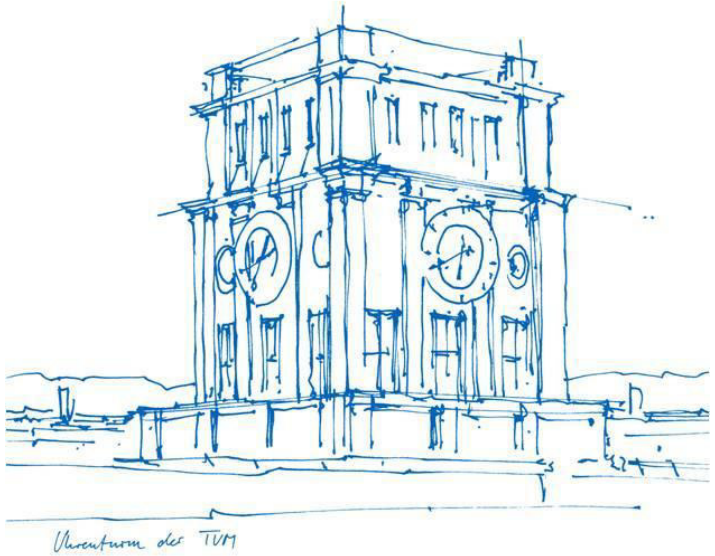
Lehrstuhl für Produktentwicklung und Leichtbau
TUM School of Engineering and Design
Technische Universität München

Numerical optimization for robot design and control – Day 01
Optimization Introduction and MATLAB Warmup

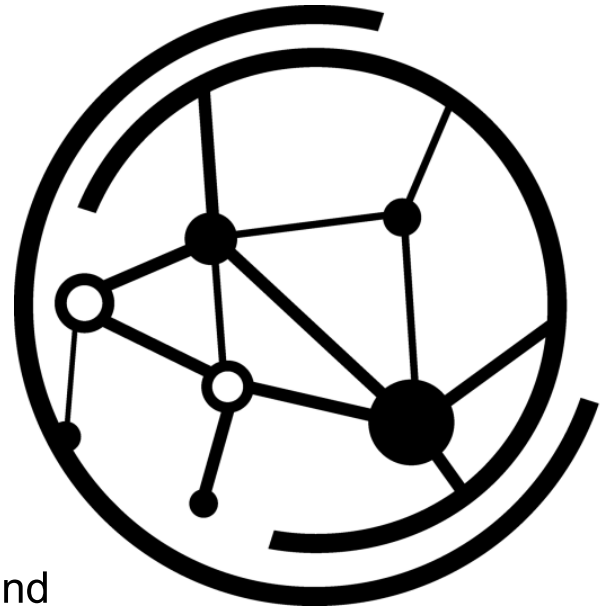
Akhil Sathuluri

Prof. Dr. Markus Zimmermann



Garching, 05th of September 2022

LPL

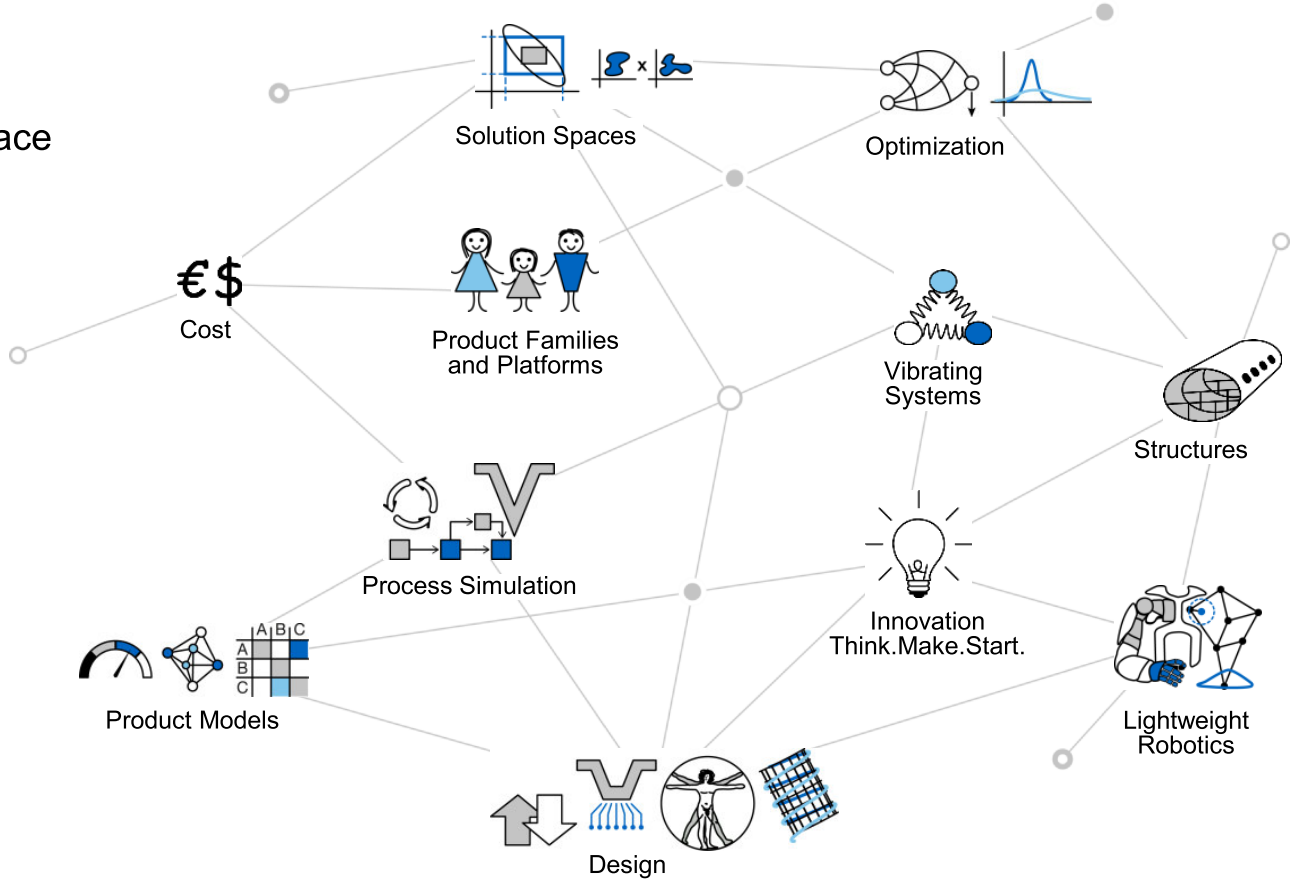


Design and Optimization of technical complex Systems

Research

Solution Space Engineering

Lightweight Structures



Methods & Processes

Robot Systems

Course overview

- Day-1: Optimization introduction and MATLAB warmup
- Day-2: Robot design optimization
- Day-3: Robot control synthesis
- Day-4: Co-design of robots
- Day-5: State-of-the-art and industrial presentation

Organization

- Course schedule:

  09:00-17:00 from Monday-Friday, Seminar Room (5506.EG.636)
  All days will require an in-person participation
  All the slides and codes will be uploaded to Moodle/Git
  Lunch break will be from 11:30-12:30

- Course mode:

  Completely interactive
  Hybrid in-class problem solving and teaching

- Questions and contact:

  You can stop me anytime if you have a question or have trouble following any explanations or mail me at akhil.sathuluri@tum.de

- Exam:

  Evaluation is done via a report submission
  The final working codes will be reviewed
  You will be required to document outcomes as a report
  Deadline for report submission is two weeks from the end of the course
  Report should be in English

Hello!

Lets hear from you.

https://www.menti.com/o9mkmcodmg

Day-01: Optimization Introduction and MATLAB warmup

Part-1: Optimization overview

1. Overview and fundamentals of optimization

Part-2: Hands-on optimization

1. Warming up in MATLAB
2. Thinking about optimization in robotics

Part-1: Overview of fundamentals of optimization

A review of optimization basics from MDO

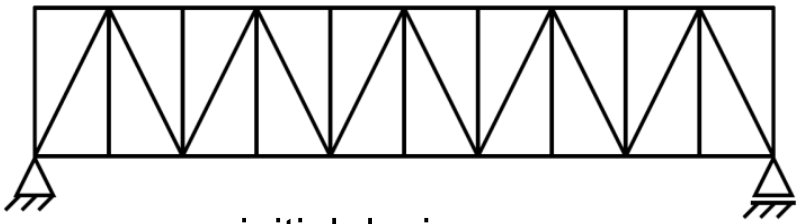
1.1.1. Definition of Design Optimization

- **Optimization is about *finding* the *best* design within the *available means*.**

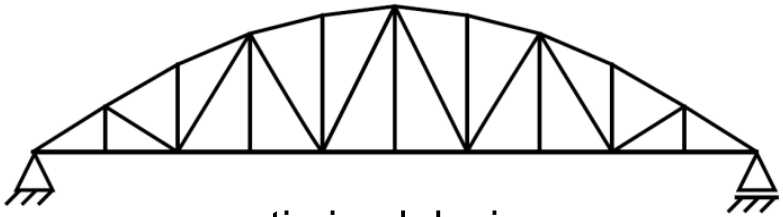
1. What is our criterion for best?
2. What are the available means?
3. How do we modify and search?

Minimum mass, best properties → **objective**
Avail. comp., design rules, bound.cond. → **constraints**
Geometry, material, etc. → **design variables**

- Example from structural design optimization:



initial design



optimized design

1.3.1. General Problem Statement

- Need universal mathematical language for interdisciplinary design work

- General problem statement:

$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{h}(\boldsymbol{x}) = 0 \\ & \boldsymbol{g}(\boldsymbol{x}) \leq 0 \\ & \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u} \end{aligned}$$

$\boldsymbol{x}$ = vector of design variables
$f(\boldsymbol{x})$ = objective function
$\boldsymbol{h}(\boldsymbol{x})$ = equality constraint functions
$\boldsymbol{g}(\boldsymbol{x})$ = inequality constraint functions
$\boldsymbol{x_l}, \boldsymbol{x_u}$ = lower and upper bounds, define design space

- The objective function and constraint functions are *mappings* from the design variables onto the quantities of interest (example to follow).

- Note: The *entire* expression $\boldsymbol{h}(\boldsymbol{x}) = 0$ is called **equality constraint** and $\boldsymbol{g}(\boldsymbol{x}) \leq 0$ **inequality constraint**!

1.3.2. Components of a Problem Statement

Symbol	Name	Description	Synonyms / examples / comments
x_i with $i = 1, \dots, d$ $\boldsymbol{x} = [x_1, \dots, x_d]^T$	Design variables	Degrees of freedom of the designer. He/she choses values for them (or requirements) in order to reach a design goal	Inputs, explanatory variables, degrees of freedom
$\boldsymbol{p} = [p_1, \dots, p_d]^T$	Design parameters	Are fixed from a designers perspective	Mass of a vehicle for crash design
Ω_{ds}	Design space	Meaningful range of design variable values	Input space
y_j with $j = 1, \dots, m$ $\boldsymbol{y} = [y_1, \dots, y_m]^T$	Quantities of interest	Measure the system performance and can be computed from the system response	Outputs, dependent variables, performance measures, objective quantities, system quantity
$f(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{g}(\boldsymbol{x}), \dots$	Output functions	Mappings of design variables onto outputs (often computed numerically)	
$f(\boldsymbol{x})$	Objective function	The function that computes the quantity of interest that is to be maximized or minimized	Cost function, „objective” (also see below)
$\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{g}(\boldsymbol{x})$	Constraint functions	Functions to compute quantities of interests that are used to express equality and inequality constraint on outputs	
$\boldsymbol{y_l} \leq \boldsymbol{y} \leq \boldsymbol{y_u}$, min $\boldsymbol{y}$	Design goal	Objective *OR* a condition satisfied by a good design	Minimization objectives are bad requirements – however often used and very practical.
$m\ddot{z} + c\dot{z} + kz = 0$	System equations	Govern the system response in dependence of the design variables $\boldsymbol{x}$	PDEs, ODEs
$z(t)$	System response / output	Solution of system equation, may be (1) closed-form = formula or (2) numerical	$u(x,t)$ displacement field

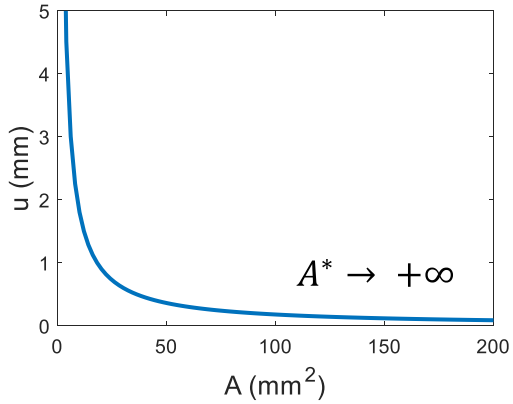
2.2.2. Well-Posedness

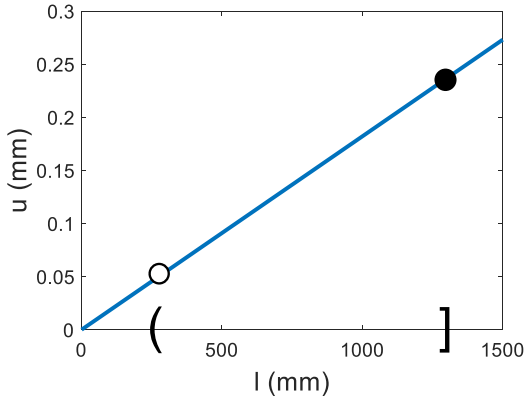
- General problem statement:

$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{h}(\boldsymbol{x}) = 0 \\ & \boldsymbol{g}(\boldsymbol{x}) \leq 0 \\ & \boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u \end{aligned}$$

- A problem is **well-posed** when a **unique** solution **exists**.

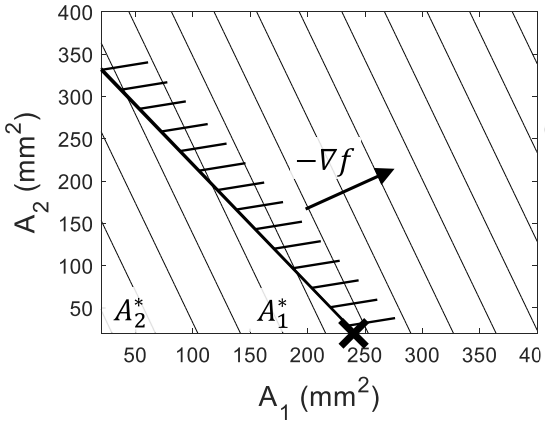
- In engineering practice:
  1) Do not distinguish between open and closed sets (typically).
  2) Make sure that you solve the right problem.



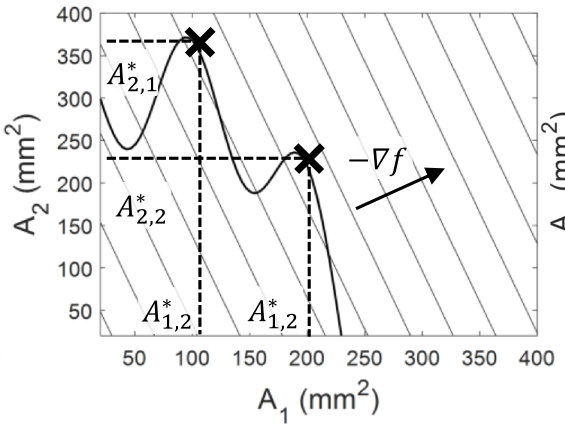


For this system responses, a minimization problem on an open set would not be well posed.

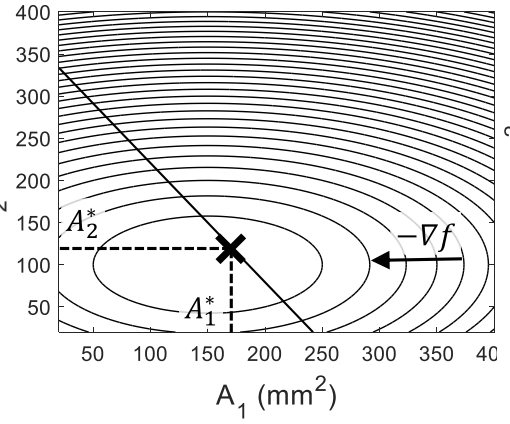
2.2.3. Linearity



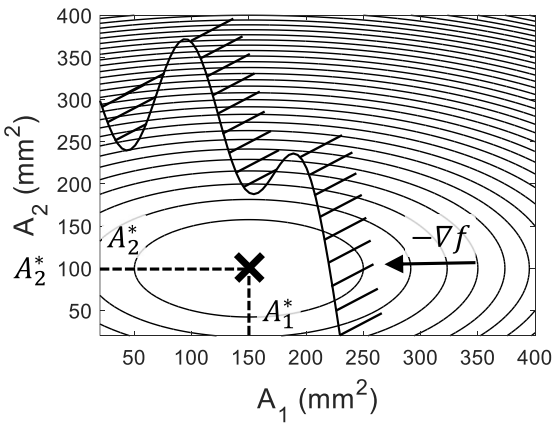
Linear objective function +
Linear constraint =
Linear optimization problem



Linear objective function +
Nonlinear constraint =
Nonlinear optimization problem



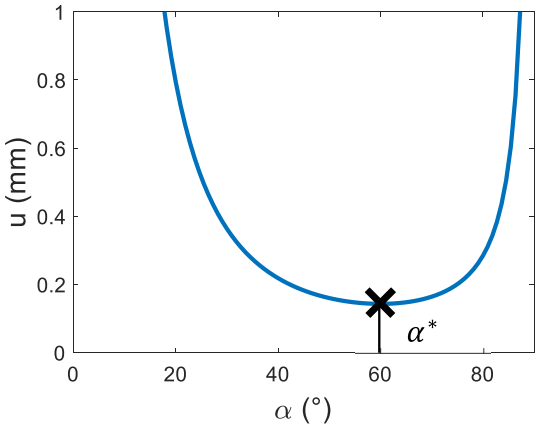
Nonlinear objective function +
Linear constraint =
Nonlinear optimization problem



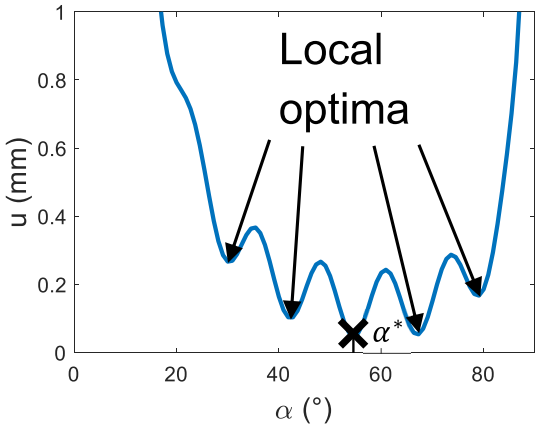
Nonlinear objective function +
Nonlinear constraint =
Nonlinear optimization problem

2.2.4. Convexity

Convex objective function



Nonconvex objective function



- A **function** $f(x)$ **is convex**, if for every two points x_1 and x_2 the connecting line lies above the graph, i.e.:
  $f\big(\boldsymbol{x}_1 + \xi(\boldsymbol{x}_2 - \boldsymbol{x}_1)\big) \leq f(\boldsymbol{x}_1) + \xi\big(f(\boldsymbol{x}_2) - f(\boldsymbol{x}_1)\big) \qquad \xi \in [0,1]$
- An **optimization problem is said to be convex**, when the objective function is convex, the inequality constraint functions are convex and the equality constraint functions are affine.
- For convex problems: local minimum = global minimum

2.2.5. Monotonicity

- Function $f(\boldsymbol{x})$ is said to be **strictly monotonically increasing** if:

$$f(x_1, \dots, x_{iA}, \dots, x_d) > f(x_1, \dots, x_{iB}, \dots, x_d) \quad \forall\, \boldsymbol{x} \in \Omega_{ds},\ x_{iA} > x_{iB}$$

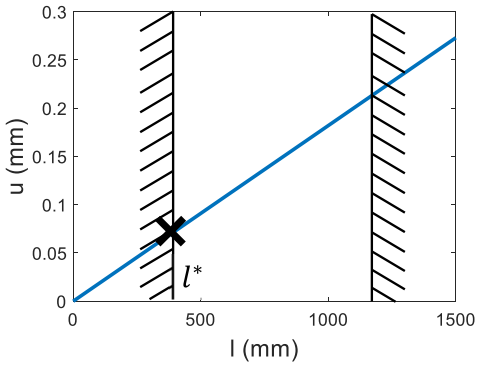
- Function $f(x)$ is said to be **strictly monotonically decreasing** if:

$$f(x_1, \dots, x_{iA}, \dots, x_d) < f(x_1, \dots, x_{iB}, \dots, x_d) \quad \forall\, \boldsymbol{x} \in \Omega_{ds},\ x_{iA} > x_{iB}$$

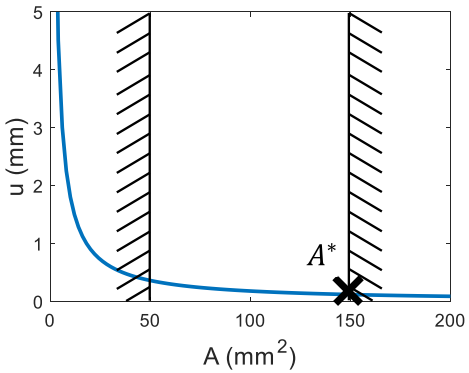
- Weakly monotonically increasing (decreasing):
  replace $>$ ($<$) by $\geq$ ($\leq$), respectively

- Monotonicity $\neq$ Linearity, but linearity implies monotonicity.

- Important monotonous functions: functions for mass, cost, and (often) stiffness.



Linear function
Monotonic function



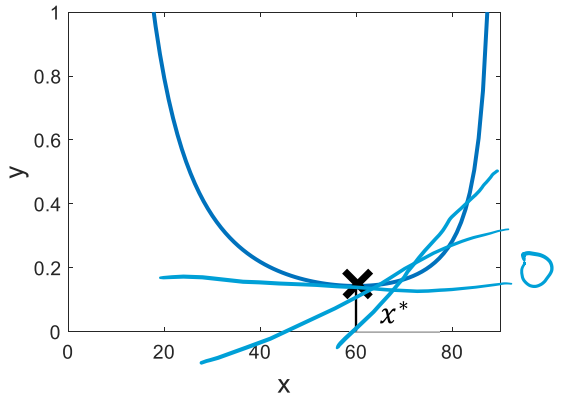
Nonlinear function
Monotonic Function

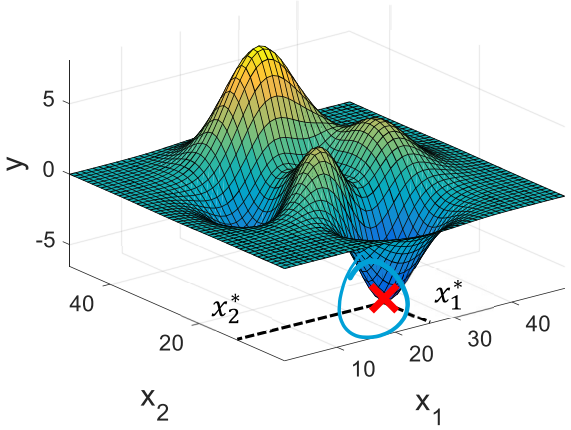
4.1.2. Interior Optima

For interior optima, the optimality conditions are:

I.	$\frac{\partial f}{\partial x}(x^*) = 0$	I. Necessary
II.	$\frac{\partial^2 f}{\partial x^2}(x^*) > 0$	I. + II. Sufficient

III.	$\nabla f(\boldsymbol{x}^*) = \frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}^*) \\ \vdots \\ \frac{\partial f}{\partial x_d}(\boldsymbol{x}^*) \end{bmatrix} = \mathbf{0}$ Gradient	III. Necessary
IV.	$H(\boldsymbol{x}^*) = \frac{\partial^2 f}{\partial \boldsymbol{x}^2}(\boldsymbol{x}^*) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\boldsymbol{x}^*) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d}(\boldsymbol{x}^*) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1}(\boldsymbol{x}^*) & \cdots & \frac{\partial^2 f}{\partial x_d^2}(\boldsymbol{x}^*) \end{bmatrix} = \text{pos. definite}$ Hessian	III. + IV. Sufficient





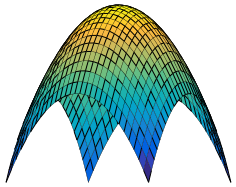
4.1.2. Interior Optima

- When $\nabla f(\boldsymbol{x}^*) = 0$, $\boldsymbol{x}^*$ is called a stationary point.

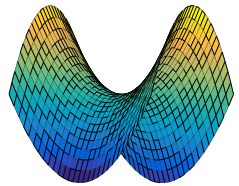
- Nature of stationary points:

Hessian matrix	**Nature of x^***	**Formula**
Positive-definite	Local minimum	$a^T H a > 0 \quad \forall a \neq 0$
Negative-definite	Local maximum	$a^T H a < 0 \quad \forall a \neq 0$
Positive-semidefinite	Not clear	$a^T H a \geq 0 \quad \forall a$
Negative-semidefinite	Not clear	$a^T H a \leq 0 \quad \forall a$
Indefinite - at least one eigenvalue positive and at least one negative	Saddlepoint	

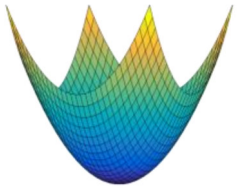
- For a *convex function* in a *convex feasible* domain, $\nabla f(\boldsymbol{x}^*) = 0$ and $a^T H a > 0 \quad \forall a \neq 0$ is sufficient for a global minimum.



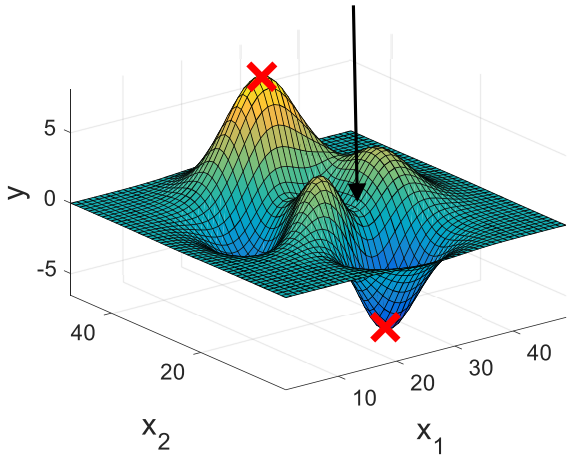
Local maximum



Saddlepoint



Local minimum



4.1.2. Interior Optima

- $\boldsymbol{H} = \boldsymbol{H}^{\boldsymbol{T}}$, because $\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}^*) = \frac{\partial^2 f}{\partial x_j \partial x_i}(\boldsymbol{x}^*)$ for sufficiently smooth functions

  → Eigenvalues λ are real with $\boldsymbol{H}\boldsymbol{e} = \lambda \boldsymbol{e}$.

- Criteria for positive definiteness (*negative definiteness*):
  - All eigenvalues λ are positive (*negative*).
  - All leading principal minors are positive (*all leading principial minors of the negative matrix are pos.*)*:*

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$ has the leading principal minors $|h_{11}|$, $\begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix}$, $\begin{vmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{vmatrix}$

  - Minor: determinant of square submatrix of H by deleting i-th row and j-th column
  - Principal minor: determinant of smaller square submatrix of H by deleting i-th row and i-th column
  - Leading principal minor: … by deleting i-th row and i-th column from bottom right

4.2.1. Lagrangian Function

What are necessary conditions for constrained problems?

- The Lagrangian Function:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \mathbf{g}(\boldsymbol{x})$$

Lagrange multipliers for inequality constraints $\boldsymbol{g}(\boldsymbol{x})$

Lagrange multipliers for equality constraints $\boldsymbol{h}(\boldsymbol{x})$

- Will help to formulate the KKT conditions:

$$\frac{\partial L}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) = \frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) + \boldsymbol{\lambda}^{*T} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) + \boldsymbol{\mu}^{*T} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) = \mathbf{0}$$

Primal feasibility	$\boldsymbol{h}(\boldsymbol{x}^*) = \mathbf{0} \quad \boldsymbol{g}(\boldsymbol{x}^*) \leq \mathbf{0}$
Dual feasibility	$\boldsymbol{\mu}^* \geq \mathbf{0}$
Complementary slackness	$\boldsymbol{\mu}^{*T} \boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0}$

4.2.8. Sufficient Conditions

- Interior optima:

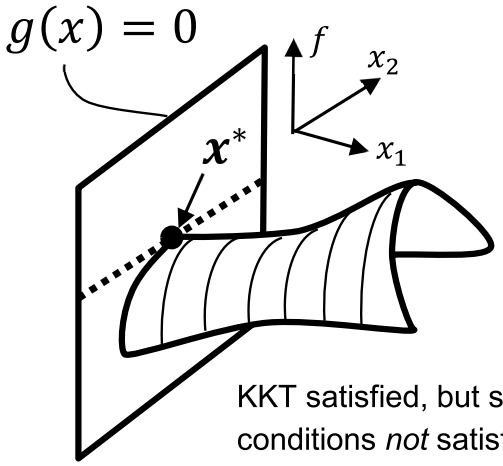
$$\nabla f(\boldsymbol{x}^*) = \frac{\partial f}{\partial \boldsymbol{x}} = 0 \quad + \quad H(\boldsymbol{x}^*) = \frac{\partial^2 f}{\partial \boldsymbol{x}^2} = \text{positive definite}$$

$$\boldsymbol{d}^T \frac{\partial^2 f}{\partial \boldsymbol{x}^2} \boldsymbol{d} > 0 \; \forall \boldsymbol{d} \neq \boldsymbol{0}$$

- Boundary optima: KKT +

$$L_{xx}(\boldsymbol{x}^*) = \frac{\partial^2 L}{\partial \boldsymbol{x}^2} = \text{positive definite on } T(\boldsymbol{x}^*)$$

$$\boldsymbol{d}^T \frac{\partial^2 L}{\partial \boldsymbol{x}^2} \boldsymbol{d} > 0 \quad \forall \boldsymbol{d} \in T(\boldsymbol{x}^*)$$



KKT satisfied, but sufficient conditions *not* satisfied

Tangent subspace:

$$T(\boldsymbol{x}^*) = \left\{ \boldsymbol{d} \neq \boldsymbol{0} \mid \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*)\boldsymbol{d} = \boldsymbol{0}, \frac{\partial \boldsymbol{g}^*}{\partial \boldsymbol{x}}(\boldsymbol{x}^*)\boldsymbol{d} = \boldsymbol{0} \right\}$$

$$g_j^*(\boldsymbol{x}) = \begin{cases} g_j(\boldsymbol{x}) & \text{if } j \text{ active} \\ 0 & \text{if } j \text{ inactive} \end{cases}$$

Tangent to equ. constraints and active inequ. constraints

Deterministic algorithms

Stochastic algorithms

5.1.2. Overview of Optimization Algorithms

	Optimization Algorithms for					
	Unconstrained Problems		**Constrained Problems**			
	d-D	**In addition, in 1-D**	**Primal methods**	**Barrier/Penalty methods**	**Dual Methods**	**Lagrange methods**
Zero Order	Random Sampling (Monte Carlo)	Bracketing	Random Sampling (Monte Carlo)	Transforming a constrained into an unconstrained optimization problem	Dual methods split the original problem into an unconstrained and a constrained problem.	
	Simulated Annealing	Sectioning	Simulated Annealing			
	Evolutionary Algorithms	Polynomial Approximation	Evolutionary Algorithms			
	Full factorial					
First Order	Steepest Descent	Polynomial Approximation	SLP			SQP
	Conjugate Gradient Method					
	Quasi-Newton's Methods					
Second Order	Modified Newton's Method					

Similar to (Bletzinger, 2014)

Local algorithms | Global algorithms

5.1.2. Overview of Optimization Algorithms

	Optimization Algorithms for					
	Unconstrained Problems		**Constrained Problems**			
	d-D	**In addition, in 1-D**	**Primal methods**	**Barrier/Penalty methods**	**Dual Methods**	**Lagrange methods**
Zero Order	Random Sampling (Monte Carlo)	Bracketing	Random Sampling (Monte Carlo)	Transforming a constrained into an unconstrained optimization problem	Dual methods split the original problem into an unconstrained and a constrained problem.	
	Simulated Annealing	Sectioning	Simulated Annealing			
	Evolutionary Algorithms	Polynomial Approximation	Evolutionary Algorithms			
	Full factorial					
First Order	Steepest Descent	Polynomial Approximation	SLP			SQP
	Conjugate Gradient Method					
	Quasi-Newton's Methods					
Second Order	Modified Newton's Method					

Similar to (Bletzinger, 2014)

5.1.3. General Structure of Optimization Algorithms

- Fundamental steps of numerical optimization:

General Optimization Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)}$.
   Set $k = k + 1$.
   Convergence Check. If violated, repeat 2.

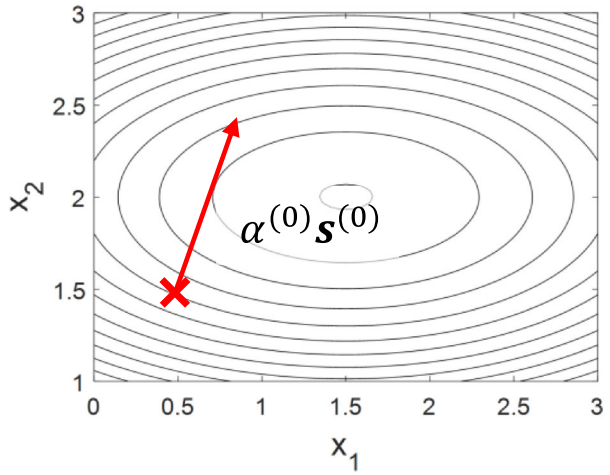
5.2.3. Step Size

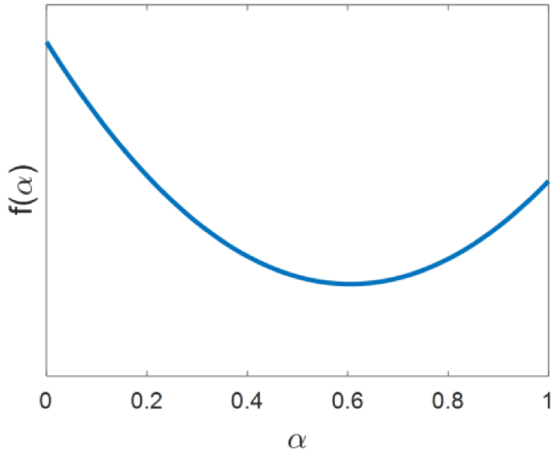
Compute step size $\alpha^{(k)}$

- 1-D Optimization Problem:

$$\min_{\alpha} f(\alpha)$$

$f(\alpha)$ replaces $f(\boldsymbol{x} + \alpha\, \boldsymbol{s})$





- Basic strategy of line search algorithms:
  1. Always **bracketing**: Find an interval $[a_0, b_0]$ that is guaranteed to contain an optimum.
  2. Possibly **sectioning**: Iterative procedure that reduces the bracket size $[a_k, b_k]$.
  3. Possibly **approximation**: Approximate the optimum, typically by an interpolation within $[a_n, b_n]$.

5.2.7. Convergence Test

Convergence Tests / Stopping Criteria

- No or little change of the value of the objective function:

$$\left|\frac{f^{(k+1)}(\boldsymbol{x})-f^{(k)}(\boldsymbol{x})}{f_{ref}}\right| \leq \epsilon$$

$$\left|\frac{f^{(k+1)}(\boldsymbol{x})-f^{(k)}(\boldsymbol{x})}{f^{(k)}(\boldsymbol{x})}\right| \leq \epsilon$$

- No or little change of the candidate design:

$$\frac{|\boldsymbol{x}^{(k+1)}-\boldsymbol{x}^{(k)}|}{|\boldsymbol{x}_{ref}|} \leq \epsilon$$

- An acceptable number of iterations was exceeded (e.g., because an acceptable solution does not exist):

$$k \geq k_{max}$$

- The algorithm cycles:

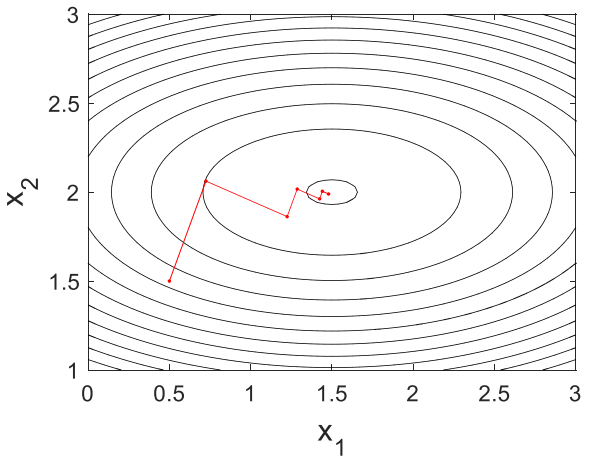
$$f^{(k)}(\boldsymbol{x}) = f^{(k+q)}(\boldsymbol{x})$$

5.2.1. Steepest Descent Overview over Algorithm

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Steepest Descent Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Set search direction $\boldsymbol{s}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right)$.
   Compute step size $\alpha^{(k)}$.
   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$.
   Set $k = k + 1$.
   Convergence test. If violated, repeat 2.



5.2.2. Steepest Descent Search Direction

Set search direction $\boldsymbol{s}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right)$

- Idea: Gradient points in the direction of the biggest increase of the objective function.
- Gradient: $-\nabla f(\boldsymbol{x}) = -\begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_d} \end{bmatrix}$
- Often gradient not known and needs to be approximated:
  - e.g. Central finite differences: $\frac{\partial f(\boldsymbol{x})}{\partial x_i} = \frac{f(x_i+\Delta x_i)-f(x_i-\Delta x_i)}{2\Delta x_i}$

    → Choose appropriate Δx_i, e.g.: resolution length of physical problem, characteristic length of FE-model, ...

5.3.1. Conjugate Gradient Overview over Algorithm

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Conjugate Gradient Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Set search direction:

   if $k = 0 : \boldsymbol{s}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right)$

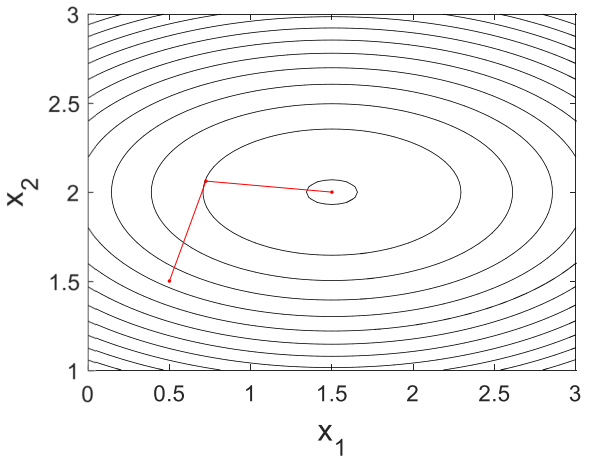
   else: $\boldsymbol{s}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right) + \beta^{(k)}\boldsymbol{s}^{(k-1)}$

   Compute step size $\alpha^{(k)}$.

   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$.

   Set $k = k + 1$.

   Convergence test. If violated, repeat 2



5.3.2. Conjugate Gradient Search Direction

Set search direction $\boldsymbol{s}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right) + \beta^{(k)}\boldsymbol{s}^{(k-1)}$

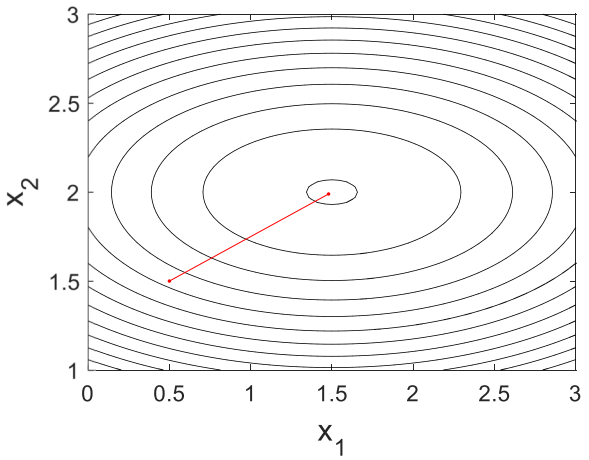
- Idea: Improve search direction by storing information of the previous iteration.
- β determines the ratio between both direction for $\boldsymbol{s}^{k}$.
- Fletcher-Reeves: $\beta^{(k)} = \frac{\nabla f^{T}\left(\boldsymbol{x}^{(k)}\right)\nabla f\left(\boldsymbol{x}^{(k)}\right)}{\nabla f^{T}\left(\boldsymbol{x}^{(k-1)}\right)\nabla f\left(\boldsymbol{x}^{(k-1)}\right)}$
  - Becomes smaller, when approaching solution.
  - Other choices possible.

5.4.1. Modified Newton's Method Overview over Algorithm

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Modified Newton's Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Set search direction $\boldsymbol{s}^{(k)} = -\boldsymbol{H}(\boldsymbol{x}^{(k)})^{-1}\nabla f\left(\boldsymbol{x}^{(k)}\right)$.
   Compute step size $\alpha^{(k)}$.
   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$.
   Set $k = k + 1$.
   Convergence test. If violated, repeat 2



5.4.2. Modified Newton's Method Search Direction

Set search direction $\boldsymbol{s}^{(k)} = -\boldsymbol{H}(\boldsymbol{x}^{(k)})^{-1}\nabla f(\boldsymbol{x}^{(k)})$

- Idea: Approximate function with Taylor expansion of 2nd order:

$$f(\boldsymbol{x}^{(k+1)}) = f(\boldsymbol{x}^{(k)}) + \nabla f^T(\boldsymbol{x}^{(k)})\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\boldsymbol{H}(\boldsymbol{x}^{(k)})\Delta\boldsymbol{x} \qquad \text{with } \Delta\boldsymbol{x} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$$

$$\frac{\partial f(\boldsymbol{x}^{(k+1)})}{\partial \boldsymbol{x}^{(k+1)}} = \nabla f(\boldsymbol{x}^{(k)}) + H(\boldsymbol{x}^{(k)})\Delta\boldsymbol{x} = 0 \Rightarrow \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{H}(\boldsymbol{x}^{(k)})^{-1}\nabla f(\boldsymbol{x}^{(k)})$$

- If the function is locally strictly convex, $\boldsymbol{H}(\boldsymbol{x}^{(k)})$ will be positive-definite and the search direction will improve the objective function (for an appropriate step size).

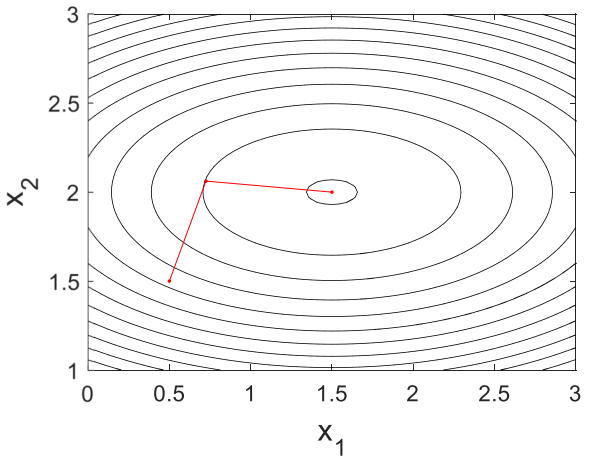
- *Modified* Newton's method, because of line search.
- Only works for non-singular Hessians $\boldsymbol{H}(\boldsymbol{x}^{(k)})$.
- Small no of iterations, however the Hessian and the increment may be expensive to compute.

5.5.1. Quasi-Newton's Method Overview over Algorithm

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Quasi-Newton's Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$ and starting matrix $\boldsymbol{B}\left(\boldsymbol{x}^{(0)}\right)$.
2. For $k \geq 0$: Set search direction $\boldsymbol{s}^{(k)} = -\boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right)\nabla f\left(\boldsymbol{x}^{(k)}\right)$.
   Compute step size $\alpha^{(k)}$.
   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$.
   Calculate $\boldsymbol{B}\left(\boldsymbol{x}^{(k+1)}\right) = \boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right) + \Delta\boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right)$.
   Set $k = k + 1$.
   Convergence test. If violated, repeat 2



5.5.2. Quasi-Newton's Method Search Direction

Set search direction $\boldsymbol{s^{(k)} = -B\left(x^{(k)}\right)\nabla f\left(x^{(k)}\right)}$

- Idea: Instead of the Hessian, an approximated Hessian $\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)$ is used:

$$\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k+1)}\right) = \widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right) + \Delta\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)$$

- $\Delta\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)$ is the update matrix, $\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(0)}\right)$ is the initial Matrix which is usually chosen as $\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(0)}\right) = \mathbf{1}$.

- As the inverse of the Hessian is needed, we define:

$$\boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right) = \widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)^{-1}$$

$$\boldsymbol{B}\left(\boldsymbol{x}^{(k+1)}\right) = \boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right) + \Delta\boldsymbol{B}\left(\boldsymbol{x}^{(k)}\right)$$

5.5.2. Quasi-Newton's Method Search Direction

Set search direction $\boldsymbol{s^{(k)} = -B\left(x^{(k)}\right)\nabla f\left(x^{(k)}\right)}$

- Update possible with the BFGS (Broydon-Fletcher-Goldfarb-Shanno) Method:

$$\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k+1)}\right) = \widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right) + \left(\frac{\Delta(\nabla f)\Delta(\nabla f)^T}{\Delta(\nabla f)^T \Delta \boldsymbol{x}}\right) - \left(\frac{\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)\Delta x\left(\widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x}\right)^T}{\Delta \boldsymbol{x}^T \widehat{\boldsymbol{H}}\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x}}\right)$$

$$\text{with } \Delta(\nabla f) = \nabla f\left(\boldsymbol{x}^{(k+1)}\right) - \nabla f\left(\boldsymbol{x}^{(k)}\right) \text{ and } \Delta \boldsymbol{x} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$$

The inverse can be obtained by applying the Sherman–Morrison formula:

$$\widehat{\boldsymbol{B}}\left(\boldsymbol{x}^{(k+1)}\right) = \widehat{\boldsymbol{B}}\left(\boldsymbol{x}^{(k)}\right) + \left(1 + \frac{\Delta(\nabla f)^T \widehat{\boldsymbol{B}}\left(\boldsymbol{x}^{(k)}\right)\Delta(\nabla f)}{\Delta \boldsymbol{x}^T \Delta(\nabla f)}\right)\left(\frac{\Delta \boldsymbol{x}\Delta \boldsymbol{x}^T}{\Delta \boldsymbol{x}^T \Delta(\nabla f)}\right) - \left(\frac{\Delta \boldsymbol{x}\Delta(\nabla f)^T \widehat{\boldsymbol{B}}\left(\boldsymbol{x}^{(k)}\right) + \widehat{\boldsymbol{B}}\left(\boldsymbol{x}^{(k)}\right)\Delta(\nabla f)\Delta \boldsymbol{x}^T}{\Delta \boldsymbol{x}^T \Delta(\nabla f)}\right)$$

Lunch

5.6.1. Barrier and Penalty Introduction

- Why not just use the known algorithms for constrained problems as well?

→ Optimal Solution is a **saddle point** in the Lagrangian design space $\Omega_{ds}^{L} = \Omega_{ds} \times \Omega_{\lambda,\mu}$

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{T}\boldsymbol{h}(\boldsymbol{x})$$

$$H(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \begin{bmatrix} \frac{\partial^2 L}{\partial \boldsymbol{x}^2}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) & \frac{\partial^2 L}{\partial \boldsymbol{\lambda}\partial \boldsymbol{x}}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \\ \left(\frac{\partial^2 L}{\partial \boldsymbol{x}\partial \boldsymbol{\lambda}}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\right)^{T} & \frac{\partial^2 L}{\partial \boldsymbol{\lambda}^2}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f}{\partial \boldsymbol{x}^2}(\boldsymbol{x}^*) + \boldsymbol{\lambda}^{*T}\frac{\partial^2 \boldsymbol{h}}{\partial \boldsymbol{x}^2}(\boldsymbol{x}^*) & \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*) \\ \left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*)\right)^{T} & 0 \end{bmatrix}$$

- Problem: Gradient methods do not converge at a saddle point.

- Solutions:
  - Constrained Algorithms (Lecture 6)
  - **Barrier and Penalty methods**

5.6.1. Barrier and Penalty Introduction

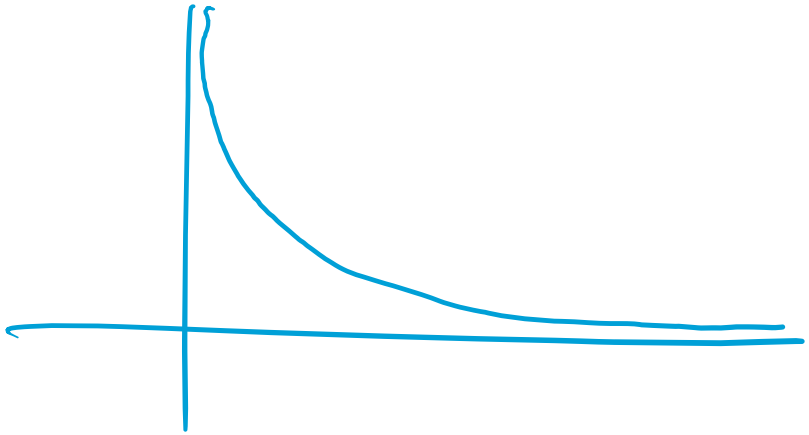
- Basic Idea: Transform a constrained into an unconstrained optimization problem.

$$\psi(\boldsymbol{x}) = f(\boldsymbol{x}) + F(\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}), r)$$

$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \\ & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \end{aligned} \longrightarrow \boxed{\min_{\boldsymbol{x}} \psi(\boldsymbol{x})}$$

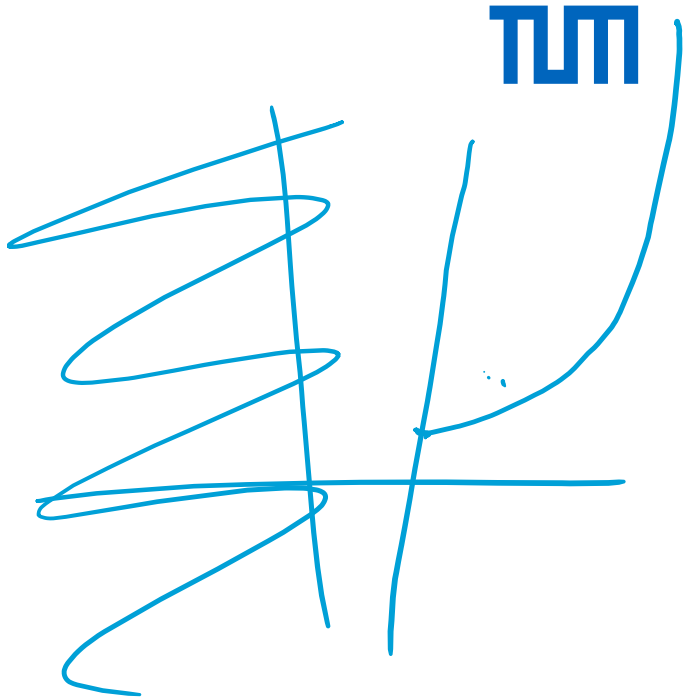
1. Barrier functions (Interior-point penalty function)
2. Penalty functions
3. Augmented Lagrangian method (Dual Method)

5.6.2. Barrier Functions



- General form: $\psi(\boldsymbol{x}) = f(\boldsymbol{x}) + r\, B(\boldsymbol{x})$, for $r > 0$
- With $B(\boldsymbol{x})$ as the barrier function
- Barrier functions cannot be used for equality constraints $\boldsymbol{h}(x)$.
- Typical functions:
  a) $B(\boldsymbol{x}) = -\sum_{j=1}^{m} \ln[-g_j(\boldsymbol{x})]$
  b) $B(\boldsymbol{x}) = -\sum_{j=1}^{m} g_j^{-1}(\boldsymbol{x})$
- When the penalty constant r tends to zero, $\psi(\boldsymbol{x})$ approaches the original solution $\boldsymbol{x}^*$ of $f(\boldsymbol{x})$.

5.6.3. Penalty Functions



- General form: $\psi(\boldsymbol{x}) = f(\boldsymbol{x}) + r^{-1}\, P(\boldsymbol{x}),$ for $r > 0$
- with $P(\boldsymbol{x})$ as the penalty function
- Typical functions for:
  a) Inequality constraints: $P(\boldsymbol{x}) = \sum_{j=1}^{m} \left[\max\{0, g_j(\boldsymbol{x})\}\right]^2$
  (defined to be non-negative in Ω and $P(x) = 0$ for an interior point)
  b) Equality constraints: $P(\boldsymbol{x}) = \sum_{j=1}^{m} \left[h_j(\boldsymbol{x})\right]^2$
- When the penalty constant r tends to zero, $\psi(\boldsymbol{x})$ approaches the original solution $\boldsymbol{x}^*$ of $f(\boldsymbol{x})$.

5.6.4. Augmented Lagrangian Method

- **Problems for pure Barrier/Penalty functions:**
  - For $r^{(k)} \to 0$, ill-conditioned Hessian makes optimization difficult.

- Remedy: Augmented Lagrangian Method
  - Add $P(\boldsymbol{x})$ as penalty function to the Lagrangian function

$$\psi(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}) + r^{-1} P(\boldsymbol{x}), \qquad \text{for } r > 0$$

  - Optimize only w.r.t. $\boldsymbol{x}$ with estimated values for $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. Then update $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ and repeat.

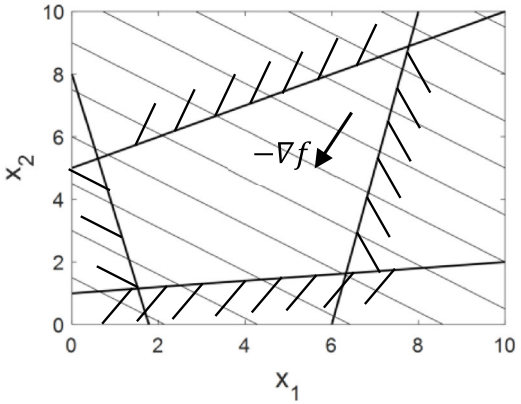
- At optimum: KKT conditions are satisfied even for finite r!

6.1.1. Linear Programming Introduction

- Linear programming is a collection of solution methods to solve a linear optimization problem.
- Will serve as module for Sequential Linear Programming

- Consider a linear optimization problem with $f(\boldsymbol{x}) = \boldsymbol{e}^T\boldsymbol{x} + z$:

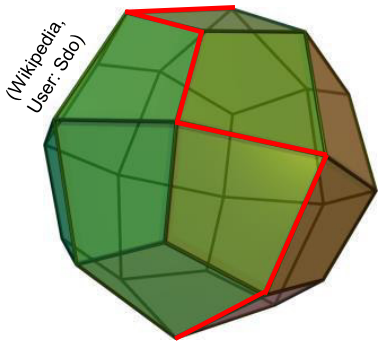
$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} = 0 \\ & \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} \leq 0 \\ & \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u} \end{aligned}$$



- Objective function $f(\boldsymbol{x})$ and constraint functions $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are linear.
- Optimal solution is always on the boundary:
  - If the solution is at a vertex of at least two constraints → only one solution
  - If one constraint is parallel to the objective function $\nabla g / \nabla h \parallel \nabla f$ → infinite number of solutions
  - If contradicting constraints exist → no solution

6.1.2. Simplex Algorithm

- Several algorithms for Linear Programming problems exist.



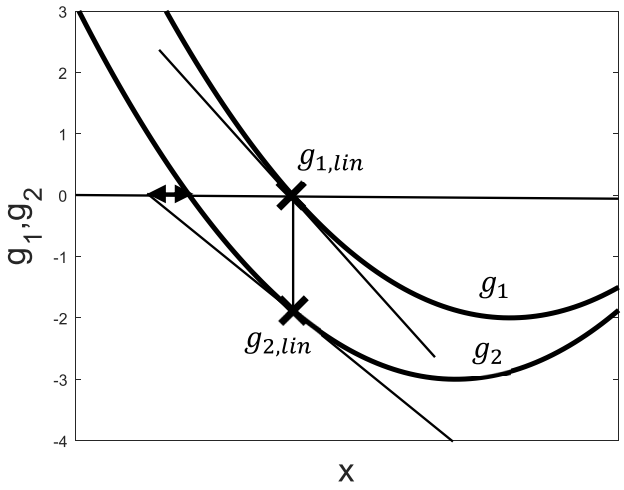
- An important representative is the Simplex algorithm, developed by George Danzig:
  - Underlying idea: move from one feasible vertex to a better neighboring feasible vertex. Finish, when there is no improvement.
  - E.g. in Matlab: linprog

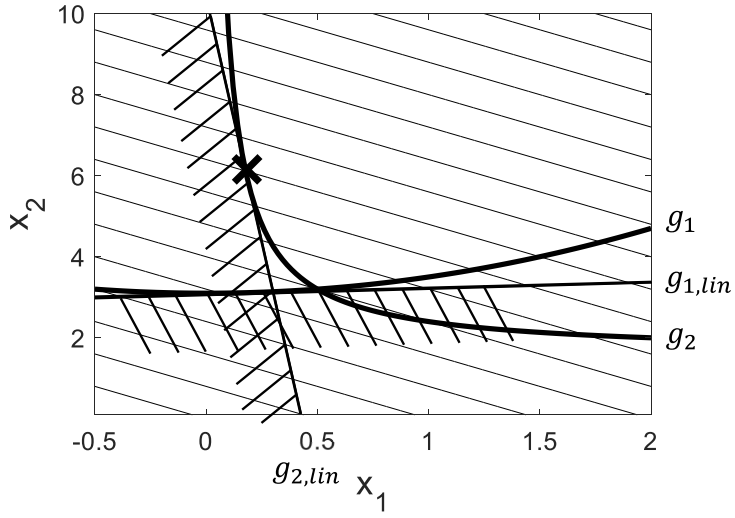
- The Simplex algorithm will be explained for the linear optimization problem with $f(\boldsymbol{x}) = \boldsymbol{e}^T\boldsymbol{x} + z$:

$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} \leq 0 \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

6.2.1. SLP Introduction

- Sequential Linear Programming (SLP): used to solve nonlinear optimization problems by a sequence of linear optimization problems





6.2.2. SLP Overview over Algorithm

SLP Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$. Choose *move limits* $\theta^{(k)}$.
2. For $k \geq 0$: Linearization of the optimization problem
   Solve linear optimization sub-problem
   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)}$
   Adapt *move limits* $\theta^{(k+1)} = \theta^{(k)} + \Delta\theta^{(k)}$ (Preparation for next step)
   Set $k = k + 1$
   Check whether new point is feasible. If violated, repeat step 2.
   Convergence test. If violated, repeat 2.

6.2.4. SLP Linearization

Definition of gradient for this lecture (different from Papalambros):

$$d\boldsymbol{v} = \left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}\right)^{\boldsymbol{T}} d\boldsymbol{x} = (\nabla \boldsymbol{v})^T d\boldsymbol{x} = \nabla \boldsymbol{v}^T d\boldsymbol{x}$$

Linearization

- Expand the objective and constraint functions as a first order Taylor series

$$\hat{f}\,(\boldsymbol{x}) = f\left(\boldsymbol{x}^{(k)}\right) + \nabla f^T\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x}^{(k)} \qquad \widehat{\boldsymbol{h}}\,(\boldsymbol{x}) = \boldsymbol{h}\left(\boldsymbol{x}^{(k)}\right) + \nabla \boldsymbol{h}^T\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x} \qquad \widehat{\boldsymbol{g}}\,(x) = \boldsymbol{g}\left(\boldsymbol{x}^{(k)}\right) + \nabla \boldsymbol{g}^T\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x}$$

$$\Delta \boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$$

- Add move limits as constraints to linearized optimization problem.

- Linearized optimization sub-problem:

$$\begin{aligned} & \min_{\boldsymbol{x}} \hat{f}(\boldsymbol{x}) \\ \text{subject to:} \quad & \widehat{\boldsymbol{h}}\,(\boldsymbol{x}) = \boldsymbol{h}\left(\boldsymbol{x}^{(k)}\right) + \nabla \boldsymbol{h}^T\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x} = \boldsymbol{0} \\ & \widehat{\boldsymbol{g}}\,(\boldsymbol{x}) = \boldsymbol{g}\left(\boldsymbol{x}^{(k)}\right) + \nabla \boldsymbol{g}^T\left(\boldsymbol{x}^{(k)}\right)\Delta \boldsymbol{x} \leq \boldsymbol{0} \\ & (1-\theta)\boldsymbol{x}^{(k)} \leq \boldsymbol{x} \leq (1+\theta)\boldsymbol{x}^{(k)} \\ & \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u} \end{aligned}$$

6.3.1. Quadratic Programming Introduction

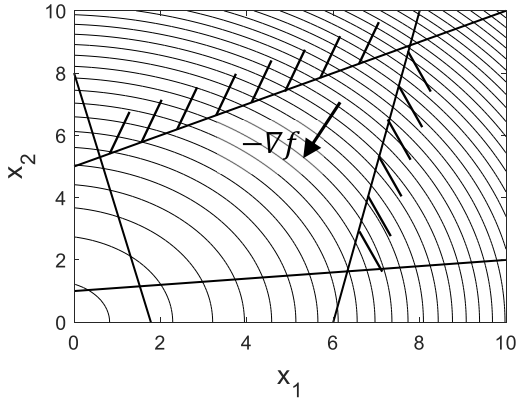
- Quadratic Programming is a collection of solution methods to solve quadratic optimization problems.
- Will serve as module for Sequential Quadratic Programming.

- Consider the quadratic optimization problem:

  with $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x^T Q x} + \boldsymbol{e}^T\boldsymbol{x} + z$

$$\begin{aligned} & \min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to:} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} = 0 \\ & \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} \leq 0 \\ & \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u} \end{aligned}$$

- Objective function $f(\boldsymbol{x})$ is quadratic and constraint functions $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are linear.



6.3.2. Quadratic Programming Overview over Algorithm

Active-Set Strategy Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Solve *Lagrange-Newton* equations for active constraints to determine search direction $\boldsymbol{s}^{(k)}$ and the corresponding Lagrange multipliers $\boldsymbol{\lambda}^{(k+1)}$ and $\boldsymbol{\mu}^{(k+1)}$.

   If $(\boldsymbol{s}^{(k)} \neq 0$ and $g_j\left(\boldsymbol{x}^{(k)} + \boldsymbol{s}^{(k)}\right) \leq 0)$: $\alpha^{(k)} = 1$, for $j \in J_g$. 1)

   elseif $(\boldsymbol{s}^{(k)} \neq 0$ and $g_j\left(\boldsymbol{x}^{(k)} + \boldsymbol{s}^{(k)}\right) > 0)$: Calculate step size $\alpha^{(k)}$. 2)
   Add most violated constraint g_j

   elseif $(\boldsymbol{s}^{(k)} = 0$ and $\min_{j \in J_{g,a}} \left(\mu_j^{(k+1)}\right) < 0)$: Remove constraint. 3)

   elseif $(\boldsymbol{s}^{(k)} = 0$ and $\min_{j \in J_{g,a}} \left(\mu_j^{(k+1)}\right) > 0)$: Stop. 4)

   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)} \boldsymbol{s}^{(k)}$, set $k = k + 1$

   Convergence test. If violated, repeat 2.

(Gerdts, 2015)

6.3.4. Lagrange-Newton Equations

Definition of gradient for this lecture (different from Papalambros):

$$d\boldsymbol{v} = \left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}\right)^{\boldsymbol{T}} d\boldsymbol{x} = (\nabla \boldsymbol{v})^T d\boldsymbol{x} = \nabla \boldsymbol{v}^T d\boldsymbol{x}$$

Solve Lagrange-Newton equations for active constraints

- The Lagrange-Newton equations = stationarity conditions of the KKT-conditions *for quadratic problems with equality constraints and active inequality constraints*:

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\boldsymbol{T}} \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^{\boldsymbol{T}} \boldsymbol{g}_{\boldsymbol{a}}(\boldsymbol{x}) \\ &= \frac{1}{2} \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{e}^T \boldsymbol{x} + z + \boldsymbol{\lambda}^{\boldsymbol{T}} (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{\mu}^{\boldsymbol{T}} (\boldsymbol{C}\boldsymbol{x} + \boldsymbol{d}) \end{aligned}$$

$$\frac{\partial L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu})}{\partial \boldsymbol{x}} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{e} + \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{\lambda} + \boldsymbol{C}^{\boldsymbol{T}}\boldsymbol{\mu} = \boldsymbol{0}$$

$$\frac{\partial L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu})}{\partial \boldsymbol{\lambda}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0}$$

$$\frac{\partial L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} = \boldsymbol{0}$$

- This can be directly solved for $\boldsymbol{x}^{(k+1)}$:

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^{\boldsymbol{T}} & \boldsymbol{C}^{\boldsymbol{T}} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{(k+1)} \\ \boldsymbol{\lambda}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{e} \\ \boldsymbol{b} \\ \boldsymbol{d} \end{bmatrix}$$

- Or for an increment $\boldsymbol{s}^{(k)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$:

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^{\boldsymbol{T}} & \boldsymbol{C}^{\boldsymbol{T}} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^{(k)} \\ \boldsymbol{\lambda}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{e} \\ \boldsymbol{b} \\ \boldsymbol{d} \end{bmatrix} - \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^{\boldsymbol{T}} & \boldsymbol{C}^{\boldsymbol{T}} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{(k)} \\ 0 \\ 0 \end{bmatrix} = - \begin{bmatrix} \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{e} \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \boldsymbol{s}^{(k)} \\ \boldsymbol{\lambda}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^{\boldsymbol{T}} & \boldsymbol{C}^{\boldsymbol{T}} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{e} \\ 0 \\ 0 \end{bmatrix}$$

6.4.2. SQP Overview over Algorithm

SQP Algorithm

1. For $k = 0$: Select start vector $\boldsymbol{x}^{(k)}$.
2. For $k \geq 0$: Derive quadratic sub-problem.
   Solve quadratic sub-problem to determine search direction $\boldsymbol{s}^{(k)}$ and the corresponding Lagrange multipliers $\boldsymbol{\lambda}^{(k+1)}$ and $\boldsymbol{\mu}^{(k+1)}$.
   Compute step size $\alpha^{(k)}$.
   Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$.
   Set $k = k + 1$.
   Convergence test. If violated, repeat 2.

6.4.3. SQP Quadratic Sub-Problem

Derive quadratic sub-problem

$$\begin{bmatrix} \boldsymbol{Q}^{(k)} & \left(\boldsymbol{A}^{(k)}\right)^T \\ \boldsymbol{A}^{(k)} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{s}^{(k)} \\ \boldsymbol{\lambda}^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \nabla f^{(k)} \\ \boldsymbol{h}^{(k)} \end{bmatrix}$$

$$\begin{aligned} \boldsymbol{Q}^{(k)}\boldsymbol{s}^{(k)} + \left(\boldsymbol{A}^{(k)}\right)^T \boldsymbol{\lambda}^{(k+1)} + \nabla f^{(k)} &= 0 \\ \boldsymbol{A}^{(k)}\boldsymbol{s}^{(k)} + \boldsymbol{h}^{(k)} &= 0 \end{aligned}$$

$$\Rightarrow \hat{f}\left(\boldsymbol{s}^{(k)}\right) = \frac{1}{2}\left(\boldsymbol{s}^{(k)}\right)^T \boldsymbol{Q}^{(k)}\boldsymbol{s}^k + \nabla f^{(k)}\boldsymbol{s}^{(k)} + f^{(k)}$$

$$\begin{aligned} & \min \hat{f}\left(\boldsymbol{s}^{(k)}\right) \\ \text{subject to:} \quad & \boldsymbol{A}^{(k)}\boldsymbol{s}^{(k)} + \boldsymbol{h}^{(k)} = \boldsymbol{0} \end{aligned}$$

with $\boldsymbol{Q}^{(k)} = \nabla^2_{\boldsymbol{xx}} L^{(k)} = \nabla^2 f^{(k)} + \left(\boldsymbol{\lambda}^{(k)}\right)^T \nabla^2 \boldsymbol{h}^{(k)}$

with $\boldsymbol{A}^{(k)} = \nabla \boldsymbol{h}^{(k)}$

- Solution vector: $\begin{bmatrix} \boldsymbol{s}^{(k)} \\ \boldsymbol{\lambda}^{(k+1)} \end{bmatrix}$
- For inequality constraints an active-set strategy is utilized according to the QP approach.

6.5.1. Evolutionary Algorithms Introduction

Typical steps in evolutionary algorithms:

- **Recombination (Crossover)** combines the characteristics of parent individuals randomly to create new children.
- **Mutation** alters properties of parent individuals to pass them on to the next generation.
- **Evaluation** assesses the fitness of each individual.
- **Selection** defines a new subset of individuals out of the population

6.5.2. Overview over Evolutionary Algorithms

Evolutionary Algorithms

1. For $k = 0$: Initialization (Select an initial set of vectors $\boldsymbol{x}_A^{(k)}$) with $A = 1 \dots N$
   N is the population size
2. For $k \geq 0$:
   - Mutation
   - Recombination
   - Evaluation
   - Selection

   (Set $\boldsymbol{x}_A^{(k+1)} = \boldsymbol{x}_A^{(k)} + \Delta\boldsymbol{x}_A^{(k)}$)

   Set $k = k + 1$.

   Convergence test. If violated, repeat 2.

- There are variations, e.g., with respect to order.

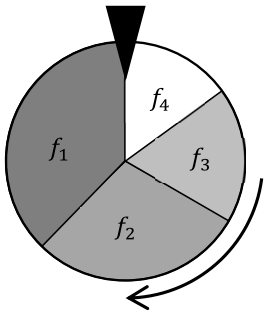
6.5.3. Genetic Algorithms

Selection (Parents):

- Pairs of individuals (Parents) are selected based on the **evaluation** of their fitness score. Individuals with high fitness scores have a better chance of being selected for reproduction.

- One Approach: **Roulette Wheel Selection** → Probability of selection p_A is directly proportional to fitness function value $f_A(\boldsymbol{x}_A^{(k)})$.

$$p_A = \frac{f_A(\boldsymbol{x}_A^{(k)})}{\sum_{A=1}^{N} f_A(\boldsymbol{x}_A^{(k)})}$$

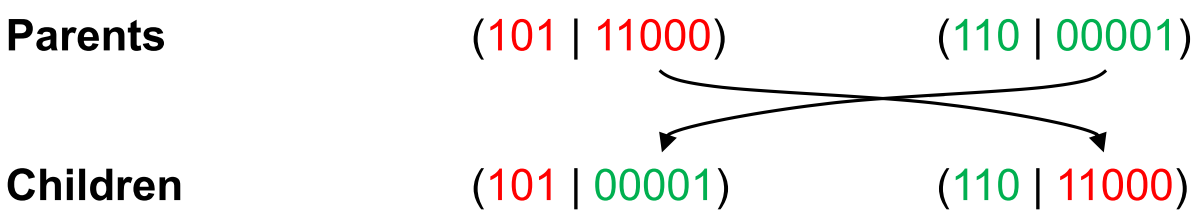


- Besides **proportional** selection operators, **elitist** approaches select only the fittest individuals for reproduction (→ Tradeoff between exploitation and exploration).

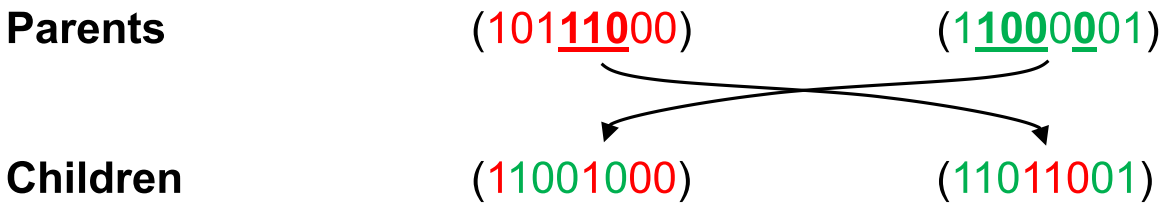
6.5.3. Genetic Algorithms

Crossover:

- **Single point crossover** can be applied to binary chromosomes. In this case the weighting factor r determines the location of a chromosome split.

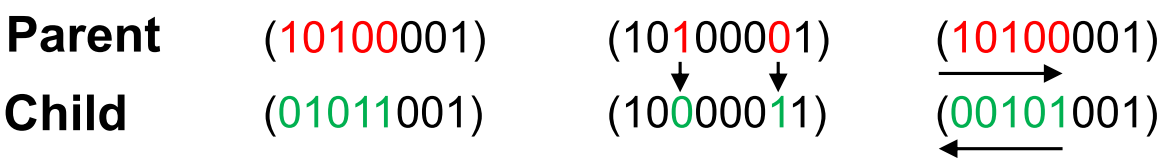


- In uniform crossover each binary gene is chosen from one parent.



6.5.3. Genetic Algorithms

Mutation:



- Mutations are defined as minor **random** tweaks to the chromosomes. Operators like **Bit Flip**, **Swap** and **Inversion** alter chromosome sections to maintain and introduce diversity to the population.

Selection (Next Generation):

- For the formation of the next generation, **age-based** and **fitness-based** approaches exist.
  - For fitness-based selectors, parents and/or children with good fitness values in the **evaluation** are maintained in the population.
  - In age-based selection an individual remains in the in the population for a finite generation, after that, it is kicked out of the population no matter how good its fitness is.

6.5.4. Differential Evolution

Mutation:

- For a **reference design** $\boldsymbol{x}_A^{(k)}$, three individuals $\boldsymbol{x}_{r1}^{(k)}$, $\boldsymbol{x}_{r2}^{(k)}$ and $\boldsymbol{x}_{r3}^{(k)}$ are randomly chosen from the population.
- A **donor vector** $\boldsymbol{v}_\mathrm{A}$ is defined as the weighted difference of two individuals to the third

$$\boldsymbol{v}_A^{(k+1)} = \boldsymbol{x}_{r1}^{(k)} + F(\boldsymbol{x}_{r2}^{(k)} - \boldsymbol{x}_{r3}^{(k)})$$

- The **mutation factor** (differential constant) is usually chosen between $F \in [0.5,1]$.

Recombination:

- In recombination the cross over constant CR is assigned to control the probability of the reference design to be mutated

$$u_{i,A}^{(k+1)} = \begin{cases} v_{i,A}^{(k+1)} & \text{if rnd} \leq CR \\ x_{i,A}^{(k)} & \text{if rnd} > CR \end{cases}$$

- The cross over constant CR is chosen between $CR \in [0,1]$
- $\boldsymbol{u}_A$ is called the trial vector

6.5.4. Differential Evolution

Evaluation and Selection:

- The reference vectors $\boldsymbol{x}_A^{(k)}$ and the trial vectors trial vectors $\boldsymbol{u}_A^{(k+1)}$ are evaluated and compared to each other
- Those with lower objective function values are selected for the next generation $(k+1)$.
- For example: $f\left(\boldsymbol{u}_{\mathbf{1}}^{(2)}\right) < f\left(\boldsymbol{x}_1^{(1)}\right) \rightarrow \boldsymbol{u}_{\mathbf{1}}^{(2)}$ is selected for $k = 2$

  $f\left(\boldsymbol{u}_{\mathbf{6}}^{(5)}\right) > f\left(\boldsymbol{x}_6^{(4)}\right) \rightarrow \boldsymbol{x}_{\mathbf{6}}^{(4)}$ is selected for $k = 5$

6.5.5. Particle Swarm Optimization

Initialization:

- Algorithm starts with an initial set of particles created randomly within feasible design space comprising the swarm.

- Each particle defined by four factors:
  1. Position $\boldsymbol{x}_{\boldsymbol{A}}^{(\boldsymbol{k})}$
  2. Velocity $\boldsymbol{v}_{\boldsymbol{A}}^{(\boldsymbol{k})}$
  3. Individual best position $\boldsymbol{p}_{\boldsymbol{A}}^{(\boldsymbol{k})}$
  4. Individual best objective value $\boldsymbol{f}(\boldsymbol{p}_{\boldsymbol{A}}^{(\boldsymbol{k})})$

6.5.5. Particle Swarm Optimization

Objective Update:

- Each particle **evaluates** the fitness function at current position $\boldsymbol{f}\left(\boldsymbol{x}_A^{(k)}\right)$.
- Individual best position $\boldsymbol{p}_A^{(k)}$ and swarm's global best position $\boldsymbol{p}_g^{(k)}$ are updated accordingly.

$$\boldsymbol{p}_A^{(k)} = \begin{cases} x_A^{(k)} & \text{if } f(x_A^{(k)}) < f(p_A^{(k-1)}) \\ p_A^{(k-1)} & \text{else} \end{cases} \qquad \boldsymbol{p}_g^{(k)} = \begin{cases} p_A^{(k)} & \text{if } f(p_A^{(k)}) < f(p_g^{(k-1)}) \\ p_g^{(k-1)} & \text{else} \end{cases}$$

6.5.5. Particle Swarm Optimization

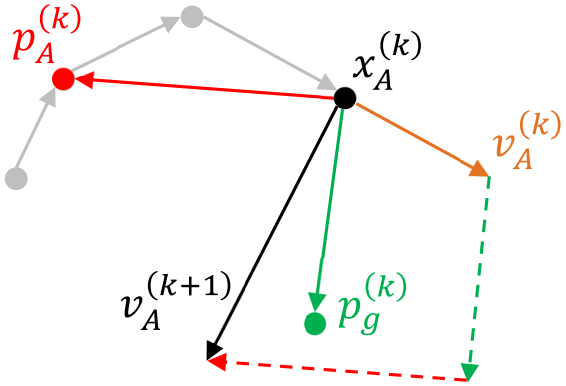
Velocity Update:

- Velocity update scheme is inspired by bird swarm analogy. New velocity $\boldsymbol{v}_A^{(k+1)}$ results from linear combination of current velocity vector $\boldsymbol{v}_A^{(k)}$, individual best position $\boldsymbol{p}_A^{(k)}$ and swarm's best position $\boldsymbol{p}_g^{(k)}$

Inertia — Cognitive Component — Social Component

$$\boldsymbol{v}_A^{(k+1)} = w\boldsymbol{v}_A^{(k)} + \xi_1\left(\boldsymbol{p}_A^{(k)} - \boldsymbol{x}_A^{(k)}\right) + \xi_2\left(\boldsymbol{p}_g^{(k)} - \boldsymbol{x}_A^{(k)}\right)$$

$$\boldsymbol{x}_A^{(k+1)} = \boldsymbol{x}_A^{(k)} + \boldsymbol{v}_A^{(k+1)}$$



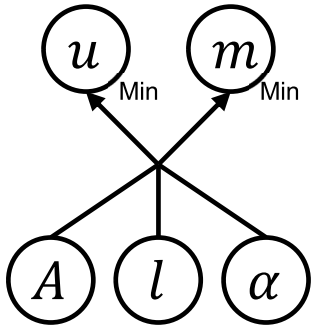
- The parameter w (weight) and the random uniform variables ξ_1, ξ_2 control particle velocity in a certain direction. Tuning of these parameters allows switching between exploration and exploitation.

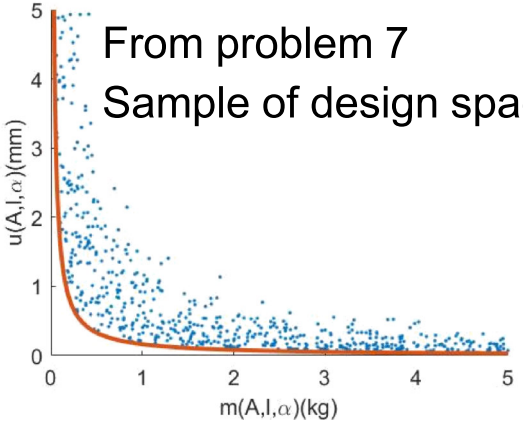
7.2.2. Pareto Optimality

A design $\boldsymbol{x}^*$ is **non-dominated** or **pareto-optimal**, if there is no other design $\boldsymbol{x}$ with $\forall j\colon\ f_j(\boldsymbol{x}) \leq f_j(\boldsymbol{x}^*)$ with $j = 1, \ldots, m$

- The Pareto front is the set of non-dominated designs.
- Advantage of Pareto optimization:
  - Provides full information to designer
  - No need for weighting & no undesired bias by weighting
- Disadvantage:
  - Many solutions → designer still needs to work
  - Visualization limited to 2d (with more effort to 3d)
- Algorithms:
  - Normal-Boundary Intersection (NBI)
  - Nondominated sorting genetic algorithm II (NSGA-II)

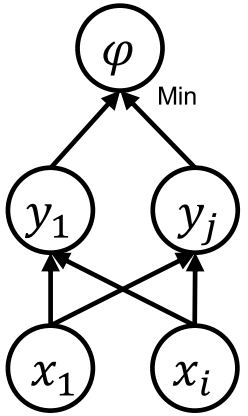
  MATLAB function: gamultiobj (see exercise)





7.3.1. Introduction

- Replace: $\min_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$ $\Rightarrow$ $\min_{\boldsymbol{x}} \varphi\big(\boldsymbol{f}(\boldsymbol{x})\big)$



- Formalism is similar to penalty/barrier functions.
  There, it was for numerical purposes.
  Here, the objective meta function expresses the designer‘s preference.

- Discuss 3 types of objective meta functions:
  (A) Linear
  (B) Quadratic
  (C) Worst-case

7.3.5. Overview

Linear:

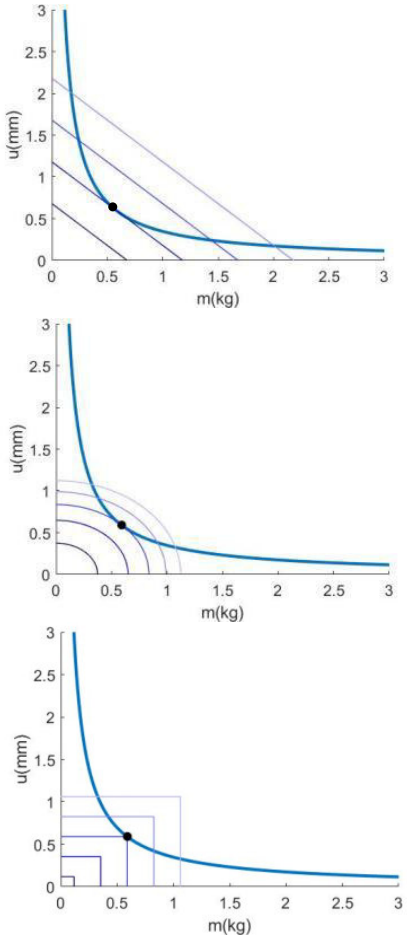
$$\varphi = \sum_{j=1}^{m} w_j y_j$$

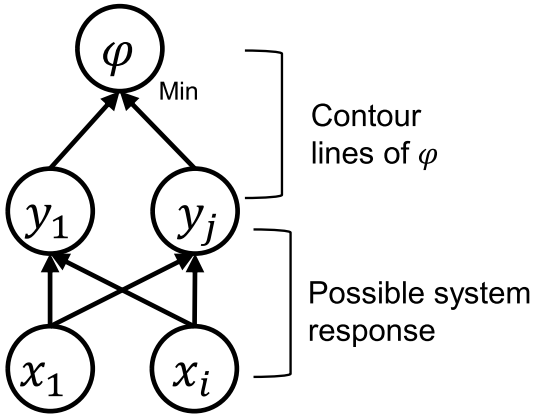
Quadratic:

$$\varphi = \sum_{j=1}^{m} w_j^2 y_j^2$$

Worst-Case:

$$\varphi = \max_j \{w_j y_j\}$$





Part-2: Warming up in Matlab

Matlab basics

- Basics warmup
- Optimisation warmup

End of Day-1

Thank you!

References

[1] Bletzinger; Munich, Lehrstuhl für Statik: Structural Optimization (Lecture Notes)