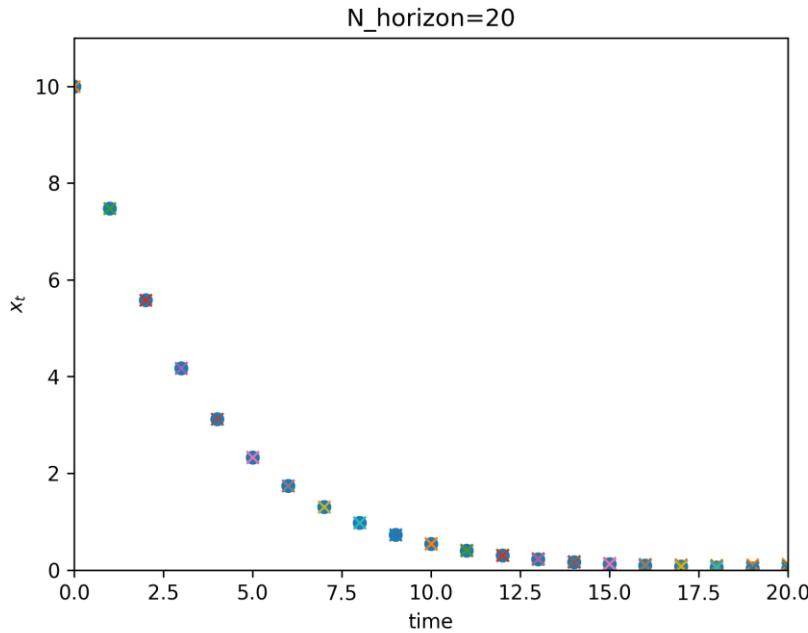
Open-Loop vs Closed-Loop: $N = 10$



Open-Loop vs Closed-Loop: $N = 15$



Open-Loop vs Closed-Loop: $N = 20$



Note: This assumes no model-plant mismatch!

Why MPC over PID control?

Advantages of MPC:

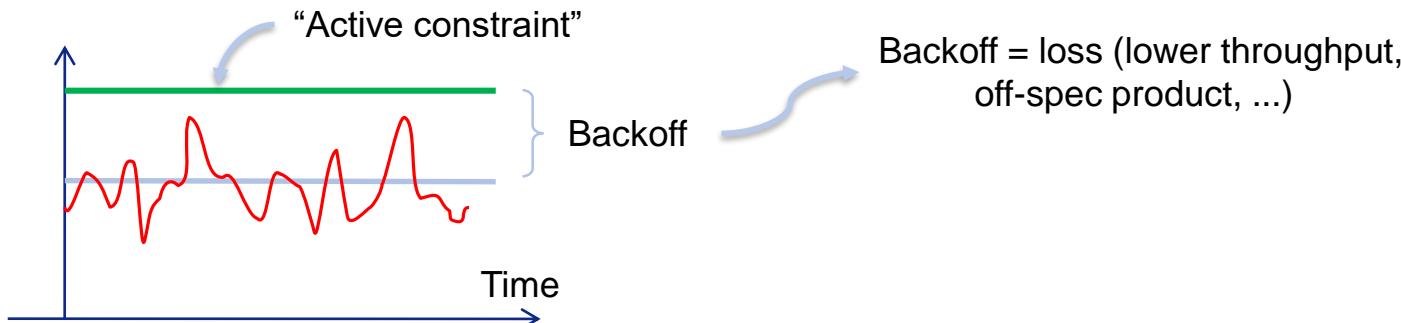
- MPC handles constraints in a transparent way
 - Physical constraints (actuator limits), performance constraints, safety limits, ...
- MPC is by design multivariable (MIMO)
- MPC gives “optimal” performance (but what is the optimal objective?)

Disadvantage with MPC

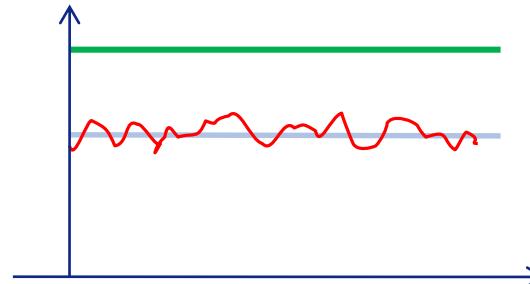
- Online complexity
- Requires models! Increased commissioning cost?
- Difficult to maintain?

“Squeeze and shift”

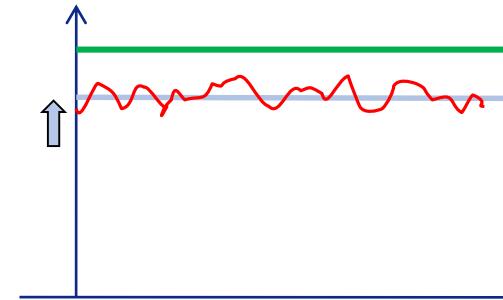
How MPC (or improved/advanced control in general) improves profitability



Backoff = loss (lower throughput,
off-spec product, ...)



Shift
(reduce backoff)





NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 11

More MPC: Output feedback, target calculation and offset-free control

Lecturer: Lars Imsland

Outline

- Recap: Model Predictive Control (MPC), Feasibility&stability

Common (necessary) features in practical MPC implementations:

- Output feedback
- Target calculation
- Offset-free MPC (integral action in MPC)

Reference: B&H Ch. 4.2.3-4.2.4

(Two articles containing more information on Blackboard – not curriculum)

Open-loop optimization with linear state-space model

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + d_{x,t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + d_{u,t} u_t + \frac{1}{2} \Delta u_t^\top S \Delta u_t$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$-\Delta u^{\text{high}} \leq \Delta u_t \leq \Delta u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

QP

where

x_0 and u_{-1} is given

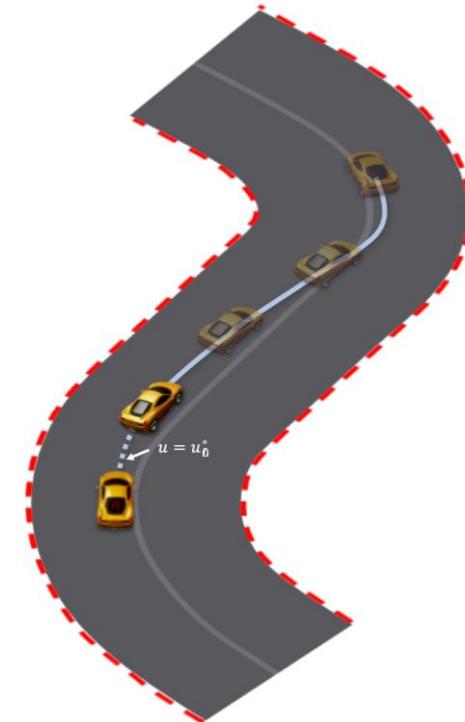
$$\Delta u_t := u_t - u_{t-1}$$

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

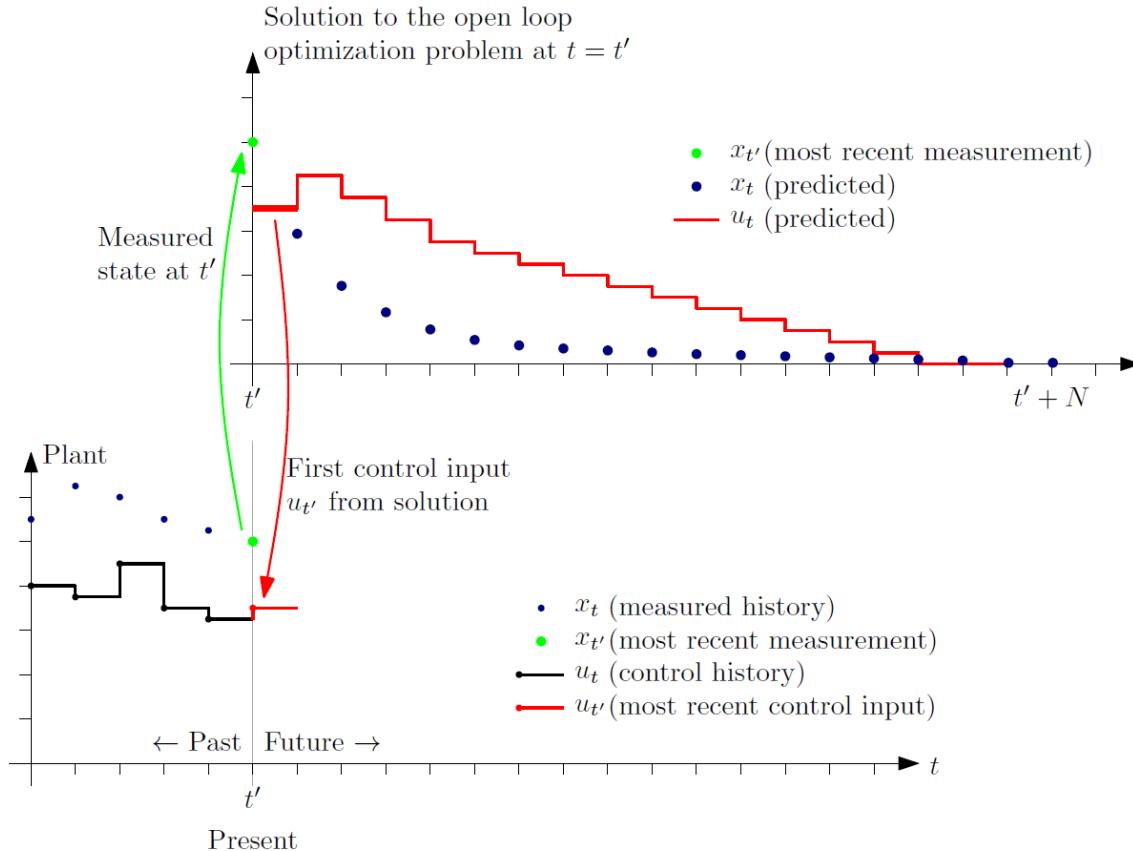
$$n = N \cdot (n_x + n_u)$$

$$Q_t \succeq 0 \quad t = \{1, \dots, N\}$$

$$R_t \succ 0 \quad t = \{0, \dots, N-1\}$$



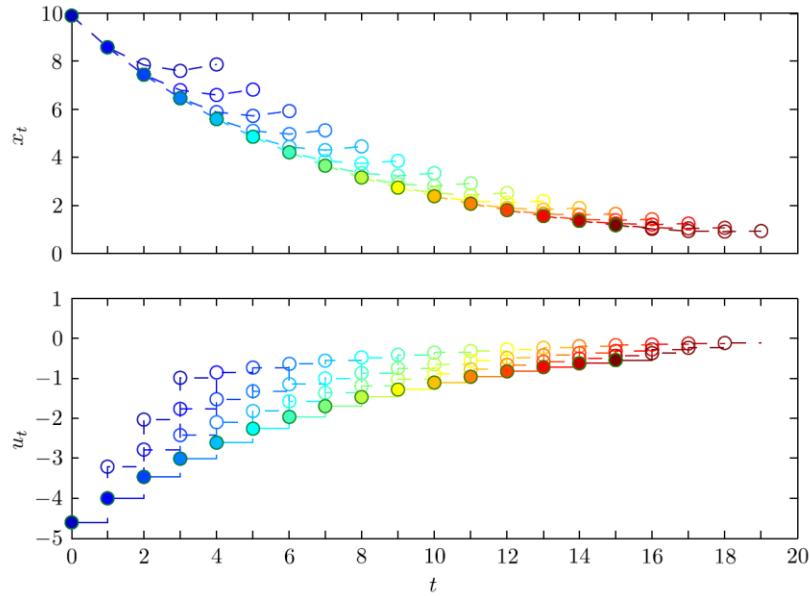
Model predictive control principle



Open-loop vs closed-loop trajectories

$$\min \sum_{t=0}^4 x_{t+1}^2 + 4 u_t^2$$

$$\text{s.t. } x_{t+1} = 1.2x_t + 0.5u_t, \quad t = 0, \dots, 4$$



- Closed-loop trajectories different from open-loop (optimized) trajectories!
- It is the closed-loop trajectories that must be analyzed for feasibility and stability.

MPC and feasibility

Is there always a solution to the MPC open-loop optimization problem?

- Not necessarily – state constraints may become infeasible, for example after a disturbance
- Practical solution: Soft constraints (or “exact penalty” formulations)
 - “Soften” state constraints by adding “slack variables”

$$\begin{aligned} \min_{z \in \mathbb{R}^n} f(z) &= \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + \rho^\top \epsilon \\ \text{s.t.} \quad x_{t+1} &= A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\} \\ x^{\text{low}} - \epsilon &\leq x_t \leq x^{\text{high}} + \epsilon, \quad t = \{1, \dots, N\}, \quad \epsilon > 0 \\ &\vdots \end{aligned}$$

MPC optimality implies stability?

$$\min \sum_{t=0}^1 x_{t+1}^2 + r u_t^2$$

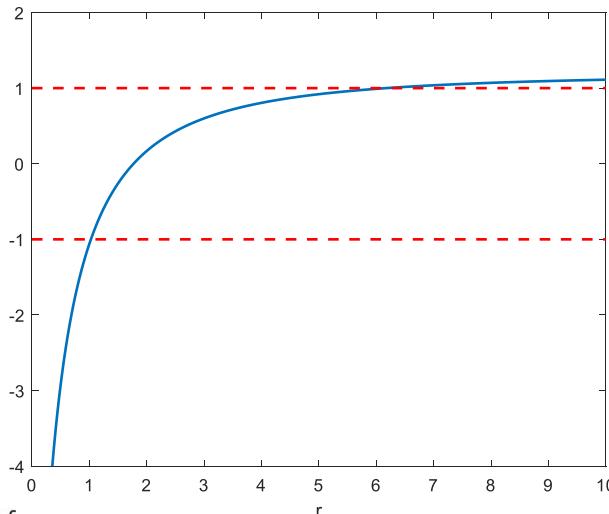
$$\text{s.t. } x_{t+1} = 1.2x_t + u_t, \quad t = 0, 1$$

MPC solution

$$u_t = -\frac{1.2 + 2.64r}{1 + 3.2r + r^2} x_t$$

MPC closed loop

$$x_{t+1} = \underbrace{\left(1.2 - \frac{1.2 + 2.64r}{1 + 3.2r + r^2}\right)}_{\text{MPC closed loop}} x_t$$



MPC and stability

Requirements for stability:

- Stabilizability ((A,B) stabilizable)
- Detectability ((A,D) detectable)
 - D is a matrix such that $Q = D^T D$ (that is, “ D is matrix square root of Q ”)
 - Detectability: No modes can grow to infinity without being “visible” through Q

How to design MPC schemes with guaranteed *nominal stability*:

- Choose prediction horizon equal to infinity (usually not possible)
- For given N , choose Q and R such that MPC is stable (cf. example)
 - Difficult, and not always possible!
- Change the optimization problem – add terminal cost/terminal constraints – such that
 - The new problem is an “upper approximation” of infinite horizon problem
 - The constraints holds after the prediction horizon
- Typically, in practice: Choose horizon N “large enough”
 - Usually works well!
 - What is “large enough”? Longer than dominating dynamics, but shorter can be OK.
 - Good practice: Choose N large enough such that open-loop predictions resembles closed-loop (test in simulations!)

MPC controller – state feedback

Output feedback MPC controller

Reference tracking

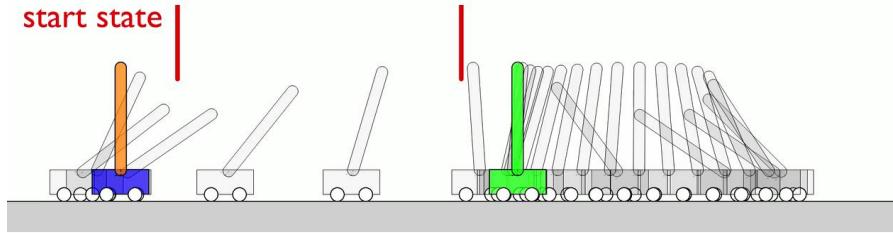
Reference tracking, cont'd

Reference tracking – target calculation

Offset-free control (= “integral action”)

Offset-free control (= “integral action”), cont’d

Offset-free MPC (or MPC with integral action)



From description:

- MPC with nonlinear model and a linear (input) disturbance model with one disturbance state: $x_t = f(x_t, u_t) + B_d d_t$. All states are measured ($y_t = x_t$).
- A linear observer is designed as a steady-state Kalman filter for the linearized augmented model at the final equilibrium.
- The forward-looking nature of the MPC controller allows to react to disturbances by considering obstacles in the environment and drastic replanning when necessary.
- From “Offset-free MPC explained: novelties, subtleties, and applications” - G. Pannocchia, M. Gabiccini, A. Artoni, NMPC 2015 - Seville, Spain September 17 - 20, 2015.



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 12

Summing up MPC and LQ

Lecturer: Lars Imsland

Outline

- MPC example: Adaptive Cruise Control
 - MPC design with offset-free control (disturbance observer + target calculation)
- LQ-control (recap), LQG, stability and robustness, separation principle

Reference: F&H Ch. 4.5-4.6

Open-loop optimization with linear state-space model

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + d_{x,t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + d_{u,t} u_t + \frac{1}{2} \Delta u_t^\top S \Delta u_t$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$-\Delta u^{\text{high}} \leq \Delta u_t \leq \Delta u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

QP

where

x_0 and u_{-1} is given

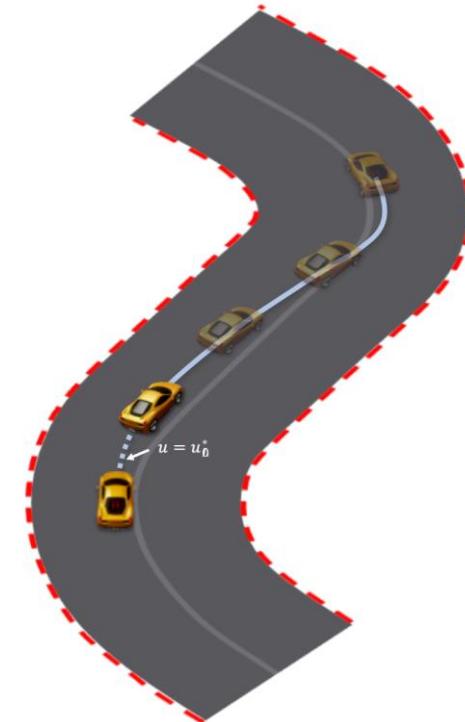
$$\Delta u_t := u_t - u_{t-1}$$

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

$$n = N \cdot (n_x + n_u)$$

$$Q_t \succeq 0 \quad t = \{1, \dots, N\}$$

$$R_t \succ 0 \quad t = \{0, \dots, N-1\}$$



LQ: MPC open loop problem without constraints

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t$$

subject to

$$\begin{aligned}x_{t+1} &= A_t x_t + B_t u_t, \quad t = 0, \dots, N-1 \\x_0 &= \text{given}\end{aligned}$$

where

$$\begin{aligned}z^\top &:= (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top) \\n &= N \cdot (n_x + n_u) \\Q_t &\succeq 0 \quad t = \{1, \dots, N\} \\R_t &\succ 0 \quad t = \{0, \dots, N-1\}\end{aligned}$$

- Solution: LTV state feedback

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$\begin{aligned}K_t &= R_t^{-1} B_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t &= 0, \dots, N-1 \\P_t &= Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t &= 0, \dots, N-1 \\P_N &= Q_N\end{aligned}$$

Linear quadratic control; some observations

- The optimal solution to LQ control is a linear, time-varying state feedback:

$$u_t = -K_t x_t$$

where the feedback gain matrix is derived by

$$\begin{aligned} K_t &= R_t^{-1} B_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t &= 0, \dots, N-1 \\ P_t &= Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t &= 0, \dots, N-1 \\ P_N &= Q_N \end{aligned}$$

- The matrix (difference) equation

$$\begin{aligned} P_t &= Q_t + A_t^\top P_{t+1} (I + B_t R_t^{-1} B_t^\top P_{t+1})^{-1} A_t, & t &= 0, \dots, N-1 \\ P_N &= Q_N \end{aligned}$$

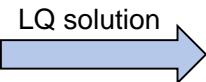
is called the (discrete-time) *Riccati equation*.

- Note that the gain matrix K_t and the Riccati equation is independent of the states. It can therefore be computed in advance (knowing A_t , B_t , Q_t , R_t).
- Note that the “boundary condition” is given at the end of the horizon, and the P_t -matrices must be found iterating backwards in time.

Example

$$\min \sum_{t=0}^{10} \frac{1}{2} x_{t+1}^2 + \frac{1}{2} r u_t^2$$

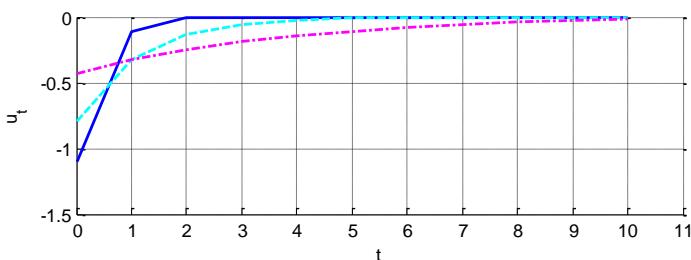
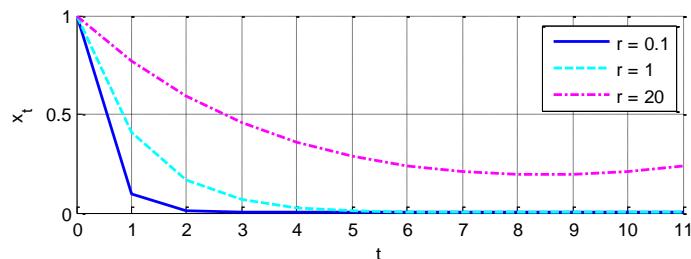
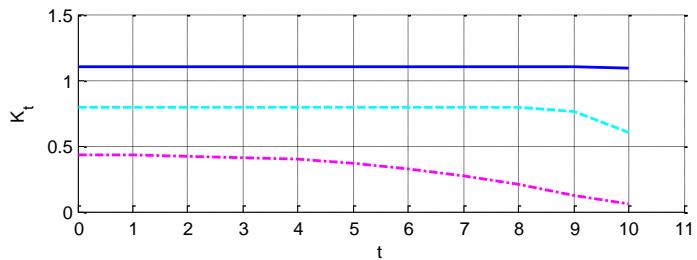
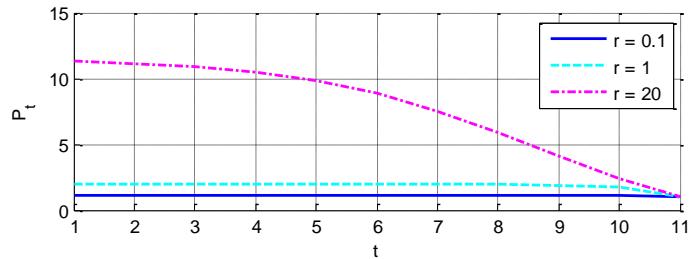
$$\text{s.t. } x_{t+1} = 1.2x_t + u_t, \quad t = 0, 1, \dots, 10$$

LQ solution 

$$P_t = 1 + \frac{1.44rP_{t+1}}{P_{t+1} + r}, \quad t = 10, \dots, 1$$

$$P_{11} = 1$$

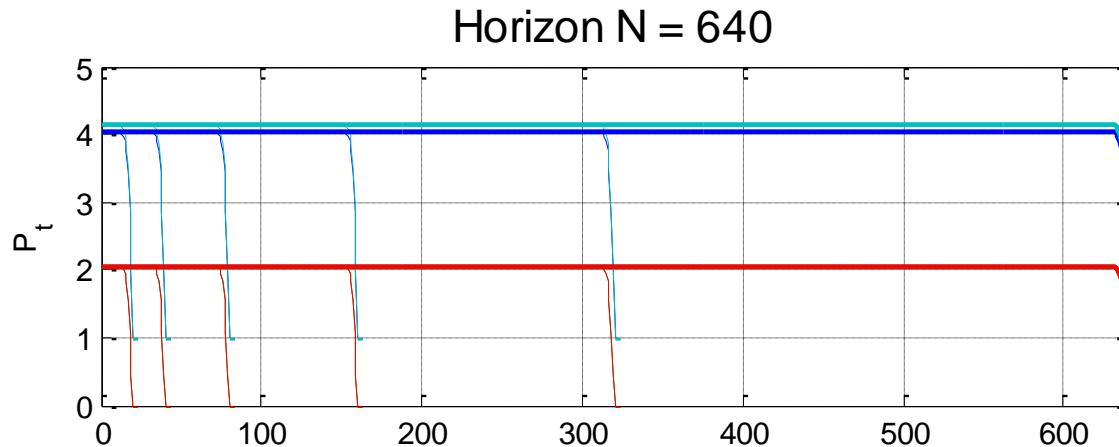
$$K_t = 1.2 \frac{P_{t+1}}{P_{t+1} + r}, \quad t = 0, \dots, 10$$



Increasing LQ horizon

$$\begin{aligned} \min & \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots, N-1 \end{aligned}$$

$$A = \begin{pmatrix} 1 & 0.5 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.125 \\ 0.5 \end{pmatrix}, \quad Q = I, \quad R = 1.$$



Infinite horizon LQ solution is steady-state finite horizon LQ solution!

Infinite horizon LQ (LQR – The Linear Quadratic Regulator)

$$\min_{z \in \mathbb{R}^\infty} f(z) = \sum_{t=0}^{\infty} \frac{1}{2} x_{t+1}^\top Q x_{t+1} + \frac{1}{2} u_t^\top R u_t$$

subject to $x_{t+1} = Ax_t + Bu_t, \quad t = 0, 1, \dots$

x_0 given

$Q \succeq 0, \quad R \succ 0$

- This has a solution provided (A, B) is **stabilizable**
- Then the optimal solution is the LTI state feedback

$$u_t = -Kx_t$$

where the feedback gain matrix is derived by

$$K = R^{-1}B^\top P(I + BR^{-1}B^\top P)^{-1}A,$$

$$P = Q + A^\top P(I + BR^{-1}B^\top P)^{-1}A; \quad P = P^\top \succ 0$$

- This solution is guaranteed to be closed-loop stable (eigenvalues of $A-BK$ stable) if (A, D) is **detectable**, where $Q = D^\top D$
- Being a state feedback solution, it implies some robustness (more on this later)

Controllability vs stabilizability

Observability vs detectability

- Stabilizable: All unstable modes are controllable
(that is: all uncontrollable modes are stable)
- Detectability: All unstable modes are observable
(that is: all unobservable modes are stable)
- Controllability implies stabilizability
- Observability implies detectability

LQR vs MPC

- LQR is MPC without constraints -> solution is “linear state feedback”
 - MPC solution is “online optimization” (QP)
- Often: Constraints can be active when far from setpoint, but become irrelevant close to setpoint
 - In other words: MPC “reduces” to LQR when close to setpoint
- Consider double integrator example:

The double integrator, two integrators in series, discretized with sample interval T_s , can be written in state-space form as

$$A = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix}.$$

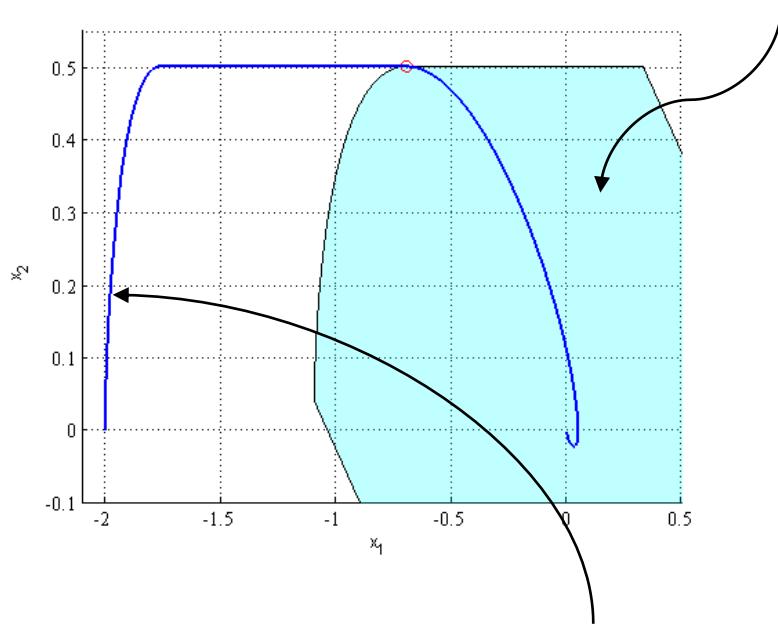
Consider an MPC cost function with

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1,$$

The constraints are $-0.5 \leq x_2 \leq 0.5$, and $-1 \leq u \leq 1$.

LQR vs MPC, II

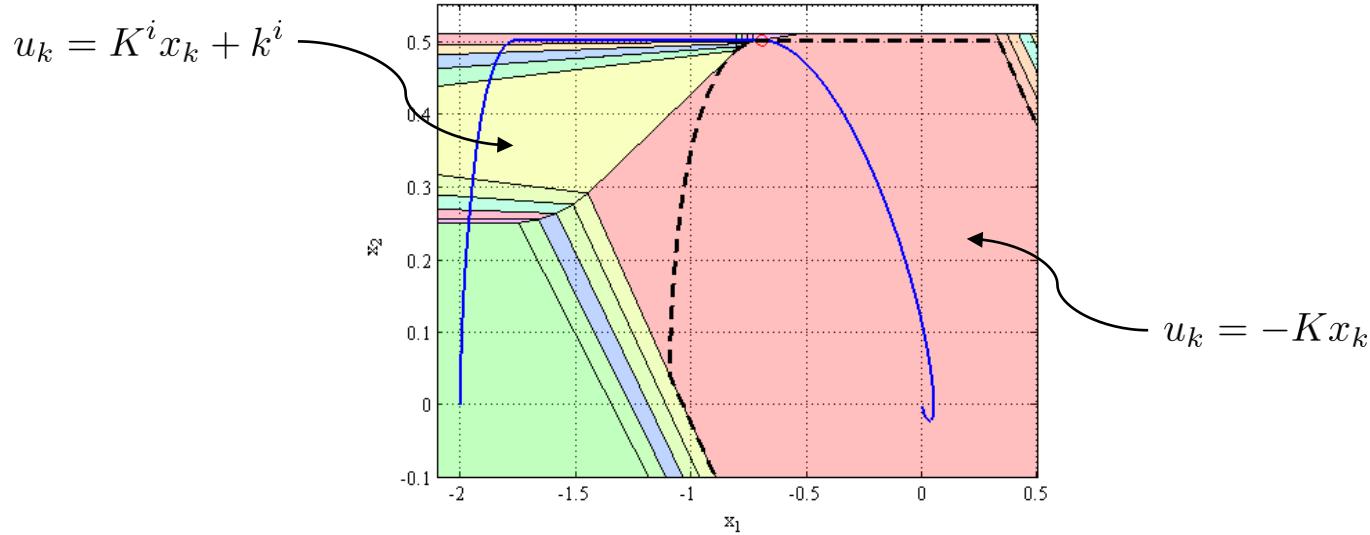
Region (polytopic set) where LQR solution is optimal
(where we can assume problem unconstrained)



MPC solution has larger feasible region than LQR solution!

LQR vs MPC, III

- In fact, the MPC solution is piecewise linear, defined on polytopic regions



- Proof/computation of this is an exercise in studying KKT conditions
 - (Not very difficult, but was not realized before ca. 2000)
 - But: solution quickly becomes very complex (many regions), except for very small systems.

LQ regulator (LQR)

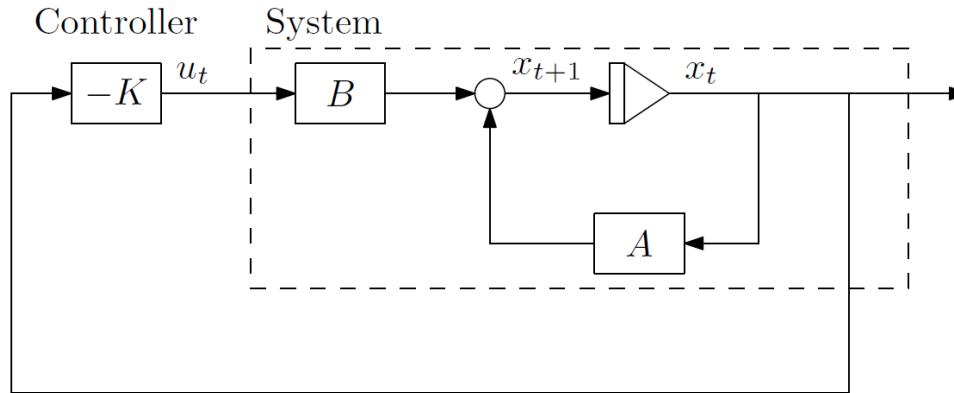


Figure 4.4: Solution of the LQ control problem, i.e., with state feedback.

LQ and robustness

- SISO LQ regulators have 60 degrees phase margin and 6dB gain margin
- Can be extended to MIMO systems

IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. AC-22, NO. 2, APRIL 1977

173

Gain and Phase Margin for Multiloop ~~LQG~~ Regulators LQ

MICHAEL G. SAFONOV, STUDENT MEMBER, IEEE, AND MICHAEL ATHANS, FELLOW, IEEE

Abstract—Multiloop linear-quadratic state-feedback (LQSF) regulators are shown to be robust against a variety of large dynamical linear time-invariant and memoryless nonlinear time-varying variations in open-loop dynamics. The results are interpreted in terms of the classical concepts of gain and phase margin, thus strengthening the link between classical and modern feedback theory.

measured in terms of multiloop generalizations of the classical notions of *gain and phase margin*. Like classical gain and phase margin, the present results consider robustness as an input-output property characterizing variations in open-loop transfer functions which will not

- However, usually one does not measure all the states...

Output feedback MPC

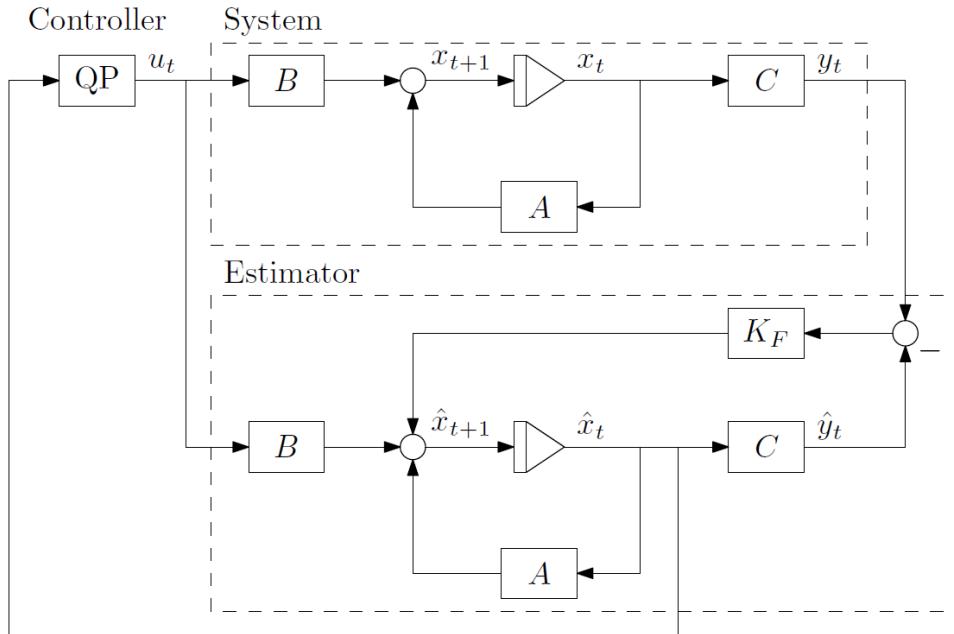


Figure 4.3: The structure of an output feedback linear MPC.

LQG: Linear Quadratic Gaussian (= LQR + KF)

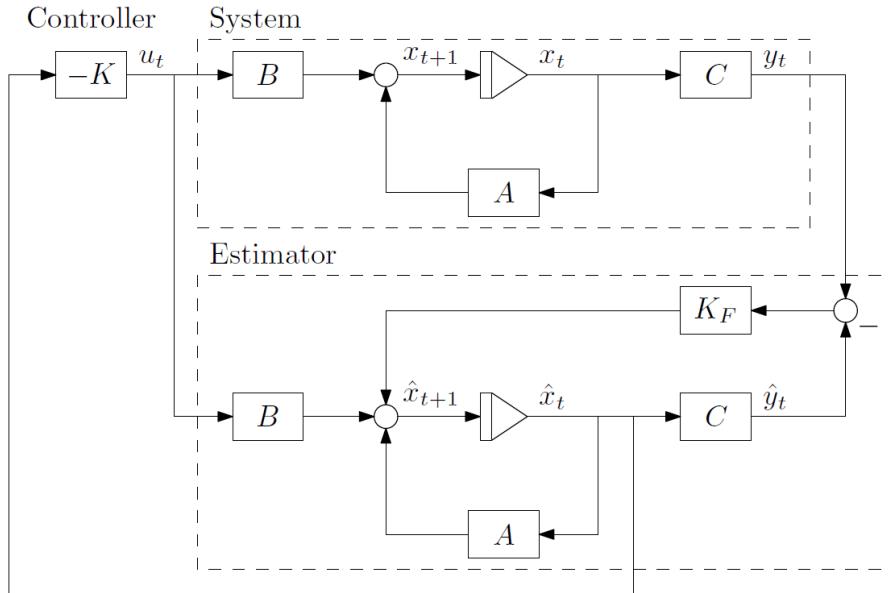


Figure 4.7: Structure of the LQG controller, i.e., output feedback LQ control.

Separation principle for linear output feedback control

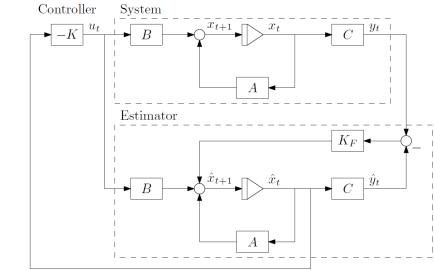


Figure 4.7: Structure of the LQG controller, i.e., output feedback LQ control.

Separation principle for linear output feedback control

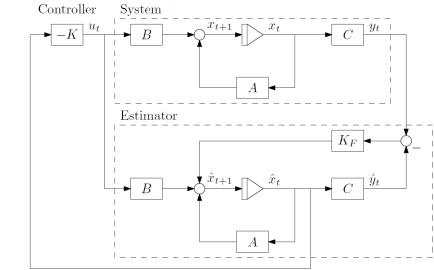


Figure 4.7: Structure of the LQG controller, i.e., output feedback LQ control.

LQG and robustness

- Doyle, 1978:

Guaranteed Margins for LQG Regulators

JOHN C. DOYLE

Abstract—There are none.

INTRODUCTION

Considerable attention has been given lately to the issue of robustness of linear-quadratic (LQ) regulators. The recent work by Safonov and Athans [1] has extended to the multivariable case the now well-known guarantee of 60° phase and 6 dB gain margin for such controllers. However, for even the single-input, single-output case there has remained the question of whether there exist any guaranteed margins for the full LQG (Kalman filter in the loop) regulator. By counterexample, this note answers that question; there are none.

A standard two-state single-input single-output LQG control problem is posed for which the resulting closed-loop regulator has arbitrarily small gain margin.

- Lead to a lot of research in robust control in the 80's (and later), not topic of this course

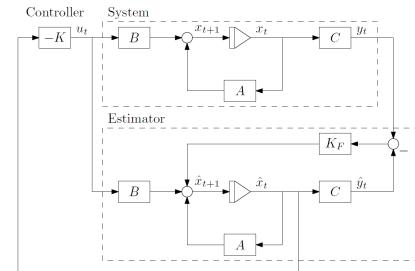
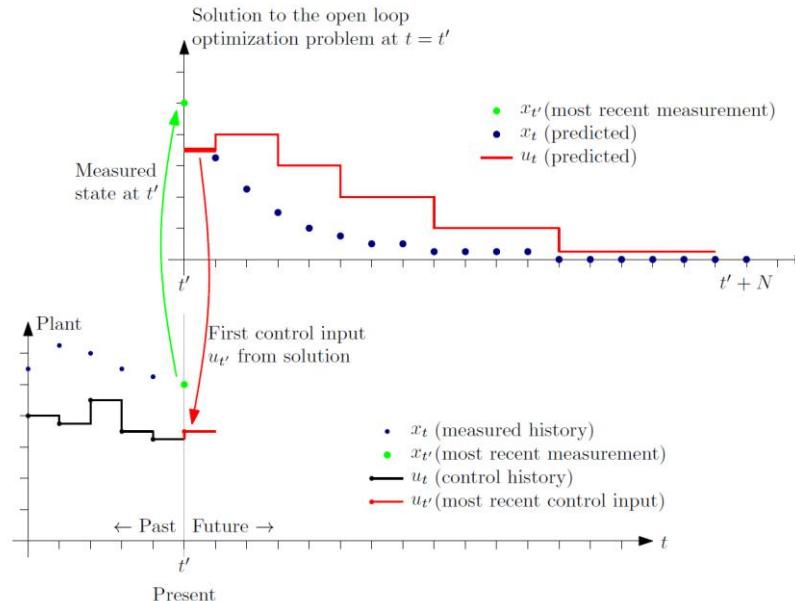


Figure 4.7: Structure of the LQG controller, i.e., output feedback LQ control.

Complexity reduction strategies in MPC

- Input blocking (or move blocking) – reduce number of QP variables



- “Incident points” – reduce number of QP constraints
 - Only check constraints at certain time instants, rather than at all times on horizon



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 13

Unconstrained optimization

Lecturer: Lars Imsland

Outline

- Optimality conditions for unconstrained optimization
- Ingredients in gradient algorithms for unconstrained optimization
 - Descent directions (steepest descent, Newton, Quasi-Newton)
 - How far to walk in descent direction (line search, trust region)
 - Termination criteria
- Scaling

Reference: N&W Ch.2.1-2.2

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

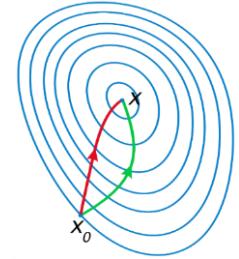
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



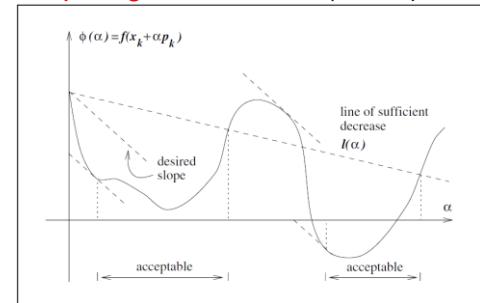
How to calculate derivatives – Ch. 8

$$\min_x f(x)$$



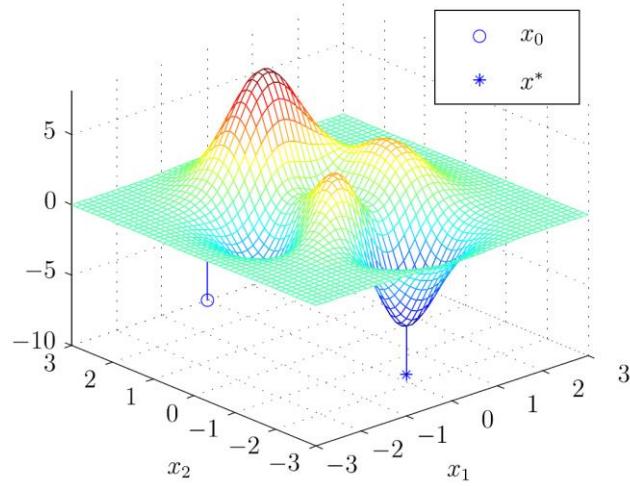
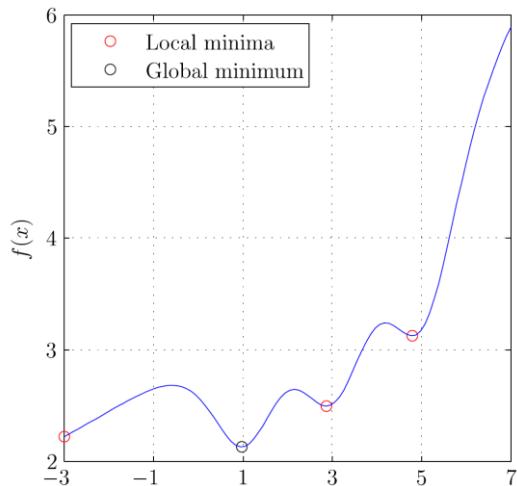
A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. ([wikipedia.org](https://en.wikipedia.org))

Step length line search (Wolfe):

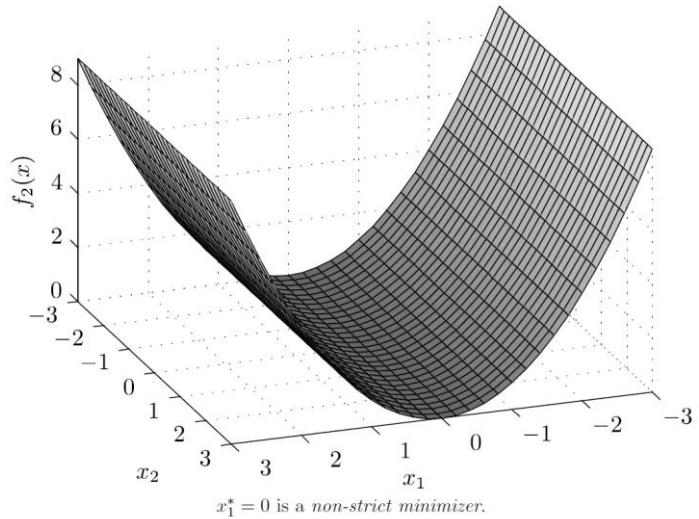
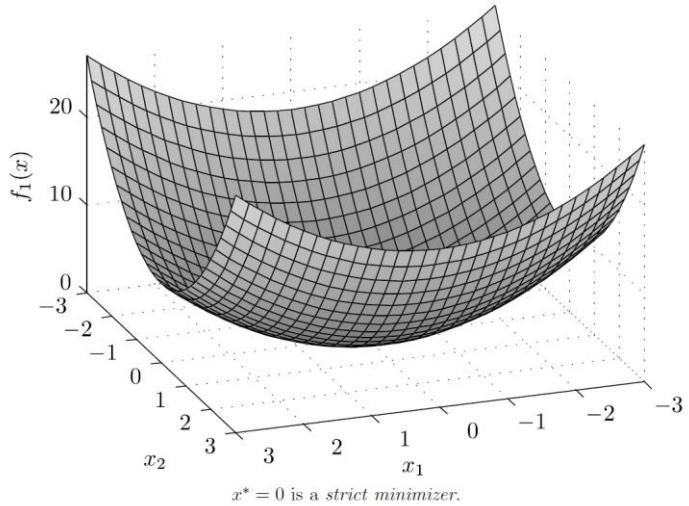


Unconstrained optimization

What is a solution? Local and global minimizers



(Strict and non-strict optimizers)



Necessary condition for optimality

$$\min_x f(x)$$

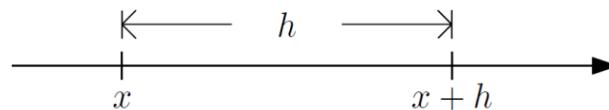
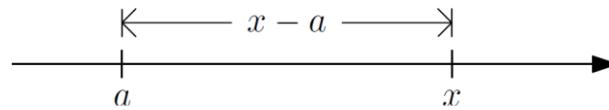
Taylor expansions

- From Calculus?

$$f(x) = f(a) + (x - a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \dots$$

- In this course:

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots$$



Taylor's theorem

$$f : \mathbb{R}^n \rightarrow \mathbb{R}, p \in \mathbb{R}^n$$

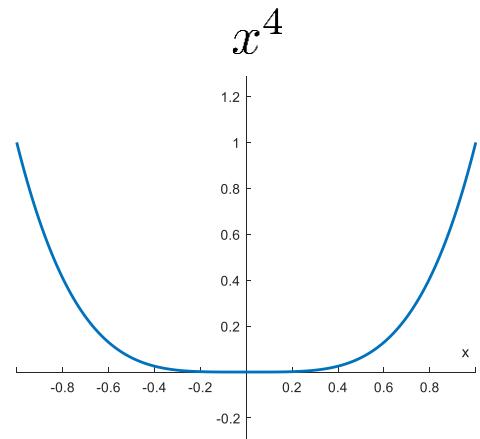
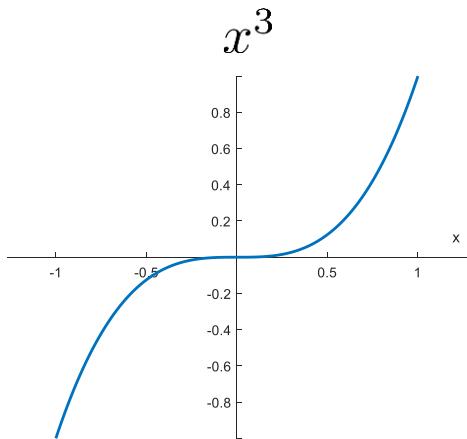
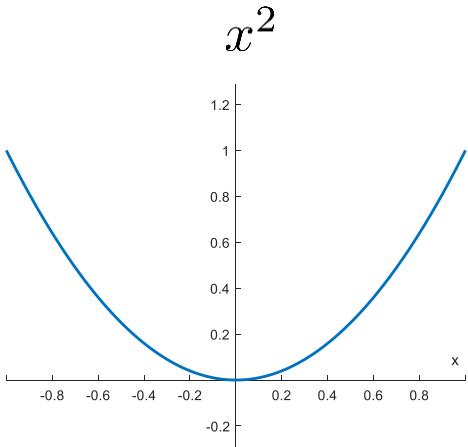
- First order: If f is continuously differentiable,

$$f(x + p) = f(x) + \nabla f(x + tp)^\top p, \quad \text{for some } t \in (0, 1)$$

- Second order: If f is twice continuously differentiable

$$f(x + p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x + tp)^\top p, \quad \text{for some } t \in (0, 1)$$

Sufficient conditions for optimality



$$\nabla f(0) = 0$$

$$\nabla^2 f(0) > 0$$

$$\nabla f(0) = 0$$

$$\nabla^2 f(0) = 0$$

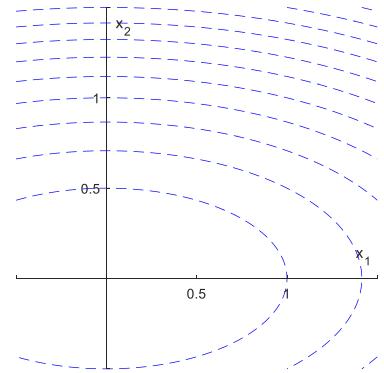
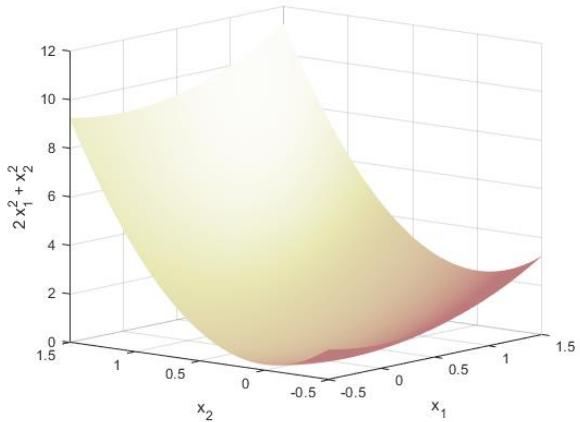
$$\nabla f(0) = 0$$

$$\nabla^2 f(0) = 0$$

General algorithm for solving $\min_x f(x)$

Termination criteria

Descent (downhill) directions



Quadratic approximation to objective function

$$f(x_k + p) \approx m_k(p) = f(x_k) + p^\top \nabla f(x_k) + \frac{1}{2} p^\top \nabla^2 f(x_k) p$$

Minimize approximation:

$$\nabla_p m_k(p) = 0 \Rightarrow p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

“Newton step”:

$$x_{k+1} = x_k + p_k = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

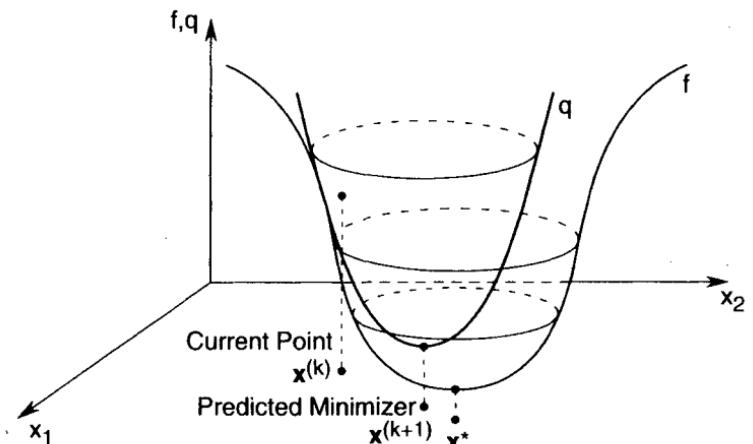
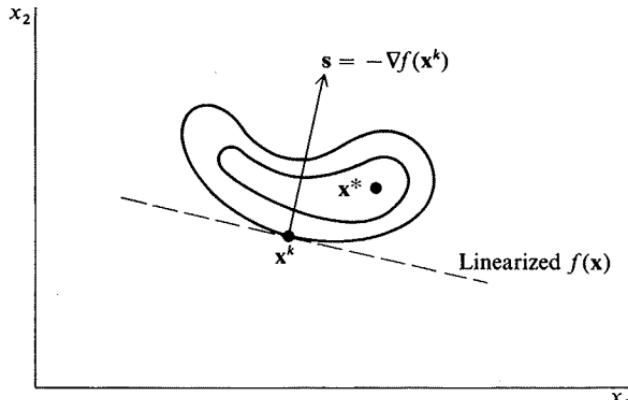
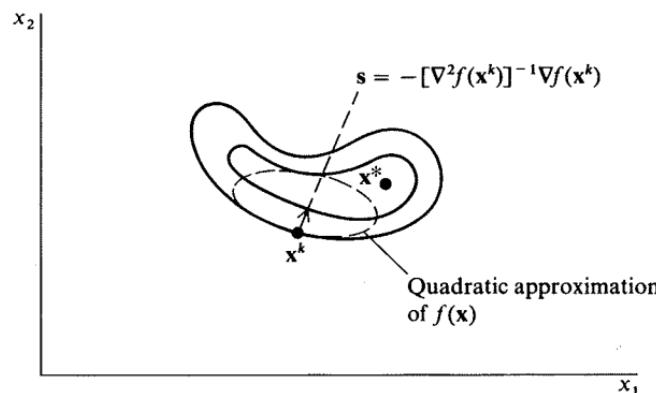


Figure 9.1 Quadratic approximation to the objective function using first and second derivatives.
Chong & Zak, “An introduction to optimization”

Steepest descent directions vs Newton directions from objective function approximations



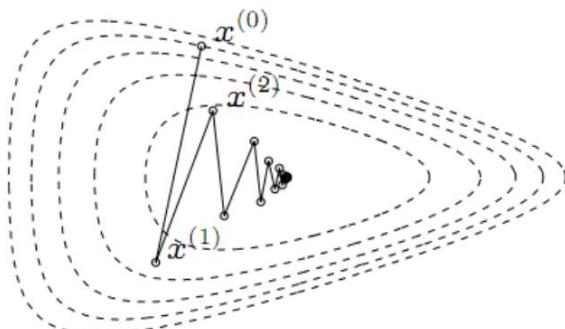
(a) Steepest descent: first-order approximation
(linearization) of $f(\mathbf{x})$ at \mathbf{x}^k



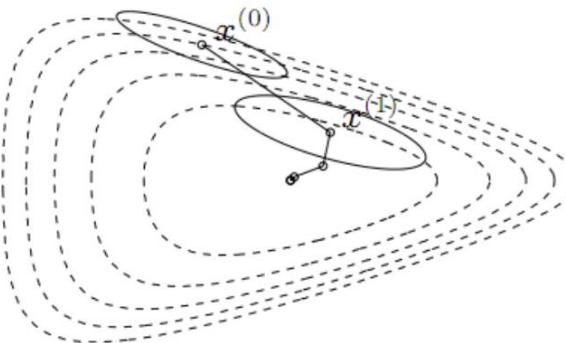
(b) Newton's method: second-order (quadratic)
approximation of $f(\mathbf{x})$ at \mathbf{x}^k

From Edgar, Himmelblau, Lasdon: "Optimization of Chemical Processes"

Steepest descent vs Newton



gradient descent with
backtracking line search

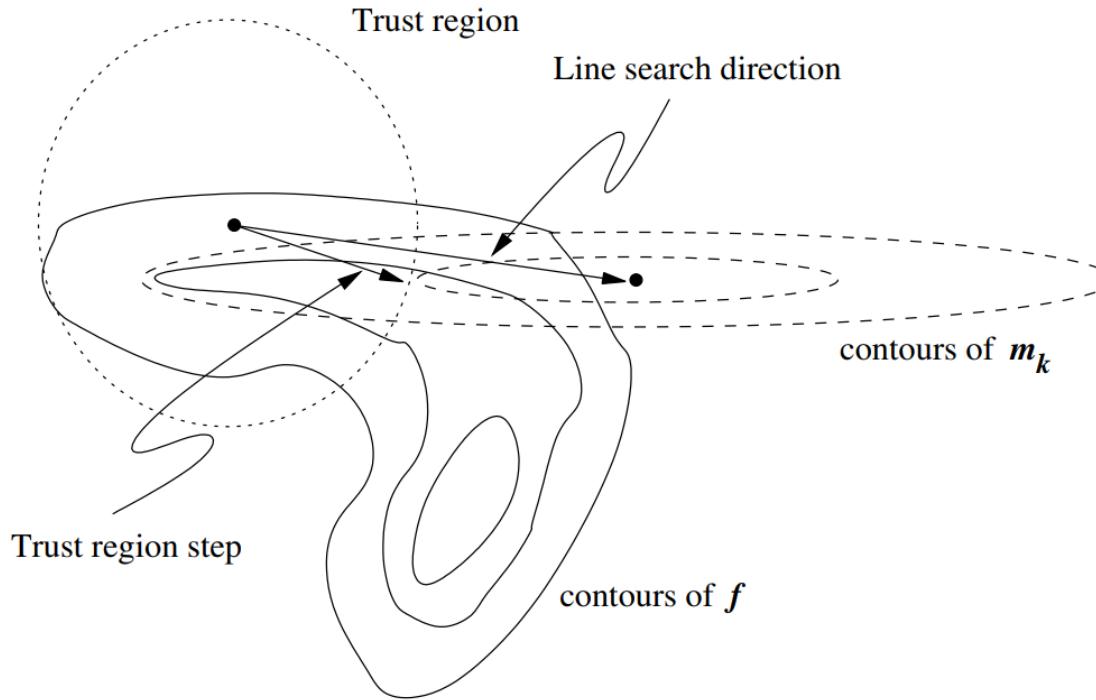


Newton's method with
backtracking line search

Boyd & Vanderberghe, P. Abbeel

How far should we walk along p_k ?

Line search and trust region steps



Scaling, scale invariance

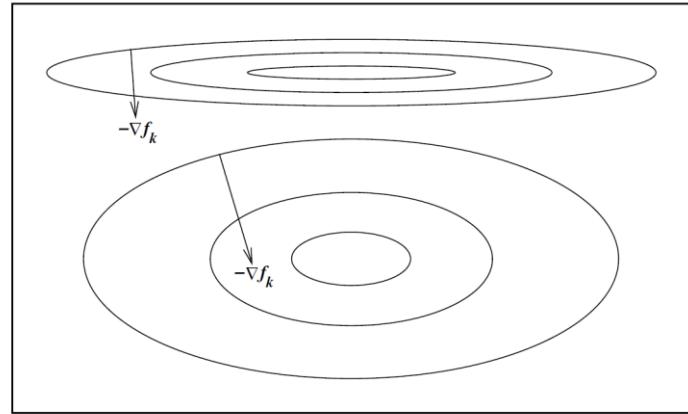


Figure 2.7 Poorly scaled and well scaled problems, and performance of the steepest descent direction.



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 14

Line search

Lecturer: Lars Imsland

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

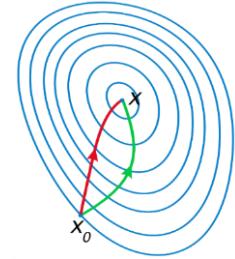
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



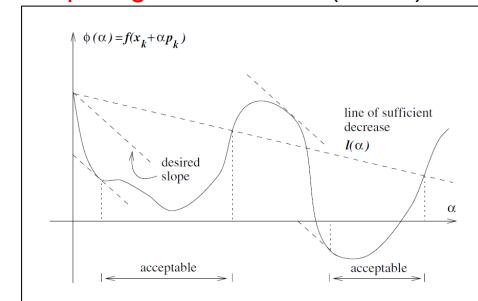
How to calculate derivatives – Ch. 8

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. (wikipedia.org)

Step length line search (Wolfe):



Outline today

Objective of line search: **make gradient algorithms work when you start far away from optimum**

- These algorithms are sometimes called globalization strategies
- Two basic globalization strategies: line search (Ch. 3) and trust-region (Ch. 4, not syllabus)
 - Note again: “globalization” does not imply that we search for global optimum, but we make the algorithm work far from a (local or global) optimum!

Line search elements:

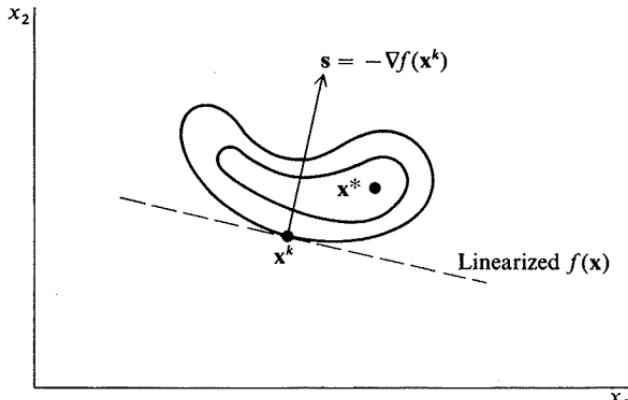
- Conditions on step-length: Wolfe conditions
- Step-length computation

Hessian modification for Newton

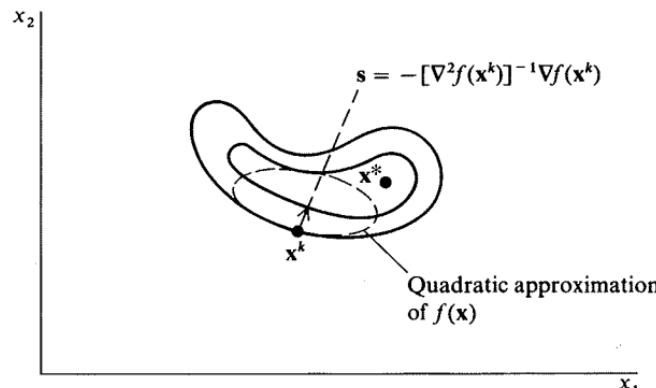
Reference: N&W Ch.3-3.1, 3.4, 3.5

Steepest descent direction vs Newton direction from objective function approximation

From Edgar, Himmelblau, Lasdon: "Optimization of Chemical Processes"

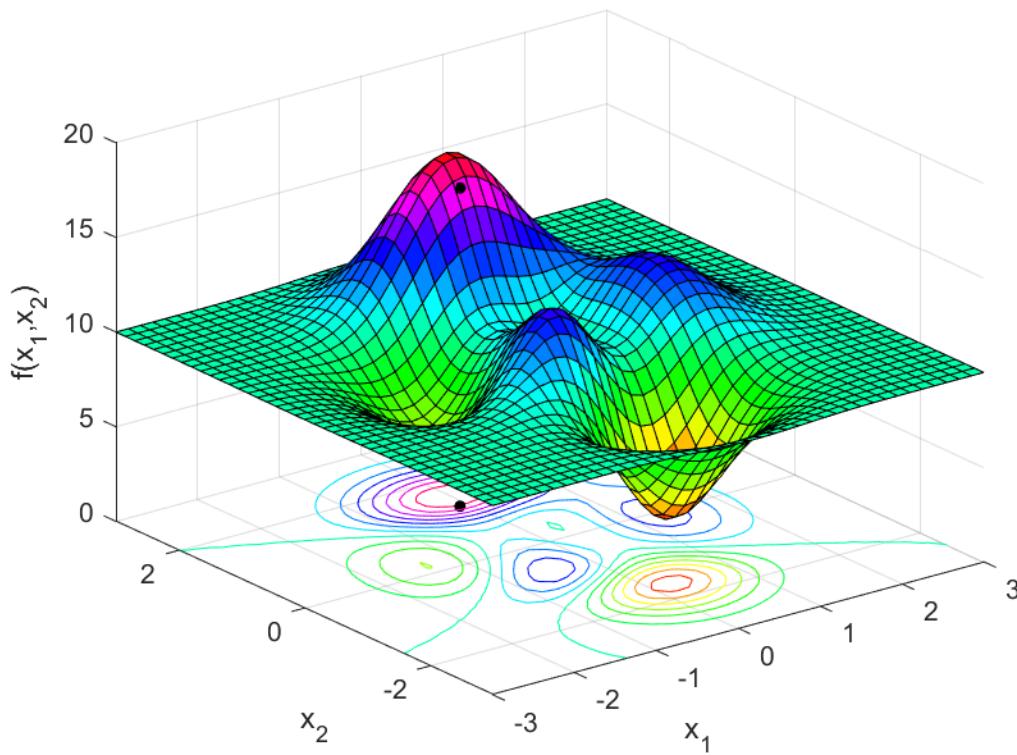


(a) Steepest descent: first-order approximation
(linearization) of $f(\mathbf{x})$ at \mathbf{x}^k

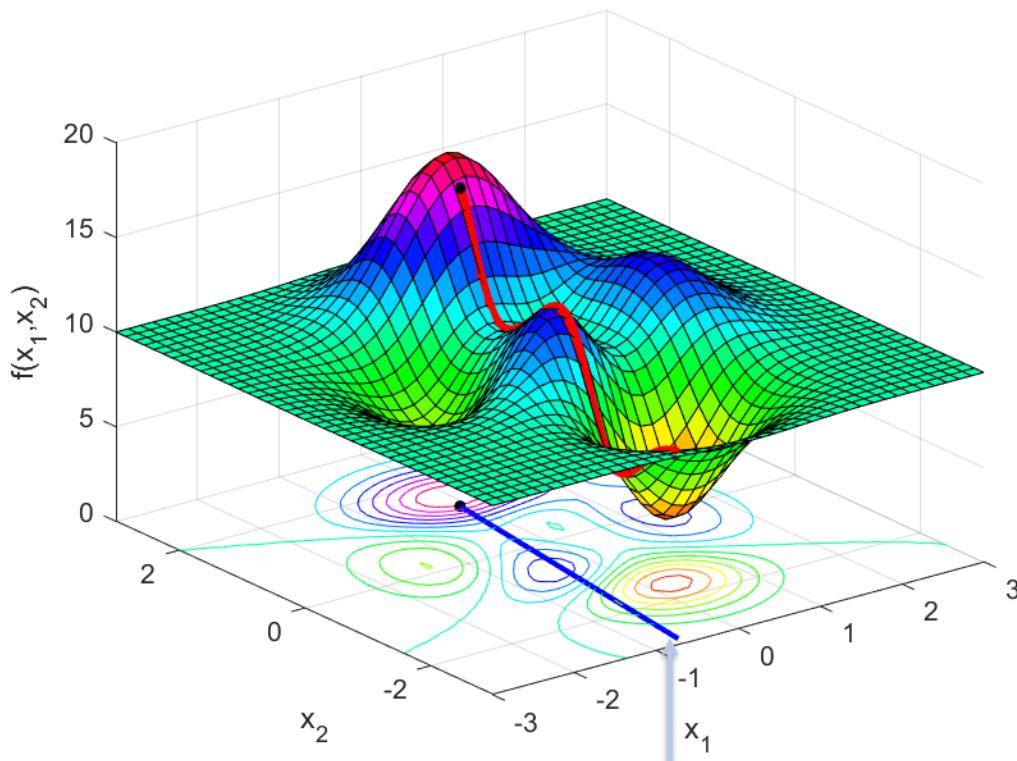


(b) Newton's method: second-order (quadratic)
approximation of $f(\mathbf{x})$ at \mathbf{x}^k

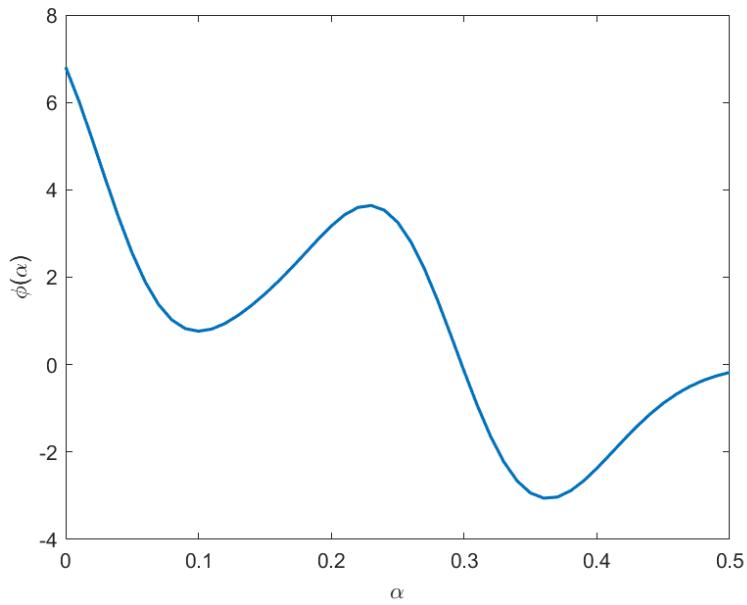
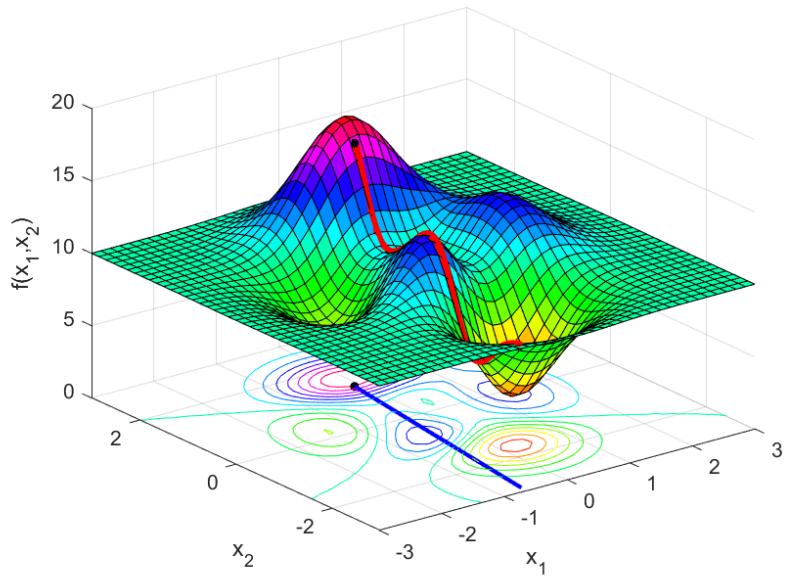
$$x_0 = (-0.1, 1.3)^\top$$



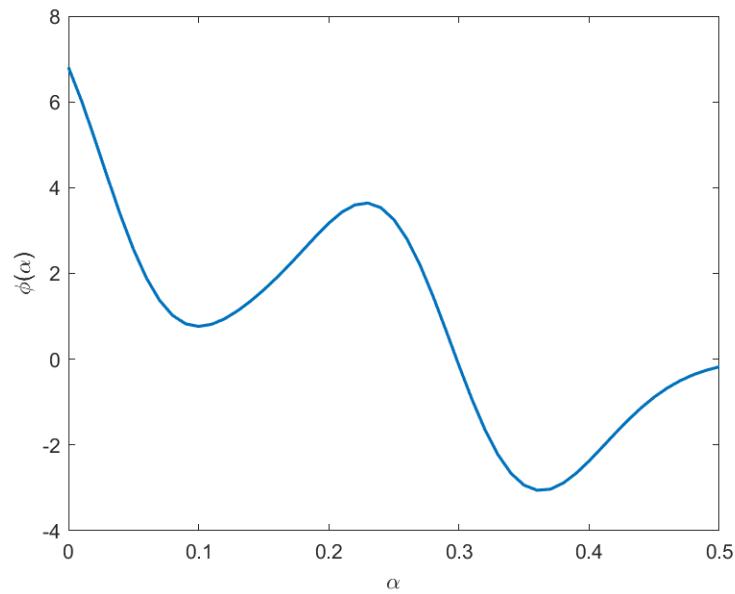
$$x_0 = (-0.1, 1.3)^\top$$



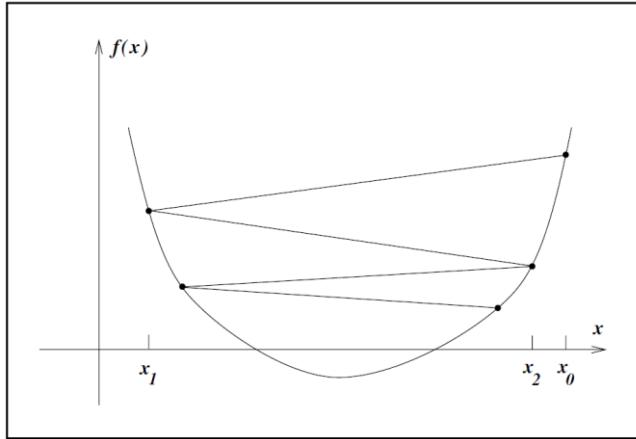
Steepest descent direction from x_0



Exact linesearch



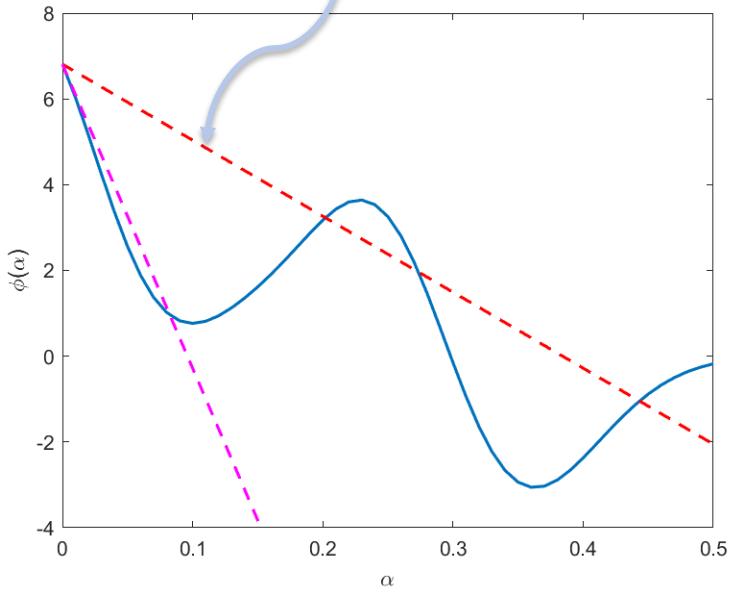
Why sufficient decrease?



- Decrease not enough, need sufficient decrease (1st Wolfe condition)

Condition 1: Sufficient decrease (*Armijo*)

$$l(\alpha) = \phi(0) + c_1 \alpha \phi'(0), \quad c_1 = 0.25$$



Sufficient decrease

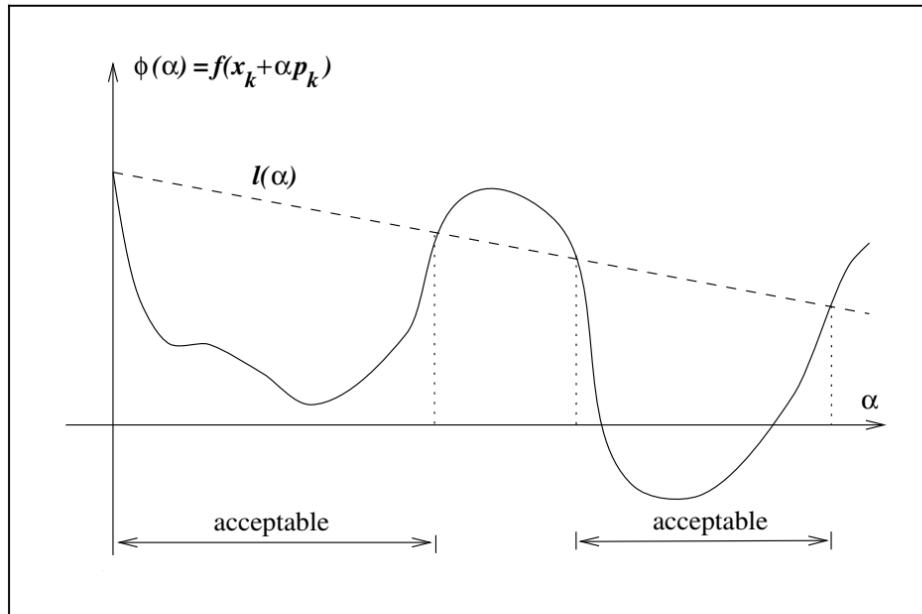
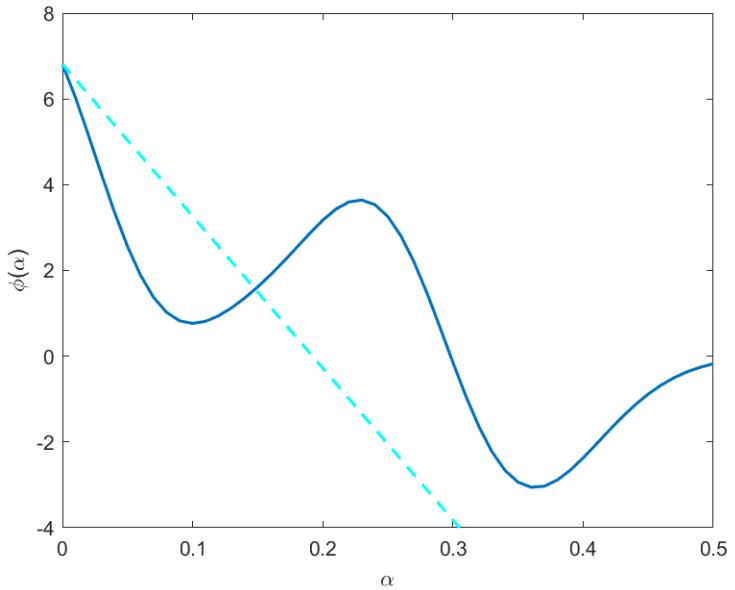


Figure 3.3 Sufficient decrease condition.

Condition 2: Curvature condition

$$l(\alpha) = \phi(0) + c_2 \alpha \phi'(0), \quad c_2 = 0.5$$



Curvature condition

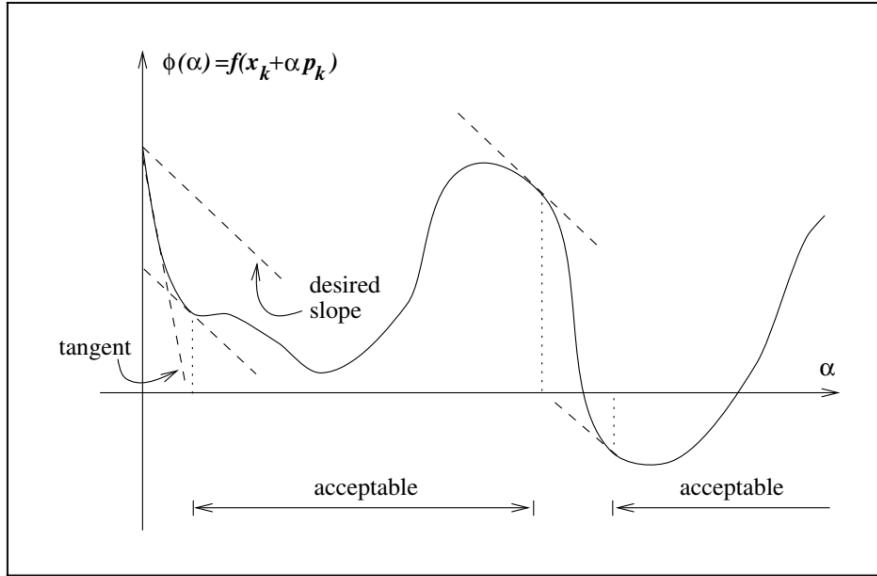


Figure 3.4 The curvature condition.

Wolfe conditions

- Good step lengths should fulfill the **Wolfe conditions**:

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^\top p_k \quad \text{Sufficient decrease (Armijo condition)}$$

$$\nabla f(x_k + \alpha_k p_k)^\top p_k \geq c_2 \nabla f_k^\top p_k \quad \text{Desired slope (Curvature condition)}$$

- How do we compute such a step length?

Backtracking Line Search

Algorithm 3.1 (Backtracking Line Search).

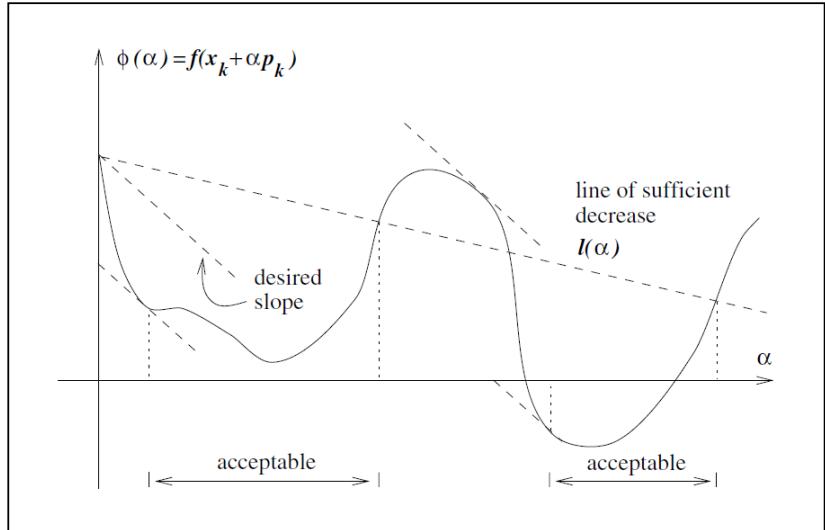
Choose $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$; Set $\alpha \leftarrow \bar{\alpha}$;

repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$

$\alpha \leftarrow \rho\alpha$;

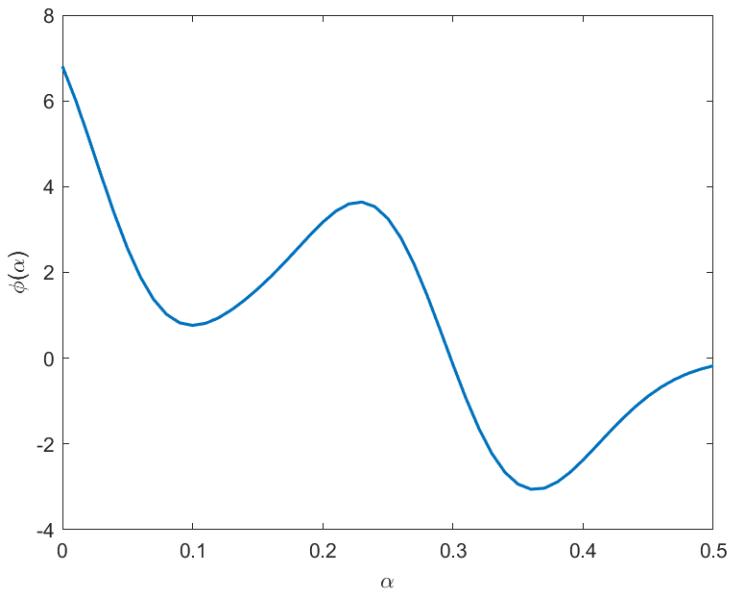
end (repeat)

Terminate with $\alpha_k = \alpha$.



Curvature condition (desired slope)
not needed since we start with long step length

Interpolation



Example: Line search for convex quadratic objective function

Newton: Hessian modification

Line search Newton

Algorithm 3.2 (Line Search Newton with Modification).

Given initial point x_0 ;

for $k = 0, 1, 2, \dots$

Factorize the matrix $B_k = \nabla^2 f(x_k) + E_k$, where $E_k = 0$ if $\nabla^2 f(x_k)$ is sufficiently positive definite; otherwise, E_k is chosen to ensure that B_k is sufficiently positive definite;

Solve $B_k p_k = -\nabla f(x_k)$;

Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions;

end

Local convergence rates (close to optimum)

Steepest descent:
Linear convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r \quad \text{for all } k \text{ sufficiently large, } r \in (0, 1)$$

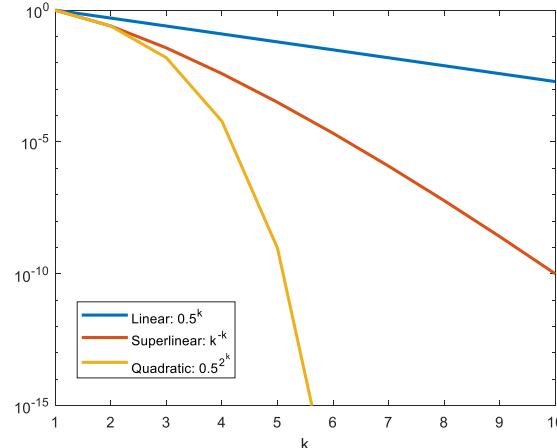
Newton:
Quadratic convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M \quad \text{for all } k \text{ sufficiently large, } M > 0$$

Quasi-Newton:
Superlinear convergence

$$\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

$$\frac{\|x_{k+1} - x^*\|}{\|x_0\|}$$





NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 15

Quasi-Newton

Lecturer: Lars Imsland

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

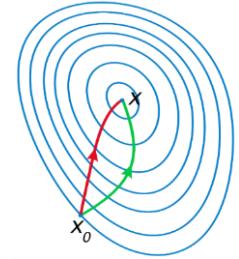
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



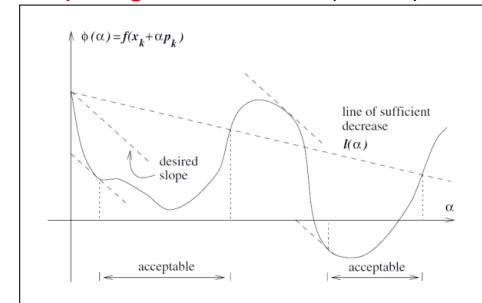
How to calculate derivatives – Ch. 8

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. (wikipedia.org)

Step length line search (Wolfe):



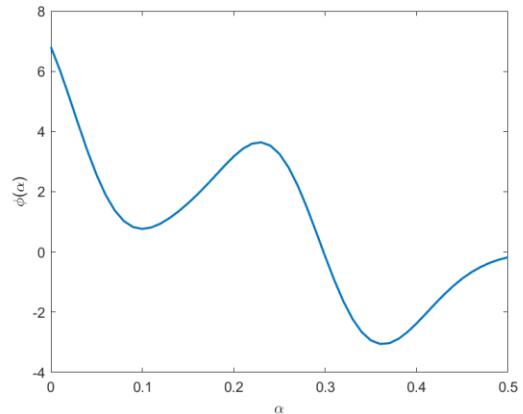
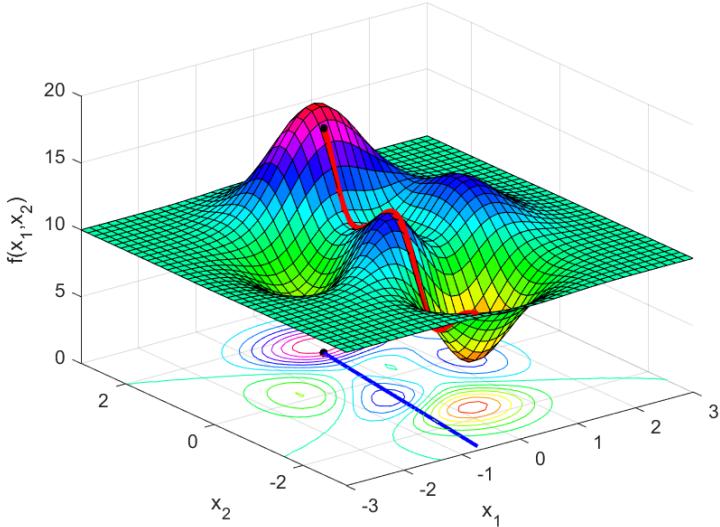
Outline today

Quasi-Newton

- Q-N efficiently produce good search directions
 - Steepest descent: Many iterations, but each iteration cheap (**need only gradient**)
 - Newton: Few iterations, but each iteration expensive (**need also Hessian**)
 - Quasi-Newton: Few iterations by **approximating the Hessian using only the gradient**
- Secant condition
- BFGS (and DFP) Hessian approximation update formulas
- What is this used for? A lot!
General optimization, nonlinear MPC, machine learning,
image processing, ...

Reference: N&W Ch.6-6.1 (Superficially 7.1)

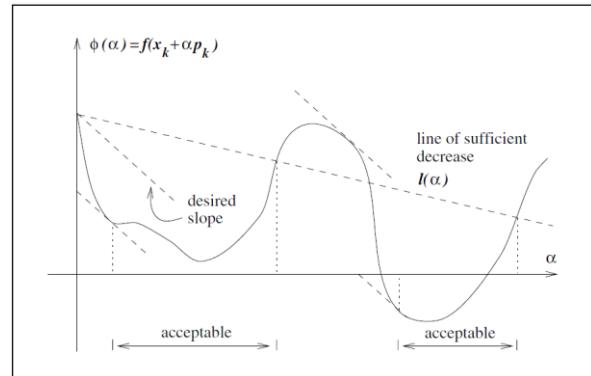
Line search



Conditions for a good step length: Wolfe conditions

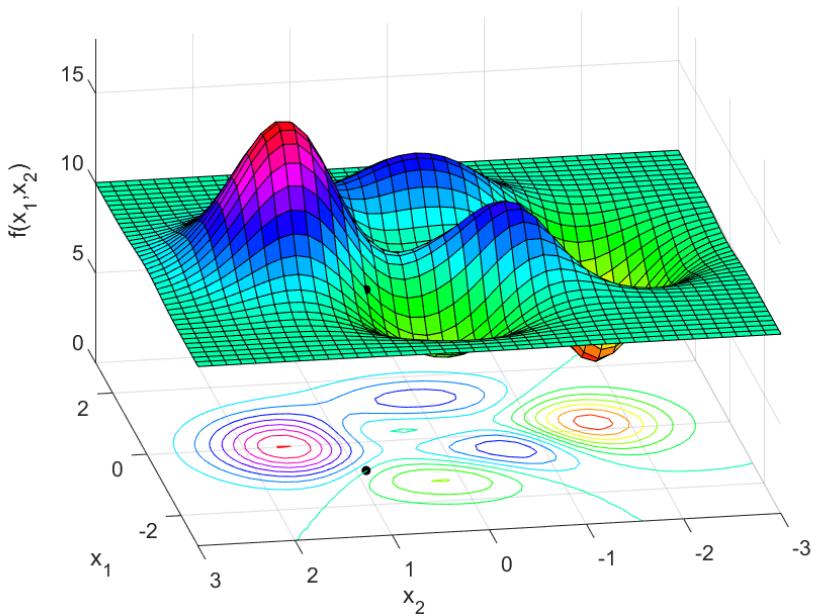
$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^\top p_k \quad \text{Sufficient decrease (Armijo condition)}$$

$$\nabla f(x_k + \alpha_k p_k)^\top p_k \geq c_2 \nabla f_k^\top p_k \quad \text{Desired slope (Curvature condition)}$$



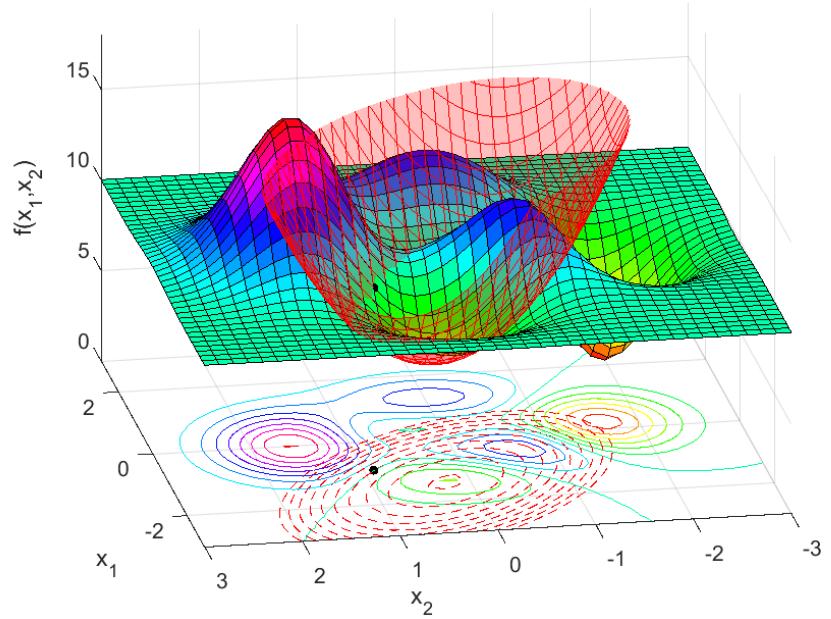
Newton's method

$$x_0 = (-0.9, 0.9)^\top$$



Newton's method

$$f(x_k + p) \approx m_k(p) = f_k + \nabla f_k^\top p + \frac{1}{2} p^\top \nabla^2 f_k p$$



Hessian modification for Newton's method

- For $p_k = -B_k^{-1}\nabla f(x_k)$ to be a descent direction, we need $B_k > 0$
- In general, this does not hold true for Newton, $B_k = \nabla^2 f(x_k)$. We therefore modify the Hessian when it is not positive definite:

Algorithm 3.2 (Line Search Newton with Modification).

Given initial point x_0 ;

for $k = 0, 1, 2, \dots$

Factorize the matrix $B_k = \nabla^2 f(x_k) + E_k$, where $E_k = 0$ if $\nabla^2 f(x_k)$ is sufficiently positive definite; otherwise, E_k is chosen to ensure that B_k is sufficiently positive definite;

Solve $B_k p_k = -\nabla f(x_k)$;

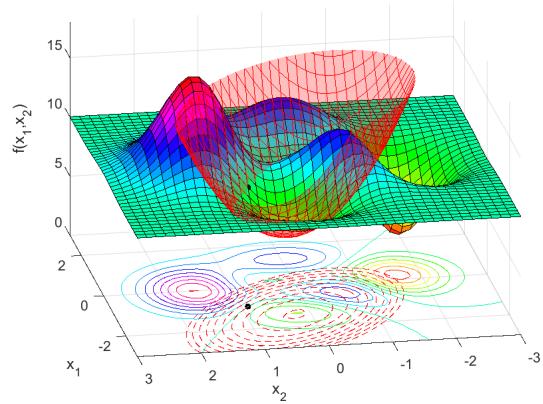
Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions;

end

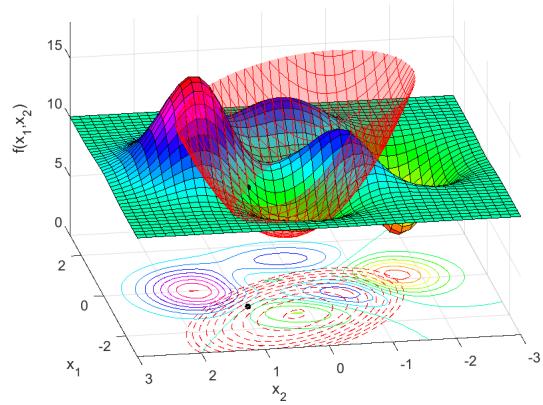
- A good, but inefficient method: $E_k = \tau_k I$, $\tau_k = \max(0, \delta - \lambda_{\min}(\nabla^2 f(x_k)))$
- More efficient to do “similar” changes during factorization process
 - E.g. “modified Cholesky” method

Newton's method

$$x_{k+1} = x_k + \alpha_k p_k; \quad p_k = -[\nabla^2 f_k]^{-1} \nabla f_k$$

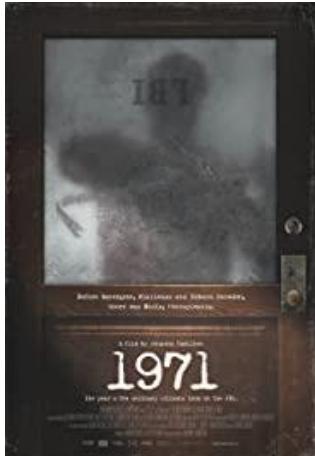
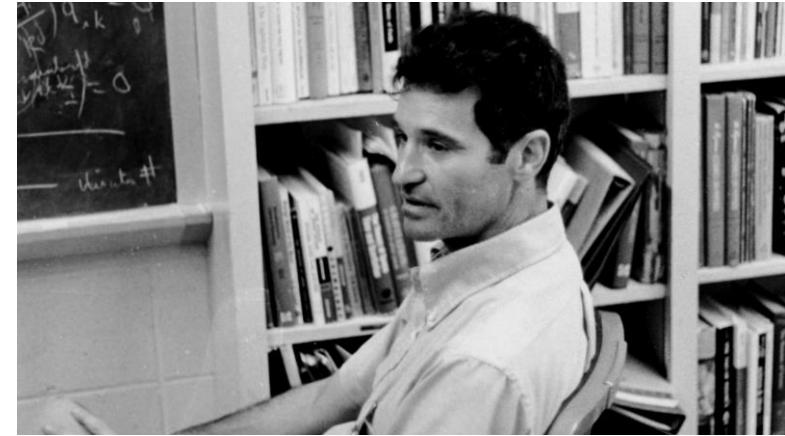


Quasi-Newton



Quasi-Newton

- Invented by Bill Davidon, physicist at Argonne National Labs, around mid 1950s
- “The Davidon-Fletcher-Powell (DFP) update formula”
- “One of the most creative ideas in nonlinear optimization”, tremendous impact



Fun fact 1: His first paper on Q-N was not accepted for publication before 1991, over thirty years later

Fun fact 2: Bill Davidon was a peace activist, mastermind behind a break-in at an FBI office in 1971 (Movie: “1971”)

Secant condition

Positive definite requirement

DFP update formula

Inverse update formula

DFP formula for updating B_k :

$$B_{k+1} = (I - \rho_k y_k s_k^\top) B_k (I - \rho_k s_k y_k^\top) + \rho_k y_k y_k^\top, \quad \rho_k = \frac{1}{y_k^\top s_k}$$

But: since we need B_k^{-1} in $p_k = -B_k^{-1}\nabla f_k$, can we update $H_k = B_k^{-1}$ instead?

Yes: Multiply out DFP as

$$B_{k+1} = B_k + (B_k s_k - y_k) \begin{pmatrix} 0 & -\rho_k \\ -\rho_k & \rho_k + s_k^\top B_k s_k \rho_k^2 \end{pmatrix} \begin{pmatrix} s_k^\top B_k \\ y_k^\top \end{pmatrix}, \quad \rho_k = \frac{1}{y_k^\top s_k},$$

use the *matrix inversion lemma* (Sherman-Morrison-Woodbury formula)

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

to obtain the inverse DFP formula:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^\top H_k}{y_k^\top H_k y_k} + \frac{s_k s_k^\top}{y_k^\top s_k}$$

BFGS update formula

BFGS

Broyden, Fletcher, Goldfarb, Shanno



BFGS method

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,

inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k;$$

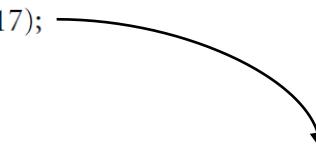
 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
 procedure to satisfy the Wolfe conditions (3.6);

 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)



$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Local convergence rates (close to optimum)

Steepest descent:
Linear convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r \quad \text{for all } k \text{ sufficiently large, } r \in (0, 1)$$

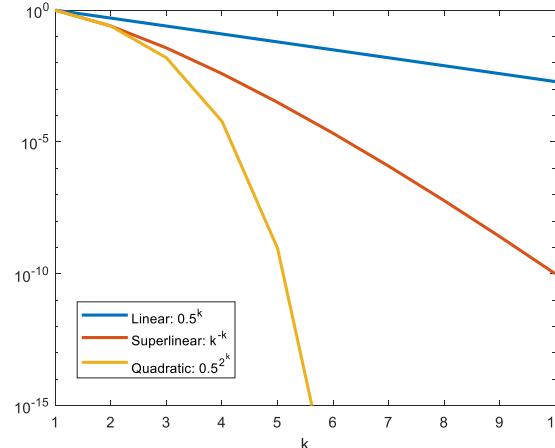
Newton:
Quadratic convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M \quad \text{for all } k \text{ sufficiently large, } M > 0$$

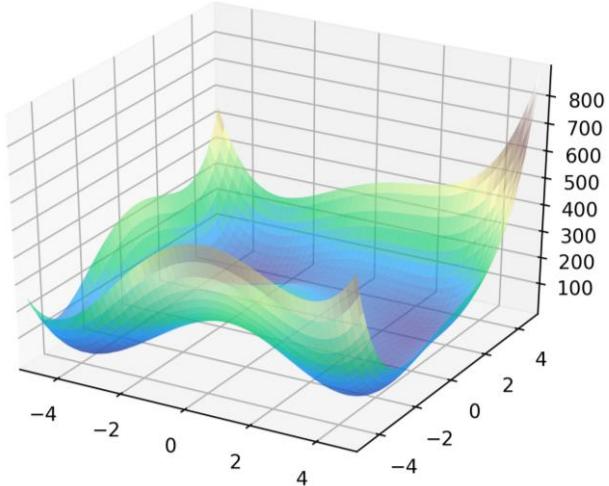
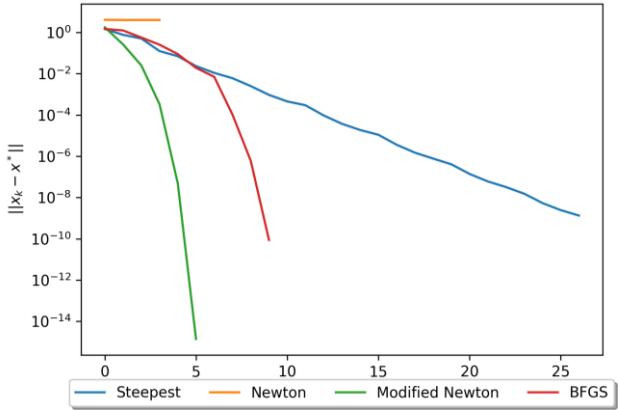
Quasi-Newton:
Superlinear convergence

$$\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

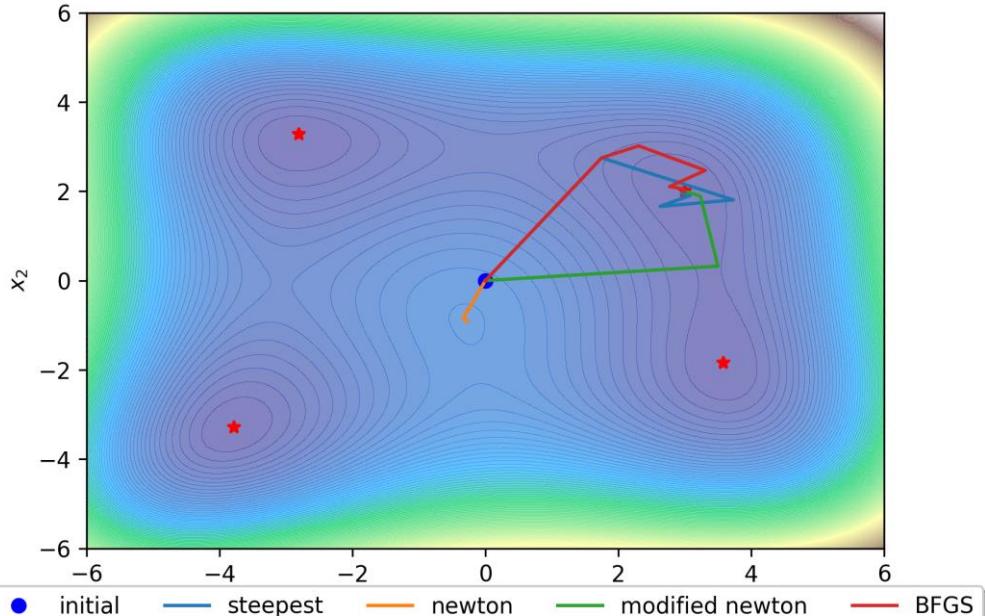
$$\frac{\|x_{k+1} - x^*\|}{\|x_0\|}$$



Example: Himmelblau



$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$



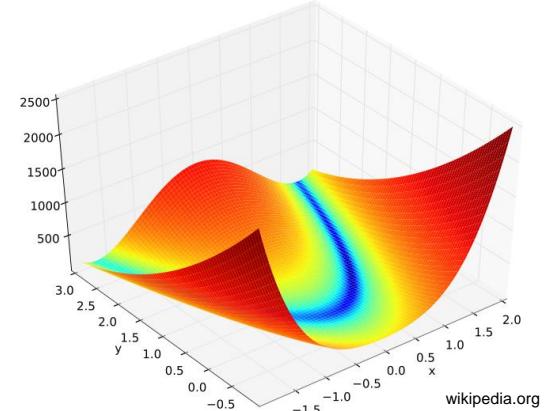
● initial — steepest — newton — modified newton — BFGS

Example (from book)

- Using steepest descent, BFGS and inexact Newton on Rosenbrock function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- Iterations from starting point (-1.2,1):
 - Steepest descent: 5264
 - BFGS: 34
 - Newton: 21
- Last iterations; value of $\|x_k - x^*\|$



wikipedia.org

steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

Example: image deblurring



Original image

Blurred image

Reconstruction

Figures from (Wang et. al, 2009)

Given corrupted $m \times n$ image represented as vector $y \in \mathbb{R}^{m \cdot n}$, find $x \in \mathbb{R}^{m \cdot n}$ by solving the optimization problem

$$\underset{x}{\text{minimize}} \quad \|K*x - y\|_2^2 + \lambda \left(\sum_{i=1}^{n-1} |x_{mi} - x_{m(i+1)}| + \sum_{i=1}^{m-1} |x_{ni} - x_{n(i+1)}| \right)$$

where $K*$ denotes 2D convolution with some filter K

Example: machine learning

Virtually all machine learning algorithms can be expressed as minimizing a *loss function* over observed data

Given inputs $x^{(i)} \in \mathcal{X}$, desired outputs $y^{(i)} \in \mathcal{Y}$, hypothesis function $h_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ defined by parameters $\theta \in \mathbb{R}^n$, and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$

Machine learning algorithms solve optimization problem

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^m \ell \left(h_\theta(x^{(i)}), y^{(i)} \right)$$

Quasi-Newton in machine learning

On optimization methods for deep learning

Quoc V. Le
Jiquan Ngiam
Adam Coates
Abhik Lahiri
Bobby Prochnow
Andrew Y. Ng

QUOCLE@CS.STANFORD.EDU
JNGIAM@CS.STANFORD.EDU
ACOATES@CS.STANFORD.EDU
ALAHIRI@CS.STANFORD.EDU
PROCHNOW@CS.STANFORD.EDU
ANG@CS.STANFORD.EDU

Computer Science Department, Stanford University, Stanford, CA 94305, USA

Abstract

The predominant methodology in training deep learning advocates the use of stochastic gradient descent methods (SGDs). Despite its ease of implementation, SGDs are difficult to tune and parallelize. These problems make it challenging to develop, debug and scale up deep learning algorithms with SGDs. In this paper, we show that more sophisticated off-the-shelf optimization methods such as Limited memory BFGS (L-BFGS) and Conjugate gradient (CG) with line search can significantly simplify and speed up the process of pretraining deep algorithms. In our experiments, the difference between L-BFGS/CG and SGDs are more pronounced if we consider algorithmic extensions (e.g., sparsity regularization) and hardware extensions (e.g., GPUs or computer clusters). Our experiments with distributed optimization support the use of L-BFGS with locally connected networks and convolutional neural networks. Using L-BFGS, our convolutional network model achieves 0.69% on the standard MNIST dataset. This is a state-of-the-art result on MNIST among algorithms that do not use distortions or pretraining.

2008; Zinkevich et al., 2010). A strength of SGDs is that they are simple to implement and also fast for problems that have many training examples.

However, SGD methods have many disadvantages. One key disadvantage of SGDs is that they require much manual tuning of optimization parameters such as learning rates and convergence criteria. If one does not know the task at hand well, it is very difficult to find a good learning rate or a good convergence criterion. A standard strategy in this case is to run the learning algorithm with many optimization parameters and pick the model that gives the best performance on a validation set. Since one needs to search over the large space of possible optimization parameters, this makes SGDs difficult to train in settings where running the optimization procedure many times is computationally expensive. The second weakness of SGDs is that they are inherently sequential: it is very difficult to parallelize them using GPUs or distribute them using computer clusters.

Batch methods, such as Limited memory BFGS (L-BFGS) or Conjugate Gradient (CG), with the presence of a line search procedure, are usually much more stable to train and easier to check for convergence. These methods also enjoy parallelism by computing the gradient on GPUs (Raina et al., 2009) and/or distributing that computation across machines (Chu et al., 2007). These methods, conventionally considered to



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 16

Calculating derivatives and

Derivative-free optimization

Lecturer: Lars Imsland

Lecture 16: Calculating derivatives (Ch. 8), and Derivative-free optimization (Ch. 9)

- Brief recap linesearch unconstrained optimization
- Calculating derivatives (gradient/Jacobian and Hessian)
- What can you do when obtaining derivatives is impractical?
 - Derivative-free optimization! For example: Nelder-Mead

Reference: N&W Ch. 8.1, (8.2), Ch. 9.1, 9.5

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

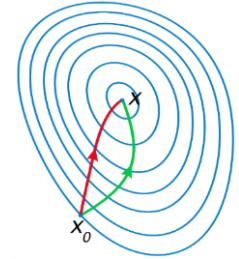
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



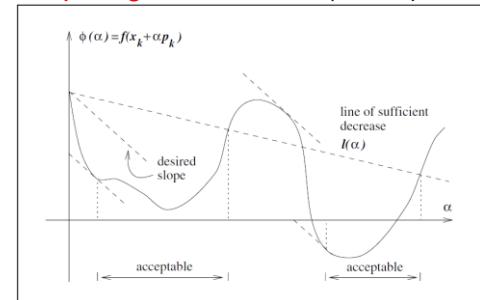
How to calculate derivatives – Ch. 8

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. ([wikipedia.org](https://en.wikipedia.org))

Step length line search (Wolfe):



Quasi-Newton: BFGS method

$$\begin{aligned} p_k &= -B_k^{-1} \nabla f(x_k) \\ B_k &\approx \nabla^2 f(x_k) \\ H_k &= B_k^{-1} \end{aligned}$$

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,

inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k;$$

 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
 procedure to satisfy the Wolfe conditions (3.6);

 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)

We use only gradient!

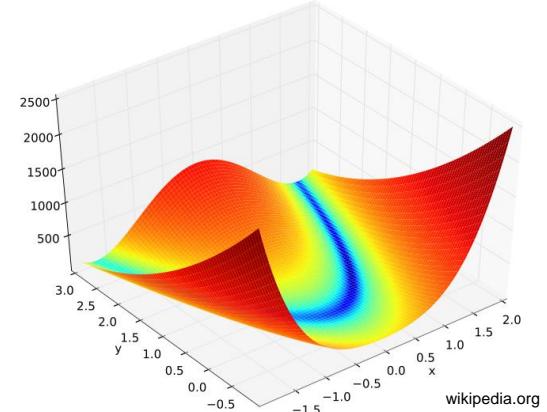
$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Example (from book)

- Using steepest descent, BFGS and inexact Newton on Rosenbrock function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- Iterations from starting point (-1.2,1):
 - Steepest descent: 5264
 - BFGS: 34
 - Newton: 21
- Last iterations; value of $\|x_k - x^*\|$



wikipedia.org

steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

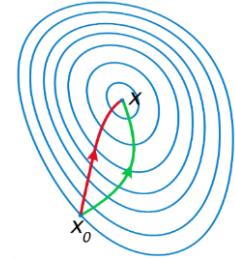
Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

Descent directions:

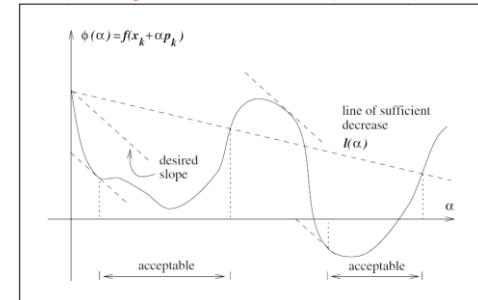
- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. ([wikipedia.org](https://en.wikipedia.org))

Step length line search (Wolfe):



Need derivatives! How to compute them?

And what if derivatives are not available, or too expensive to compute?

Four ways of finding derivatives

- By hand
 - Time consuming and (very!) error prone for large problems
- Symbolic differentiation
 - Computer algebra systems (CAS)
 - Maple, Mathematica, Matlab symbolic toolbox, ...
 - Typically result in very long code that is expensive to evaluate (and compile!)
- **Numerical differentiation (finite differences)**
 - Easy to implement (do it yourself), but may have low accuracy
- **Automatic (Algorithmic) Differentiation (AD)**
 - Best option!
 - Relatively easy to implement using the right software
 - Exact up to machine precision

Numerical differentiation

Theoretical accuracy of numerical differentiation

Taylor's theorem

$$f : \mathbb{R}^n \rightarrow \mathbb{R}, p \in \mathbb{R}^n$$

- First order: If f is continuously differentiable,

$$f(x + p) = f(x) + \nabla f(x + tp)^\top p, \quad \text{for some } t \in (0, 1)$$

- Second order: If f is twice continuously differentiable

$$f(x + p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x + tp)p, \quad \text{for some } t \in (0, 1)$$

Finite differences and accuracy

One-sided difference:

$$\frac{\partial f}{\partial x_i} = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} + O(\epsilon)$$

Two-sided difference:

$$\frac{\partial f}{\partial x_i} = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} + O(\epsilon^2)$$

```
>> eps  
ans =  
2.2204e-16
```

Numerical differentiation (finite differences)

- Scalar $f : \mathbb{R} \rightarrow \mathbb{R}$: For some small ϵ ,

$$\frac{df}{dx} \approx \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

- Directional derivative of $f : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\nabla f^\top p \approx \frac{f(x + \epsilon p) - f(x)}{\epsilon}$$

- Full gradient ∇f : Directional derivatives along all axes $p = \epsilon e_i$
($e_1 = (1, 0, 0, \dots)^\top, e_2 = (0, 1, 0, \dots)^\top, \dots$)

– Note: Not necessary to calculate full gradient if you only need directional derivative!
(Also valid for AD!)

- How to choose epsilon?

- Theoretical error proportional to ϵ , but too small ϵ gives numerical noise
- Rule of thumb: $\epsilon = \sqrt{\text{eps}}$, where eps is machine precision (or precision of computing f)
(IEEE double precision: $\epsilon = 10^{-8}$)

Approximating the Hessian

- In many cases, the gradient is available, but not the Hessian. We can then use finite differences on the gradient:

$$\nabla^2 f(x)p \approx \frac{\nabla f(x + \epsilon p) - \nabla f(x)}{\epsilon}$$

- If the gradient is not available, use finite differences “twice” for Hessian:

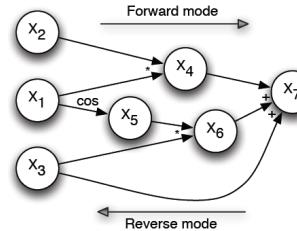
$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x + \epsilon e_i) - f(x + \epsilon e_j) + f(x)}{\epsilon^2} + O(\epsilon)$$

(but usually better to use Quasi-Newton then...)

AD – Automatic (algorithmic) differentiation

- Software tools that automatically computes derivatives of your code
- The principle is simple: Extensive/automated use of ‘chain rule’
- Two modes; forward and reverse
- Two (main) implementation variants
 - Source code transformation
 - **Operator overloading** (object oriented language)
- Requires (more or less) that your implementation is differentiable
- Example software: ADOL-C, CppAD, CasADI, ...

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

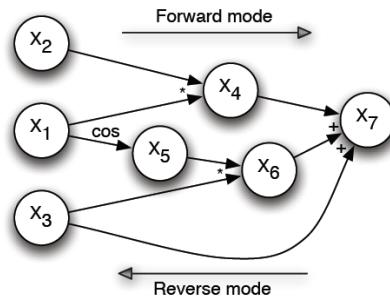


Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward and reverse

- Forward mode
 - Both x_i and ∇x_i are calculated by forward traversing computation graph

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

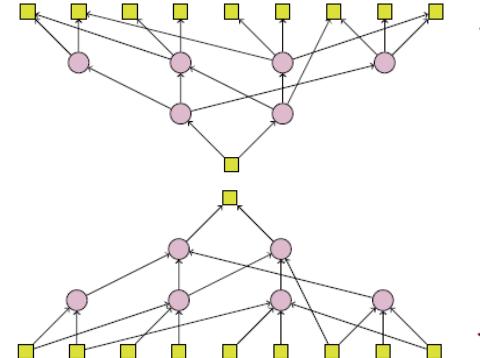


Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

- Reverse mode
 - First, calculate x_i by traversing graph forward
 - Then, calculate derivatives by traversing graph backward

AD – forward vs. reverse

- Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
 - Costs of calculating derivatives with AD:
 - Forward mode (one “column”): $\text{cost}(\nabla f^\top p) \leq 2 \text{ cost}(f)$
 - Forward mode (entire Jacobian): $\text{cost}(\nabla f) \leq 2n \text{ cost}(f)$
 - Reverse mode (one “row”): $\text{cost}(\lambda^\top \nabla f) \leq 3 \text{ cost}(f)$
 - Reverse mode (entire Jacobian): $\text{cost}(\nabla f) \leq 3m \text{ cost}(f)$
 - Forward mode: Similar cost as numerical differentiation, but more accurate
 - If $m \gg n$, forward mode is fastest
 - If $n \gg m$, reverse mode is fastest



How AD software is implemented

- Prototype procedure:
 - Decompose original code into “intrinsic” functions (e.g. x_1x_2 , $\sin(x)$, $\ln(x)$, etc.)
 - Differentiate the intrinsic functions ('symbolically', or make a lookup table) ($\sin(x)' = \cos(x)$, etc.)
 - Put everything together according to the chain rule
(either forward or reverse mode)
- How to automatically transform your program into a program with derivatives? Two approaches:
 - Source code transformation (Typical: C, Fortran)
 - **Operator overloading** (C++, Fortran 90, Java, Matlab, Python, ...)

Example (C/C++)

$$f(x_1, x_2, x_3) = x_1x_2 + x_3 \cos(x_1) + x_3$$

function.c

```
double f(double x1, double x2, double x3) {
    double x4, x5, x6, x7;

    x4 = x1*x2;
    x5 = cos(x1);
    x6 = x3*x5;
    x7 = x4 + x6 + x3;

    return x7;
}
```

function.c

Source code transformation (forward mode)

```
diff_function.c

double* f(double x1, double x2, double x3, double dx1, double dx2, double dx3) {
    double x4, x5, x6, x7, dx4, dx5, dx7, df[2];

    x4 = x1*x2;
    dx4 = dx1*x2 + x1*dx2;
    x5 = cos(x1);
    dx5 = -sin(x1)*dx1;
    x6 = x3*x5;
    dx6 = dx3*x5 + x3*dx5;
    x7 = x4 + x6 + x3;
    dx7 = dx4 + dx6 + dx3;

    df[0] = x7;
    df[1] = dx7;

    return df;
}
```



Operator overloading example (using CppAD)

- Implement function as you do normally, but with other types (here: using C++ templates):

```
function.cpp

template <class vector>
vector f(vector x) {
    vector ... // possible temporary variables

    return x[1]*x[2] + x[3]*cos(x[1]) + x[3];
}
```

- Record operation sequence when you use the function:

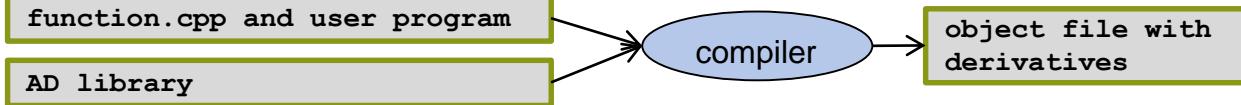
```
...
CppAD::vector<ADdouble> x(3), f_res;
x[1] = pi; x[2] = 4; x[3] = 3;

// declare that x contains the independent variables (and start recording)
CppAD::Independent(x);

f_res = f(x);

// create the AD function object F : x -> f_res (and stop recording)
CppAD::ADFun<double> F(x, f_res);

std::vector<double> jac( NS*NS ) = F.Jacobian;
...
```



Software etc.

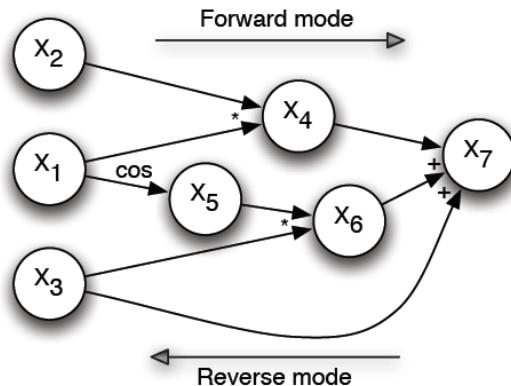
- General information
 - <http://www.autodiff.org/>
 - http://en.wikipedia.org/wiki/Automatic_differentiation
- Many libraries of different maturity/robustness/performance, for different languages and for different applications
- Some mature libraries
 - C++: ADOL-C, CppAD
 - Developed for control&optimization: CasADi (Matlab/Octave, Python, C++)
- Book:
 - A. Griewank, **A. Walther**, “Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation”, 2nd edition. SIAM, 2008.



AD – example (from R. Ringset)

- Calculate gradient of $f(x_1, x_2, x_3) = x_1x_2 + x_3 \cos(x_1) + x_3$

at $\begin{bmatrix} \pi & 4 & 3 \end{bmatrix}^T$



Variables	Derivatives
$x_4 = x_1x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

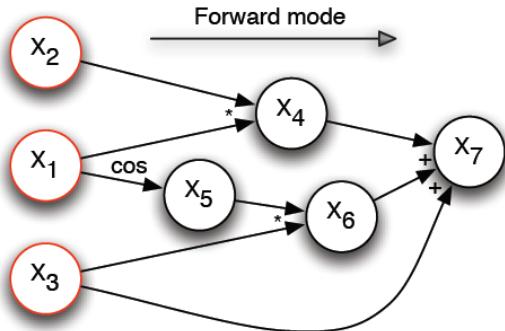
AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\begin{aligned}\nabla x_1 &= e_1 \\ \nabla x_2 &= e_2 \\ \nabla x_3 &= e_3 \\ \nabla x_4 &=? \\ \nabla x_5 &=? \\ \nabla x_6 &=? \\ \nabla x_7 &=?\end{aligned}$$

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

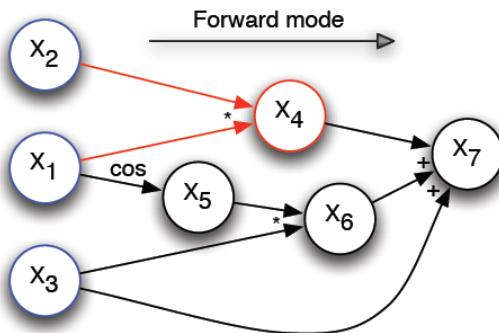
$$\nabla x_3 = e_3$$

$$\nabla x_4 = \frac{\partial x_4}{\partial x_1} \nabla x_1 + \frac{\partial x_4}{\partial x_2} \nabla x_2 = [\begin{array}{ccc} x_2 & x_1 & 0 \end{array}]^T = [\begin{array}{ccc} 4 & \pi & 0 \end{array}]^T$$

$$\nabla x_5 = ?$$

$$\nabla x_6 = ?$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

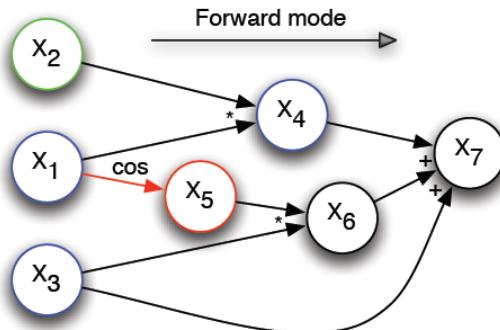
$$\nabla x_3 = e_3$$

$$\nabla x_4 = [\begin{array}{ccc} 4 & \pi & 0 \end{array}]^T$$

$$\nabla x_5 = \frac{\partial x_5}{\partial x_1} \nabla x_1 = -\sin(x_1) e_1 = 0$$

$$\nabla x_6 = ?$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

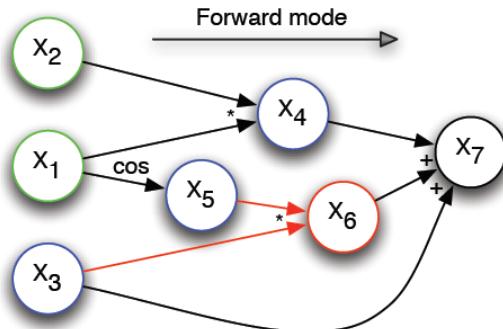
$$\nabla x_3 = e_3$$

$$\nabla x_4 = [\begin{array}{ccc} 4 & \pi & 0 \end{array}]^T$$

$$\nabla x_5 = 0$$

$$\nabla x_6 = \frac{\partial x_6}{\partial x_3} \nabla x_3 + \frac{\partial x_6}{\partial x_5} \nabla x_5 = x_5 e_3 + x_3 0 = \cos(\pi) e_3 = -e_3$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

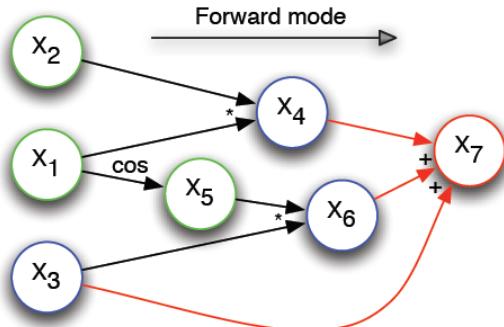
$$\nabla x_3 = e_3$$

$$\nabla x_4 = [\begin{array}{ccc} 4 & \pi & 0 \end{array}]^\top$$

$$\nabla x_5 = 0$$

$$\nabla x_6 = -e_3$$

$$\nabla x_7 = \nabla f(x) = \frac{\partial x_7}{\partial x_4} \nabla x_4 + \frac{\partial x_7}{\partial x_6} \nabla x_6 + \frac{\partial x_7}{\partial x_3} \nabla x_3 = \left[\begin{array}{c} 4 \\ \pi \\ 0 \end{array} \right] - e_3 + e_3 = \left[\begin{array}{c} 4 \\ \pi \\ 0 \end{array} \right]$$

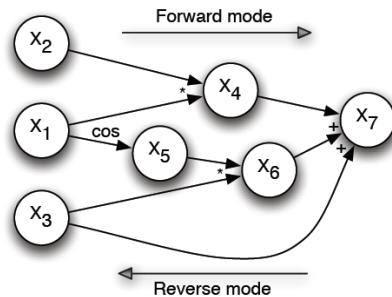


Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward and reverse

- Forward mode
 - Both x_i and ∇x_i are calculated by forward traversing computation graph

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

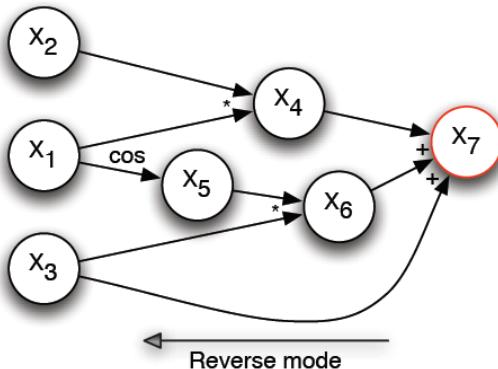
- Reverse mode
 - First, calculate x_i by traversing graph forward
 - Then, calculate derivatives by traversing graph backward

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$



$$\frac{\partial f}{\partial x_i} = \sum_{x_n \text{ child of } x_i} \frac{\partial f}{\partial x_n} \frac{\partial x_n}{\partial x_i}$$

Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

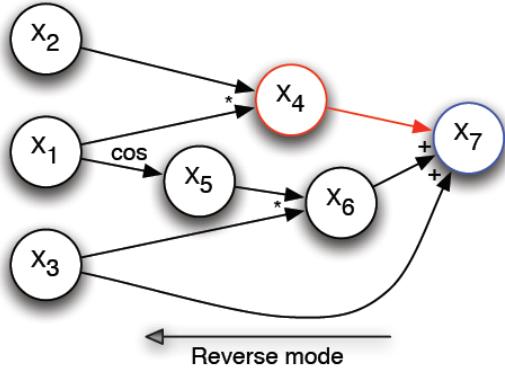
AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

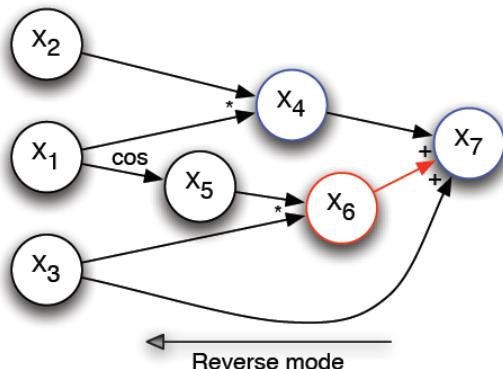
$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

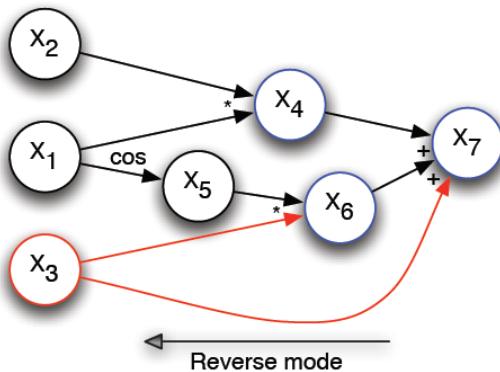
$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = 0$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

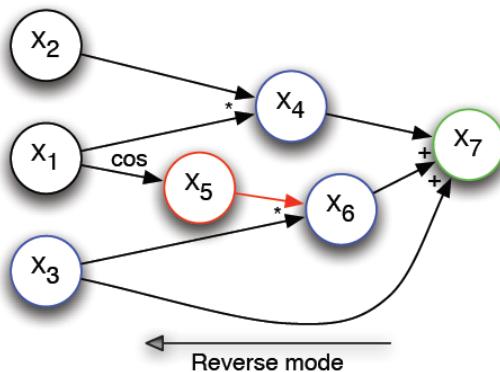
$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = 0$$

$$\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_5} = x_3 = 3$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

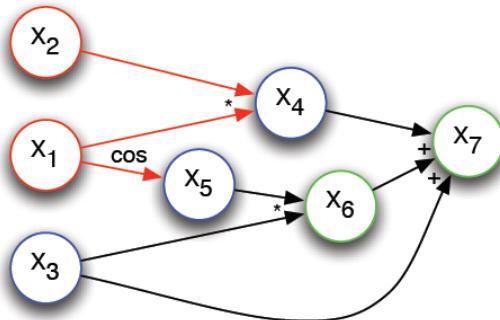
$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = 0$$

$$\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_5} \frac{\partial x_6}{\partial x_5} = x_3 = 3$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_4} \frac{\partial x_4}{\partial x_1} + \frac{\partial f}{\partial x_5} \frac{\partial x_5}{\partial x_1} = x_2 - 3 \sin(x_1) = 4 - 3 \sin(\pi) = 4$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_4} \frac{\partial x_4}{\partial x_3} = x_1 = \underline{\pi}$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

Example: optimization using CasADI

- CasADI (<https://casadi.org/>)
 - “CasADI is a symbolic framework for numeric optimization implementing automatic differentiation in forward and reverse modes on sparse matrix-valued computational graphs.”

$$\begin{aligned} \min_{x,y,z} \quad & x^2 + 100z^2 \\ \text{s.t. } \quad & z + (1-x)^2 - y = 0 \end{aligned}$$

Define variables

Define objective and constraints

Create solver object

Solve the opt problem

```
rosenbrock.m

import casadi.*

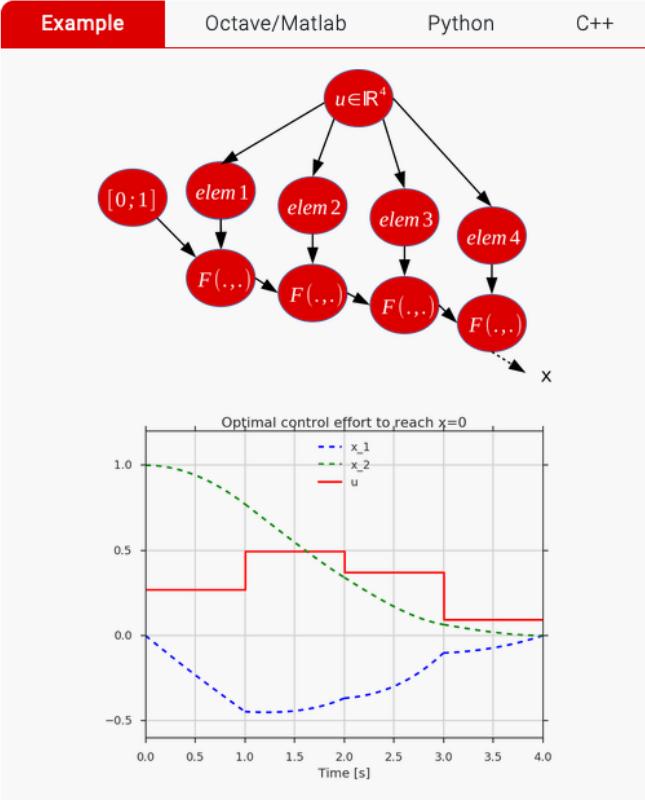
% Create NLP: Solve the Rosenbrock problem:
%   minimize    x^2 + 100*z^2
%   subject to  z + (1-x)^2 - y == 0
x = SX.sym('x');
y = SX.sym('y');
z = SX.sym('z');
v = [x;y;z];
f = x^2 + 100*z^2;
g = z + (1-x)^2 - y;
nlp = struct('x', v, 'f', f, 'g', g);

% Create IPOPT solver object
solver = nlpsol('solver', 'ipopt', nlp);

% Solve the NLP
res = solver('x0', [2.5 3.0 0.75],... % solution guess
             'lbx', -inf,... % lower bound on x
             'ubx', inf,... % upper bound on x
             'lbg', 0,... % lower bound on g
             'ubg', 0); % upper bound on g

% Print the solution
f_opt = full(res.f)           % >> 0
x_opt = full(res.x)           % >> [0; 1; 0]
lam_x_opt = full(res.lam_x)   % >> [0; 0; 0]
lam_g_opt = full(res.lam_g)   % >> 0
```

Example from CasADi



Example Octave/Matlab **Python** C++

```
from casadi import *

x = MX.sym('x',2) # Two states
p = MX.sym('p'); # Free parameter

# Expression for ODE right-hand side
z = 1-x[1]**2;
rhs = vertcat(z*x[0]-x[1]+2*tanh(p),x[0])

# ODE declaration with free parameter
ode = {'x':x,'p':p,'ode':rhs}

# Construct a Function that integrates over 1s
F = integrator('F','cvodes',ode,{tf:1})

# Control vector
u = MX.sym('u',4,1)

x = [0,1] # Initial state
for k in range(4):
    # Integrate 1s forward in time:
    # call integrator symbolically
    res = F(x0=x,p=u[k])
    x = res["xf"]

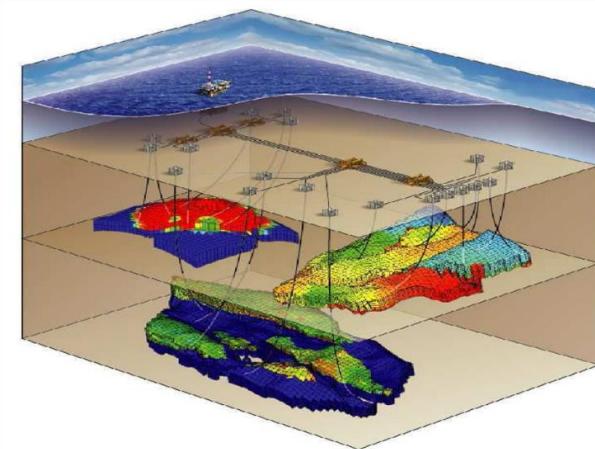
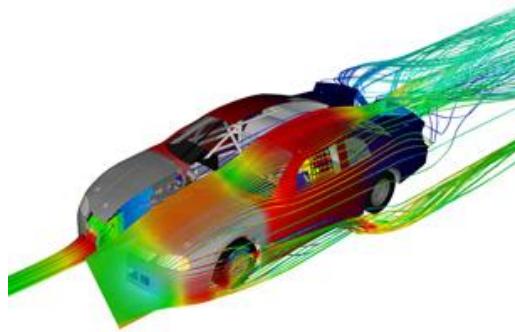
# NLP declaration
nlp = {'x':u,'f':dot(u,u),'g':x};

# Solve using IPOPT
solver = nlpSolver('solver','ipopt',nlp)
res = solver(x0=0.2,lbx=0,ubx=0)

plot(res["x"])
```

Derivative-free optimization

- If you have derivatives (gradients, possibly Hessian), use them!
 - “Always” more efficient than not using them!
- However, sometimes, obtaining derivatives is prohibitive
 - Typically: Objective function (and constraints) are calculated using “heavy” simulators
 - Often models from computational fluid dynamics (CFD)

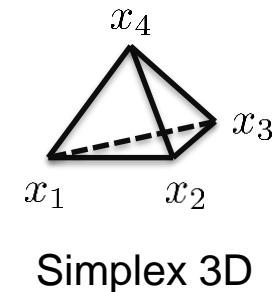
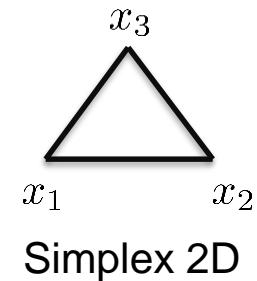


- This motivates “derivative-free optimization” methods”

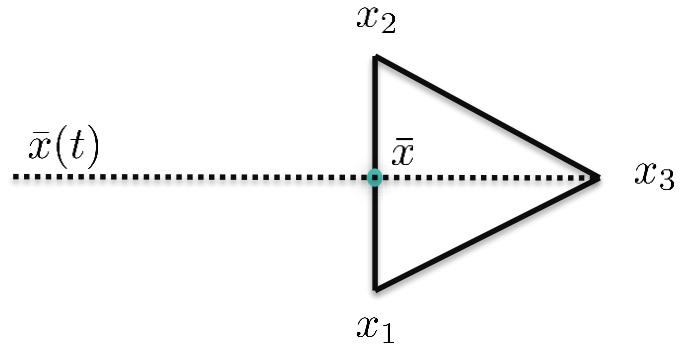
Derivative-free optimization (DFO)

- DFO use function values at a set of **sample points** to determine new iterates.
- Coarsely, two different classes of methods:
 1. “**Model-based**”: sample points -> approximate model -> search directions
 2. “**Metaheuristics**”: Often inspired by processes in nature, such as “genetic algorithm”, “simulated annealing”, “particle swarm optimization”, “wolf pack optimization”, ...
- Many of the metaheuristic methods claim to do “global optimization” and tackle “non-differentiable problems”, but this must be interpreted with care. Guarantees are seldom given.
- DFO generally works best if the number of optimization variables is relatively small
- Here: Nelder-Mead (old&simple, but fairly good)

Nelder-Mead Simplex method (Ch. 9.5)

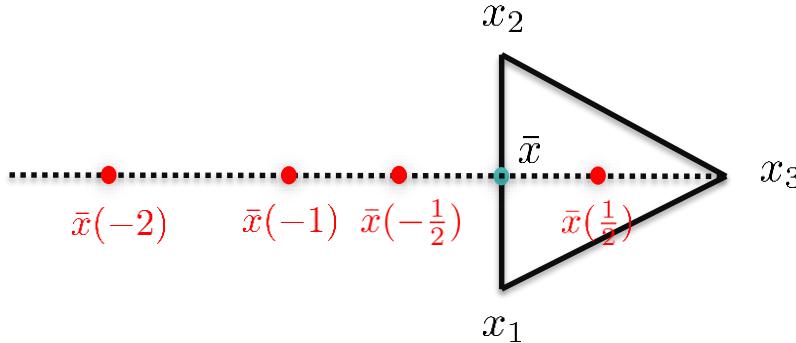


Nelder-Mead Simplex method (Ch. 9.5)



One iteration of Nelder-Mead

$$\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$$

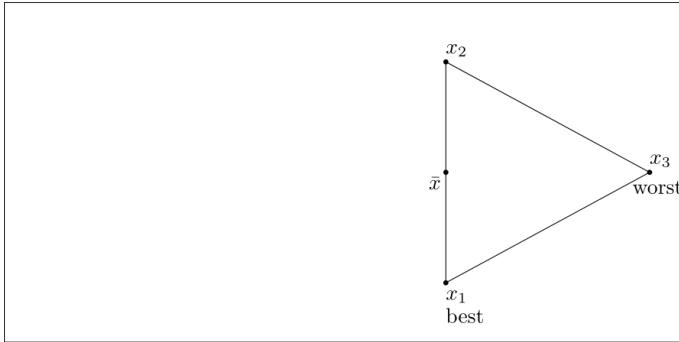


One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

 replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

 evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

 replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

 replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

 evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

 replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

 evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

 replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

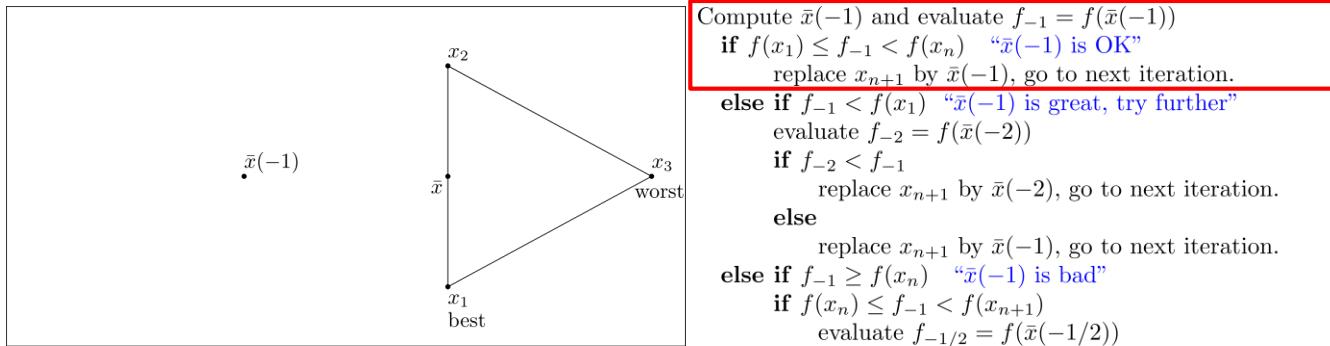
 replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Reflection

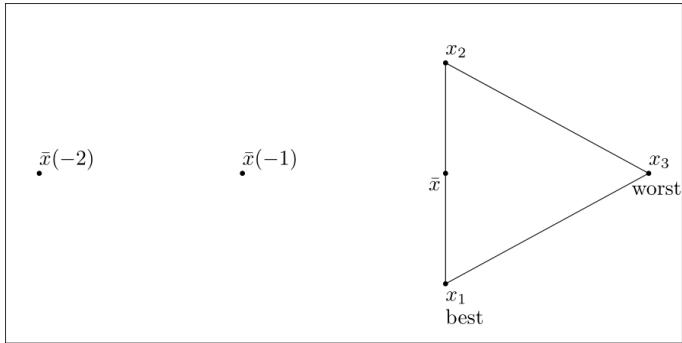
```
Compute  $\bar{x}(-1)$  and evaluate  $f_{-1} = f(\bar{x}(-1))$ 
if  $f(x_1) \leq f_{-1} < f(x_n)$  " $\bar{x}(-1)$  is OK"
    replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} < f(x_1)$  " $\bar{x}(-1)$  is great, try further"
    evaluate  $f_{-2} = f(\bar{x}(-2))$ 
    if  $f_{-2} < f_{-1}$ 
        replace  $x_{n+1}$  by  $\bar{x}(-2)$ , go to next iteration.
    else
        replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} \geq f(x_n)$  " $\bar{x}(-1)$  is bad"
    if  $f(x_n) \leq f_{-1} < f(x_{n+1})$ 
        evaluate  $f_{-1/2} = f(\bar{x}(-1/2))$ 
        if  $f_{-1/2} \leq f_{-1}$ 
            replace  $x_{n+1}$  by  $\bar{x}(-1/2)$ , go to next iteration.
        else ( $f_{-1} > f(x_{n+1})$ )
            evaluate  $f_{1/2} = f(\bar{x}(1/2))$ 
            if  $f_{1/2} < f_{n+1}$ 
                replace  $x_{n+1}$  by  $\bar{x}(1/2)$ , go to next iteration.
            replace  $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$ ,  $i = 2, 3, \dots, n + 1$  "shrink"
```

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Expansion

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

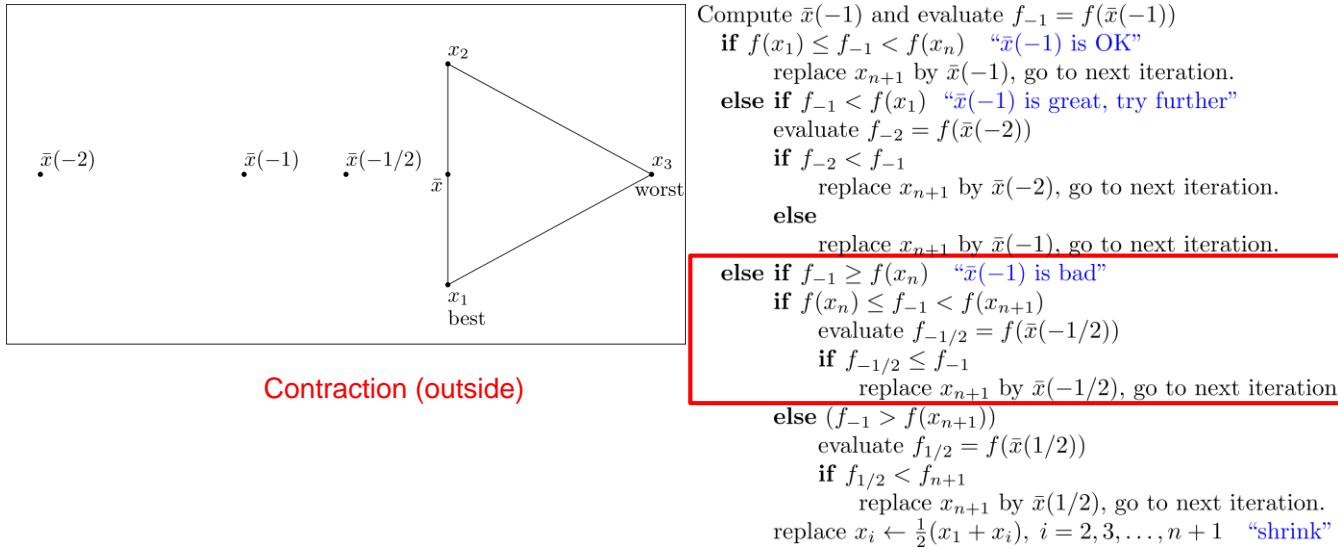
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$

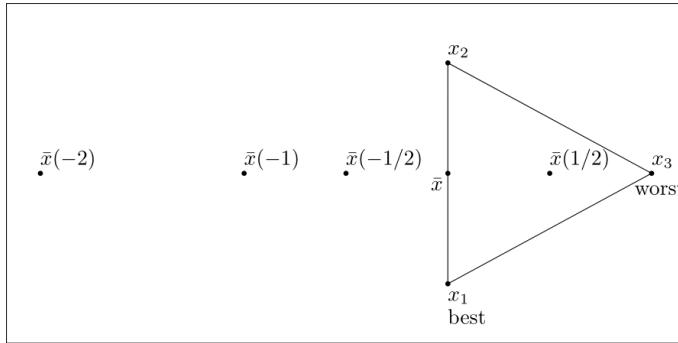


One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Contraction (inside)

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”
 replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”
 evaluate $f_{-2} = f(\bar{x}(-2))$
 if $f_{-2} < f_{-1}$
 replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.
 else
 replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”
 if $f(x_n) \leq f_{-1} < f(x_{n+1})$
 evaluate $f_{-1/2} = f(\bar{x}(-1/2))$
 if $f_{-1/2} \leq f_{-1}$
 replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)
 evaluate $f_{1/2} = f(\bar{x}(1/2))$
 if $f_{1/2} < f_{n+1}$
 replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

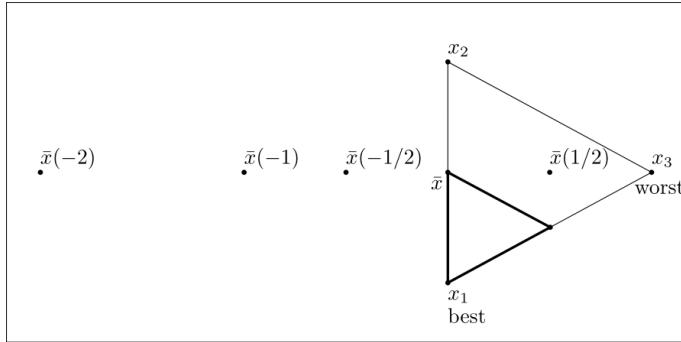
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Shrinkage

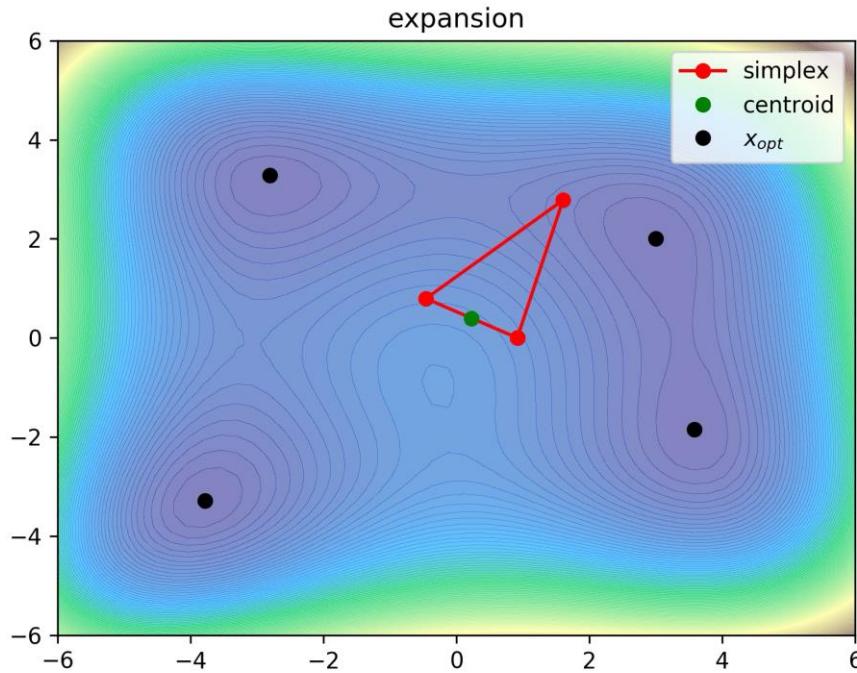
```
Compute  $\bar{x}(-1)$  and evaluate  $f_{-1} = f(\bar{x}(-1))$ 
if  $f(x_1) \leq f_{-1} < f(x_n)$  " $\bar{x}(-1)$  is OK"
    replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} < f(x_1)$  " $\bar{x}(-1)$  is great, try further"
    evaluate  $f_{-2} = f(\bar{x}(-2))$ 
    if  $f_{-2} < f_{-1}$ 
        replace  $x_{n+1}$  by  $\bar{x}(-2)$ , go to next iteration.
    else
        replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} \geq f(x_n)$  " $\bar{x}(-1)$  is bad"
    if  $f(x_n) \leq f_{-1} < f(x_{n+1})$ 
        evaluate  $f_{-1/2} = f(\bar{x}(-1/2))$ 
        if  $f_{-1/2} \leq f_{-1}$ 
            replace  $x_{n+1}$  by  $\bar{x}(-1/2)$ , go to next iteration.
        else ( $f_{-1} > f(x_{n+1})$ )
            evaluate  $f_{1/2} = f(\bar{x}(1/2))$ 
            if  $f_{1/2} < f_{n+1}$ 
                replace  $x_{n+1}$  by  $\bar{x}(1/2)$ , go to next iteration.
            replace  $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$ ,  $i = 2, 3, \dots, n + 1$  "shrink"
```

Termination and convergence

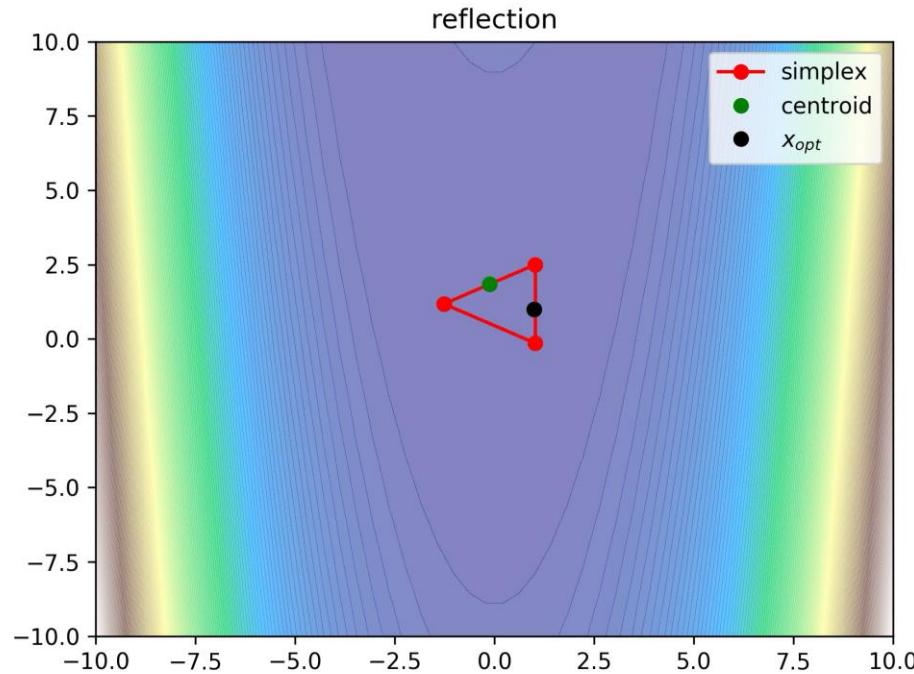
Termination: $|f(x_1) - f(x_{n+1})| \leq \text{tol}$

Convergence: The average value $\frac{1}{n+1} \sum_{i=1}^{n+1} f(x_i)$ decrease in most iterations

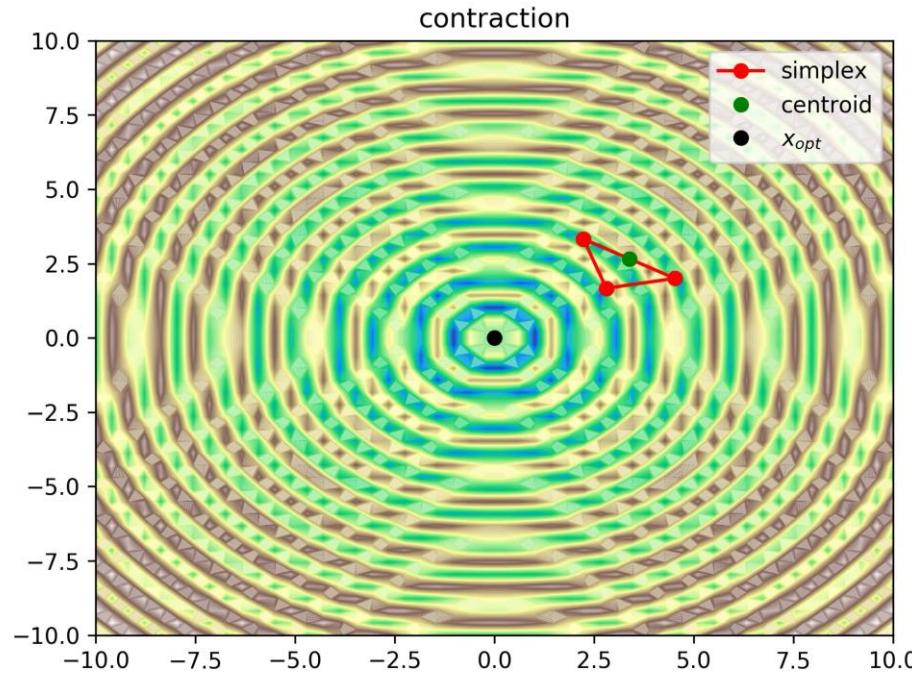
Examples: Nelder-Mead, Himmelblau



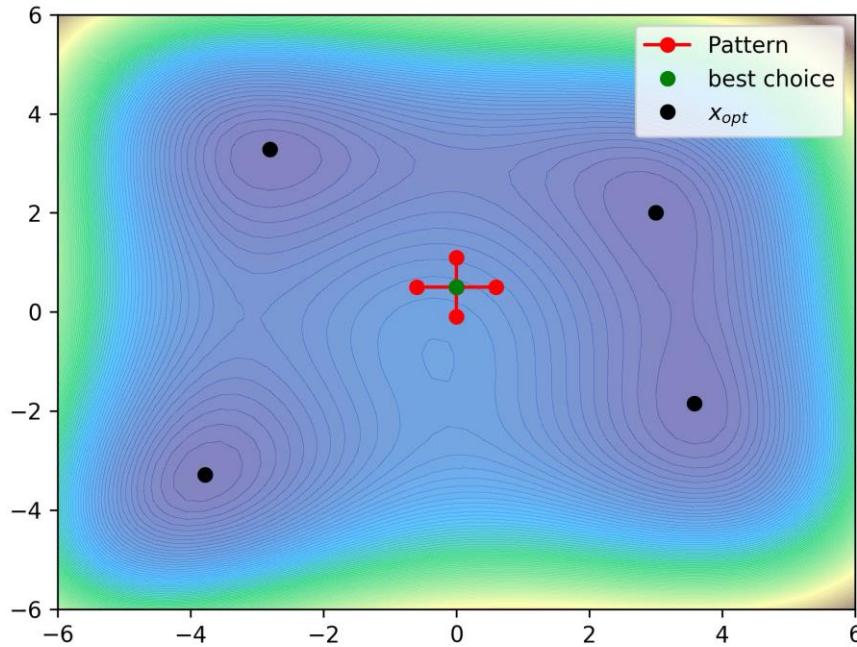
Example: Nelder-Mead, Rosenbrock



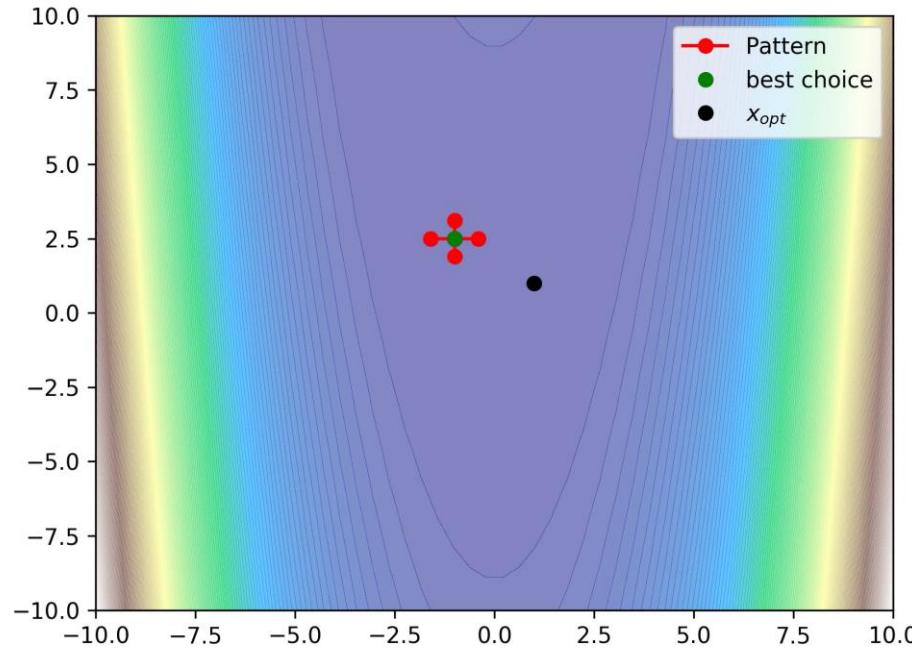
Example: Nelder-Mead, Salomon



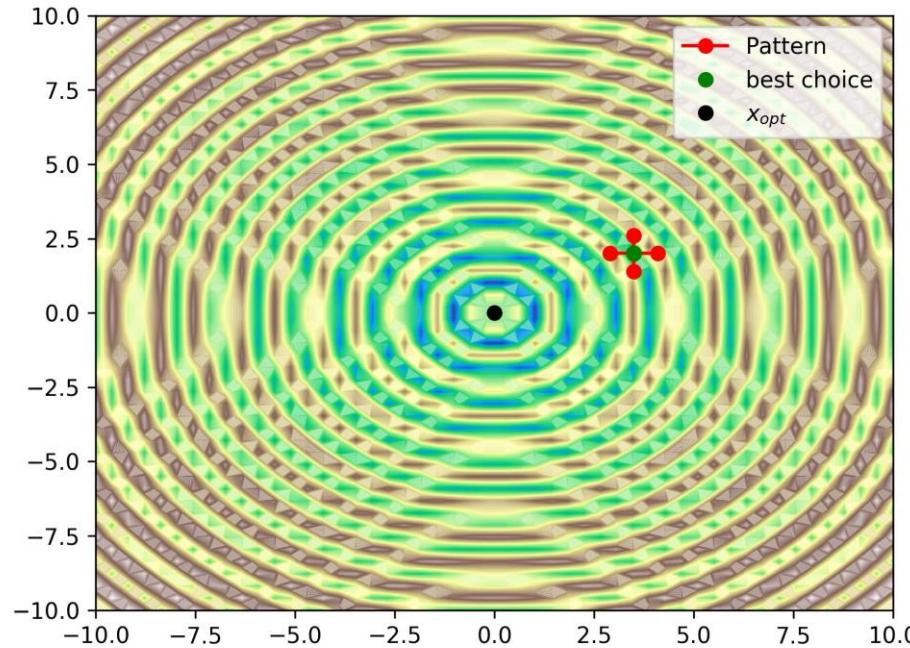
Example: Coordinate Descent, Himmelblau



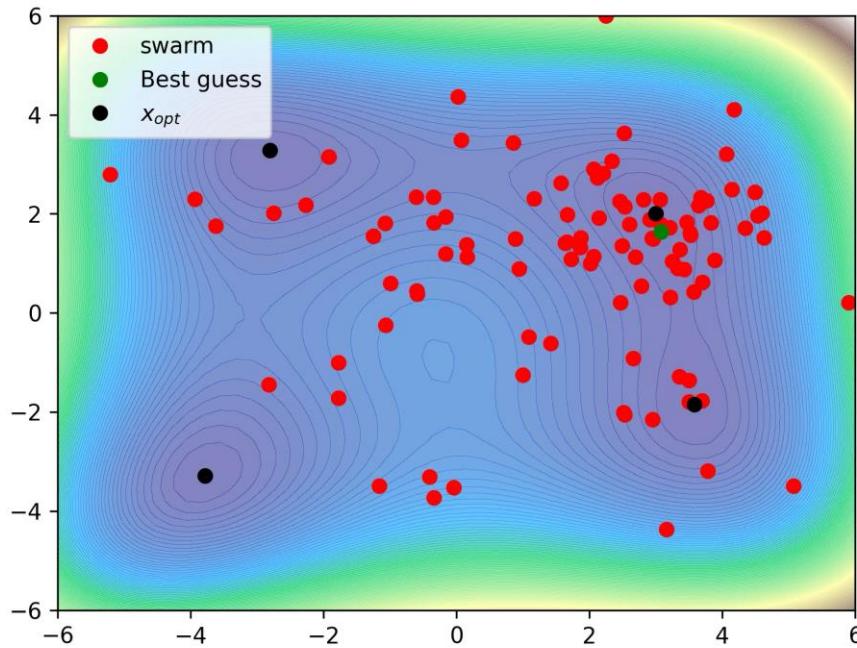
Example: Coordinate Descent, Rosenbrock



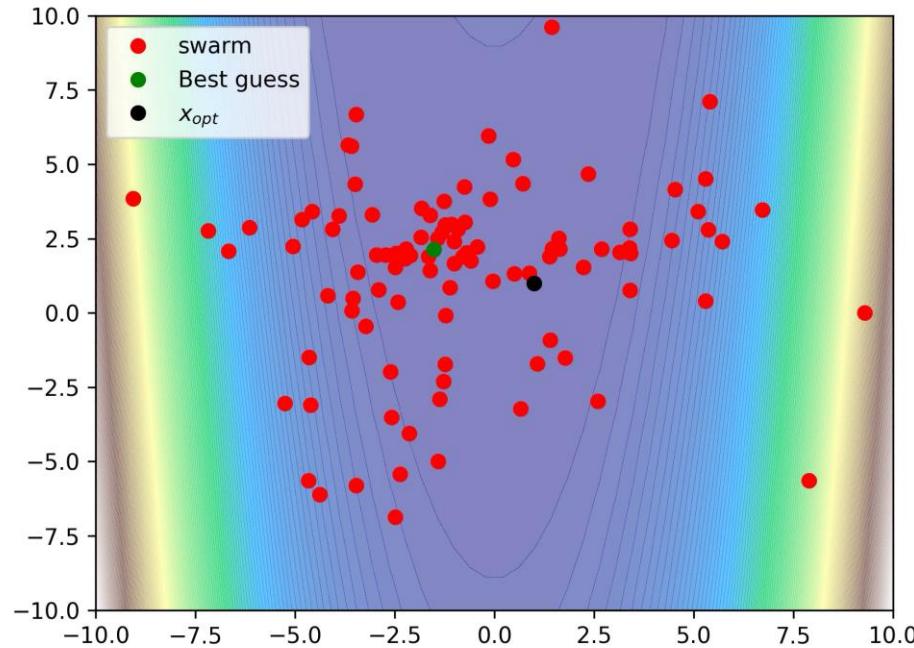
Example: Coordinate Descent, Salomon



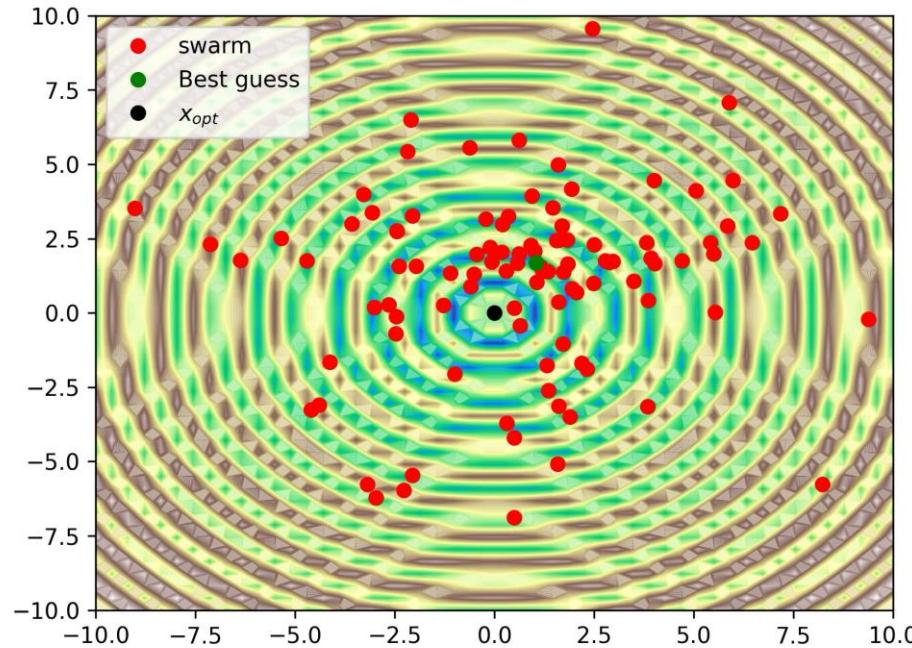
Example: Particle Swarm, Himmelblau



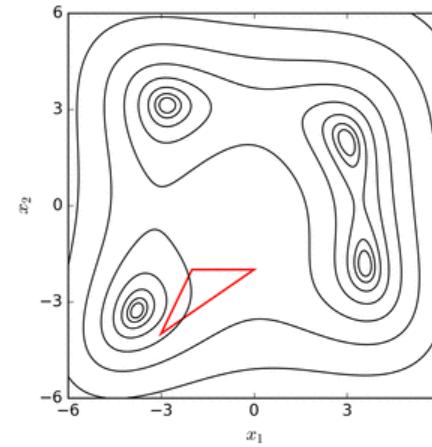
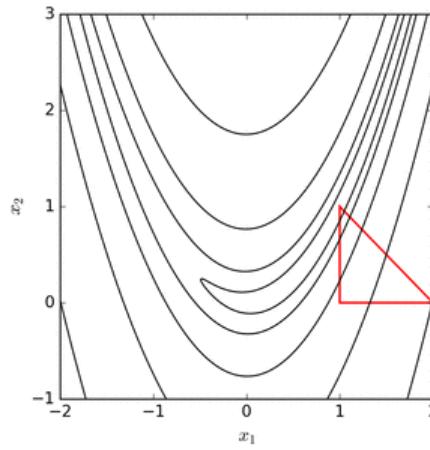
Example: Particle Swarm, Rosenbrock



Example: Particle Swarm, Salomon



Examples





NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 17

Nonlinear Equations

Lecturer: Lars Imsland

Outline

- A brief summary of Ch. 10: (Nonlinear) Least Squares
- **Nonlinear equations** (Ch. 11)
 - Newton's method for solving nonlinear equations
 - Convergence
 - Merit functions

Reference: N&W Ch. 11-11.1

Gradient and Jacobian

- The *gradient* of a scalar function $f(x)$ of several variables is

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix}^\top$$

- Say $f(x) = (f_1(x) \quad f_2(x) \quad \dots \quad f_m(x))^\top$. We define the *Jacobian* as the m by n matrix

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(x)^\top \\ \nabla f_2(x)^\top \\ \vdots \\ \nabla f_m(x)^\top \end{pmatrix}$$

A brief aside: Nonlinear least squares (Ch. 10)

- Consider the following problem: We have a number of (noisy) data

$$(u_1, y_1), (u_2, y_2), \dots, (u_m, y_m)$$

and want to fit the function

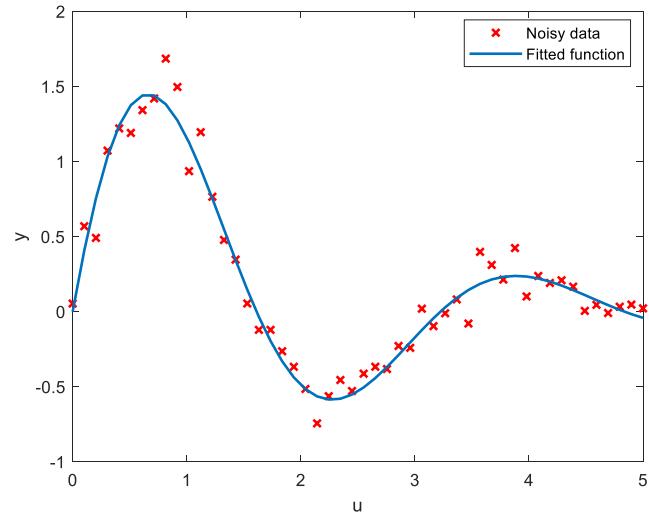
$$y = \theta_1 e^{\theta_2 u} \sin(\theta_3 u)$$

to the data

- (Nonlinear) least squares formulation:

$$\theta = \arg \min_{\theta \in \mathbb{R}^3} \sum_{j=1}^m \underbrace{(y_j - \theta_1 e^{\theta_2 u_j} \sin(\theta_3 u_j))^2}_{\text{residual } r_j(\theta)}$$

- Generalizations:
 - (Statistical) Machine Learning: **Regression**, or parametric learning
 - Control theory: **System identification** (fitting dynamic models to data)



How to solve nonlinear least squares problems

This is an **unconstrained optimization problem** (with typically $m \gg n$):

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} \sum_{j=1}^m r_j(x)^2$$

Say we want to use Newton's method. We need gradient and Hessian of objective function:

- First find gradient of **residuals** $r_j(x)$:

$$r(x) = (r_1(x) \quad r_2(x) \quad \dots \quad r_m(x))^{\top} \quad J(x) = \begin{pmatrix} \nabla r_1(x)^{\top} \\ \nabla r_2(x)^{\top} \\ \vdots \\ \nabla r_m(x)^{\top} \end{pmatrix}$$

- Gradient and Hessian of **objective** $f(x) = \frac{1}{2} \|r(x)\|^2$:

$$\nabla f(x) = \sum_{j=1}^m r_j(x) \nabla r_j(x) = J(x)^{\top} r(x)$$

$$\nabla^2 f(x) = \sum_{j=1}^m \nabla r_j(x) \nabla r_j(x)^{\top} + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x) = J(x) J(x)^{\top} + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x)$$

Gauss-Newton method

- For these problems, a good approximation of the Hessian is

$$\nabla^2 f(x) = J(x)J(x)^\top + \sum_{j=1}^m r_j(x)\nabla^2 r_j(x) \approx J(x)J(x)^\top$$

- The **Gauss-Newton method for nonlinear least squares** problems: Use **Newton's method with this Hessian approximation**
 - Note: Only first-order derivatives are needed!
 - Make it work far from solution: Use linesearch with Wolfe-conditions, etc. (same as before)
- (Using the same approximation with trust-region instead of linesearch is the *Levenberg-Marquardt* algorithm – implemented in Matlab-function `lsqnonlin`)

Linear least squares

- Say you want to fit a polynomial $y = \theta_1 + \theta_2 u + \theta_3 u^2 + \dots$ to data $(u_1, y_1), (u_1, y_1), \dots, (u_m, y_m)$
- Define $x = (\theta_1 \ \theta_2 \ \theta_3 \ \dots)^\top$ and formulate least squares optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} \sum_{j=1}^m r_j(x)^2 = \frac{1}{2} \sum_{j=1}^m (y_j - (1 \ u_j \ u_j^2 \ \dots) x)^2 = \frac{1}{2} \|y - Ax\|^2$$

where the *regressor matrix* A is

$$A = \begin{pmatrix} 1 & u_1 & u_1^2 & \dots \\ 1 & u_2 & u_2^2 & \dots \\ \vdots & \vdots & \vdots & \dots \\ 1 & u_m & u_m^2 & \dots \end{pmatrix}$$

Linear in parameters!

- Easy to show that the solution is given from

$$A^\top A x = A^\top y \quad \Rightarrow \quad x = (A^\top A)^{-1} A^\top y$$

- Solve by Cholesky or (better) QR (see book 10.2). Matlab: $\mathbf{x} = \mathbf{A}\backslash\mathbf{y}$.

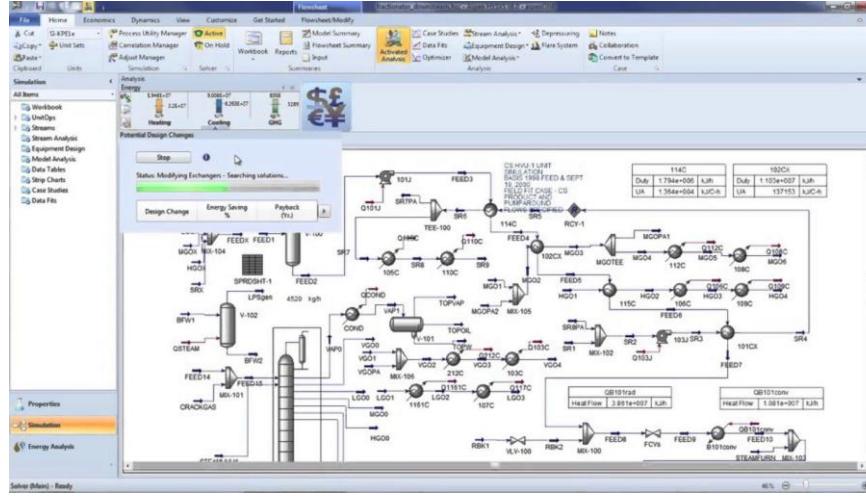
Observe: The Gauss-Newton approximation $A^\top A$ is exact for linear problems!

Nonlinear equations

Nonlinear equations

Why study nonlinear equations? – Examples

- Given nonlinear system $\dot{x} = f(x)$, the steady state is found by solving $f(x) = 0$
- Flowsheet analysis in chemical/process engineering (steady state simulators)



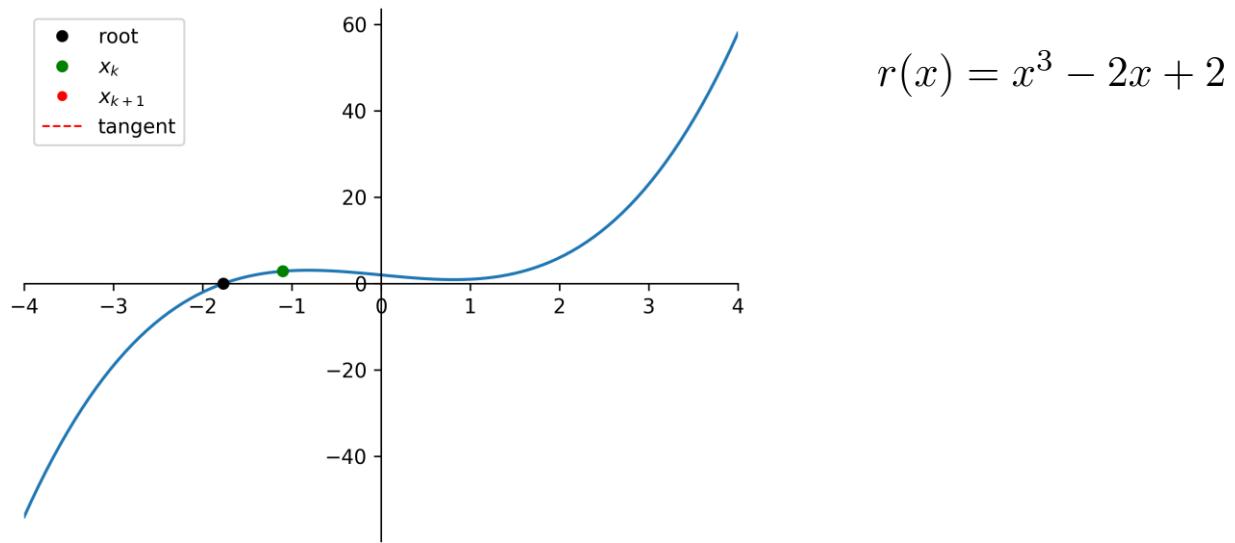
- Simulation methods (ModSim): For implicit Runge-Kutta, we need to solve nonlinear equations
- Newton's method for nonlinear equations is important for SQP methods (next lecture)

Derivation of Newton's method for nonlinear equations

Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

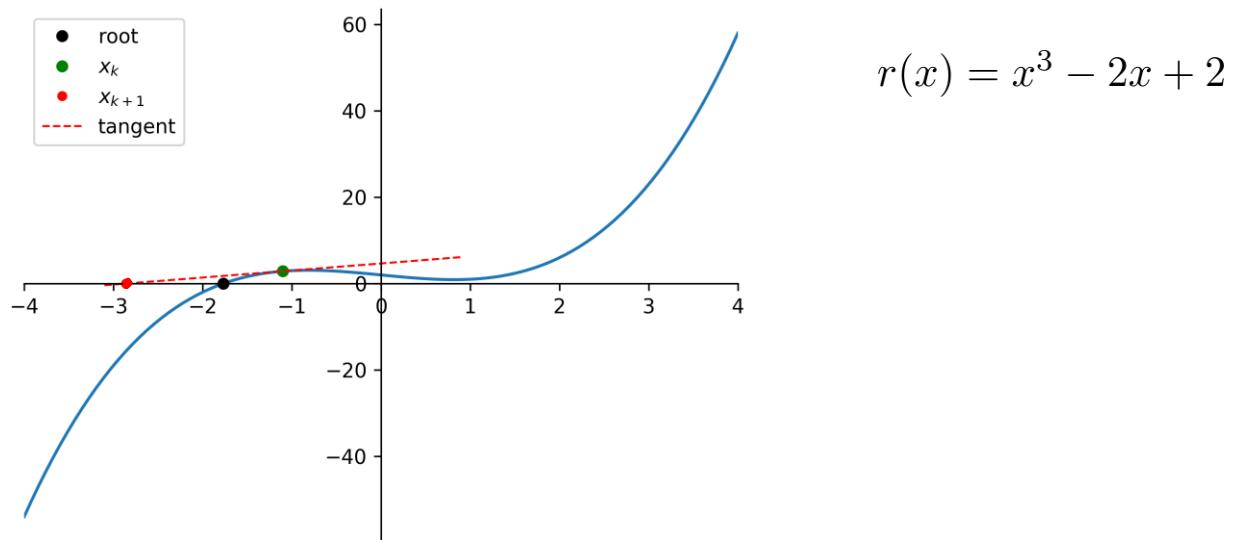
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

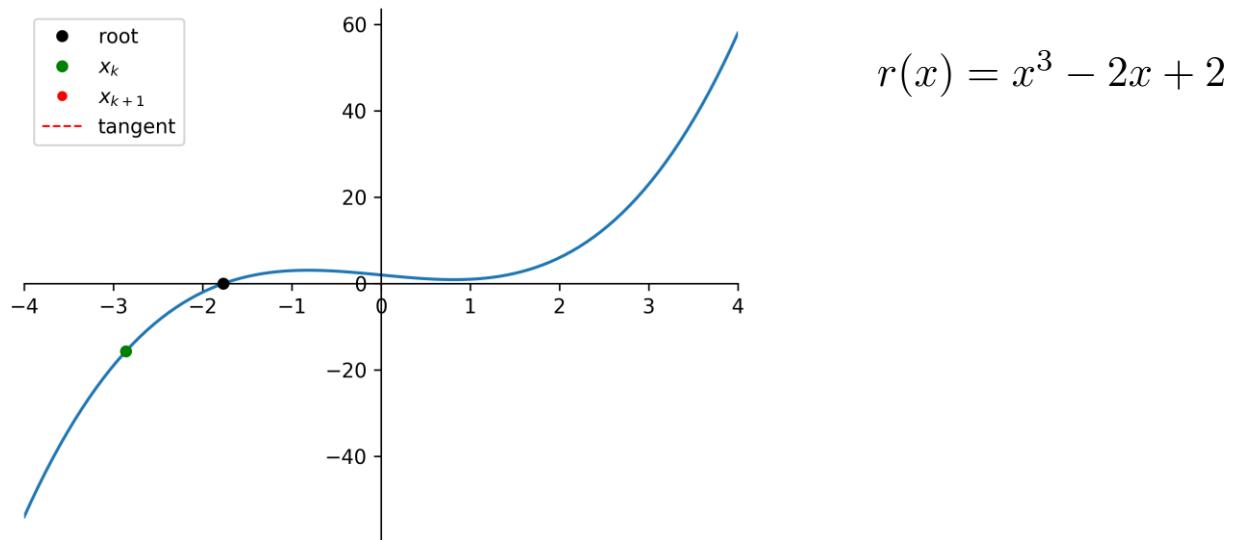
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

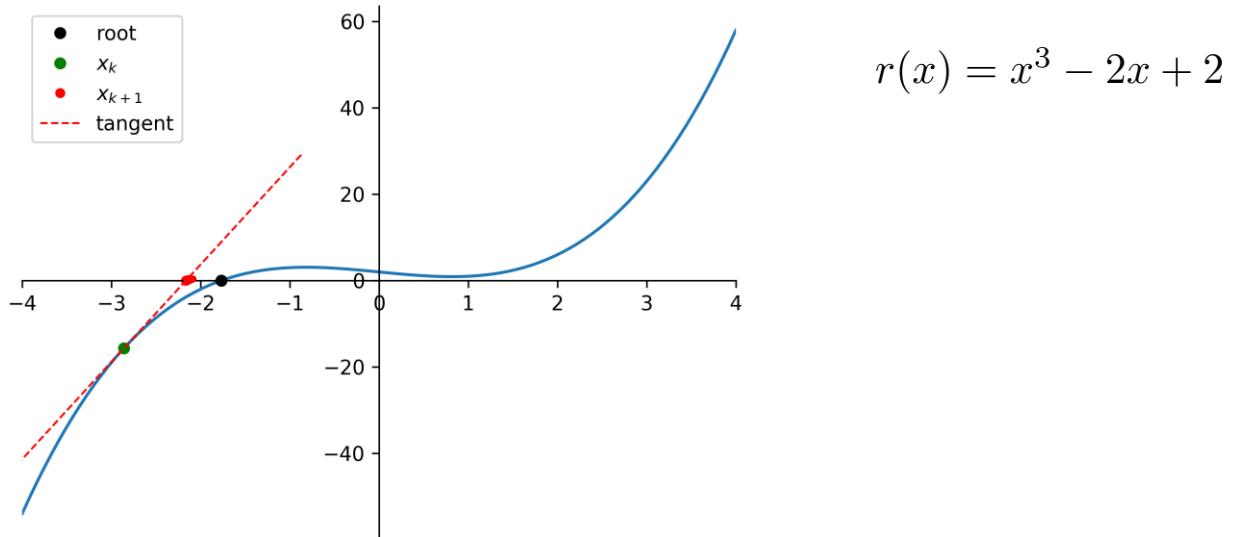
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

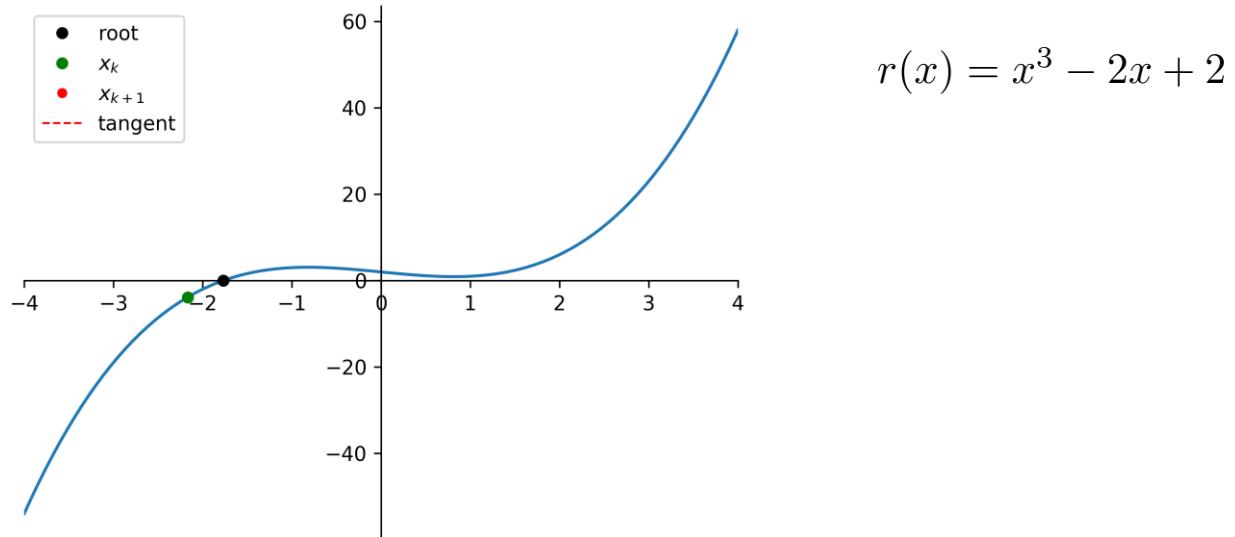
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

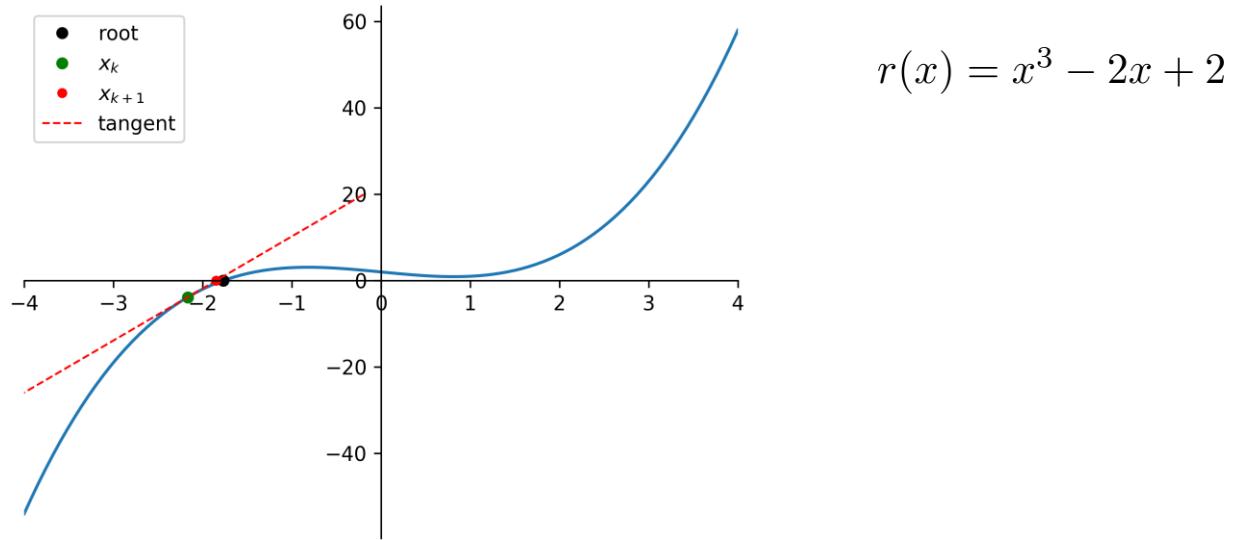
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

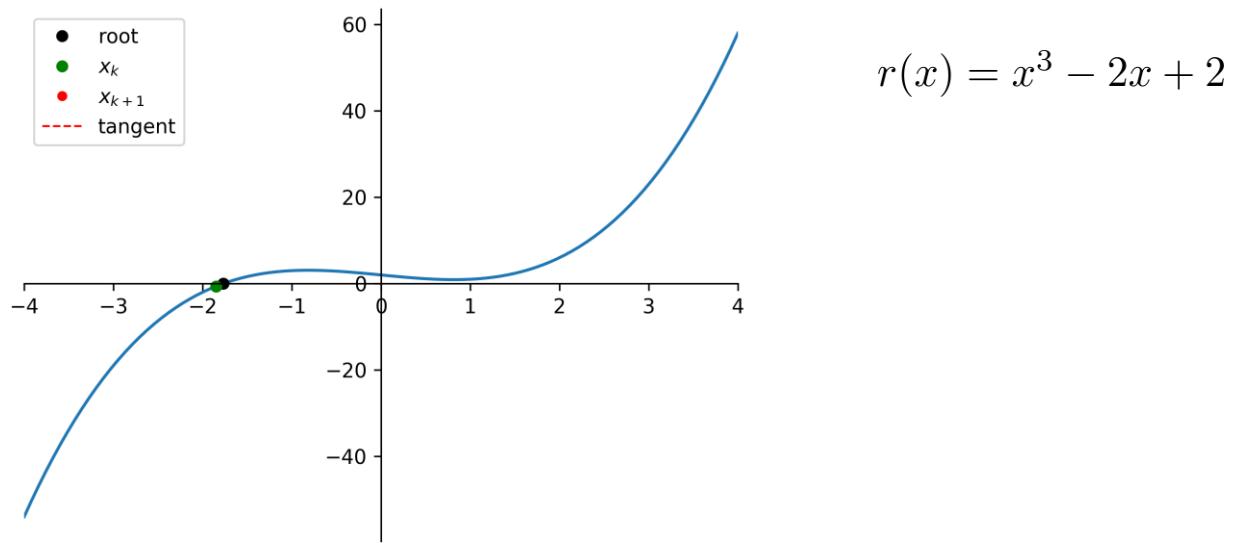
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

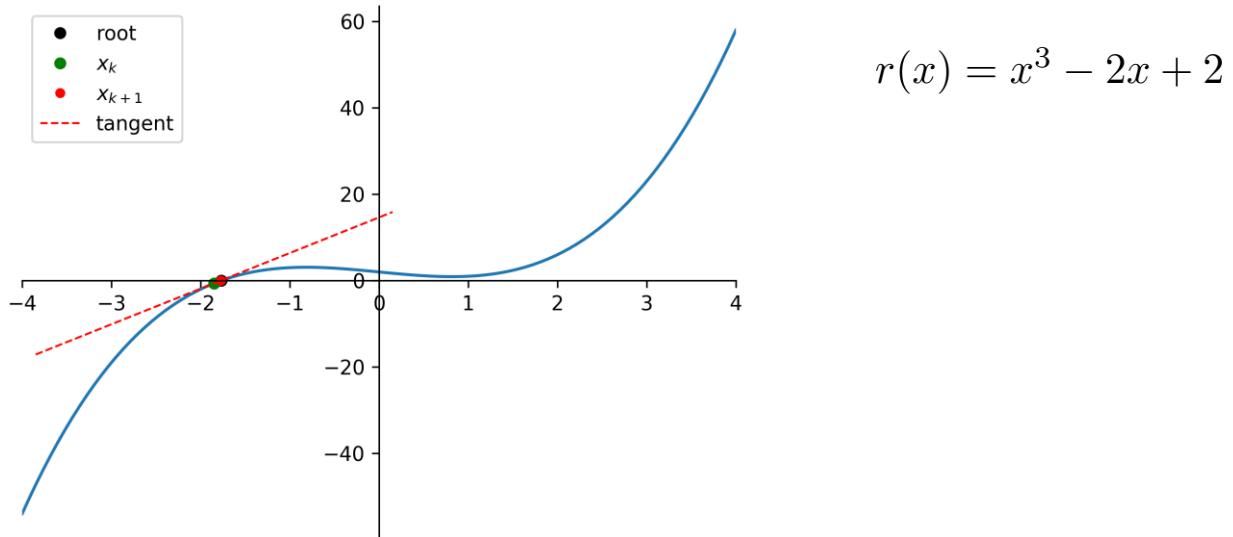
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

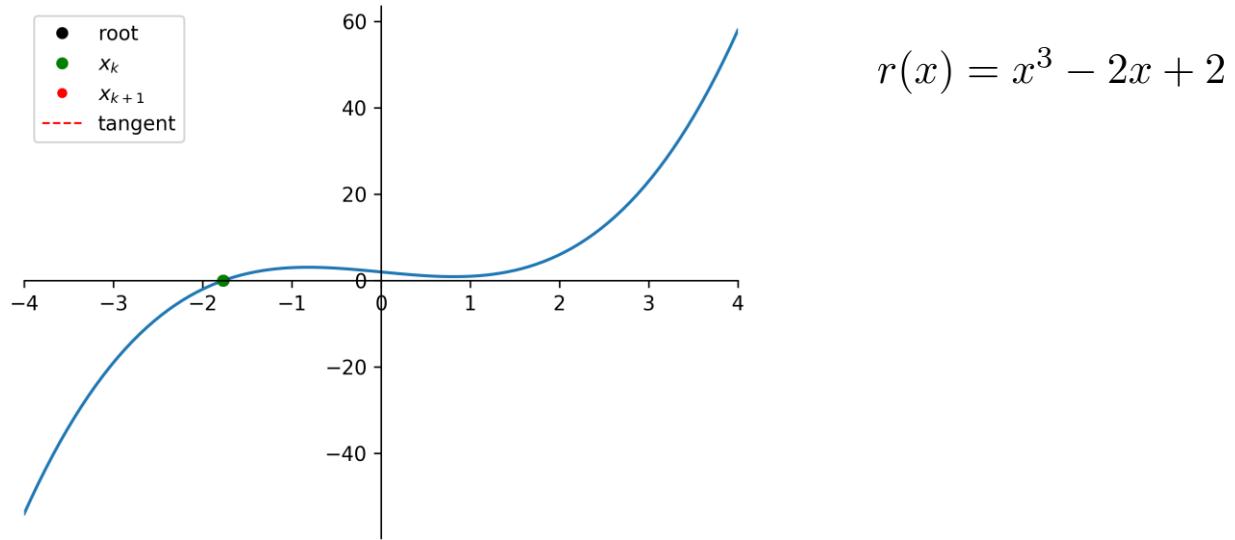
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

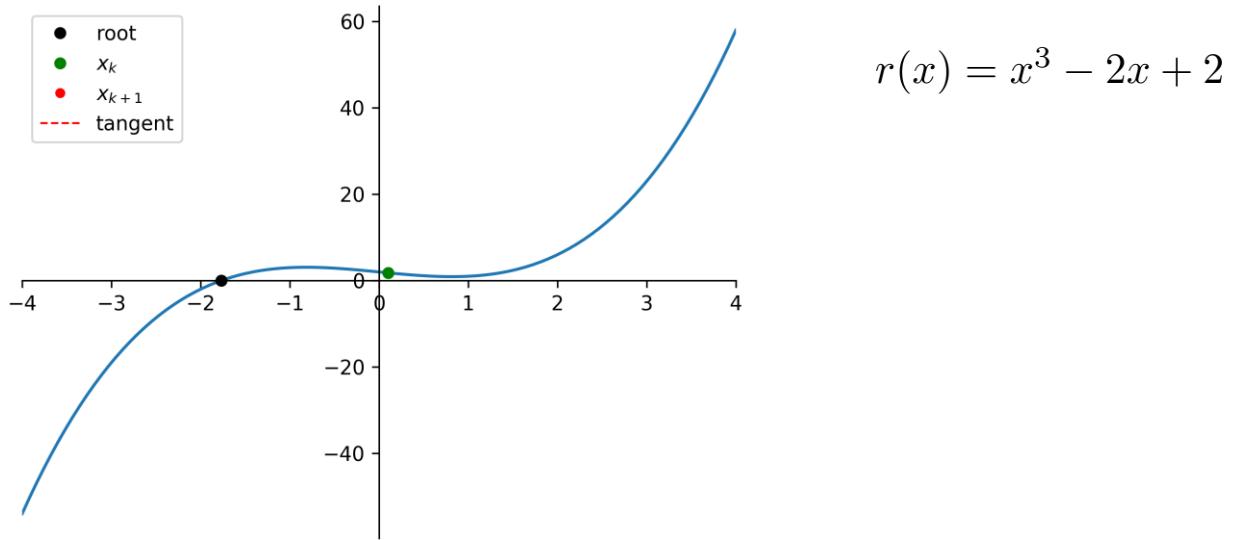
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

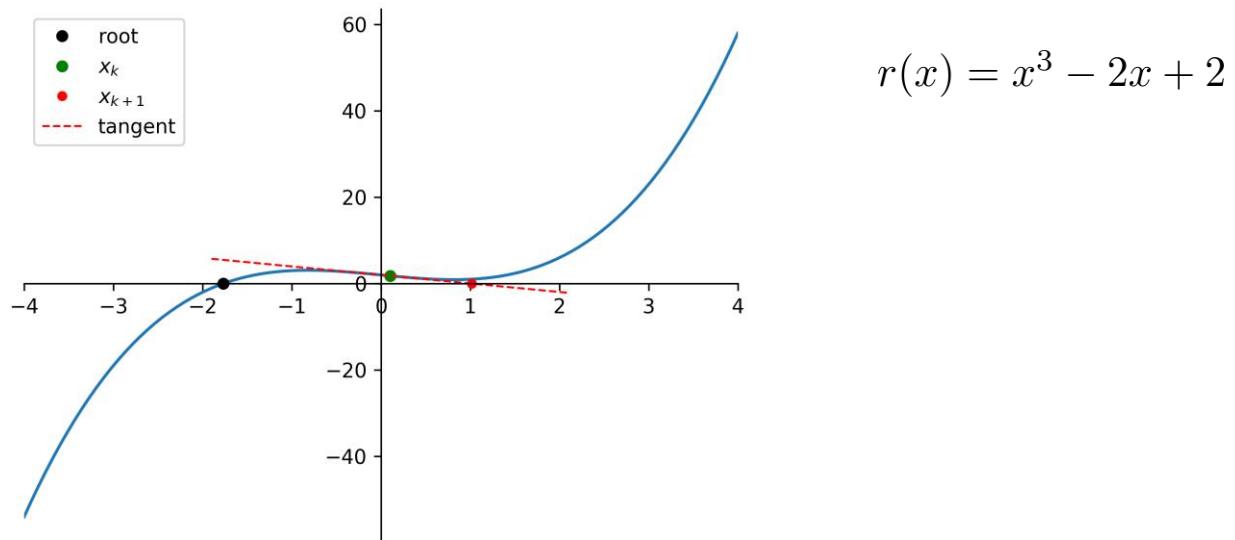
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

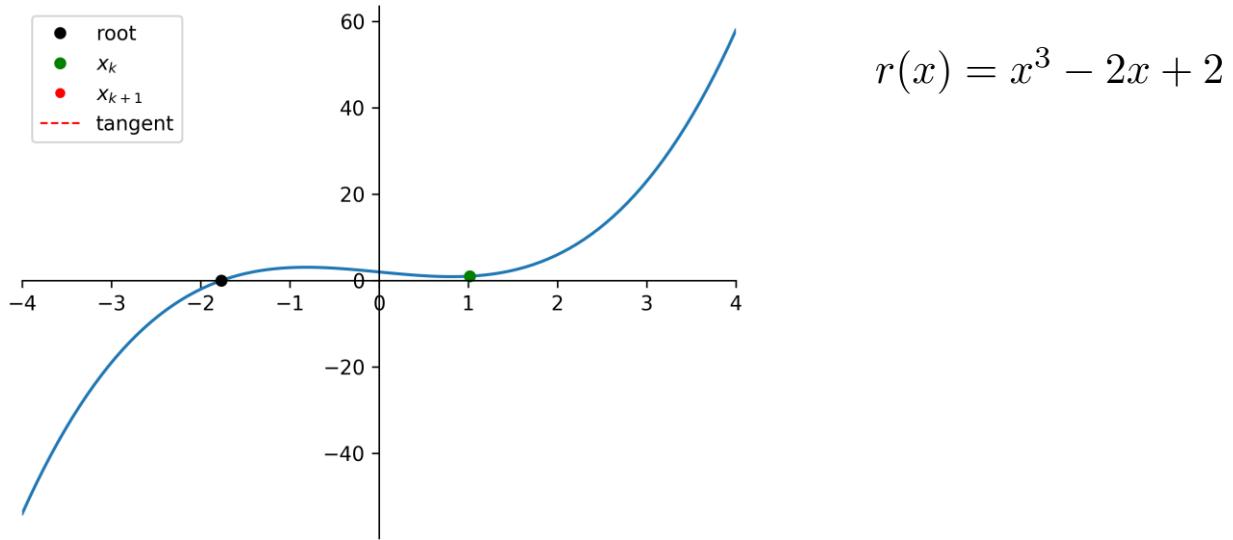
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

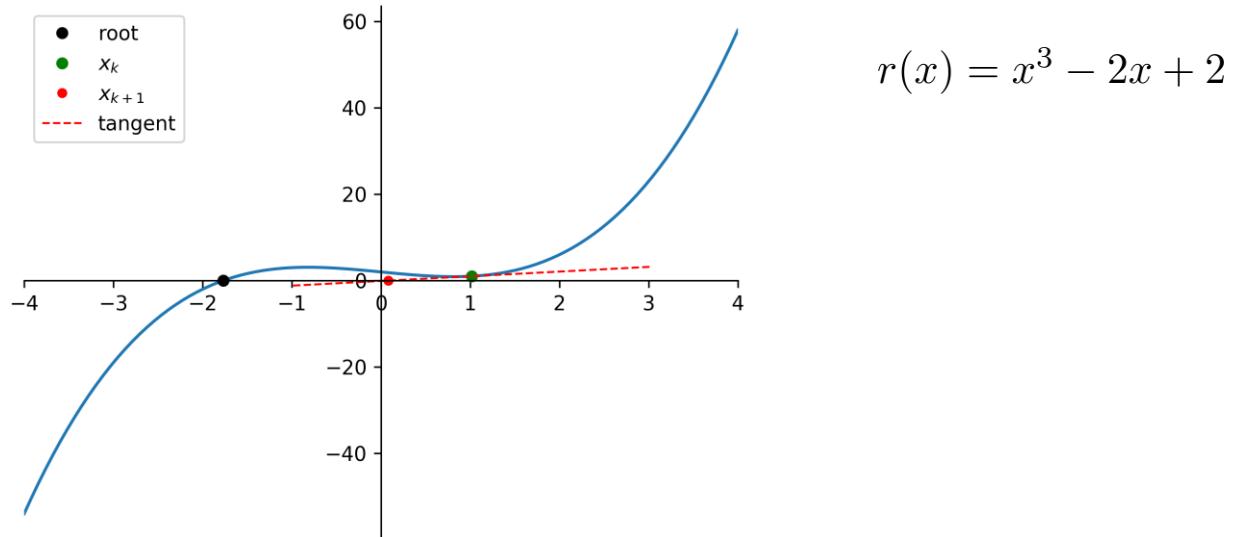
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

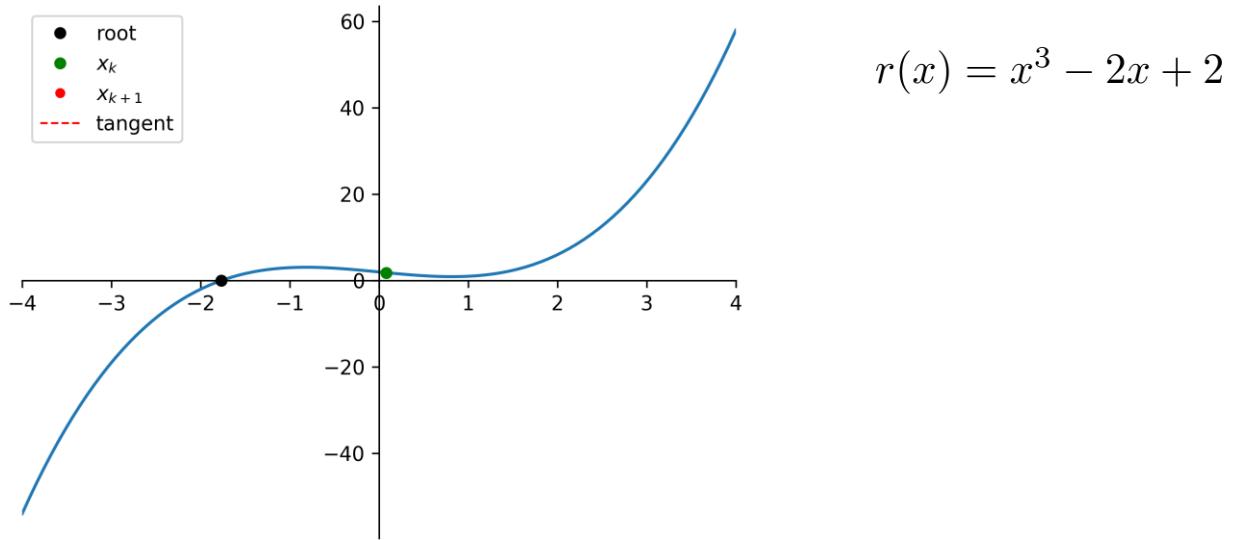
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

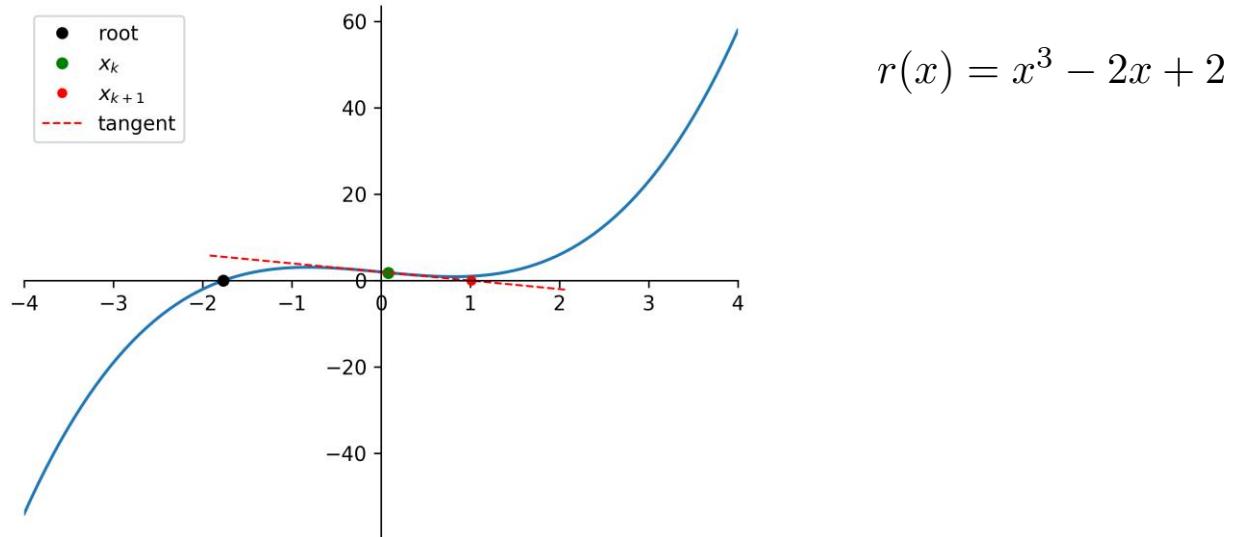
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

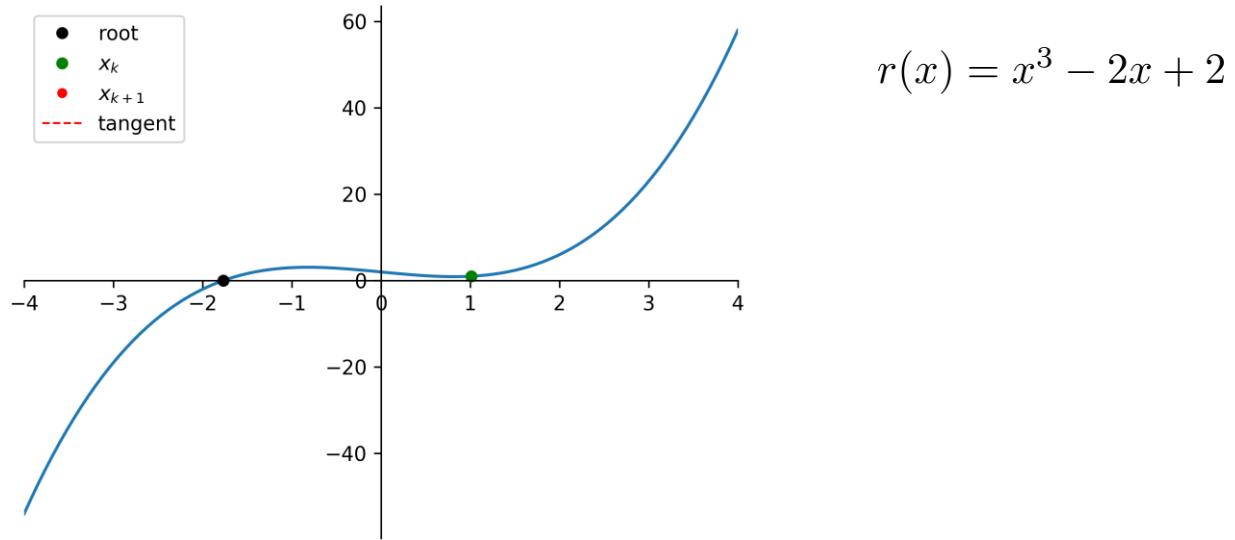
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

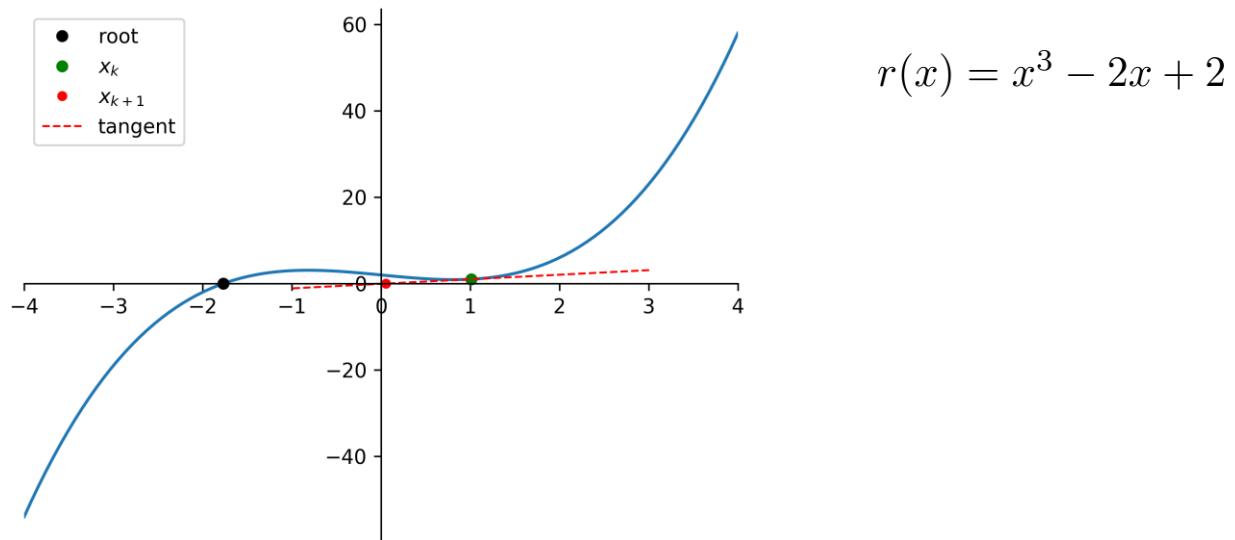
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

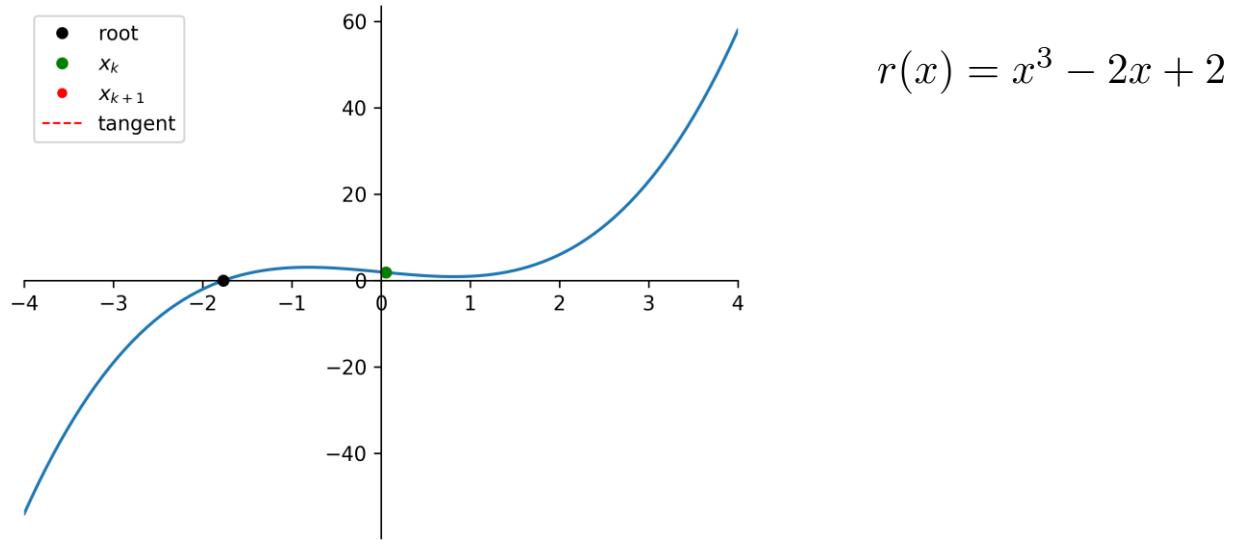
Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's Method (aka Newton-Raphson method)

$$x_{k+1} = x_k + p_k, \quad p_k = -J(x_k)^{-1}r(x_k)$$

Scalar case: $x_{k+1} = x_k - \frac{r(x_k)}{r'(x_k)}$



Newton's method for nonlinear equations (Alg. 11.1)

Convergence of Newton's method

Practical issues with Newton's method: Jacobian

Practical issues with Newton's method: Merit function



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 18

Sequential Quadratic Programming (SQP)

Lecturer: Lars Imsland

Outline

- Recap: Newton's method for solving nonlinear equations
- Recap: Equality-constrained QPs
- SQP for *equality-constrained* nonlinear programming problems
 - Next time: SQP for general

Reference: N&W Ch.18-18.1

Types of Constrained Optimization Problems

- Linear programming
 - Convex problem
 - Feasible set polyhedron
- Quadratic programming
 - Convex problem if $P \geq 0$
 - Feasible set polyhedron
- Nonlinear programming
 - In general non-convex!

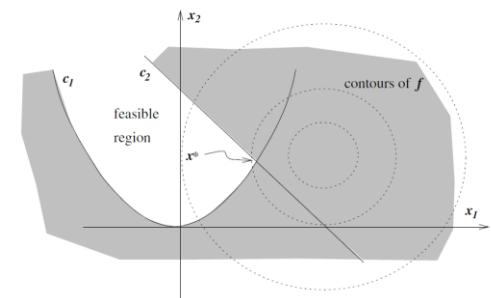
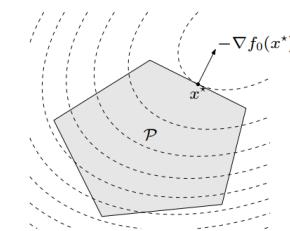
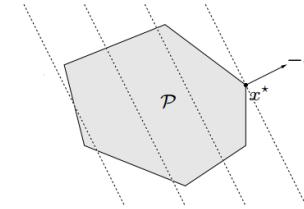
$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\begin{aligned} & \min c^\top x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

$$\begin{aligned} & \min \frac{1}{2} x^\top Px + q^\top x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

$$\begin{aligned} & \min f(x) \\ \text{subject to } & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$

$$\begin{aligned} & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

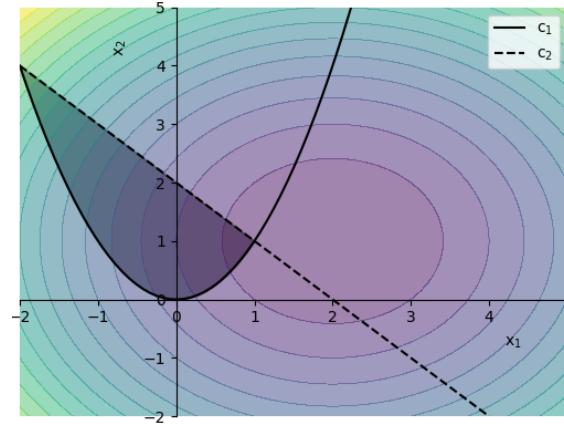
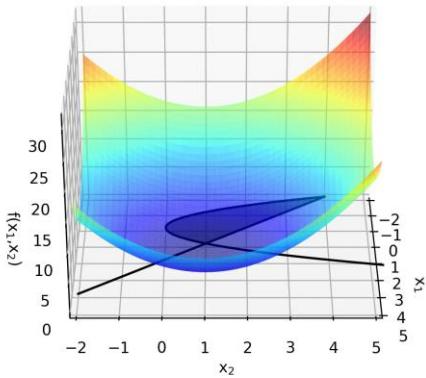


General Optimization Problem

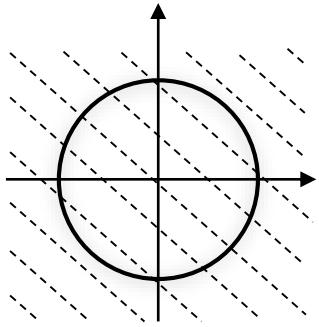
$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned}$$

- Example:

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \quad \text{subject to} \quad \begin{aligned} x_1^2 - x_2 &\leq 0, \\ x_1 + x_2 &\leq 2. \end{aligned}$$



Today: Only equality constraints



The Lagrangian

For constrained optimization problems, introduce modification of objective function:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

- Multipliers for *equality* constraints may have both signs in a solution
- Multipliers for *inequality* constraints cannot be negative (cf. shadow prices)
- For (inequality) constraints that are *inactive*, multipliers are zero

KKT conditions (Theorem 12.1)

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

KKT conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

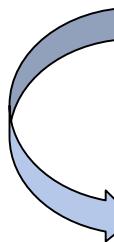
$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

}

(primal feasibility)

(dual feasibility)

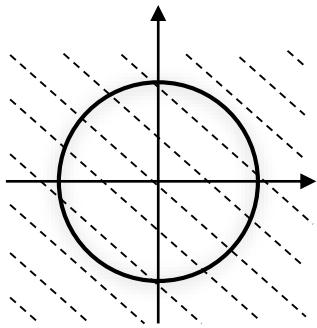
(complementarity condition/
complementary slackness)



Either $\lambda_i^* = 0$ or $c_i(x^*) = 0$

Example KKT system

$$\min_{x \in \mathbb{R}^2} -x_1 - x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 1 = 0$$



Today: Equality-constrained NLP

Newton's method for solving nonlinear equations (Ch. 11)

- Solve equation system $r(x) = 0$, $r(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n$
- Assume Jacobian $J(x) \in \mathbb{R}^{n \times n}$ exists and is continuous
- Taylor: $r(x + p) = r(x) + J(x)p + O(\|p\|^2)$

$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Algorithm 11.1 (Newton's Method for Nonlinear Equations).

Choose x_0 ;

for $k = 0, 1, 2, \dots$

 Calculate a solution p_k to the Newton equations

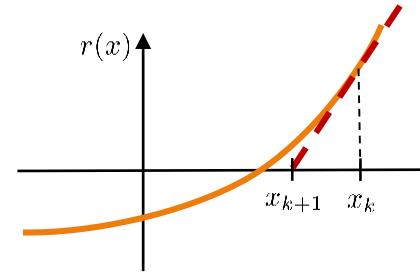
$$J(x_k)p_k = -r(x_k);$$

$$x_{k+1} \leftarrow x_k + p_k;$$

end (for)

- Convergence rate (Thm 11.2): **Quadratic convergence** if $J(x)$ is invertible
(**quadratic convergence is very good**, but only holds close to the solution)
- If we set $r(x) = \nabla f(x)$, then this method corresponds to Newton's method for minimizing $f(x)$

$$p_k = -J(x_k)^{-1}r(x_k) \quad \longleftrightarrow \quad p_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$$



Newton's method to solve $\mathbf{F}(\mathbf{x}, \lambda) = \mathbf{0}$

$$F(x, \lambda) = \begin{pmatrix} \nabla f(x) - A^\top(x)\lambda \\ c(x) \end{pmatrix}$$

Equality-constrained QP (EQP)

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2} x^\top G x + c^\top x \\ \text{subject to} \quad & Ax = b, \quad A \in \mathbb{R}^{m \times n} \end{aligned}$$

Basic assumption:
A full row rank

- KKT-conditions (KKT system, KKT matrix):

$$\begin{pmatrix} G & -A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -c \\ b \end{pmatrix} \quad \text{or, if we let } x^* = x + p, \quad \begin{pmatrix} G & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} -p \\ \lambda^* \end{pmatrix} = \begin{pmatrix} c + Gx \\ Ax - b \end{pmatrix}$$

- Solvable when $Z^\top G Z > 0$ (columns of Z basis for nullspace of A)
- That is: QP with only equality constraints is solved by solving a set of linear equations (*linear system*)

Alternative “derivation” of KKT-system

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad c(x) = 0$$

Local SQP-algorithm for solving equality-constrained NLPs

$$\begin{aligned} & \min f(x) \\ & \text{subject to } c(x) = 0 \end{aligned}$$

Algorithm 18.1 (Local SQP Algorithm for solving (18.1)).

Choose an initial pair (x_0, λ_0) ; set $k \leftarrow 0$;

repeat until a convergence test is satisfied

Evaluate f_k , ∇f_k , $\nabla_{xx}^2 \mathcal{L}_k$, c_k , and A_k ;

Solve (18.7) to obtain p_k and l_k ;

Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow l_k$;

end (repeat)

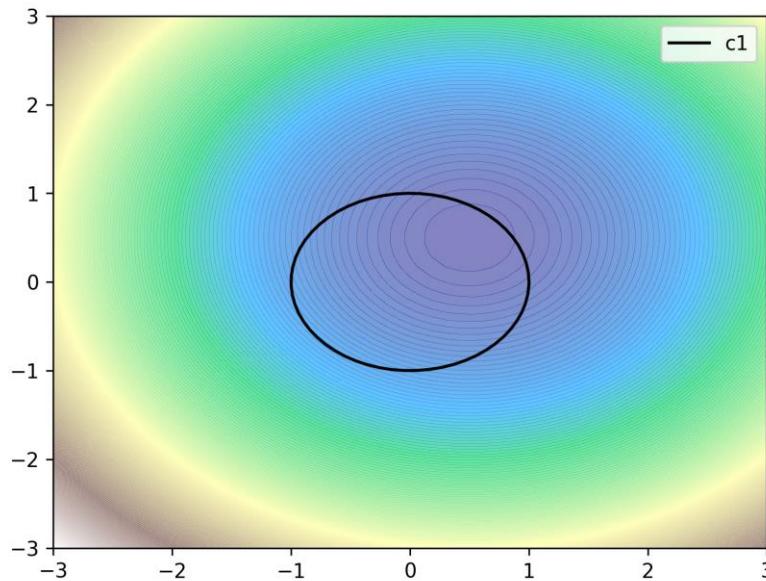
EQP:

$$\begin{aligned} & \min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ & \text{subject to} \quad A_k p + c_k = 0. \end{aligned}$$

Local SQP

$$f(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2$$

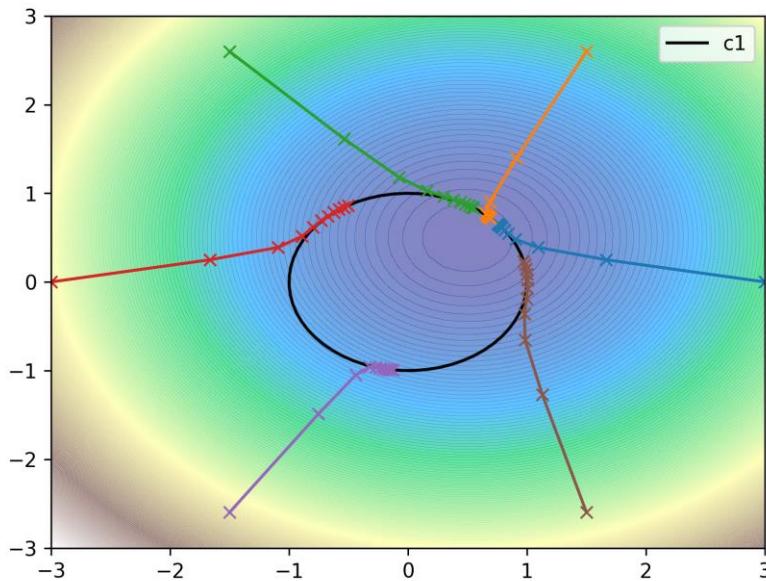
$$c_1(x) = x_1^2 + x_2^2 - 1 = 0$$



Local SQP

$$f(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2$$

$$c_1(x) = x_1^2 + x_2^2 - 1 = 0$$



$$\begin{aligned} \min_p \quad & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to} \quad & A_k p + c_k = 0. \end{aligned}$$



NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 19

Practical SQP algorithms for nonlinear programming

Lecturer: Lars Imsland

Outline

- Recap: Local SQP algorithms for equality-constrained NLPs
 - Extension to inequalities
- Globalization («make it work when starting far from optimum»):
 - Computation/approximation of the Hessian
 - Linesearch
- Other issues
 - The Maratos effect
 - Infeasible linearized constraints

Reference: N&W Ch. 18.2, 18.3, 15.4

Newton's method for solving nonlinear equations (Ch. 11)

- Solve equation system $r(x) = 0$, $r(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n$
- Assume Jacobian $J(x) \in \mathbb{R}^{n \times n}$ exists and is continuous
- Taylor: $r(x + p) = r(x) + J(x)p + O(\|p\|^2)$

$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Algorithm 11.1 (Newton's Method for Nonlinear Equations).

Choose x_0 ;

for $k = 0, 1, 2, \dots$

 Calculate a solution p_k to the Newton equations

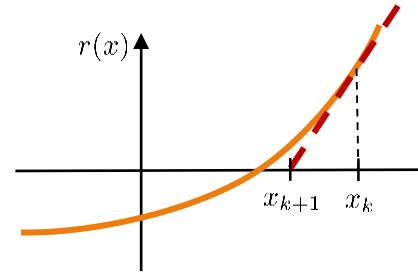
$$J(x_k)p_k = -r(x_k);$$

$$x_{k+1} \leftarrow x_k + p_k;$$

end (for)

- Convergence rate (Thm 11.2): **Quadratic convergence** if $J(x)$ is invertible
(quadratic convergence is very good, but only holds close to the solution)
- If we set $r(x) = \nabla f(x)$, then this method corresponds to Newton's method for minimizing $f(x)$

$$p_k = -J(x_k)^{-1}r(x_k) \quad \longleftrightarrow \quad p_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$$



Equality-constrained NLPs – Newton

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \lambda^\top c(x)$
- KKT conditions: $F(x, \lambda) = \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ c(x) \end{pmatrix} = 0$
- To solve: Use Newton's method for nonlinear equations on KKT conditions:

$$\begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} + \begin{pmatrix} p_k \\ p_{\lambda_k} \end{pmatrix} \quad \text{where} \quad \underbrace{\begin{pmatrix} \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k) & -A^\top(x_k) \\ A(x_k) & 0 \end{pmatrix}}_{\text{Jacobian of } F(x, \lambda) \text{ at } (x_k, \lambda_k)} \begin{pmatrix} p_k \\ p_{\lambda_k} \end{pmatrix} = \underbrace{\begin{pmatrix} -\nabla f(x_k) + A^\top(x_k)\lambda_k \\ -c(x_k) \end{pmatrix}}_{-F(x_k, \lambda_k)}$$

$A(x)^\top = (\nabla c_1(x), \dots, \nabla c_m(x))$

- Very efficient method for solving equality-constrained NLPs locally

Equality-constrained NLP – QP

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Consider now this quadratic approximation to the objective (or Lagrangian):

$$\min_{p \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^\top p + \frac{1}{2} p^\top \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k) p \quad \text{subject to} \quad c(x_k) + A(x_k)^\top p = 0$$

- KKT conditions:

$$\begin{pmatrix} \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k) & -A^\top(x_k) \\ A(x_k) & 0 \end{pmatrix} \begin{pmatrix} p_k \\ l_k \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -c(x_k) \end{pmatrix}$$

- If we let $l_k = p_{\lambda_k} + \lambda_k = \lambda_{k+1}$, it is easy to show that the two KKT systems give equivalent solutions
 - Newton-viewpoint: **quadratic convergence locally**
 - QP-viewpoint: practical implementation and **extension to inequality constraints**
- Assumptions for the above:
 - $A(x_k)$ full row rank (**LICQ**),
 - $d^\top \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k) d > 0$ for all $d \neq 0$ s.t. $A(x_k)d = 0$ (“**pos.def. on tangent space of constraints**”)

Local SQP-algorithm for solving NLPs

Only equality constraints:

$$\begin{aligned} \min f(x) \\ \text{subject to } c(x) = 0 \end{aligned}$$

$$\begin{aligned} \min_p & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to } & A_k p + c_k = 0. \end{aligned}$$

Algorithm 18.1 (Local SQP Algorithm for solving (18.1)).

Choose an initial pair (x_0, λ_0) ; set $k \leftarrow 0$;

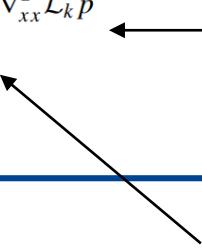
repeat until a convergence test is satisfied

Evaluate $f_k, \nabla f_k, \nabla_{xx}^2 \mathcal{L}_k, c_k$, and A_k ;

Solve (18.7) to obtain p_k and λ_k ;

Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow \lambda_k$;

end (repeat)



With inequality constraints (IQP method):

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

$$\begin{aligned} \min_p & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to } & \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \\ & \nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

Thm 18.1: Alg. 18.1 identifies (eventually) the optimal active set of constraints (under assumptions). After, it behaves like Newton's method for equality constrained problems.

Alternatively (EQP method): Maintain a “working set” (approximation of the active set) in Alg. 18.1, solve equality-constrained QP in each iteration. May be more efficient for large-scale problems.

Globalization: Newton unconstrained optimization

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

- Quadratic approximation:
- Valid direction:
- How to ensure valid direction:
- Line search:

Globalization: SQP for constrained optimization

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

- Quadratic approximation:
- Valid solution:
- How to ensure valid solution:
- Line search:

Quasi-Newton for unconstrained problems

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

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,

inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k;$$

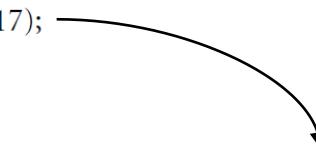
 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
 procedure to satisfy the Wolfe conditions (3.6);

 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)



$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Quasi-Newton for SQP

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.11a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \quad (18.11c)$$

- SQP needs Hessian of Lagrangian, but this require second derivatives of objective and constraints, which may be expensive
- Quasi-Newton (BFGS) very successful for unconstrained optimization – can we do the same in the constrained case?

Unconstrained case:

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \quad y_k = \nabla f_{k+1} - \nabla f_k, \quad (6.5)$$

$$(BFGS) \quad H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad (6.17)$$

$$H_k \approx [\nabla^2 f(x_k)]^{-1}$$

Constrained case:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \quad (18.13)$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}. \quad (18.16)$$

$$B_k \approx \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$$

- Possible problem: BFGS gives positive definite Hessian approximation, while Hessian of Lagrangian is not necessarily positive definite (not even close to a solution). That is, the approximation may be bad.
- Possible solution: Approximate “reduced Hessian” (Hessian on nullspace of constraints) instead. This reduced Hessian is much more likely to be positive definite (recall sufficient conditions).

J, merit function

Descent direction of merit function

Line search on merit function

Line search – Merit function

“Globalization”

- How far to walk along p ? Linesearch (or trust region)!
- Unconstrained optimization: The Armijo (Wolfe) condition ensure sufficient decrease of objective function:

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k, \quad (3.4)$$

- Constrained optimization: Must check both objective and constraint!
- Merit function (for line search): Function that measure progress in both:

I_1 merit function: $\phi_1(x; \mu) = f(x) + \mu \sum_{i \in \mathcal{E}} |c_i(x)| + \mu \sum_{i \in \mathcal{I}} [c_i(x)]^-$, $\mu^* = \max\{|\lambda_i^*|, i \in \mathcal{E} \cup \mathcal{I}\}$

Definition 15.1 (Exact Merit Function).

A merit function $\phi(x; \mu)$ is exact if there is a positive scalar μ^* such that for any $\mu > \mu^*$, any local solution of the nonlinear programming problem (15.1) is a local minimizer of $\phi(x; \mu)$.

- Thm 18.2: $D(\phi_1(x_k; \mu); p_k) \leq -p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k - (\mu - \|\lambda_{k+1}\|_\infty) \|c_k\|_1$
 - That is: p_k is a descent direction for merit function if Hessian of Lagrangian is positive definite and μ is large enough

A practical line search SQP method

Algorithm 18.3 (Line Search SQP Algorithm).

Choose parameters $\eta \in (0, 0.5)$, $\tau \in (0, 1)$, and an initial pair (x_0, λ_0) ;

Evaluate $f_0, \nabla f_0, c_0, A_0$;

If a quasi-Newton approximation is used, choose an initial $n \times n$ symmetric positive definite Hessian approximation B_0 , otherwise compute $\nabla_{xx}^2 \mathcal{L}_0$;

repeat until a convergence test is satisfied

 Compute p_k by solving (18.11); let $\hat{\lambda}$ be the corresponding multiplier;

 Set $p_\lambda \leftarrow \hat{\lambda} - \lambda_k$;

 Choose μ_k to satisfy (18.36) with $\sigma = 1$;

 Set $\alpha_k \leftarrow 1$;

while $\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \eta \alpha_k D_1(\phi(x_k; \mu_k)) p_k$

 Reset $\alpha_k \leftarrow \tau_\alpha \alpha_k$ for some $\tau_\alpha \in (0, \tau]$;

end (while)

 Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k p_\lambda$;

 Evaluate $f_{k+1}, \nabla f_{k+1}, c_{k+1}, A_{k+1}$, (and possibly $\nabla_{xx}^2 \mathcal{L}_{k+1}$);

 If a quasi-Newton approximation is used, set

$s_k \leftarrow \alpha_k p_k$ and $y_k \leftarrow \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$,

 and obtain B_{k+1} by updating B_k using a quasi-Newton formula;

end (repeat)

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.11a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \quad (18.11c)$$

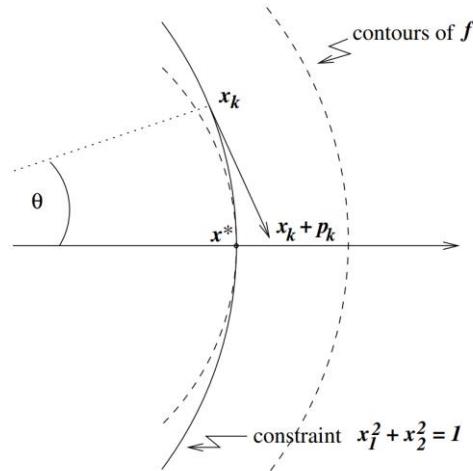
$$\mu \geq \frac{\nabla f_k^T p_k + (\sigma/2) p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k}{(1-\rho) \|c_k\|_1}. \quad (18.36)$$

(choose μ_k such that p_k is a descent direction for $\phi_1(x_k; \mu_k)$)

Maratos effect

- Maratos effect: A merit function may reject good steps:

$$\min f(x_1, x_2) = 2(x_1^2 + x_2^2 - 1) - x_1, \quad \text{subject to} \quad x_1^2 + x_2^2 - 1 = 0. \quad (15.34)$$



p_k good step even if both objective and constraint violation increase!

- Remedy:
 - Use a merit function that does not suffer from the Maratos effect
 - Use “non-monotone” strategy (temporarily allow increase in merit function)
 - Use “second-order correction” (when Maratos effect occurs)

Inconsistent linearizations

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

In each SQP iteration, solve:

$$\min_p \quad f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.11a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \quad (18.11c)$$

NLP software

- SNOPT
 - “solves large-scale linear and nonlinear problems; especially recommended if some of the constraints are highly nonlinear, or constraints respectively their gradients are costly to evaluate and second derivative information is unavailable or hard to obtain; assumes that the number of “free” variables is modest.”
 - Licence: Commercial
- IPOPT
 - “interior point method for large-scale NLPs”
 - License: Open source
- WORHP
 - SQP solver for very large problems, IP at QP level, exact or approximate second derivatives, various linear algebra options, varius interfaces
 - Licence: Commercial, but free for academia
- KNITRO
 - trust region interior point method, efficient for NLPs of all sizes, various interfaces
 - License: Commercial
- (...and several others, including `fmincon` in Matlab Optimization Toolbox)
- «Decision tree for optimization software»: <http://plato.asu.edu/sub/nlores.html>

Example: optimization using CasADI

- CasADI (<https://casadi.org/>)
 - “CasADI is a symbolic framework for numeric optimization implementing automatic differentiation in forward and reverse modes on sparse matrix-valued computational graphs.”

$$\begin{aligned} \min_{x,y,z} \quad & x^2 + 100z^2 \\ \text{s.t. } & z + (1-x)^2 - y = 0 \end{aligned}$$

Define variables

Define objective and constraints

Create solver object

Solve the opt problem

```
rosenbrock.m

import casadi.*

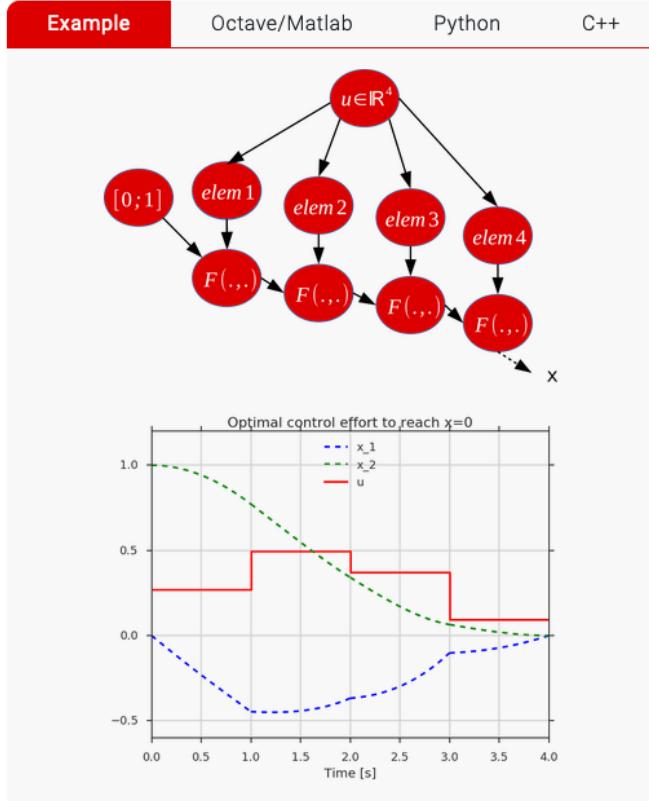
% Create NLP: Solve the Rosenbrock problem:
%   minimize    x^2 + 100*z^2
%   subject to  z + (1-x)^2 - y == 0
x = SX.sym('x');
y = SX.sym('y');
z = SX.sym('z');
v = [x;y;z];
f = x^2 + 100*z^2;
g = z + (1-x)^2 - y;
nlp = struct('x', v, 'f', f, 'g', g);

% Create IPOPT solver object
solver = nlpsol('solver', 'ipopt', nlp);

% Solve the NLP
res = solver('x0', [2.5 3.0 0.75],... % solution guess
             'lbx', -inf,... % lower bound on x
             'ubx', inf,... % upper bound on x
             'lbg', 0,... % lower bound on g
             'ubg', 0); % upper bound on g

% Print the solution
f_opt = full(res.f)           % >> 0
x_opt = full(res.x)           % >> [0; 1; 0]
lam_x_opt = full(res.lam_x)   % >> [0; 0; 0]
lam_g_opt = full(res.lam_g)   % >> 0
```

Example from CasADI



Example Octave/Matlab **Python** C++

```
from casadi import *

x = MX.sym('x',2); # Two states
p = MX.sym('p'); # Free parameter

# Expression for ODE right-hand side
z = 1-x[1]**2;
rhs = vertcat(z*x[0]-x[1]+2*tanh(p),x[0])

# ODE declaration with free parameter
ode = {'x':x,'p':p,'ode':rhs}

# Construct a Function that integrates over 1s
F = integrator('F','cvodes',ode,{tf:1})

# Control vector
u = MX.sym('u',4,1)

x = [0,1] # Initial state
for k in range(4):
    # Integrate 1s forward in time:
    # call integrator symbolically
    res = F(x0=x,p=u[k])
    x = res["xf"]

# NLP declaration
nlp = {'x':u,'f':dot(u,u),'g':x};

# Solve using IPOPT
solver = nlpSolver('solver','ipopt',nlp)
res = solver(x0=0.2,lbx=0,ubx=0)

plot(res["x"])
```




NTNU | Norwegian University of
Science and Technology

TTK4135 – Lecture 20

Summing up / Nonlinear MPC / Control applications

Lecturer: Lars Imsland

Outline

- Summing up optimization
- Today: Control and (nonlinear) optimization
 - Nonlinear MPC, some “practical”/industrial issues on MPC
 - Reference: F&H 4.5, 4.6
- Some examples from literature using concepts from this course

Numerical optimization in a nutshell (in this course)

- Unconstrained optimization
 - Search directions: Steepest descent, Newton, Quasi-Newton
 - Globalization: Line-search and (possibly) Hessian modification (**Alg. 6.1**)
- Constrained optimization
 - Optimality conditions, KKT
 - Linear programming: Standard form, KKT conditions, BFPs, SIMPLEX method (**Proc. 13.1**)
 - Quadratic programming: EQP/QP, KKT system, Active set method (**Alg. 16.3**)
 - Nonlinear programming: Nonlinear equations (**Alg. 11.1**), SQP-method (**Alg. 18.3**)

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

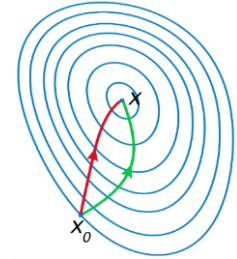
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



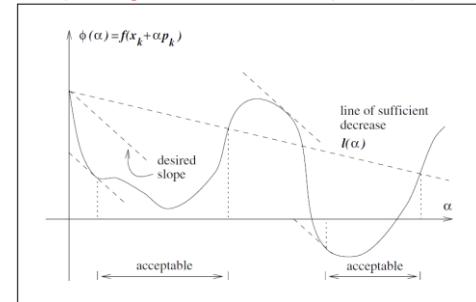
How to calculate derivatives – Ch. 8

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**.
Newton's method uses curvature information to take a more direct route. ([wikipedia.org](https://en.wikipedia.org))

Step length line search (Wolfe):



Quasi-Newton: BFGS method

$$\begin{aligned} p_k &= -B_k^{-1} \nabla f(x_k) \\ B_k &\approx \nabla^2 f(x_k) \\ H_k &= B_k^{-1} \end{aligned}$$

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,

inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k;$$

 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
 procedure to satisfy the Wolfe conditions (3.6);

 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)

We use only gradient!

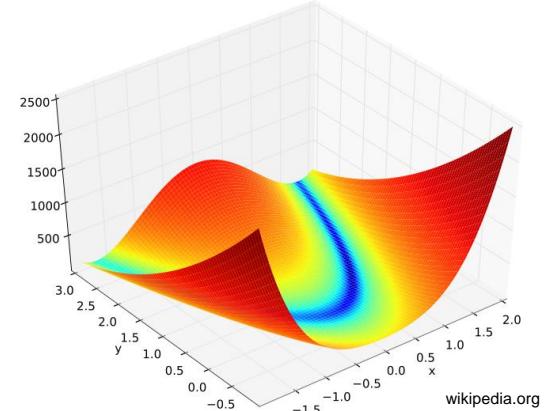
$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Example (from book)

- Using steepest descent, BFGS and inexact Newton on Rosenbrock function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- Iterations from starting point (-1.2,1):
 - Steepest descent: 5264
 - BFGS: 34
 - Newton: 21
- Last iterations; value of $\|x_k - x^*\|$



wikipedia.org

steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

Types of Constrained Optimization Problems

- Linear programming
 - Convex problem
 - Feasible set polyhedron
- Quadratic programming
 - Convex problem if $P \geq 0$
 - Feasible set polyhedron
- Nonlinear programming
 - In general non-convex!

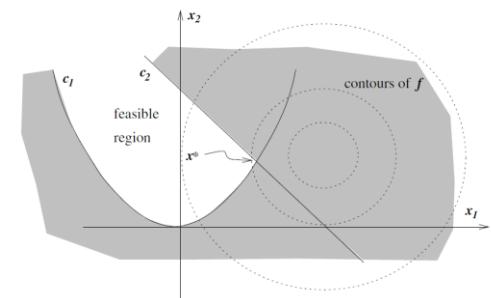
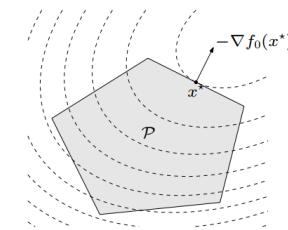
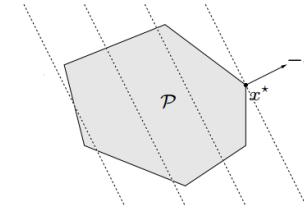
$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\begin{aligned} & \min c^\top x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

$$\begin{aligned} & \min \frac{1}{2} x^\top Px + q^\top x \\ \text{subject to } & Ax \leq b \\ & Cx = d \end{aligned}$$

$$\begin{aligned} & \min f(x) \\ \text{subject to } & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$

$$\begin{aligned} & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$



KKT conditions (Theorem 12.1)

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, & i \in \mathcal{E}, \\ c_i(x) &\geq 0, & i \in \mathcal{I}. \end{aligned}$$

KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, && \text{(stationarity)} \\ \begin{aligned} c_i(x^*) &= 0, & \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, & \forall i \in \mathcal{I}, \end{aligned} && \left. \right\} \text{(primal feasibility)} \\ \lambda_i^* &\geq 0, & \forall i \in \mathcal{I}, & \text{(dual feasibility)} \\ \lambda_i^* c_i(x^*) &= 0, & \forall i \in \mathcal{E} \cup \mathcal{I}. & \text{(complementarity condition/} \\ &&& \text{complementary slackness)} \end{aligned}$$

Starting point for all algorithms for constrained optimization in this course!

Linear programming, standard form and KKT

LP: $\min_{x \in \mathbb{R}^n} c^T x$ subject to $\begin{cases} a_i x = b_i, & i \in \mathcal{E} \\ a_i x \geq b_i, & i \in \mathcal{I} \end{cases}$

LP, standard form: $\min_{x \in \mathbb{R}^n} c^T x$ subject to $\begin{cases} Ax = b \\ x \geq 0 \end{cases}$

Lagrangian: $\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x$

KKT-conditions (LPs: necessary *and* sufficient for optimality):

$$A^T \lambda^* + s^* = c,$$

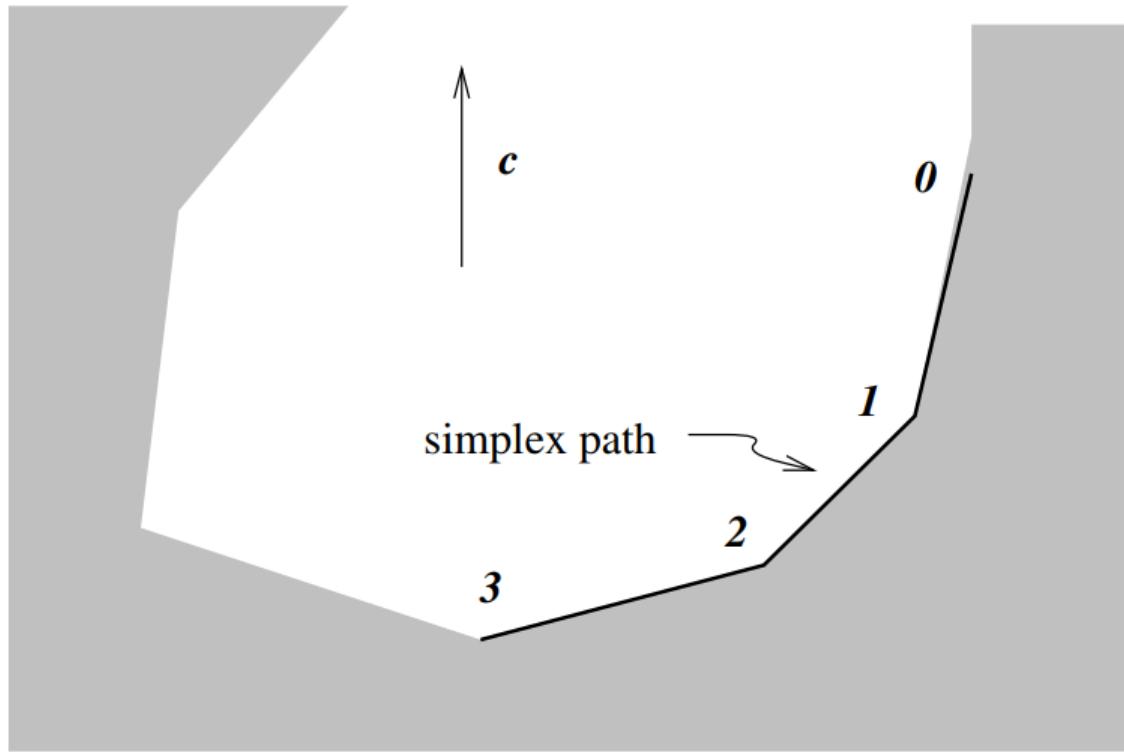
$$Ax^* = b,$$

$$x^* \geq 0,$$

$$s^* \geq 0,$$

$$x_i^* s_i^* = 0, \quad i = 1, 2, \dots, n$$

Simplex method: BFP and KKT



General QP problem

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^\top G x + x^\top c \\ \text{s.t. } & a_i^\top x = b_i, \quad i \in \mathcal{E} \\ & a_i^\top x \geq b_i, \quad i \in \mathcal{I} \end{aligned}$$

- Lagrangian

$$\mathcal{L}(x^*, \lambda^*) = \frac{1}{2} x^* \top G x + x^* \top c - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i (a_i^\top x^* - b_i)$$

- KKT conditions

General:

$$\begin{aligned} Gx^* + c - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* a_i &= 0 \\ a_i^\top x^* &= b_i, \quad i \in \mathcal{E} \\ a_i^\top x^* &\geq b_i, \quad i \in \mathcal{I} \\ \lambda_i^* &\geq 0, \quad i \in \mathcal{I} \\ \lambda_i^* (a_i^\top x^* - b_i) &= 0, \quad i \in \mathcal{E} \cup \mathcal{I} \end{aligned}$$

Defined via active set:

$$\begin{aligned} \mathcal{A}(x^*) &= \mathcal{E} \cup \left\{ i \in \mathcal{I} \mid a_i^\top x^* = b_i \right\} \\ Gx^* + c - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* a_i &= 0 \\ a_i^\top x^* &= b_i, \quad i \in \mathcal{A}(x^*) \\ a_i^\top x^* &\geq b_i, \quad i \in \mathcal{I} \setminus \mathcal{A}(x^*) \\ \lambda_i^* &\geq 0, \quad i \in \mathcal{A}(x^*) \cap \mathcal{I} \end{aligned}$$

Active set method for convex QP

Algorithm 16.3 (Active-Set Method for Convex QP).

Compute a feasible starting point x_0 ;

Set \mathcal{W}_0 to be a subset of the active constraints at x_0 ;

for $k = 0, 1, 2, \dots$

 Solve (16.39) to find p_k ;

if $p_k = 0$

 Compute Lagrange multipliers $\hat{\lambda}_i$ that satisfy (16.42),
 with $\hat{\mathcal{W}} = \mathcal{W}_k$;

if $\hat{\lambda}_i \geq 0$ for all $i \in \mathcal{W}_k \cap \mathcal{I}$

stop with solution $x^* = x_k$;

else

$j \leftarrow \arg \min_{j \in \mathcal{W}_k \cap \mathcal{I}} \hat{\lambda}_j$;

$x_{k+1} \leftarrow x_k$; $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$;

else (* $p_k \neq 0$ *)

 Compute α_k from (16.41);

$x_{k+1} \leftarrow x_k + \alpha_k p_k$;

if there are blocking constraints

 Obtain \mathcal{W}_{k+1} by adding one of the blocking
 constraints to \mathcal{W}_k ;

else

$\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$;

end (for)

$$\min_p \quad \frac{1}{2} p^T G p + g_k^T p \quad (16.39a)$$

$$\text{subject to} \quad a_i^T p = 0, \quad i \in \mathcal{W}_k. \quad (16.39b)$$

$$\sum_{i \in \hat{\mathcal{W}}} a_i \hat{\lambda}_i = g = G \hat{x} + c, \quad (16.42)$$

$$\alpha_k \stackrel{\text{def}}{=} \min \left(1, \min_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k} \right). \quad (16.41)$$

No degeneracy and $G > 0$: Active set method converges
in finite number of iterations.

Example 16.4

$$\min_x q(x) = (x_1 - 1)^2 + (x_2 - 2.5)^2$$

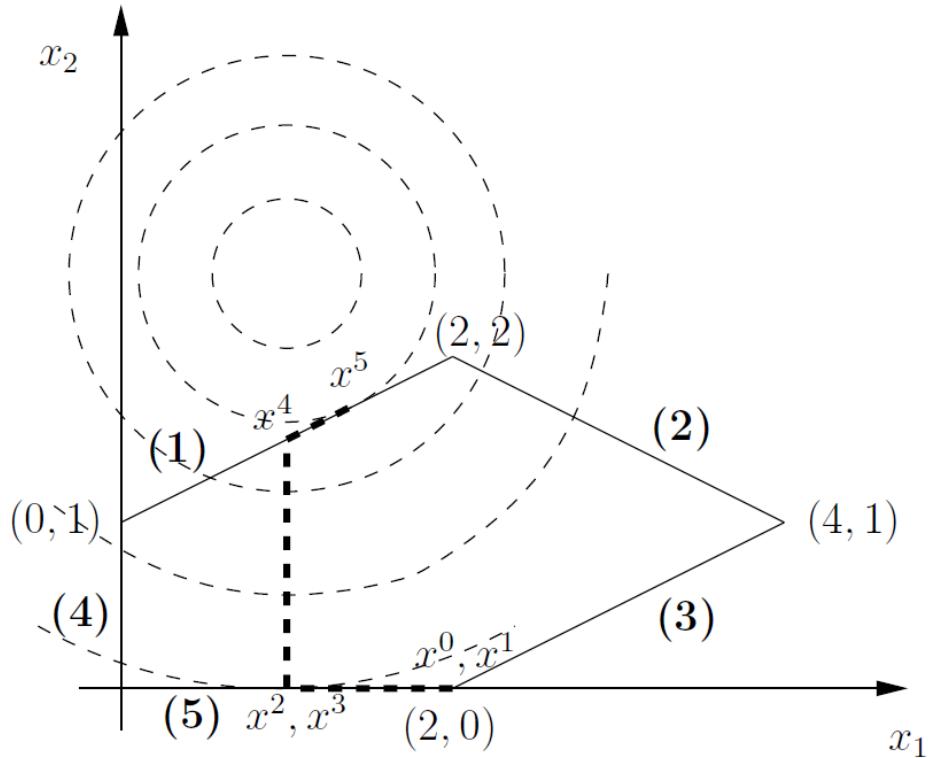
$$\text{subject to} \quad x_1 - 2x_2 + 2 \geq 0 \quad (1)$$

$$-x_1 - 2x_2 + 6 \geq 0 \quad (2)$$

$$-x_1 + 2x_2 + 2 \geq 0 \quad (3)$$

$$x_1 \geq 0 \quad (4)$$

$$x_2 \geq 0 \quad (5)$$



$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -2 \\ -5 \end{bmatrix}$$

$$a_1 = [1 \quad -2]^T, \quad b_1 = -2$$

$$a_2 = [-1 \quad -2]^T, \quad b_2 = -6$$

$$a_3 = [-1 \quad 2]^T, \quad b_3 = -2$$

$$a_4 = [1 \quad 0]^T, \quad b_4 = 0$$

$$a_5 = [0 \quad 1]^T, \quad b_5 = 0$$

Local SQP-algorithm for solving NLPs

Only equality constraints:

$$\begin{aligned} \min f(x) \\ \text{subject to } c(x) = 0 \end{aligned}$$

$$\begin{aligned} \min_p & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to } & A_k p + c_k = 0. \end{aligned}$$

Algorithm 18.1 (Local SQP Algorithm for solving (18.1)).

Choose an initial pair (x_0, λ_0) ; set $k \leftarrow 0$;

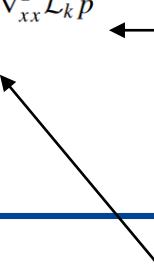
repeat until a convergence test is satisfied

Evaluate $f_k, \nabla f_k, \nabla_{xx}^2 \mathcal{L}_k, c_k$, and A_k ;

Solve (18.7) to obtain p_k and λ_k ;

Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow \lambda_k$;

end (repeat)



With inequality constraints:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad & \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases} \end{aligned}$$

$$\begin{aligned} \min_p & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \\ \text{subject to } & \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \\ & \nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

Thm 18.1: Alg. 18.1 identifies (eventually) the optimal active set of constraints (under assumptions). After, it behaves like Newton's method for equality constrained problems.

A practical line search SQP method

Algorithm 18.3 (Line Search SQP Algorithm).

Choose parameters $\eta \in (0, 0.5)$, $\tau \in (0, 1)$, and an initial pair (x_0, λ_0) ;

Evaluate $f_0, \nabla f_0, c_0, A_0$;

If a quasi-Newton approximation is used, choose an initial $n \times n$ symmetric positive definite Hessian approximation B_0 , otherwise compute $\nabla_{xx}^2 \mathcal{L}_0$;

repeat until a convergence test is satisfied

 Compute p_k by solving (18.11); let $\hat{\lambda}$ be the corresponding multiplier;

 Set $p_\lambda \leftarrow \hat{\lambda} - \lambda_k$;

 Choose μ_k to satisfy (18.36) with $\sigma = 1$;

 Set $\alpha_k \leftarrow 1$;

while $\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \eta \alpha_k D_1(\phi(x_k; \mu_k)) p_k$

 Reset $\alpha_k \leftarrow \tau_\alpha \alpha_k$ for some $\tau_\alpha \in (0, \tau]$;

end (while)

 Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k p_\lambda$;

 Evaluate $f_{k+1}, \nabla f_{k+1}, c_{k+1}, A_{k+1}$, (and possibly $\nabla_{xx}^2 \mathcal{L}_{k+1}$);

 If a quasi-Newton approximation is used, set

$s_k \leftarrow \alpha_k p_k$ and $y_k \leftarrow \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$,

 and obtain B_{k+1} by updating B_k using a quasi-Newton formula;

end (repeat)

$$\min_p f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \quad (18.11a)$$

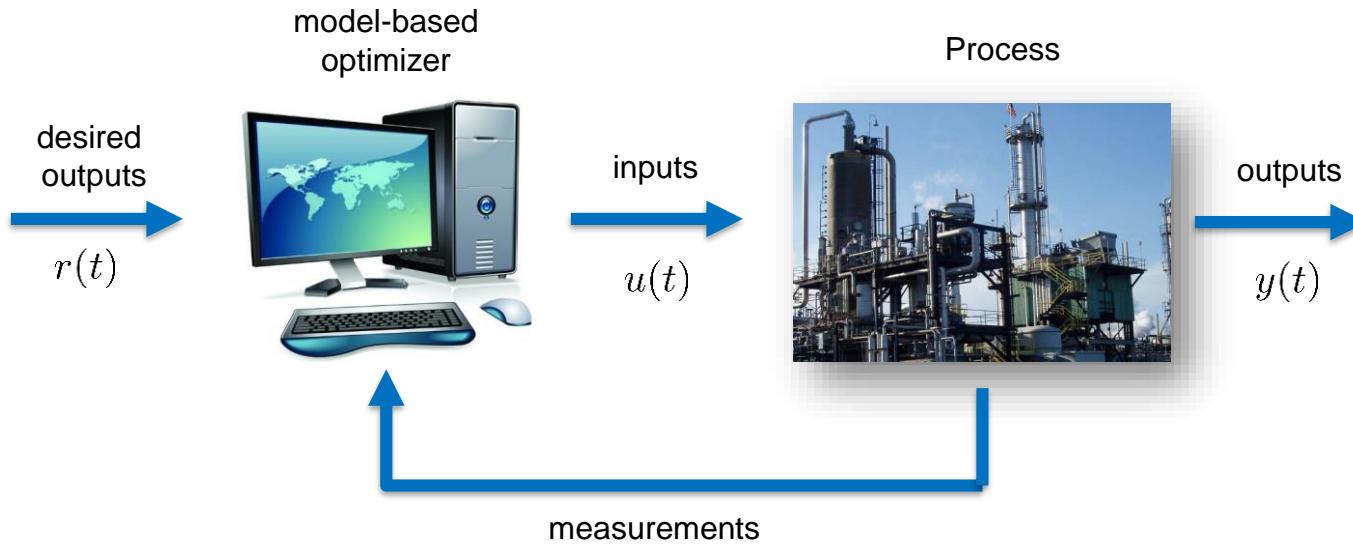
$$\text{subject to } \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \quad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \quad (18.11c)$$

Compare Alg. 6.1:

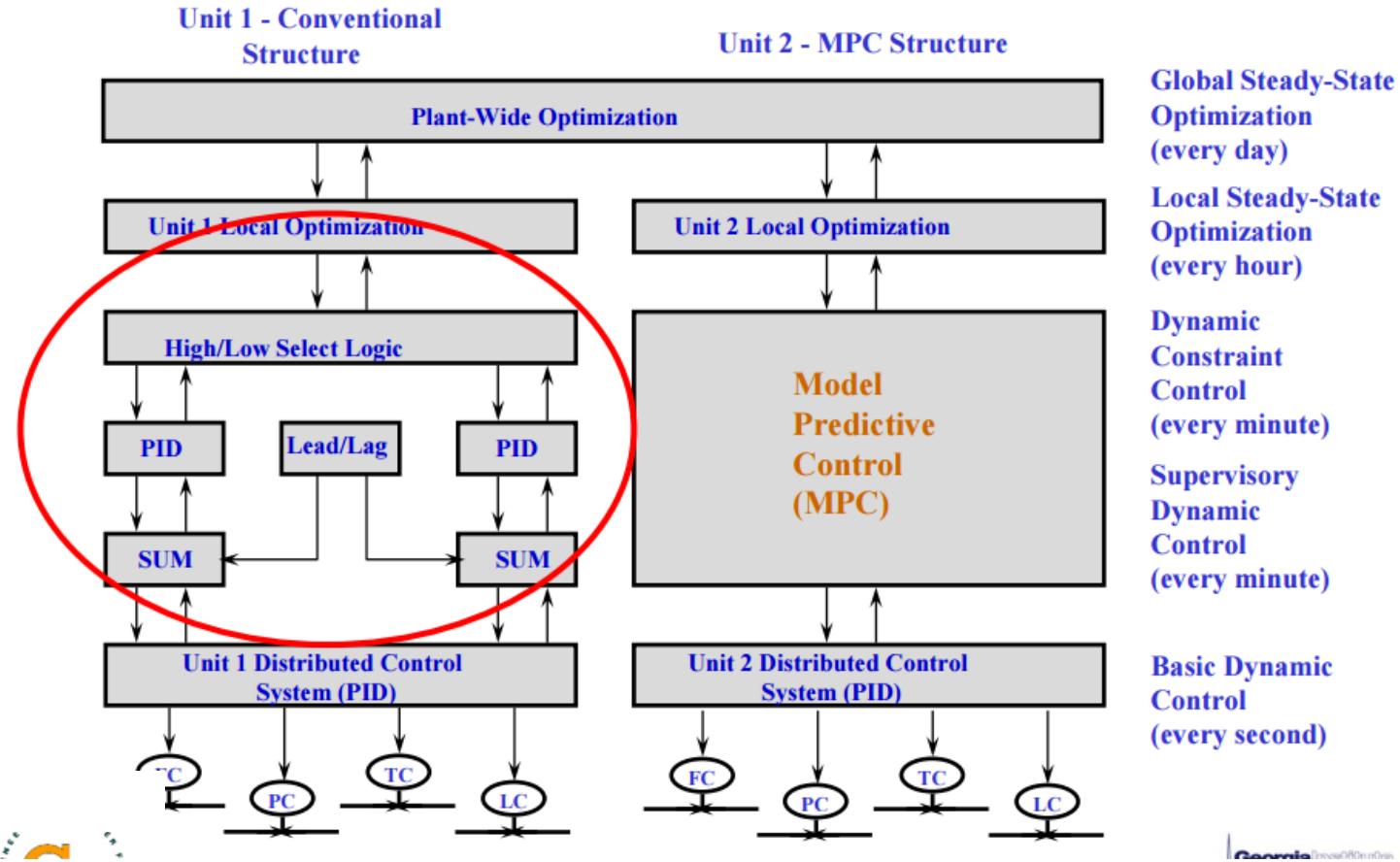
- Search direction from quadratic approximation
- Line search
- Update Hessian using BFGS

Model predictive control



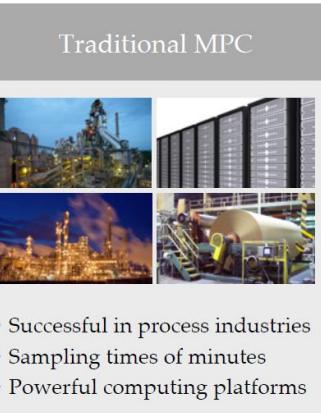
A **model** of the process is used to compute the **control signals** (inputs) that optimize **predicted** future process behavior

Process control hierarchy before and after MPC



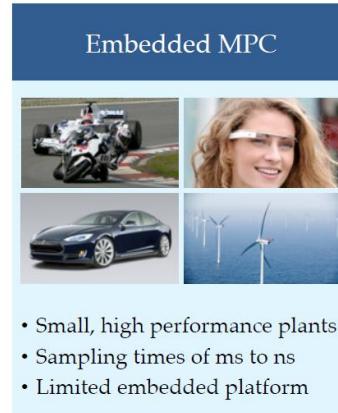
Embedded Model Predictive Control

PhD project Giorgio Kufoalor



- Successful in process industries
- Sampling times of minutes
- Powerful computing platforms

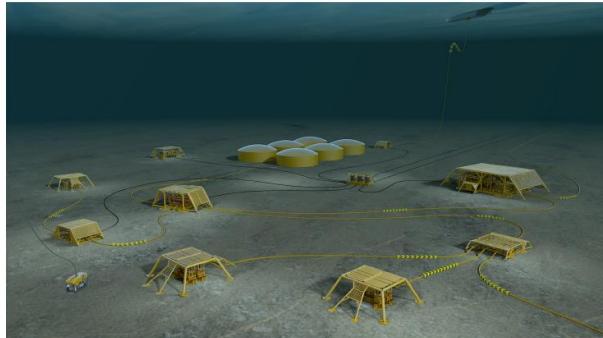
(M. Morari, 2013)



- Small, high performance plants
- Sampling times of ms to ns
- Limited embedded platform

(M. Morari, 2013)

Embedded MPC for new industrial applications

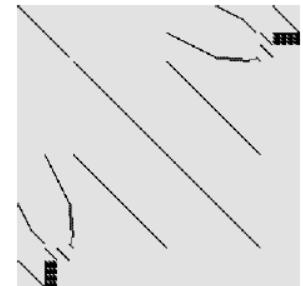


(Statoil subsea factory)

Main contributions to fill the gap

PhD project Giorgio Kufoalor

- Step-response MPC on ultra-reliable, resource constrained, industrial hardware
- Detailed study on *MPC formulations* and *solver methods* to achieve fast and reliable solutions
 - Achieve significant savings, both in computations and memory usage
 - Exploiting problem structure and specifics of computing platform
 - Automatic code generation (almost...)
- Development of new multistage QP framework, tailored to step-response MPC
- All extensively tested on a realistic subsea compact separation simulator using *hardware-in-the-loop* testing



Open-loop optimization with linear state-space model

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{t=0}^{N-1} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + d_{x,t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t + d_{u,t} u_t + \frac{1}{2} \Delta u_t^\top S \Delta u_t$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = \{0, \dots, N-1\}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}}, \quad t = \{1, \dots, N\}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

$$-\Delta u^{\text{high}} \leq \Delta u_t \leq \Delta u^{\text{high}}, \quad t = \{0, \dots, N-1\}$$

QP

where

x_0 and u_{-1} is given

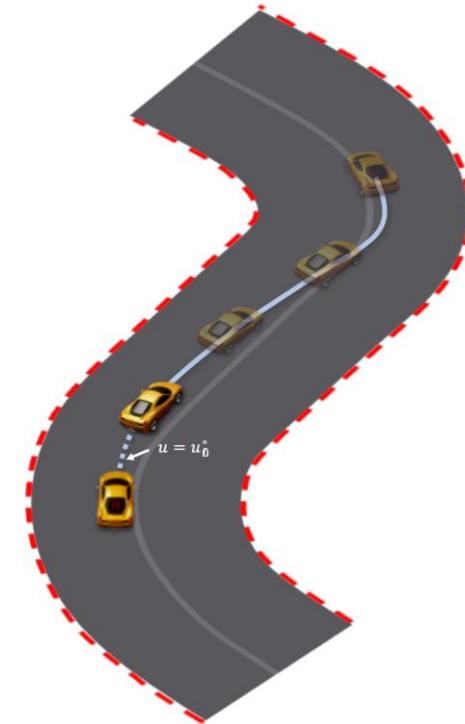
$$\Delta u_t := u_t - u_{t-1}$$

$$z^\top := (u_0^\top, x_1^\top, \dots, u_{N-1}^\top, x_N^\top)$$

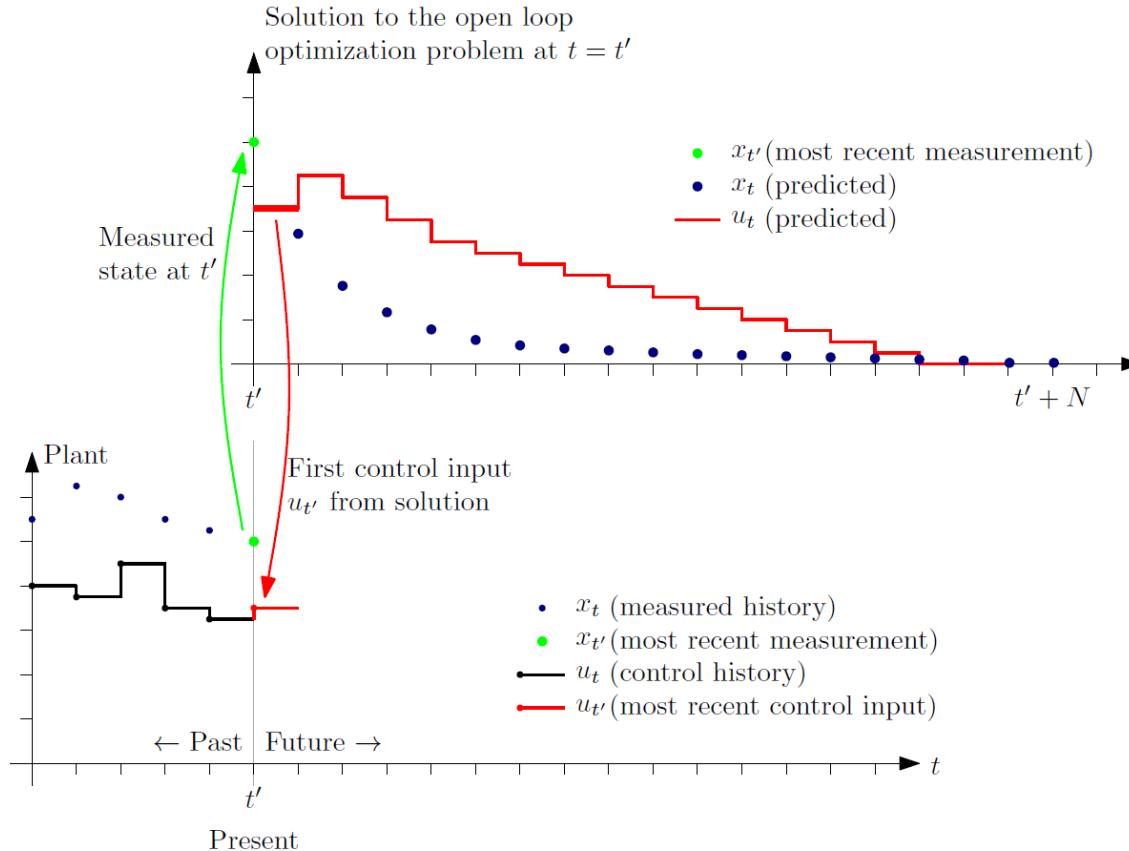
$$n = N \cdot (n_x + n_u)$$

$$Q_t \succeq 0 \quad t = \{1, \dots, N\}$$

$$R_t \succ 0 \quad t = \{0, \dots, N-1\}$$



Model predictive control principle



Nonlinear MPC

The three ways of implementing NMPC

- Sequential (single shooting) methods
 - Only inputs as optimization variables, simulate to calculate objective and state constraints (and gradients)
 - “Small” optimization problem with no structure
 - Standard SQP methods are suitable
- Simultaneous methods
 - Both inputs and states as optimization variables, include model as equality constraints
 - Huge optimization problem, but constraints and gradients are very structured (“sparse”, a lot of zeros)
 - Must use solvers that exploit this structure (e.g. IPOPT)
- In-between method: Multiple shooting
 - Divide horizon into “sub-horizons”, use single-shooting on each sub-horizon and add equality constraints to “glue” each sub-horizon together
 - Results in medium-sized “block-structured” optimization problem
 - Ideally use solvers that exploit this structure (but not many exists)
- What is best? Depends...

NMPC example: van der Pol

- Controlled van der Pol oscillator

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_1 + e(1 - x_1^2)x_2 + u\end{aligned}$$

- Discretization (here: Euler)
- Stability dependent on horizon length
- Importance of “warm-start”

Output feedback MPC

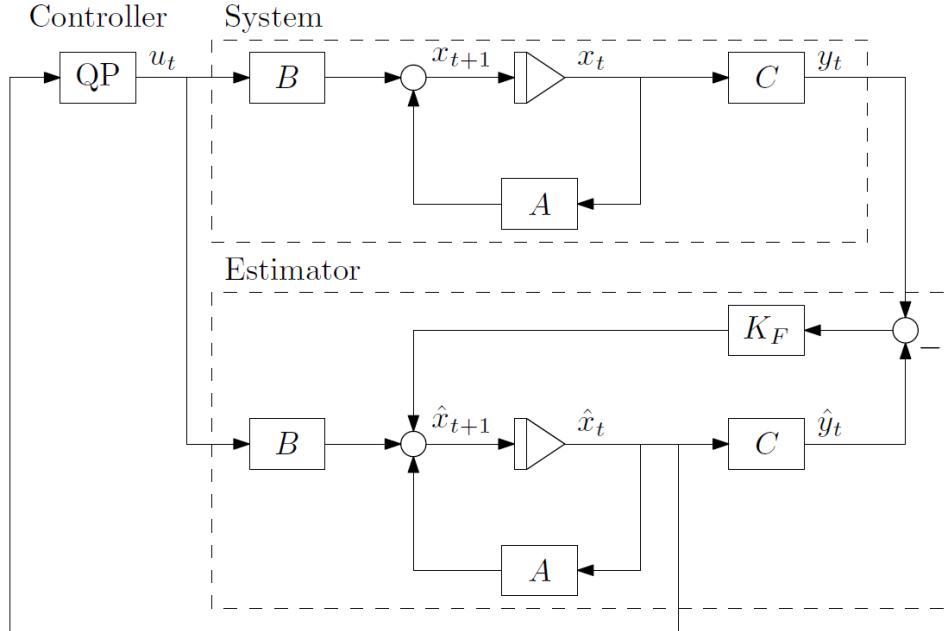
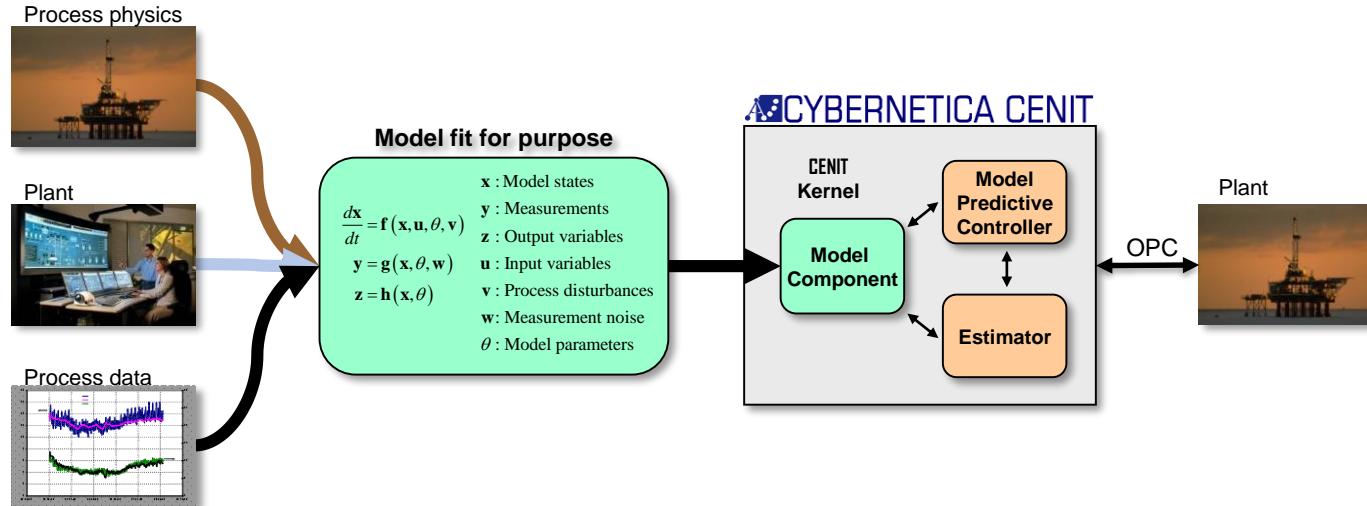


Figure 4.3: The structure of an output feedback linear MPC.

Output feedback NMPC

Cybernetica

- Cybernetica provides advanced model-based control systems for the process industry
 - Based on non-linear first principles (mechanistic) models
 - Nonlinear state- and parameter estimation (EKF, MHE)
 - Online dynamic optimization (nonlinear model predictive control, NMPC)



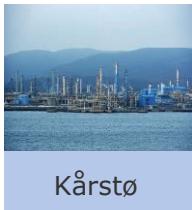
SEPTIC

Statoil Estimation and Prediction Tool
for Identification and Control

Total in Statoil:
92 MPC Applications



#2



#25



#28

Åsgard
Norne
Heidrun

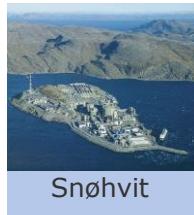
#7



#5

Kalundborg

#21



Snøhvit

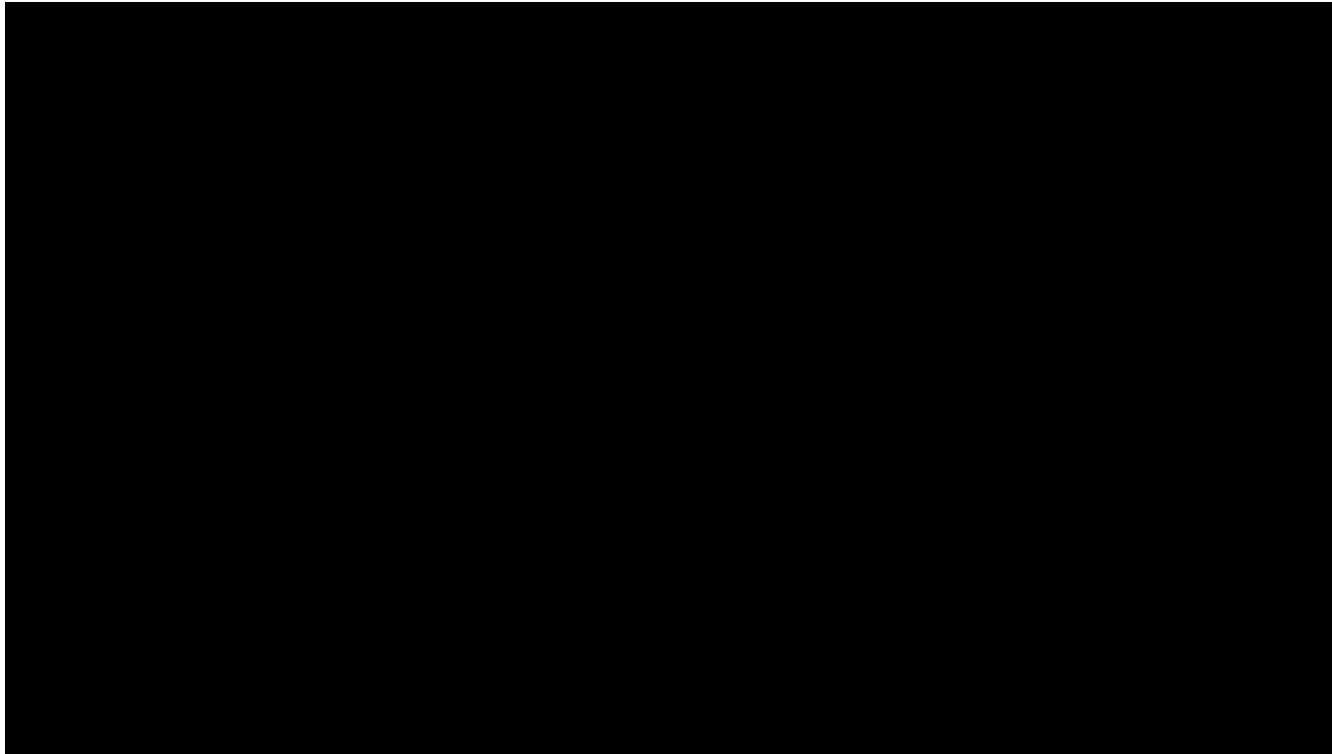
#4



Stig Strand, Statoil

SOME EXAMPLES

Applied LQR



Applied LQR

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = S^T\tau + J_c(q)^T f_c$$

1. Modify dynamics to null-space of the contact constraint
2. Linearize the model around desired pose (q_0, \dot{q}_0, τ_0)

$$\dot{x} = Ax + Bu, \quad x^T = [\Delta q^T, \Delta \dot{q}^T], \quad u^T = [\Delta \tau]$$

3. Calculate the infinite horizon linear quadratic regulator

$$J = \int_0^{\infty} x^T Q x + u^T R u$$

4. Apply the input

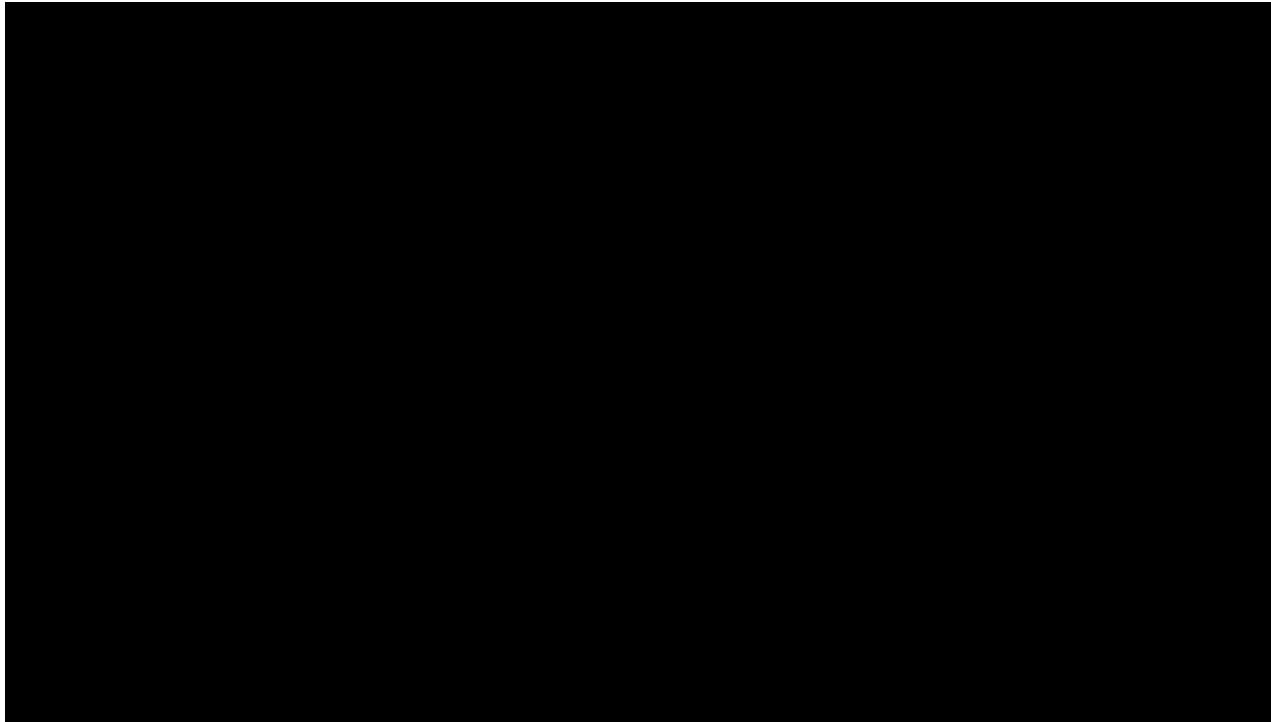
$$\tau = \tau_0 - Kx$$



Credit: Luke Fisher Photography

Fig. 1: Hydraulically actuated torque controlled Sarcos humanoid used for experiments.

Applied Open-Loop Dynamic Optimization



Applied Open-Loop Dynamic Optimization

1. Split the problem:
 - Lap time offline
 - Tracking online
2. Solve a “periodic” problem

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}} \quad & \sum_{k=0}^N j_{\text{LTO}}(\mathbf{x}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = f_s^d(\mathbf{x}_k, \mathbf{u}_k) \\ & f_s^d(\mathbf{x}_N, \mathbf{u}_N) = \mathbf{x}_0 \end{aligned}$$

3. Apply NMPC for closed-loop

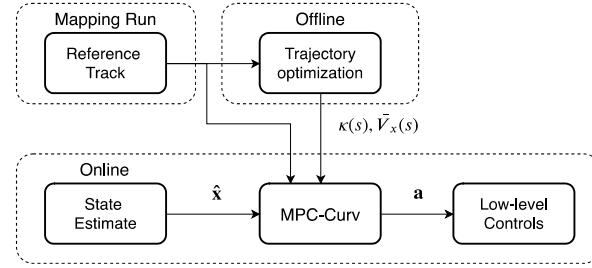
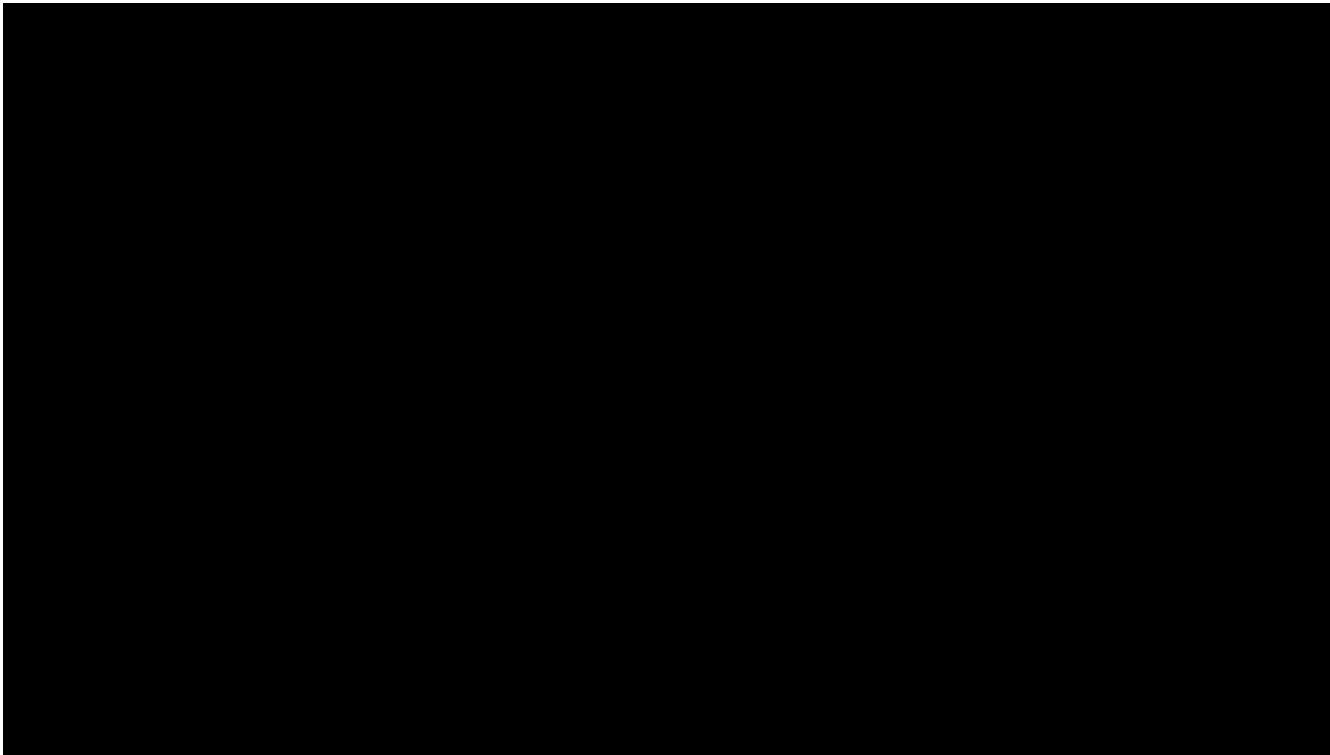


Fig. 3. The hierarchical controller uses the reference track in both stages of the hierarchical controller. First the lap time optimization (LTO) problem computes a reference path, whose curvature $\kappa(s)$ and speed profile $\bar{V}_x(s)$ are later used by the NMPC.

MPC using distributed SQP



MPC using distributed SQP

Algorithm 1 Schematic description of the Distributed SQP.
The arguments of the functions are removed for brevity.

- 1: Coordinator initializes the problem.
- 2: **while** exit conditions not fulfilled **do**
- 3: Coordinator broadcasts T .
- 4: Each vehicle solves (8)
- 5: Each vehicle returns $\nabla V_i, \nabla^2 V_i, g_i, \nabla g_i, \nabla^2 g_i$.
- 6: Coordinator solves the SQP sub-problem (21).
- 7: Coordinator and vehicles compute α .
- 8: Coordinator takes step (20).

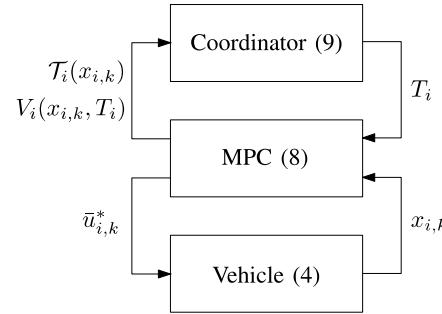
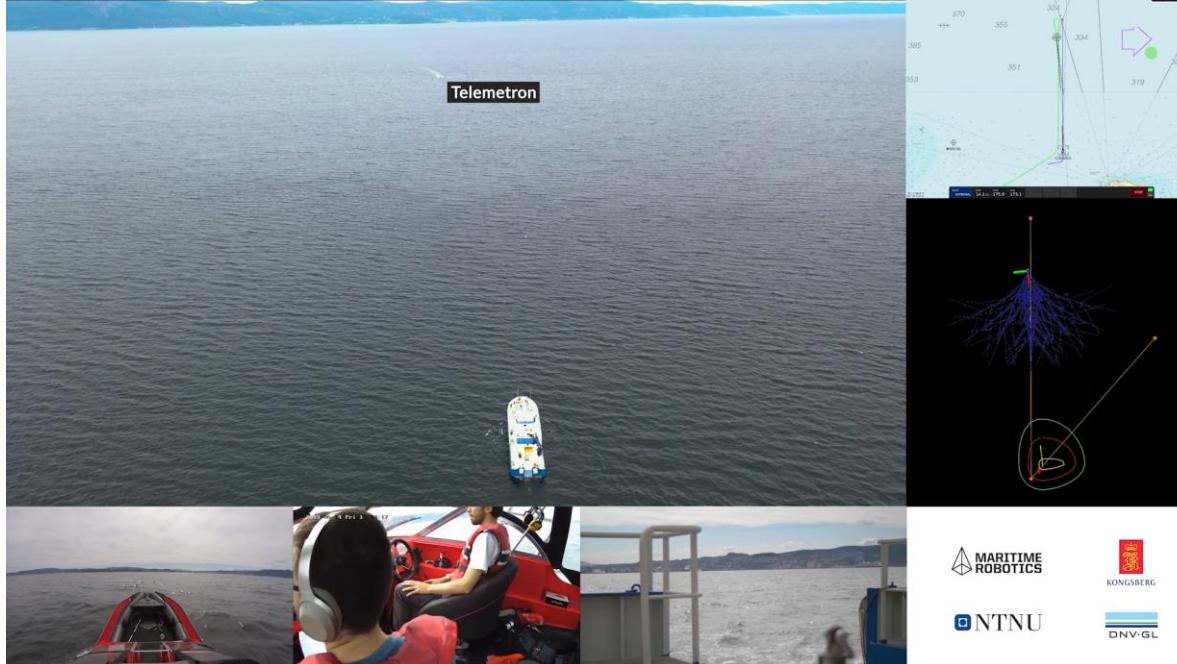


Fig. 2: Schematic illustration of the bi-level control structure for one vehicle. The coordinator is in closed-loop with all vehicles in the same way.

Branching Course MPC



Source: Bjørn-Olav Holtung Eriksen
<https://doi.org/10.1002/rob.21900>

Branching Course MPC

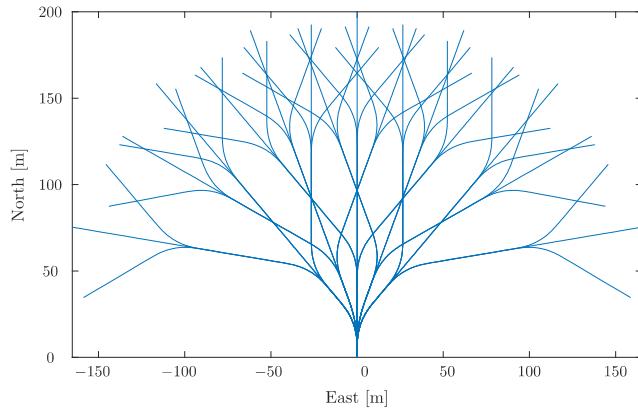


Figure 10: A set of predicted pose trajectories with 3 levels.

