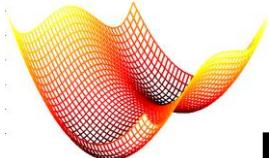




Optimierung

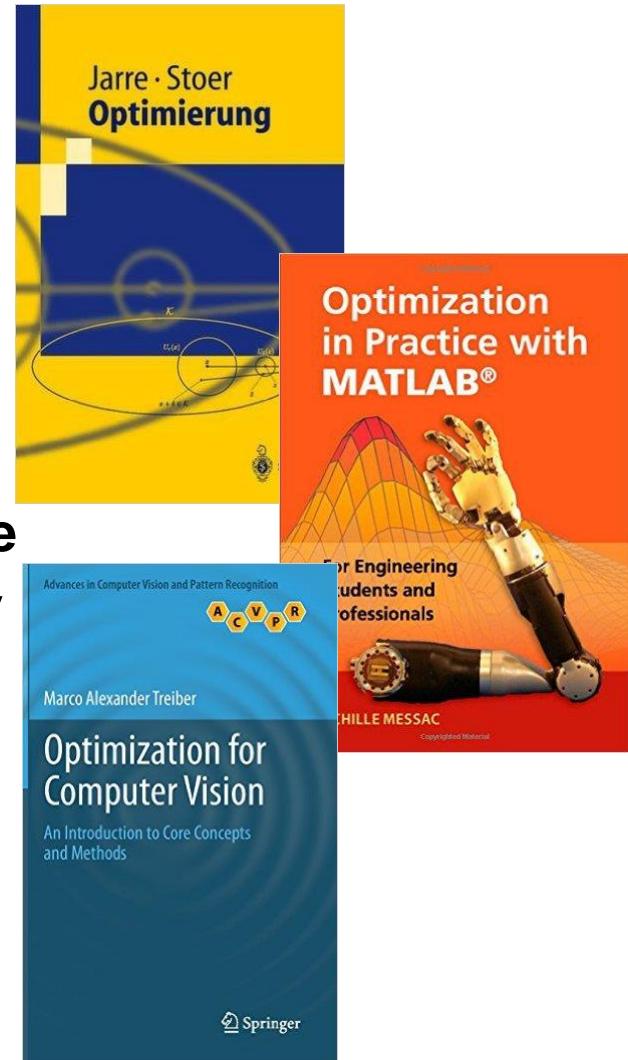
WS 2017/2018

- **Einleitung**
- **„Black-Box-Optimierung“ mit MATLAB**
- **Optimierung univariater Funktionen**
- **Optimierung multivariater Funktionen ohne NB**
- **Numerische Aspekte**
- **Ausgleichsrechnung (Regression)**
- **Lineare Optimierung**
- **Restringierte nichtlineare Optimierung**
- **Globale Optimierung**
- **Multikriterielle Optimierung**
- **Diskrete Optimierung**



Literatur

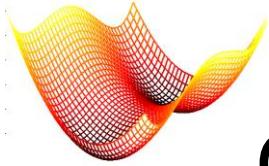
- **F.Jarre / J.Stoer:** “Optimierung”, Springer-Verlag 2004, ISBN 978-3540435754
- **A. Messac:** „Optimization in Practice with MATLAB“, Cambridge University Press, 2015, ISBN 978-1107109186
- **M. Treiber:** „Optimization for Computer Vision“, Springer-Verlag 2013, ISBN 978-1447152828





Einleitung

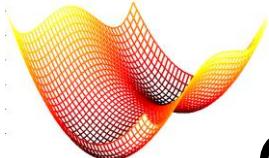
- **Motivation, Begriffsdefinitionen**
- **Einführendes Beispiel: Shading-Korrektur**
- **Modellierungsarten der Optimierungsprobleme**
- **(grundlegende) Lösungsstrategien**
- **Einteilung von Optimierungsproblemen**
- **Optimierung mit MATLAB**
- **Optimalitätsbedingungen**
- **(relevante) Eigenschaften von Zielfunktionen**
- **Vorgehensweise bei mathematischer Optimierung**



Optimierung – Begriffsdefinition (1)

Was ist Optimierung?

- “Die Kunst, die Dinge bestmöglich zu machen”
- Wikipedia: “Unter einem ***Optimum*** versteht man das beste erreichbare Resultat im Sinne eines Kompromisses zwischen verschiedenen Parametern oder Eigenschaften unter dem Aspekt einer Anwendung, einer Nutzung oder eines Ziels. Die Suche nach dem Optimum unter gegebenen Voraus- und Zielsetzungen nennt man ***Optimierung***“
- **Finde eine Lösung zu einem Problem, die hinsichtlich bestimmter Kriterien “am Besten” ist**



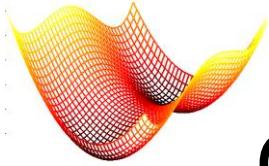
Optimierung – Begriffsdefinitionen (2)

Aspekt	Beispiel
Problem	Navigation
Kriterium	Weglänge, Zeit
Lösungsmenge	Straßennetz
Optimum	Kürzester / schnellster Weg
Randbedingungen	Keine Maut, Staus vermeiden, ...



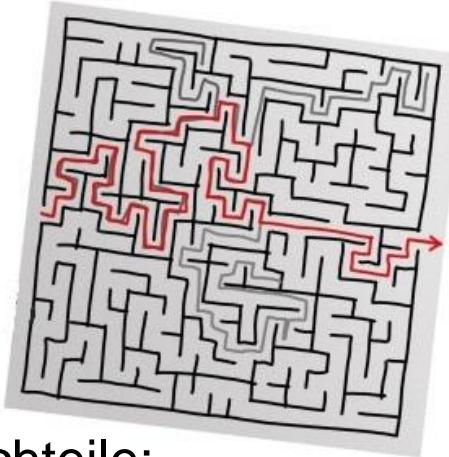
Marco Treiber, Fakultät FK 07, marco.treiber@hm.edu



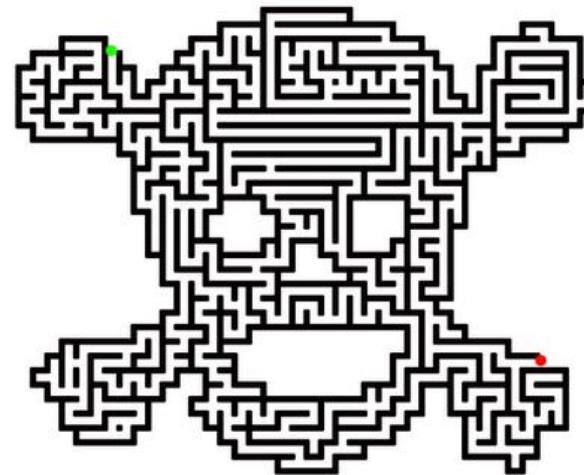


Optimierungsstrategien (1)

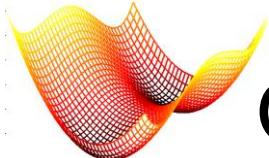
- In der Praxis oft anzutreffendes Prinzip: **Expertenwissen**
- Im Prinzip: „intelligentes“ try and error



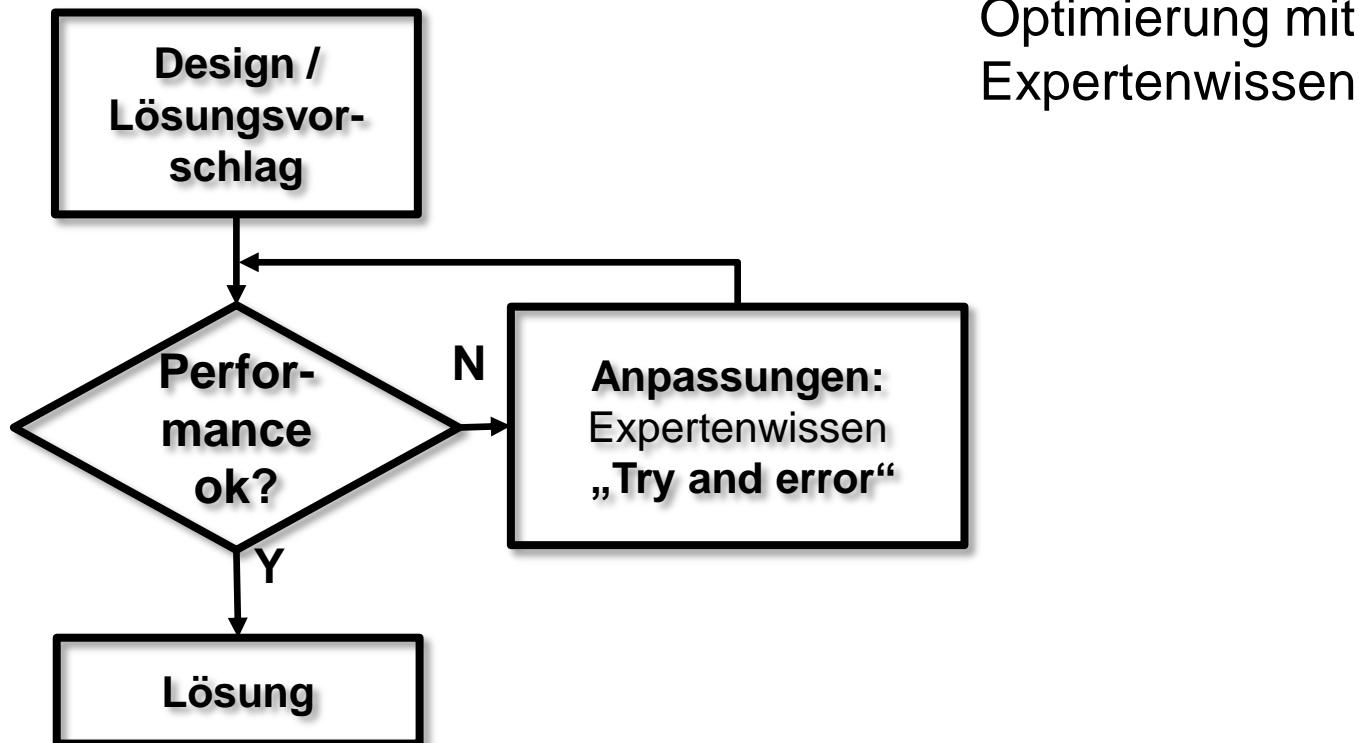
Nachteile:

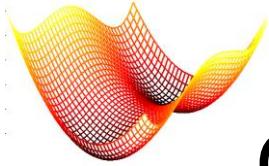


- Unstrukturiert → ineffektiv, redundant, zeitaufwändig
- Lösung wird nur mit Glück gefunden

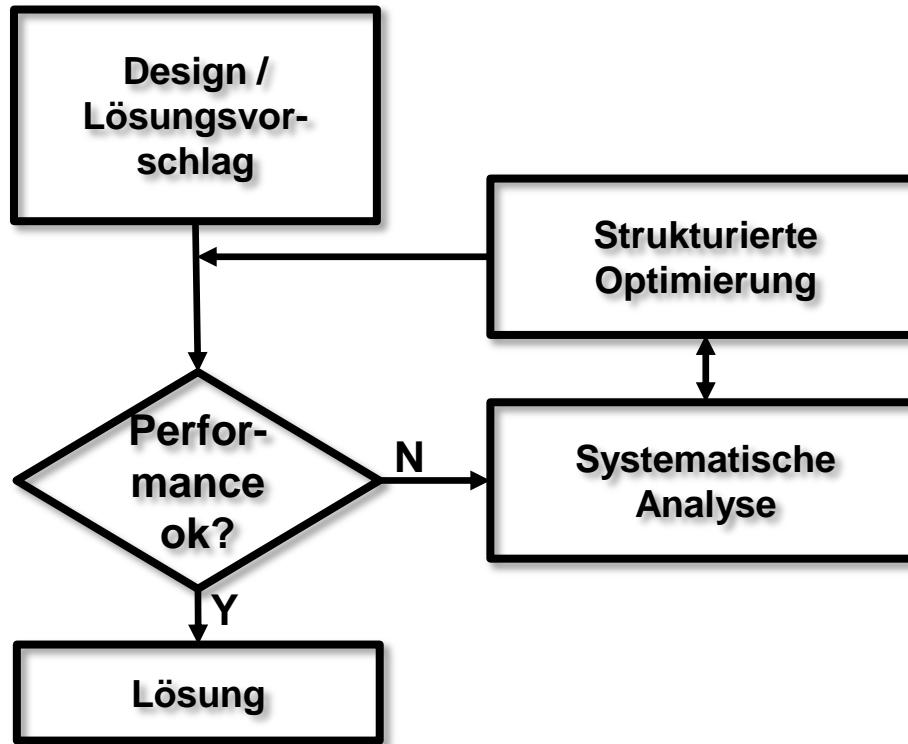


Optimierungsstrategien (2): Expertenwissen





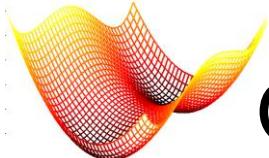
Optimierungsstrategien (3)



„systematische“
Optimierung

Vorteile:

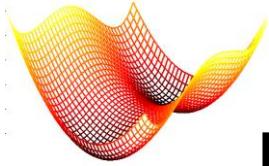
- systematisch → effizient
- computergestützt
- verlässlich / „sicher“



Optimierung – mathematische Formulierung

Aspekt	Beispiel	Mathem. Formulierung
Problem	Navigation	
Kriterium	Weglänge, Zeit	Zielfunktion $f(\mathbf{x})$
Lösungsmenge	Straßennetz	S , z.B. \mathbb{R}^N
Optimum	Kürzester / schnellster Weg	$\mathbf{x}^* = \underset{\mathbf{x} \in S}{\operatorname{argmin}} f(\mathbf{x})$
Randbedingungen	Keine Maut, Staus vermeiden, ...	$g(\mathbf{x}) \leq 0$ $h(\mathbf{x}) = 0$

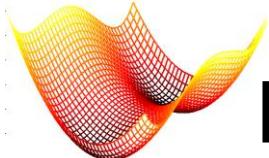
\mathbf{x} : Zielvariablen, „freie“ Variablen



Mathematische Optimierung

Mathematische Optimierung ist der Prozess des Maximierens (oder) Minimierens einer (oder mehrerer) Zielfunktionen unter der Beachtung des Einhaltens von Randbedingungen durch systematische Modifikation von Variablen, die sowohl die Zielfunktion als auch die Randbedingungen beeinflussen (können).

- Mathematische Formulierung erlaubt:
 - systematische (effiziente) Lösungssuche
 - computergestützte (numerische) Lösungen
 - systematische Tests, Sensitivitätsanalyse



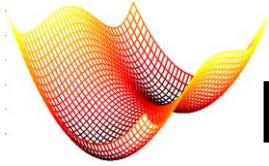
Mathematische Optimierung - Gefahren



"Sweetheart, my neural net
predicts that you and I are
98.9% compatible.
Will you be my Valentine?"

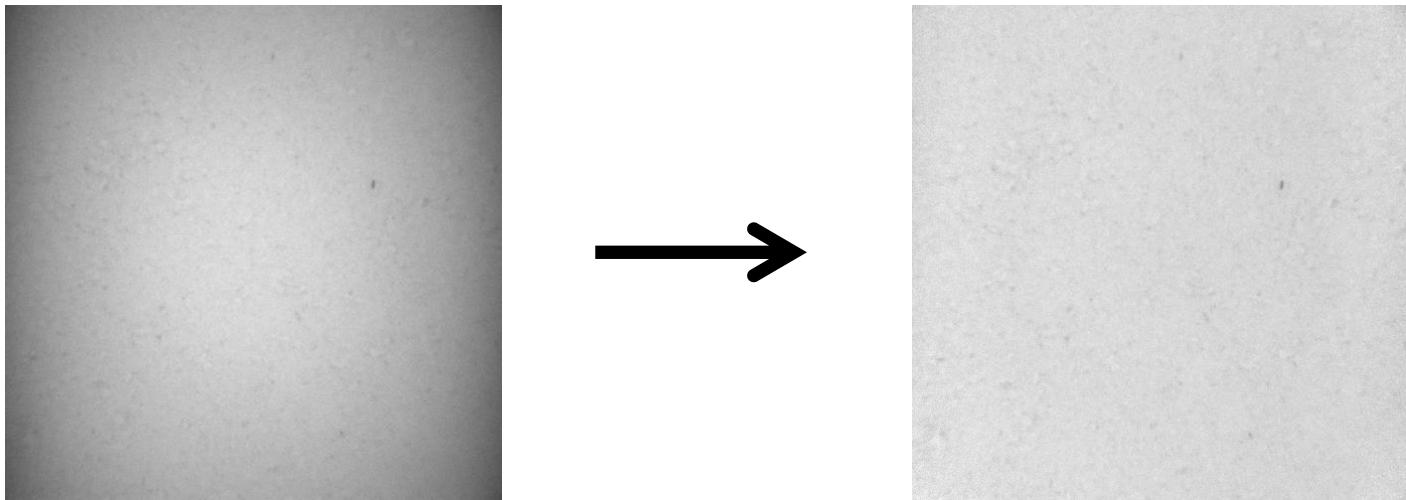
**Modellierung ist
genauso wichtig
wie Optimierungs-
verfahren selbst!**

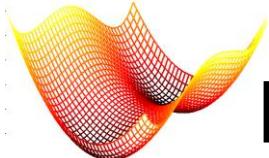
- Mathematische Lösung ausschließlich im Rahmen des definierten Modells
- Zeitbedarf hängt von der Modellierung ab
- Fehler durch Numerik



Einführendes Beispiel: Shading-Korrektur (1)

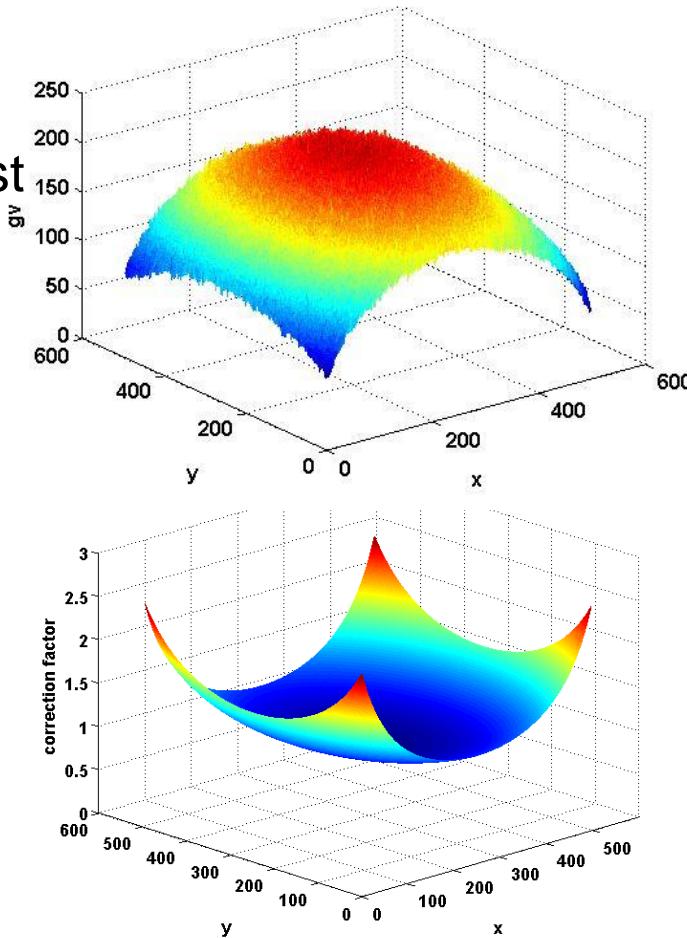
- **Aufgabe:** Bilder sind aufgrund physikalischer Effekte und inhomogener Beleuchtung an den Rändern dunkler („shading“) → **Korrektur, so dass ein homogenes Objekt auch mit gleichbleibender Helligkeit erscheint**

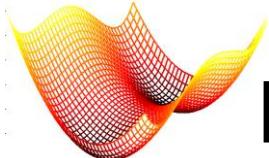




Einführendes Beispiel: Shading-Korrektur (2)

- 3D-Darstellung zeigt: Bild $I(x, y)$ lässt sich gut als Polynom approximieren
 - Bestimme Koeffizienten c eines Polynoms $p(c, x, y)$ so, dass **Fehler** zwischen $p(c, x, y)$ und $I(x, y)$ **minimiert** wird
 - **Optimierungsaufgabe**
- Korrektur mit Kehrwert $1/p(c, x, y)$ des Polynoms:
$$I_{eq}(x, y) = I(x, y) \cdot 1/p(c, x, y)$$
- **Hoffnung:** diese Korrektur lässt sich auf andere Bilder übertragen





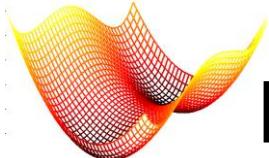
Einführendes Beispiel: Shading-Korrektur (3)

- **Zielfunktion:** Fehlerquadratsumme zwischen Trainingsbild (Aufnahme eines Objektes konstanter Helligkeit) und approximierendem Polynom
- Substitution $u = x - x_c$, $v = y - y_c$ ((x_c, y_c) ist Bildmittelpunkt) sowie Normierung des Eingangsbildes $\tilde{I}(x, y) = I(x, y)/I_{\max}$ ergibt:

$$f_{sc}(\mathbf{c}) = \sum_{u,v} \left(p(\mathbf{c}, u, v) - \tilde{I}(u, v) \right)^2$$

- Beobachtung: Shading symmetrisch zur Bildmitte → nur **Komponenten gerader Ordnung notwendig:**

$$p(c, u, v) = c_0 + c_1 \cdot u^2 + c_2 \cdot v^2 + c_3 \cdot u^2 \cdot v^2 + c_4 \cdot u^4 + c_5 \cdot v^4 + \dots$$



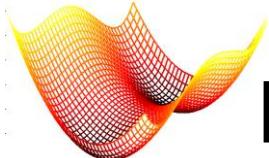
Einführendes Beispiel: Shading-Korrektur (4)

- $p(\mathbf{c}, u, v)$ ist **linear** in den (gesuchten) Koeffizienten \mathbf{c} → lineare Ausgleichsrechnung (**lin. Regression**). In Vektor-Notation:

$$p(\mathbf{c}, u, v) = \mathbf{a}^T \cdot \mathbf{c}; \mathbf{a}^T = [1 \quad u^2 \quad v^2 \quad u^2v^2 \quad u^4 \quad v^4 \quad \dots]$$

- Zielfunktion $f_{sc}(\mathbf{c})$ als **Matrix-Vektor-Produkt** bei Datenpunkten:

$$\begin{aligned} f_{sc}(\mathbf{c}) &= \sum_{n=1}^N (\mathbf{a}_n^T \cdot \mathbf{c} - \tilde{I}_n)^2 \\ &= \sum_{n=1}^N [\mathbf{a}_n^T \cdot \mathbf{c} - \tilde{I}_n]^T \cdot [\mathbf{a}_n^T \cdot \mathbf{c} - \tilde{I}_n] \\ &= [\mathbf{A} \cdot \mathbf{c} - \tilde{\mathbf{I}}]^T \cdot [\mathbf{A} \cdot \mathbf{c} - \tilde{\mathbf{I}}] \\ &= \mathbf{c}^T \cdot \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{c} - 2\mathbf{c}^T \cdot \mathbf{A}^T \cdot \tilde{\mathbf{I}} + \tilde{\mathbf{I}}^T \cdot \tilde{\mathbf{I}} \end{aligned}$$



Einführendes Beispiel: Shading-Korrektur (5)

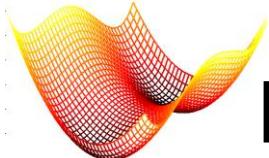
- Zielfunktion $f_{sc}(\mathbf{c})$ ist **quadratisch**:
 - Hat **genau ein Extremum** (hier: Minimum)
 - Position des Minimums dort, wo die Ableitung $\partial f_{sc}(\mathbf{c})/\partial \mathbf{c} = \mathbf{0}$ wird:

$$\partial f_{sc}(\mathbf{c})/\partial \mathbf{c} = 2\mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{c} - 2\mathbf{A}^T \cdot \tilde{\mathbf{I}} \equiv \mathbf{0}$$

$$\mathbf{A} = \begin{bmatrix} 1 & u_1^2 & v_1^2 & \dots \\ 1 & u_2^2 & v_2^2 & \dots \\ 1 & u_3^2 & v_3^2 & \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

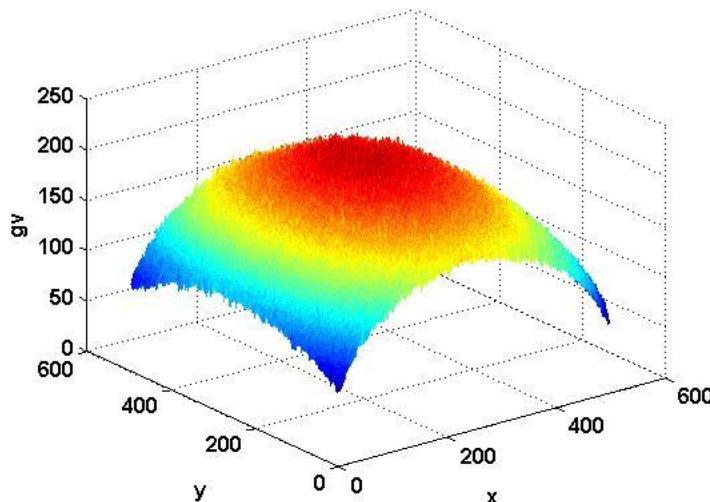
- Minimum \mathbf{c}^* lässt sich durch **Lösen eines lin. Gleichungssystems** bestimmen:

$$\mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{c}^* = \mathbf{A}^T \cdot \tilde{\mathbf{I}}$$

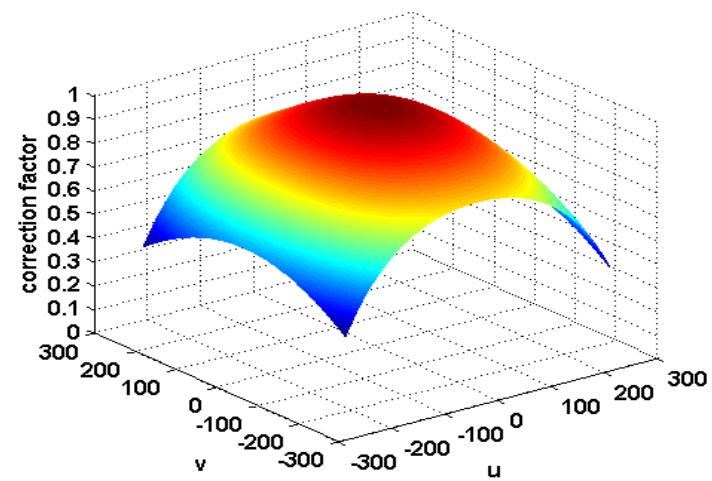


Einführendes Beispiel: Shading-Korrektur (6)

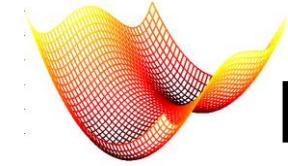
- Approximation durch quadratische Funktion ist sehr gut möglich:



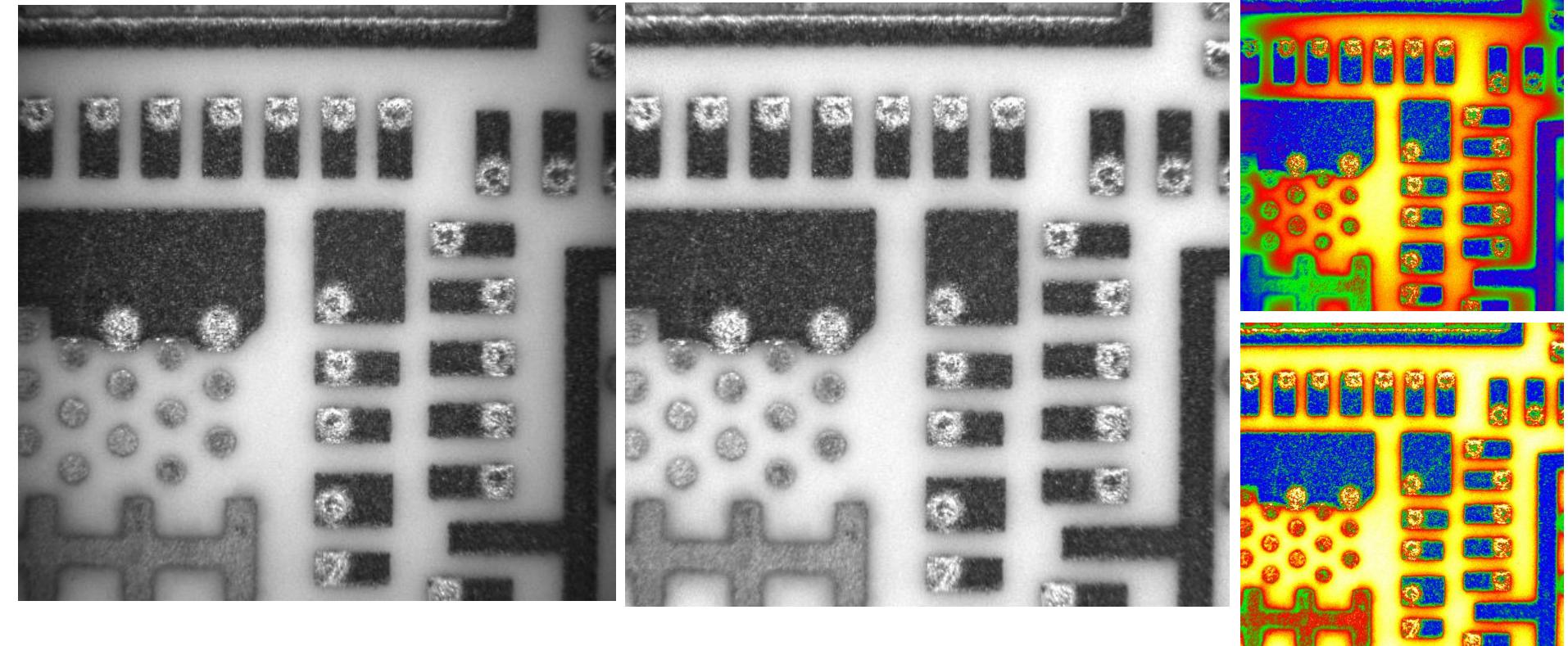
3D-Darstellung Trainingsbild

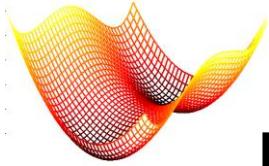


Quadratische Approximation



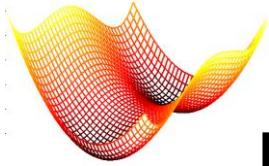
Einführendes Beispiel: Shading-Korrektur (7)





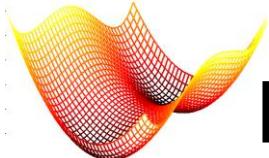
Modellierungsarten (1)

- **Physikalisch/Analytisch:** Zielfunktion leitet sich aus der mathem. Formulierung von physikalischen Gesetzmäßigkeiten ab
- **Simulationsbasiert:** Zielfunktion wird durch ein Simulationsmodell beschrieben. Optimierung durch wiederholtes Durchführen der Simulation
 - Geeignet zur Berücksichtigung von zufälligen Ereignissen.
 - Bsp: Wettervorhersage, Finite Elemente
 - In der Regel komplex



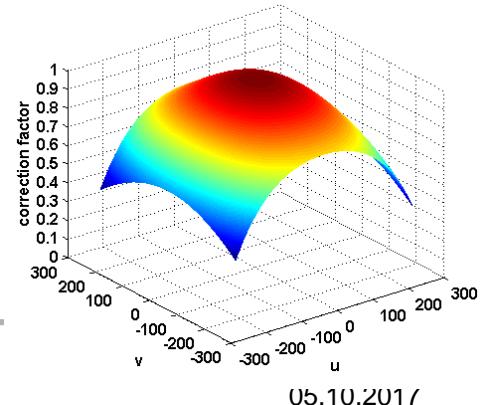
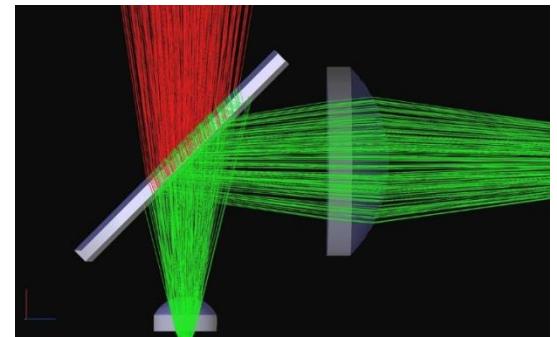
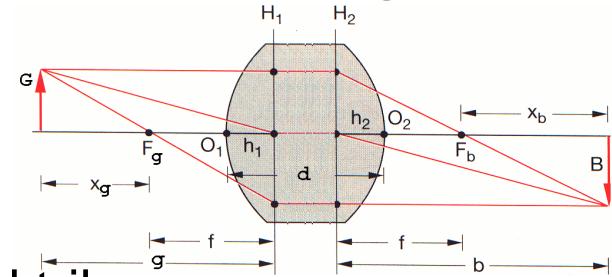
Modellierungsarten (2)

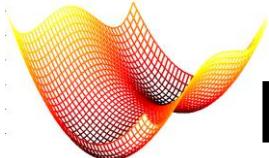
- „**surrogate Modelling**“: „**Ersatz-Modellierung**“ der Zielfunktion durch Optimierung der Parameter von vordefinierten Klassen von Funktionen
 - Bestimmen der Parameter anhand von Trainings-Beispielen
 - In der Regel keine physikalische Interpretation möglich, aber:
 - praktikable Komplexität des Modells (per Design)
 - Damit: Abstimmung auf Optimierungsverfahren und somit hinreichend schnelle Optimierung möglich



Modellierungsarten: Beispiel Shading-Korrektur

- Physikalisch:
 - Modellierung von Optik, Sensor, Elektrik,...
 - sehr komplex
- Simulation:
 - Raytracing
 - Variation der Beleuchtung
 - Zeitaufwändig
- Trainings-basiertes „Ersatz-Modell“:
 - effizient
 - Aber: unklare Gültigkeit bei sich verändernden Parametern!

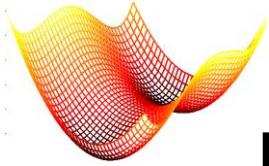




Modellierung: Ersatz-Modell bei Objekterkennung

- „Aufgabe: Erkennen von K Objektklassen
- Klassifikation anhand Wahrscheinlichkeits-Funktionen $p_k[\theta, I(x, y)]$
- Modellierung der $p_k[I(x, y)]$ durch „surrogate model“, da die direkte Abhangigkeit der p_k von $I(x, y)$ nicht darstellbar.
- Bestimmen der Parameter θ anhand von Trainings-Beispielen/Bildern: ermittle θ so, dass eine Test-Menge von Bildern bekannter Klassen optimal klassifiziert wird.

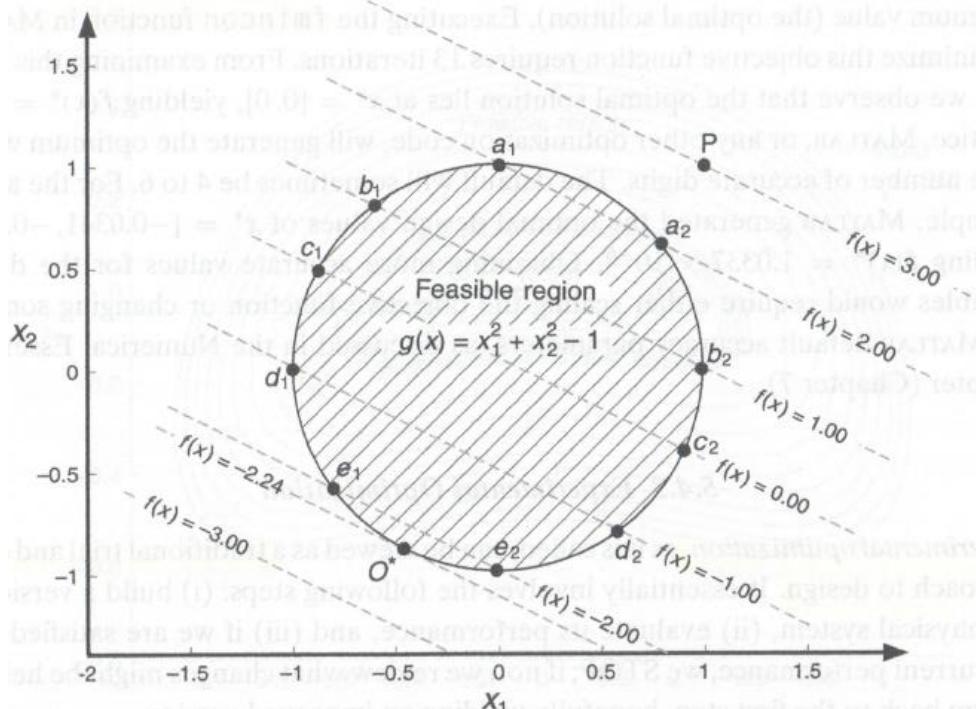
→ Ersatz-Modellierung ist auch dann anwendbar, wenn die „eigentliche“ Zielfunktion nicht modelliert werden kann



Lösungsstrategien (1)

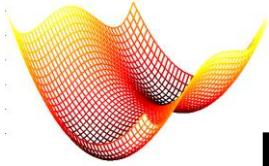
- **Experimentell:** traditionelles try-and-error, Einbeziehung von Expertenwissen
 - Nicht systematisch → ineffektiv, redundant, zeitaufw.
 - Keine Kontrolle, „wie gut“ die gefundene Lösung tatsächlich ist → Optimum wird nur mit Glück gefunden
 - Innovative Lösungen unwahrscheinlich
- **Graphisch:** „Ablesen“ der Lösung durch graphische Darstellung der Zielfunktion
 - Unmittelbare Lösung möglich
 - Nur bei einfachen Problemen anwendbar
 - Nicht automatisierbar

Lösungsstrategien (2)



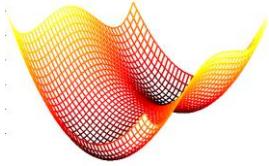
- Problemstellung:
 $\min f(\mathbf{x}) = x_1 + 2x_2$
s.t. $x_1 + x_2 \leq 1$

© Messac / Cambridge University Press



Lösungsstrategien (3)

- **Analytisch:** Finden der Lösung in geschlossener Form, d.h. analytische Berechnung des Minimums der Zielfunktion
 - Bsp.: lineare Ausgleichsrechnung
 - Schnell und exakt, aber für viele reale Probleme nicht realisierbar
- **Numerisch:** strukturierte Suche innerhalb des Lösungsraums, bis eine praktikable Lösung gefunden wurde
 - Algorithmen-basiert → systematisch, nicht „zufällig“
 - Iterativ
 - Computergest. Realisierung der Algorithmen → effizient
 - „Moderne“ Art der Optimierung



„Optimierungsprogramme“ (1)

- Optimierungsproblem wird häufig als „**Programm**“ bezeichnet (historischer Ursprung: „Program of Actions“ ist Bezeichnung für militärisches Logistik-Problem)
- **Mathematische Formulierung** eines Programms:

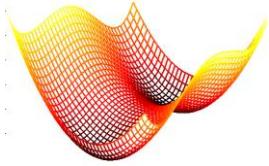
$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

Nebenbedingungen:

$$\mathbf{g}(\mathbf{x}) \leq 0$$

$$\mathbf{h}(\mathbf{x}) = 0$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$

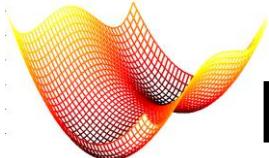


„Optimierungsprogramme“ (2)

- **Nebenbedingungen** (NB) stellen die **Machbarkeit** der Lösung sicher, z.B. positive Masse, Ausschluss nicht finanziabler Kosten, etc.
- Viele Modellierungen von Optimierungsproblemen werden erst durch Nebenbedingungen „sinnvoll“/plausibel und damit in der Praxis anwendbar
- **Maximierung ist äquivalent zur Minimierung der negativen Zielfunktion:**

$$\max_{\mathbf{x} \in S} f(\mathbf{x}) \triangleq \min_{\mathbf{x} \in S} -f(\mathbf{x})$$

→ Im weiteren Verlauf werden o.B.d.A. ausschließlich Minimierungsprobleme behandelt



Einteilung der Optimierungsprogramme (1)

- **linear vs. nichtlinear:** Sind f , g , und h alle linear in \mathbf{x} , so spricht man von einem „linearen Programm“ (LP), andernfalls liegt ein „nichtlineares Programm“ (NP) vor.
 - Lineare Programme sind einfacher und schneller zu lösen als NP.
 - Je höher die Nichtlinearität, umso „unsicherer“ ist die Lösung und umso größer ist die Empfindlichkeit auf numerische Effekte.
 - Mathematische Formulierung eines LP:

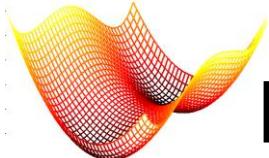
$$\min_{\mathbf{x} \in S} \mathbf{c}^T \cdot \mathbf{x}$$

$$\text{NB: } \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}; \mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}; \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$



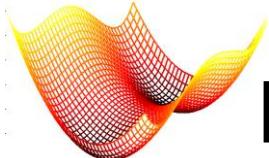
Einteilung der Optimierungsprogramme (2)

- **restringiert vs. nicht-restringiert:** nicht-restringierte Programme haben keine Nebenbedingungen
 - Die meisten realen Probleme haben Nebenbedingungen
 - Nebenbedingungen verkomplizieren die Lösung
 - Lösungsstrategie bei restringierten Programmen: Umformung in ein nicht-restringiertes Programm mit möglichst äquivalenter Lösung
 - Lineare Programme sind nur bei gleichzeitiger Existenz von Nebenbedingungen „sinnvoll“ lösbar. (ohne NB liegt die Lösung von LP stets im Unendlichen!)



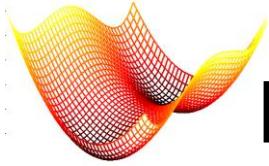
Einteilung der Optimierungsprogramme (3)

- **kontinuierlich vs. diskret:** Bei kontinuierlichen Problemen ist die Lösungsmenge aller Zielvariablen (Elemente x_i von \mathbf{x} ; $i = [1, \dots, N]$) eine Teilmenge von \mathbb{R}^N . Diskrete Probleme:
 - Binäre Programme: $x_i \in [0; 1]$
 - Integer-Programme: alle x_i ganzzahlig: $x_i \in \mathbb{Z}$
 - Allg. diskrete Programme: Lösungsmenge S enthält endliche Wertemenge (Elemente von S müssen aber nicht ganzzahlig sein)
 - Kombinatorische Probleme: gesucht: opt. Zuordnung
 - Diskrete Optimierungsprobleme sind i.d.R. schwieriger zu lösen als kontinuierliche Optimierungsprobleme



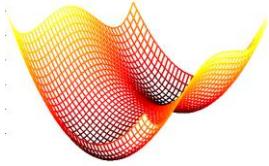
Einteilung der Optimierungsprogramme (4)

- Kontinuierliche Programme haben gelegentlich in der Praxis nur eine diskrete Lösungsmenge S , z.B. minimiere den Durchmesser eines Rohres, wobei nur endlich viele (genormte) Durchmesser verfügbar
- **Anzahl der Optima:** multimodale Zielfunktionen haben (im Gegensatz zu unimodalen Zielfunktionen) mehrere (lokale) Optima
 - Bei unimodalen Zielfunktionen genügt „lokale“ Minimierung, d.h. Suche nach Positionen, welche lokal den niedrigsten (höchsten) Funktionswert aufweisen
 - Multimodale Zielfunktionen sind weit schwieriger zu optimieren



Einteilung der Optimierungsprogramme (5)

- **Ein Kriterium vs. multikriterielle Optimierung:** bei mehreren (gleichzeitig) zu optimierenden Zielgrößen spricht man von multikriterieller Optimierung. Die Zielfunktion $f(\mathbf{x})$ ist dann vektorwertig.
 - In der Praxis sind oft mehrere Größen zu optimieren (z.B. Auto: max. Leistung, min. Kosten, min. Kraftstoffverbrauch, etc.)
 - In der Regel konkurrierende Ziele → Trade-Off nötig
 - Techniken zur Reduktion multikrit. Programme auf ein Kriterium, z.B. gewichtete Summe



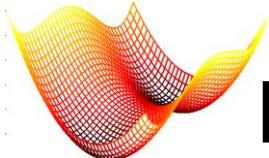
„wichtige“ Optimierungsstrategien

Von zentraler Bedeutung sind Optimierungsstrategien zur Lösung von Programmen mit folgenden Eigenschaften:

- kontinuierliche Lösungsmenge
- unimodale Zielfunktion → lokale Optimierung ausreichend
- skalare Zielfunktion → nur eine Größe zu optimieren
- keine Nebenbedingungen

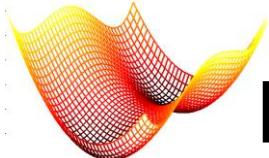
weil für o.g. Programme entwickelte Algorithmen durch Umformung bzw. Modifikationen auch zur Lösung anderer, praxisrelevanter Programme verwendet werden können.

- Beispiele: Erweiterung der Zielfunktion zur „Auflösung“ von NB, gewichtete Summe bei multikrit. Optimierung, etc.



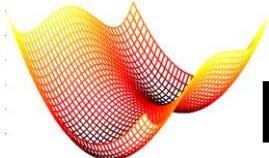
Lösen von Optimierungsaufgaben mit MATLAB (1)

- „**Black-Box-Optimierung**“: Zielfunktionen (+ ggf. Nebenbedingungen) müssen definiert werden, den Rest erledigt MATLAB ☺
- Funktion **fminsearch** und **fminbnd** in MATLAB integriert, viele weitere Funktionen in Add-on „Optimization Toolbox“, u.a.:
- Funktion **fmincon** zur nicht-linearen Optimierung (mit oder ohne NB)
- Funktion **linprog** zur linearen Optimierung (mit oder ohne NB)



Lösen von Optimierungsaufgaben mit MATLAB (2)

- Parametrierung der Optimierung mittels Struktur **optimset**, u.a.:
 - Maximale Anzahl Iterationen
 - Konvergenz-Kriterien
 - Verwendetes Optimierungsverfahren
 - Display-Optionen
 - Gleitkomma-Genauigkeit
 - Verfahren zur (numerischen) Berechnung der Ableitungen
 - ...



Nicht-Lineare Optimierung mit MATLAB (1)

- MATLAB-Funktion **fmincon** löst folgende nichtlinearen Optimierungsprobleme:

$$\underset{\mathbf{x} \in S}{\min} f(\mathbf{x})$$

Nebenbedingungen:

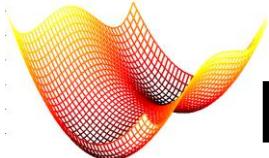
$$\mathbf{g}(\mathbf{x}) \leq 0$$

$$\mathbf{h}(\mathbf{x}) = 0$$

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$$

$$\mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$

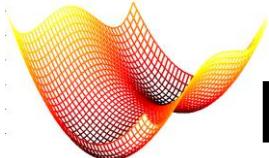


Nicht-Lineare Optimierung mit MATLAB (2)

- Syntax von **fmincon**:

```
[xopt, fopt] = fmincon('objfun', x0, A, b,  
Aeq, beq, xl, xu, 'nlconstrfun')
```

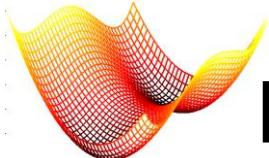
- [xopt, fopt]: Returnwerte: gesuchte Optimalposition \mathbf{x}^* bzw. Wert der Zielfunktion an dieser Stelle $f(\mathbf{x}^*)$
- x0: Startposition der Optimierung
- 'objfun' (oder @objfun): Name der Zielfunktion (wird in gleichnamigen m-file definiert)
- 'nlconstrfun' (oder @ nlconstrfun): Name der nichtlinearen NB (gleichnamiges m-file)



Nicht-Lineare Optimierung mit MATLAB (3)

- A , b : Koeffizienten der linearen Ungleichungen als NB
- A_{eq} , b_{eq} : Koeffizienten der linearen Gleichungen als NB
- x_l , x_u : untere (x_l) bzw. obere Begrenzung (x_u) des Lösungsraumes
- Explizite Definition der linearen NB aus Performance-Gründen
- NB können auch „weggelassen“ werden, z.B. $A = []$
- Syntax von `objfun` (zu definieren in `objfun.m`):

```
function funcval = objfun(x)
    funcval = < $f(\mathbf{x})$ >;
```



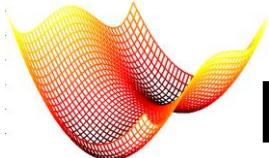
Nicht-Lineare Optimierung mit MATLAB (4)

- Syntax von `nlconstrfun` (zu definieren in `nlconstrfun.m`):

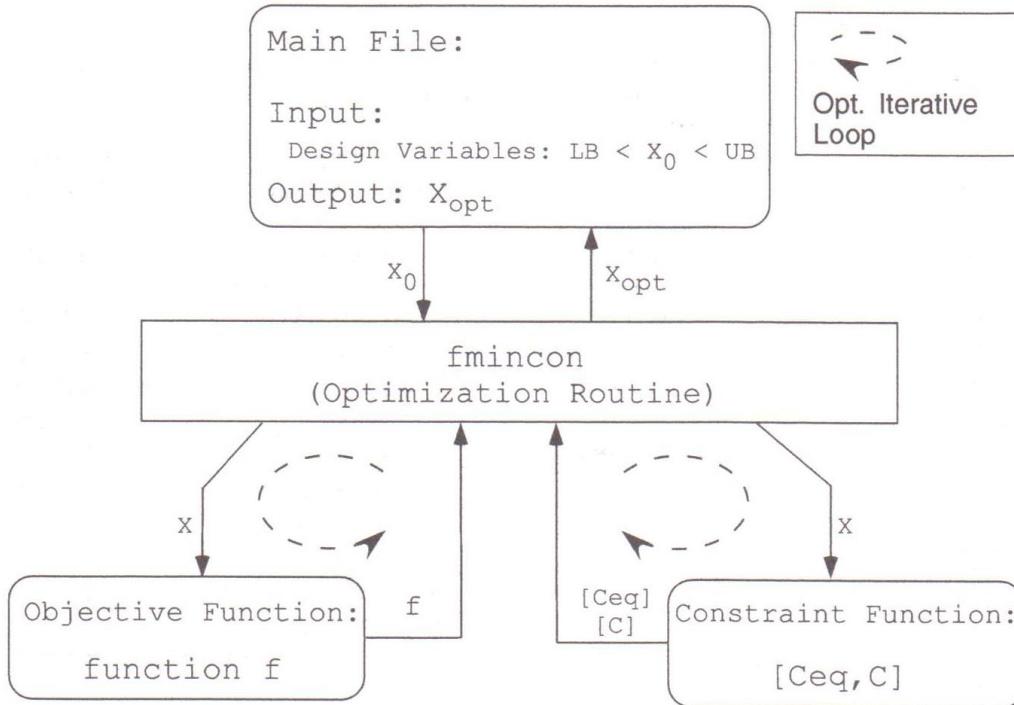
```
function [neq_constr, eq_constr] =  
nlconstrfun(x)  
    neq_constr(1) = <g1(x)>;  
    neq_constr(2) = <g2(x)>;  
    eq_constr(1) = <h1(x)>;  
    ...
```

- Wenn z.B. keine nichtlin. NB vorhanden sind:

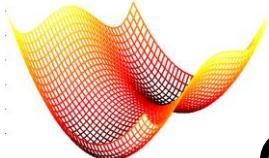
```
eq_constr = [];
```



Nicht-Lineare Optimierung mit MATLAB (5)



© Messac / Cambridge University Press



Optimierung mit MATLAB: Beispiel (1)

- Beispiel: Optimiere $f(\mathbf{x}) = x_1^2 + 10x_2^2 - 3x_1x_2$

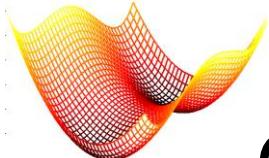
NB: $2x_1 + x_2 \geq 4$

$x_1 + x_2 \geq -5$

$-5 \leq x_1, x_2 \leq 5$

main.m:

```
x0 = [0; 0];  
A = [] ; b = [] ; %A=[-2 -1; -1 -1] ; b=[-4; 5];  
Aeq = [] ; beq= [] ;  
xl = [-5; -5] ; xu = [5; 5] ;  
[xopt, fopt] = fmincon('objfun', x0, A, b,  
Aeq, beq, xl, xu, 'nlconstrfun')
```



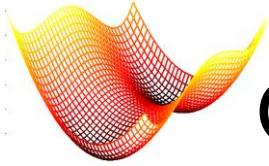
Optimierung mit MATLAB: Beispiel (2)

- objfun.m:

```
function funcval = objfun(x)  
funcval = x(1)^2 + 10*x(2)^2 -3*x(1)*x(2);
```

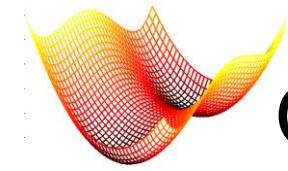
- nlconstrfun.m:

```
function [neq_constr, eq_constr] =  
nlconstrfun(x)  
  
neq_constr(1) = -2*x(1) - x(2) + 4;  
  
neq_constr(2) = -x(1) - x(2) - 5;  
  
eq_constr= [ ];
```

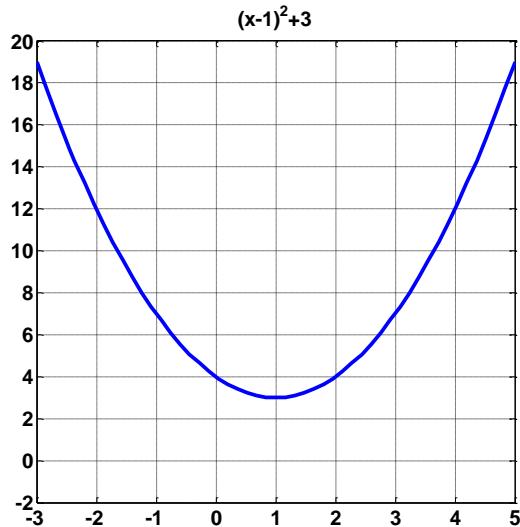


Optimalitätsbedingungen: univariate Funktionen (1)

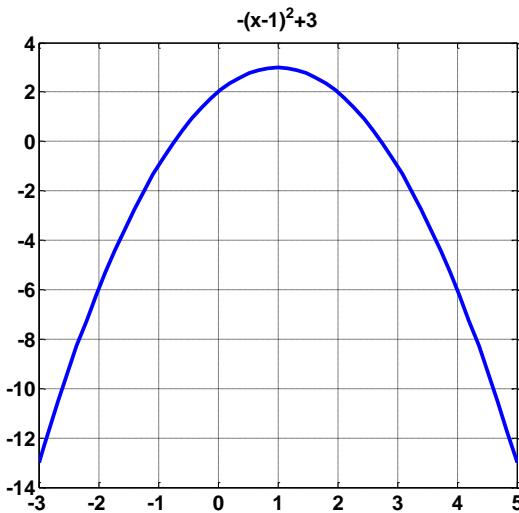
- **Notwendige Bedingung** für stationären Punkt x^* von $f(x)$:
1. Ableitung gleich 0: $f'(x^*) = 0$
- **Hinreichende Bedingung für Minimum**: 2. Ableitung
größer 0: $f''(x_{min}) > 0$ (Funktion ist in der Umgebung von x_{min} **konvex**)
- **Hinreichende Bedingung für Maximum**: 2. Ableitung
kleiner 0: $f''(x_{max}) < 0$ (Funktion ist in der Umgebung von x_{max} **konkav**)



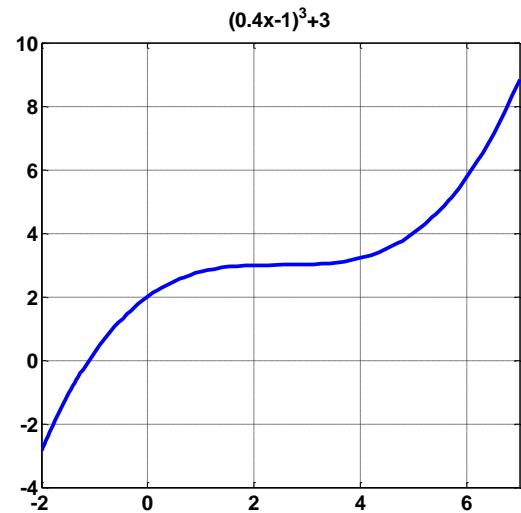
Optimalitätsbedingungen: univariate Funktionen (2)



$f'' > 0 \rightarrow \text{Min.}$

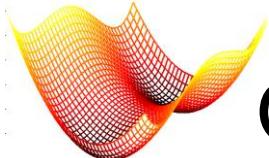


$f'' < 0 \rightarrow \text{Max.}$



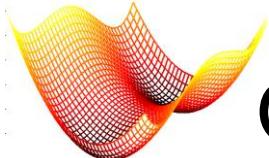
$f'' = 0 \rightarrow \text{Wendepkt.}$





Optimalitätsbedingungen: multivariate Funktionen (1)

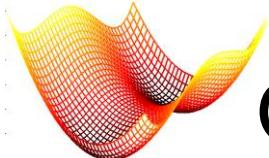
- **Multivariate Funktionen:** Der Funktionswert $f(\mathbf{x})$ hängt von **mehreren** Variablen ab.
- **Analogien:**
 - 1. Ableitung $f'(\mathbf{x}) \rightarrow$ Gradient $\nabla f(\mathbf{x})$ bei multivariaten Funktionen
 - 2. Ableitung $f''(\mathbf{x}) \rightarrow$ Hesse-Matrix $\nabla^2 f(\mathbf{x}) = \mathbf{H}(\mathbf{x})$ bei multivariaten Funktionen („Matrix der 2. Ableitungen“)
 - Hesse-Matrix ist symmetrisch, da Ableitungsoperator linear, d.h. $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$



Optimalitätsbedingungen: multivariate Funktionen (2)

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_N} \end{bmatrix}^T$$

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_N \partial x_N} \end{bmatrix}$$



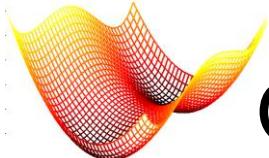
Optimalitätsbedingungen: multivariate Funktionen (3)

- **Notwendige Bedingung:** im Optimum muss gelten:

$$\nabla f(\mathbf{x}^*) = \mathbf{0}$$

- **Geometrische Interpretation:** sonst gäbe es ausgehend von \mathbf{x}^* eine Richtung, in der f weiter abnehmen würde
- Approximation von $f(\mathbf{x}^*)$ durch Taylor-Reihe:
$$f(\mathbf{x}^* + \Delta\mathbf{x}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}^*)\Delta\mathbf{x} + HOT$$
- Umformung und Einsetzen von $\nabla f(\mathbf{x}^*) = \mathbf{0}$ ergibt für Änderung $\Delta f(\mathbf{x})$:

$\Delta f(\mathbf{x}) = \frac{1}{2}\Delta\mathbf{x}^T \mathbf{H}(\mathbf{x}^*)\Delta\mathbf{x}$: muss im Punkt \mathbf{x}^* für alle $\Delta\mathbf{x} \geq 0$ sein!



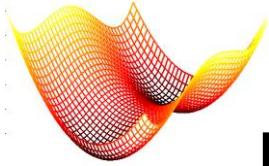
Optimalitätsbedingungen: multivariate Funktionen (4)

Notwendige Bedingung 1. Ordnung für ein Minimum: $\nabla f(\mathbf{x}^*) = \mathbf{0}$

Notwendige Bedingung 2. Ordnung für ein Minimum: $\Delta \mathbf{x}^T \mathbf{H}(\mathbf{x}^*) \Delta \mathbf{x} \geq 0$: $\mathbf{H}(\mathbf{x}^*)$ positiv semi-definit

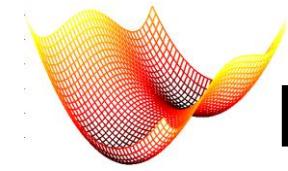
Hinreichende Bedingung 2. Ordnung für ein Minimum: $\Delta \mathbf{x}^T \mathbf{H}(\mathbf{x}^*) \Delta \mathbf{x} > 0$: $\mathbf{H}(\mathbf{x}^*)$ positiv definit

Bei Maximum: $\mathbf{H}(\mathbf{x}^*)$ negativ (semi-)definit

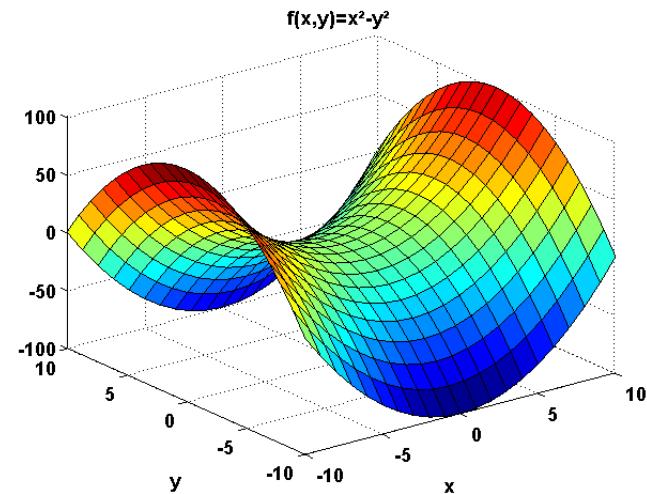
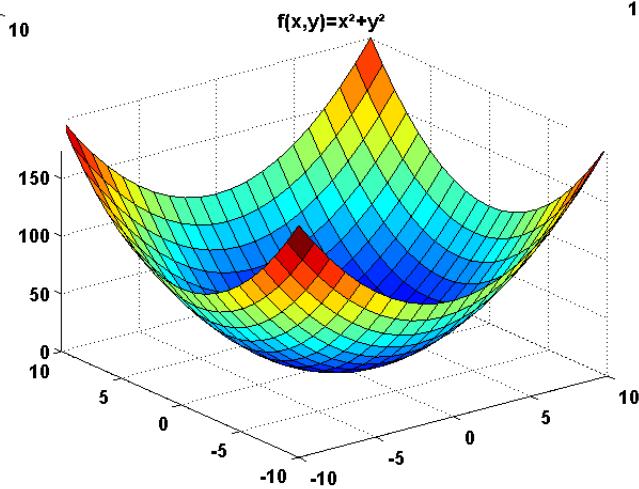
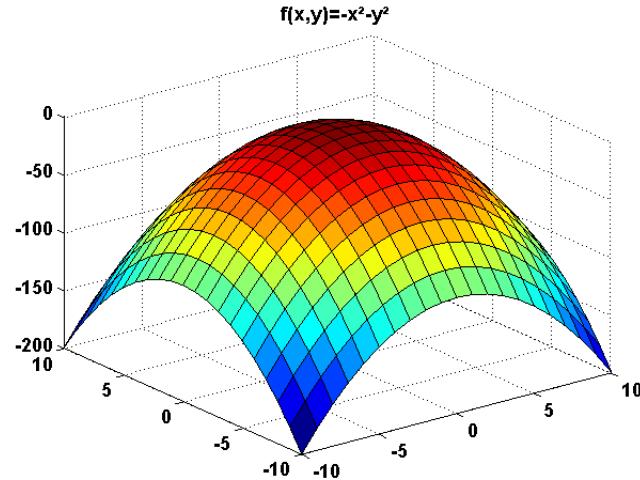


Hesse-Matrix: Eigenschaften

- Hesse-Matrix ist stets **symmetrisch**, da Ableitung eine lineare Operation ist. Deshalb gilt: $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$
- Per Definition ist Hesse-Matrix gleich der Transponierten der Jacobi-Matrix des Gradienten: $\mathbf{H}(\mathbf{x}) = \mathbf{J}(\nabla f(\mathbf{x}))^T$
- **Definitheit** wird durch **Eigenwerte** λ_i der Hesse-Matrix definiert:
 - Alle $\lambda_i \geq 0 (> 0)$: positiv semi-definit (positiv definit)
 - Alle $\lambda_i \leq 0 (< 0)$: negativ semi-definit (negativ definit)
 - Es existieren sowohl $\lambda_i \geq 0$ als auch $\lambda_i \leq 0$: indefinit
 - Funktion hat **Sattelpunkt** bei **indefiniter Hesse-Matrix**



Hesse-Matrix und Arten von stationären Punkten



Konvexe (und konkave) Funktionen

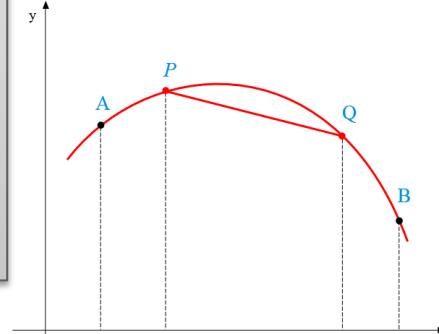
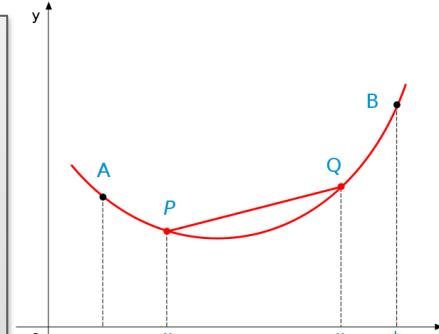
Eine Funktion $f(\mathbf{x})$ heißt für einen (zusammenhängenden) Definitionsbereich $D \subseteq \mathbb{R}^N$ **konvex**, wenn für alle $\mathbf{x}_1, \mathbf{x}_2 \in D$ und $t \in [0..1]$ gilt:

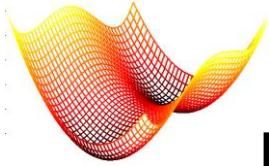
$$f(t \cdot \mathbf{x}_1 + (1 - t) \cdot \mathbf{x}_2) \leq t \cdot f(\mathbf{x}_1) + (1 - t) \cdot f(\mathbf{x}_2)$$

Gilt $f(t \mathbf{x}_1 + (1 - t) \mathbf{x}_2) < t f(\mathbf{x}_1) + (1 - t) f(\mathbf{x}_2)$, so ist $f(\mathbf{x})$ **streng konvex**

„Funktionswerte sind unterhalb der Verbindungsgerade von $f(\mathbf{x}_1)$ und $f(\mathbf{x}_2)$ “

Gilt $f(t \mathbf{x}_1 + (1 - t) \mathbf{x}_2) > t f(\mathbf{x}_1) + (1 - t) f(\mathbf{x}_2)$, so ist $f(\mathbf{x})$ **streng konkav**



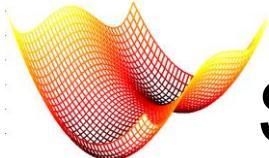


Extrema, Konvexität und Hesse-Matrix

Es gilt: Ist eine Funktion $f(\mathbf{x})$ im ganzen Lösungsbereich S streng konvex, so besitzt sie (höchstens) ein (globales) Minimum (unimodale Funktion). Ihre Hesse-Matrix $\mathbf{H}(\mathbf{x})$ ist dann für alle $\mathbf{x} \in S$ positiv definit

Ist eine Funktion $f(\mathbf{x})$ im ganzen Lösungsbereich S streng konkav, so besitzt sie (höchstens) ein (globales) Maximum (unimodale Funktion). Ihre Hesse-Matrix $\mathbf{H}(\mathbf{x})$ ist dann für alle $\mathbf{x} \in S$ negativ definit.

- Von zentraler Bedeutung, da bei konvexen Funktionen lokale Optimierungsstrategien ausreichend
- Deshalb: Zielfunktion wird oft durch konvexe Funktion approximiert

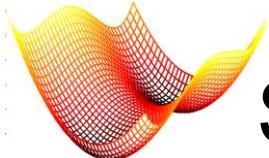


Strategie multivariater Optimierung ohne NB (1)

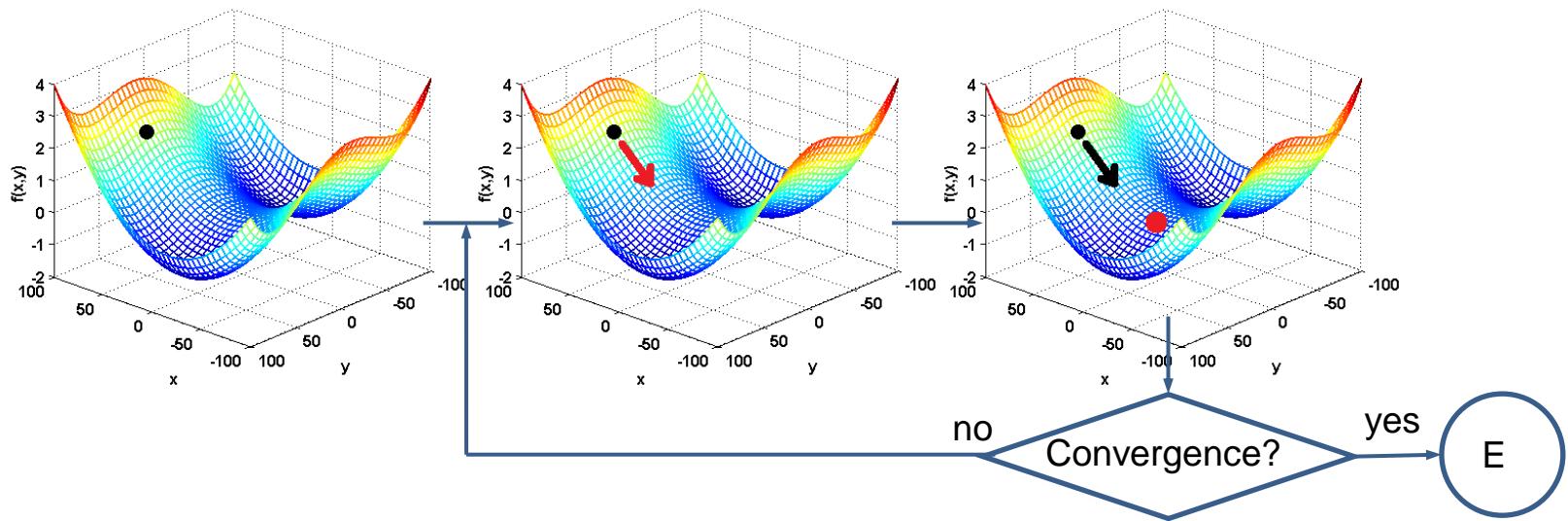
Grundlegende Optimierungsstrategie bei nichtlinearer,
multivariater Optimierung ohne Nebenbedingungen: Iteratives,
2-stufiges Vorgehen:

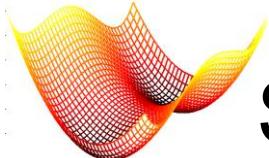
Wiederhole, bis Konvergenz (z.B. $f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) < \varepsilon$):

- 1) Finde „vielversprechende“ Suchrichtung \mathbf{s}_k
- 2) 1-dimensionale Optimierung entlang \mathbf{s}_k : Finde „verbesserte“ Lösung \mathbf{x}_{k+1} entlang $\mathbf{x}_k + t \cdot \mathbf{s}_k$, d.h. finde t_k^* , welches $f(\mathbf{x}_k + t \cdot \mathbf{s}_k)$ optimiert



Strategie multivariater Optimierung ohne NB (2)





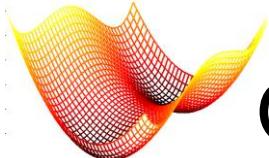
Strategie multivariater Optimierung ohne NB (3)

```
function optimizeContinuousMultidim (in objective function  $f(\mathbf{x})$ ,  
in initial solution  $\mathbf{x}^0$ , in convergence criterion  $\varepsilon$ , out  
refined solution  $\mathbf{x}^*$ )  
  
 $k \leftarrow 0$   
// main optimization loop  
repeat  
    calculate search direction  $\mathbf{s}^k$   
    find one-dimensional minimum along direction  $\mathbf{s}^k$   
    OR take one step of a fixed predefined or estimated size in  
    the search direction, yielding  $\mathbf{x}^{k+1}$   
     $k \leftarrow k+1$   
until  $\frac{f(\mathbf{x}^k) - f(\mathbf{x}^{k+1})}{f(\mathbf{x}^k) + f(\mathbf{x}^{k+1})} \leq \varepsilon$  OR  $k > K_{\max}$   
 $\mathbf{x}^* \leftarrow \mathbf{x}^k$ 
```



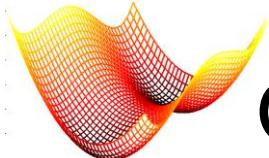
Eindimensionale Optimierung

- **Intervallreduktionsverfahren**
 - **Bisektion**
 - **Goldener Schnitt**
- **Newton-Verfahren**
- **„Armijo Line Search“**
- **Polynom-Approximation**



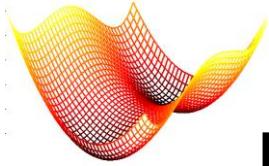
Optimierung univariater Funktionen: Einführung (1)

- **Univariate Funktionen:** Funktionen **einer** Veränderlicher $f(t)$
- Optimierung multivariater Funktionen $f(\mathbf{x})$ entlang einer bestimmten Suchrichtung \mathbf{s}_k ist 1-dimensionale Optimierung: $f(\mathbf{x}_k + t \cdot \mathbf{s}_k) = f(t)$, da in diesem Schritt \mathbf{x}_k und \mathbf{s}_k konstant bleiben und ausschließlich t variiert.
- Da Suchrichtung \mathbf{s}_k konstant: „**line search Verfahren**“
- **Annahme:** Funktion ist im untersuchten Bereich **unimodal** (d.h. hat nur ein Minimum (\rightarrow konvexe Fkt.) bzw. Maximum (\rightarrow konkave Fkt.))



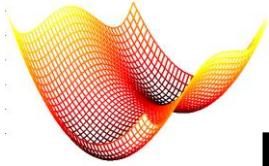
Optimierung univariater Funktionen: Einführung (2)

- Verwendete Prinzipien:
 - Reduktion von Intervallen
 - Approximation durch andere Funktionen (z.B. durch Polynome)
 - Suche nach der Nullstelle der ersten Ableitung (Newton-Verfahren)
 - „Armijo Line Search“ (inexakt)



Bisektionsverfahren (1)

- **Intervall-Reduktionsverfahren** → bei Start muss untere und obere Grenze (t_l und t_u) des untersuchten Lösungsraumes bekannt sein
- **Iteratives Verfahren:** Intervall wird immer kleiner, bis Konvergenz erreicht
- **Annahme:** Zielfunktion f ist in diesem Bereich unimodal und stetig differenzierbar
- **Grundidee: Teile Intervall in zwei Hälften und entscheide, in welchem Bereich das gesuchte Minimum liegt**

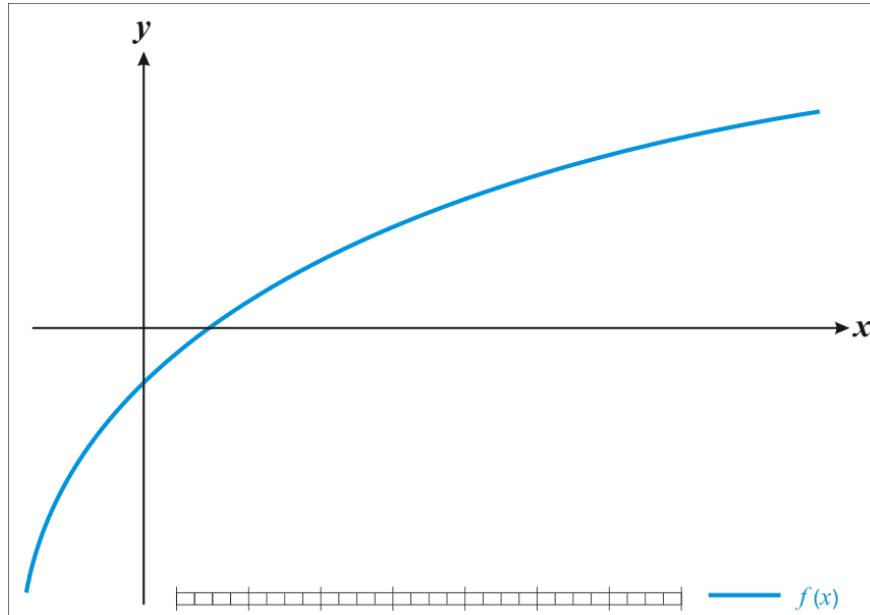


Bisektionsverfahren (2)

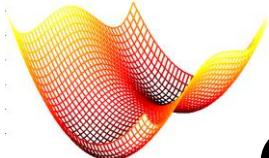
- Hierzu: betrachte 1. Ableitung von $f(t)$ in der Mitte des Intervalls zwischen t_l und t_u , d.h. $f'\left(\frac{t_l+t_u}{2}\right)$
 - Falls $f'\left(\frac{t_l+t_u}{2}\right) < 0$: f nimmt mit zunehmendem t ab
→ untersuche „rechtes“ (Teil-)Intervall in nächster Iteration
 - Falls $f'\left(\frac{t_l+t_u}{2}\right) > 0$: f nimmt mit zunehmendem t zu
→ untersuche „linkes“ (Teil-)Intervall in nächster Iteration
- Konvergenz z.B. bei
 - $(t_u - t_l) < \epsilon_I$ oder
 - $f'\left(\frac{t_l+t_u}{2}\right) < \epsilon_D$.
- Dann ist $\frac{t_l+t_u}{2}$ das gesuchte Minimum.

Bisektionsverfahren (3)

- Suche des Minimums von $f(t)$ ist äquivalent zur Suche der Nullstelle der ersten Ableitung $f'(t)$ von $f(t)$

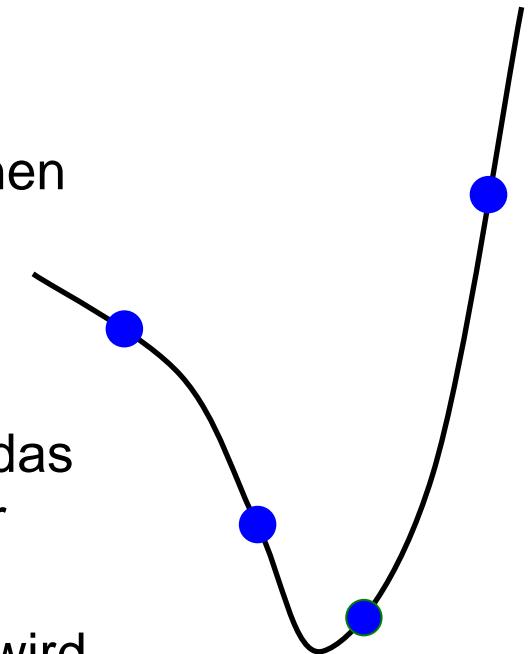


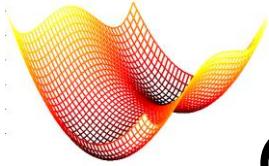
© Wikipedia/Ralf Pfeifer



Goldener Schnitt (1)

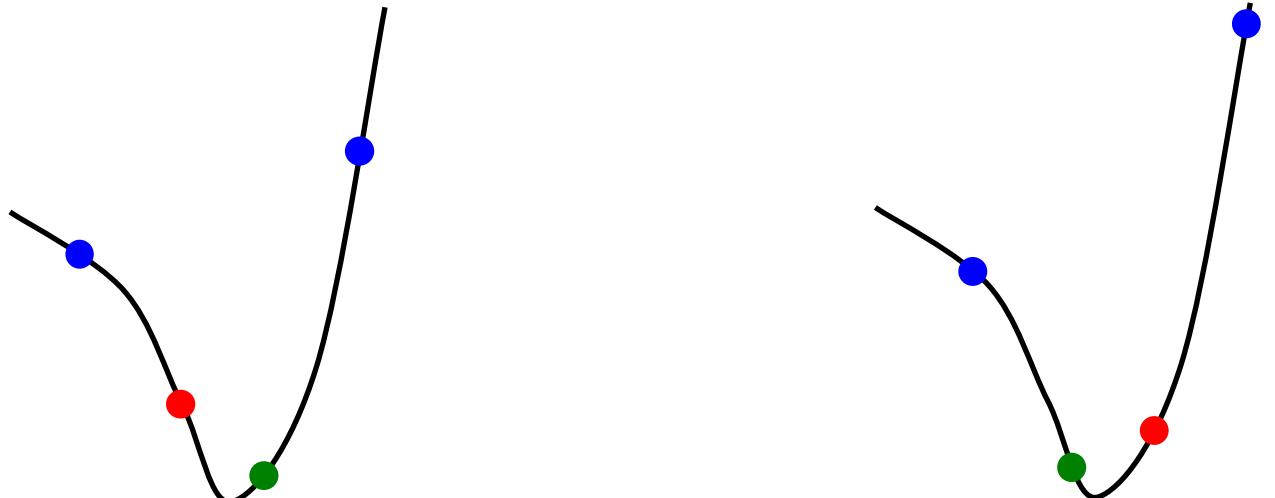
- **Reduktion** des durch $\mathbf{x}_l = \mathbf{x}_k + t_l \cdot \mathbf{s}_k$ und $\mathbf{x}_u = \mathbf{x}_k + t_u \cdot \mathbf{s}_k$ beschränkten **Intervalls** $[\mathbf{x}_l, \mathbf{x}_u]$ bzw. $[t_l, t_u]$
- Annahme: es sind zwei weitere Punkte innen im Intervall bekannt: $\mathbf{x}_1 = \mathbf{x}_k + t_1 \cdot \mathbf{s}_k$ und $\mathbf{x}_l = \mathbf{x}_2 + t_2 \cdot \mathbf{s}_k$
- Es muss gelten: $t_l < t_1 < t_2 < t_u$
- „**Iterative Refinement**“: reduziere iterativ das Intervall, indem in jedem Schritt i einer der beiden inneren Punkte \mathbf{x}_1^i oder \mathbf{x}_2^i (bzw. t_1^i oder t_2^i) als neue Intervallgrenze definiert wird.
- Konvergenz, wenn das Intervall klein genug



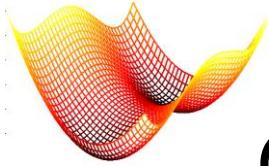


Goldener Schnitt (2)

- **Vorgehen** bei Intervallreduktion: vergleiche $f(t_1^i)$ mit $f(t_2^i)$:
 - $f(t_1^i) > f(t_2^i)$: t_1^i wird neue untere Grenze: $t_l^{i+1} = t_1^i$
 - $f(t_1^i) < f(t_2^i)$: t_2^i wird neue obere Grenze: $t_u^{i+1} = t_2^i$

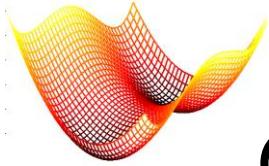


Blau: momentane Intervallgrenze; rot: neue Intervallgrenze



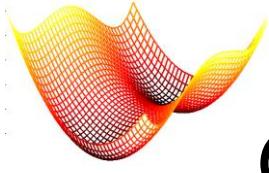
Goldener Schnitt (3)

- Optimale Wahl der „inneren“ Punkte t_1^i und t_2^i gemäß folgender Bedingungen:
 - „**symmetrische**“ Lage im Intervall, d.h. $t_1^i - t_l^i = t_u^i - t_2^i$
→ funktionsunabhängige, „**feste**“ **Intervall-Reduktion**, „worst-case“ wird vermieden
 - Reduktion der Funktionsberechnungen, d.h. **Wiederverwenden** der Punkte samt zugehöriger Funktionswerte
- Daraus ergibt sich: $t_1^i = (1 - \tau) \cdot t_l^i + \tau \cdot t_u^i$
 $t_2^i = (1 - \tau) \cdot t_u^i + \tau \cdot t_l^i$ mit $\tau = \frac{3-\sqrt{5}}{2} \approx 0.38197$



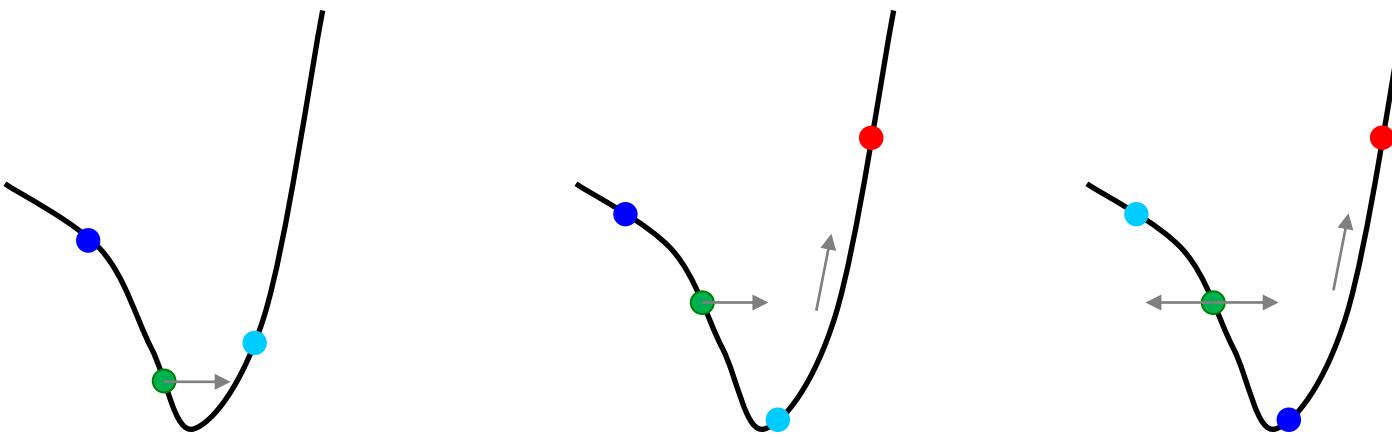
Goldener Schnitt (4)

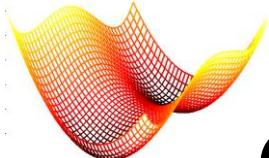
- In jeder Iteration wird daher das Intervall um ca. 38% reduziert
 - Verhältnis der Distanzen zwischen den Punkten entspricht dem „**Goldenen Schnitt**“
 - **Fixe Anzahl** N von **Iterationen**, wenn Intervallgröße am Ende ein fester Bruchteil ϵ der ursprünglichen Größe sein soll: $N \approx -2.078 \cdot \ln\epsilon + 3$, z.B. bei $\epsilon = 0.001$ ergibt sich $N = 18$
 - Bei Konvergenz sind 4 Punkte samt zugehöriger Funktionswerte bekannt
- **Verbesserung** der Minimumposition durch **kubische Interpolation** möglich/sinnvoll



Goldener Schnitt (5)

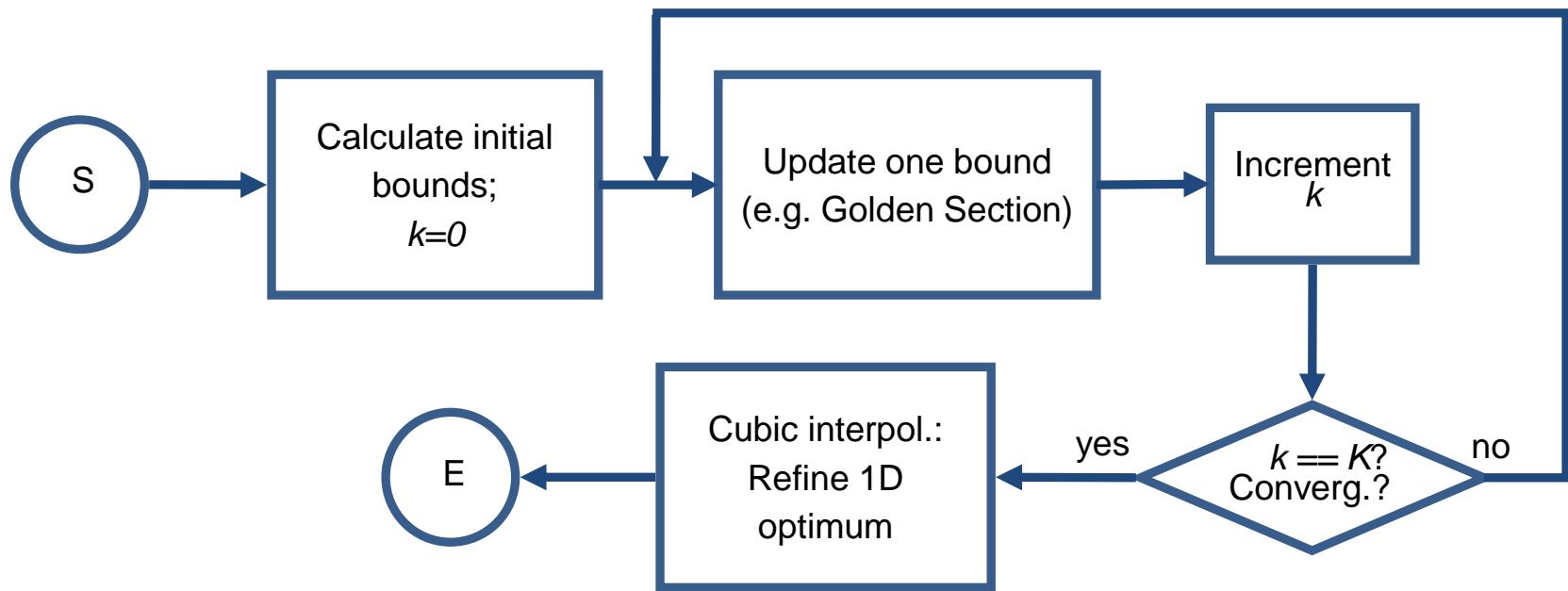
- **Festlegen des Startintervalls** bei bekanntem x_k (grün) und Suchrichtung (Pfeil): Suche, bis auf jeder Seite von x_k eine Stelle mit größerem Funktionswert gefunden
- Ggf. Tausch von unterer (blau) und oberer (hellblau) Intervallgrenze (rechtes Bild)

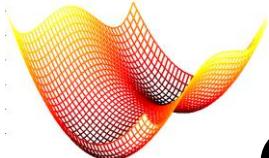




Goldener Schnitt (6)

- Flussdiagramm für eindimensionale Optimierung mittels Intervallreduktion



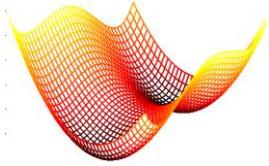


Goldener Schnitt (7): Pseudocode

```

function opt1D_GoldenSection (in objective function  $f(\mathbf{x})$ , in current solution  $\mathbf{x}^k$ , in search direction  $\mathbf{s}^k$ ,
                           in convergence criterion  $\varepsilon$ , out refined solution  $\mathbf{x}^{k+1}$ )
    // step 1: estimate initial lower and upper bounds  $\mathbf{x}_l^0$  and  $\mathbf{x}_u^0$ 
    calculate  $\mathbf{x}_l^0 = \mathbf{x}^k - \mathbf{s}^k$  and  $\mathbf{x}_u^0 = \mathbf{x}^k + \mathbf{s}^k$ 
    if  $f(\mathbf{x}_l^0) < f(\mathbf{x}^k) < f(\mathbf{x}_u^0)$  then // "wrong" search direction (right picture)
         $\mathbf{s}^k \leftarrow -\mathbf{s}^k$ ; interchange  $\mathbf{x}_l^0$  and  $\mathbf{x}_u^0$  // invert search direction
    end if
    if  $f(\mathbf{x}_u^0) < f(\mathbf{x}^k)$  then // upper bound is still not found (middle picture)
        // iterative search until a "valid" upper bound is found
        repeat
             $\mathbf{x}_l^0 \leftarrow \mathbf{x}_u^0$ 
            calculate  $\mathbf{x}_{u,new}^0 = (1+\alpha) \cdot \mathbf{x}_u^0 - \alpha \cdot \mathbf{x}_l^0$  with  $\alpha = (1 + \sqrt{5})/2$ 
            until  $f(\mathbf{x}_{u,new}^0) > f(\mathbf{x}^k)$ 
    end if
    // iterative refinement of the bounds (golden section method)
    calculate  $N \approx -2.078 \cdot \ln \varepsilon + 3$  as well as  $\alpha_l^0$  and  $\alpha_u^0$ , based on  $\mathbf{x}_l^0$ ,  $\mathbf{x}_u^0$ , and  $\mathbf{s}^k$ .
    calculate  $\mathbf{x}_1^0 = (1-\tau) \cdot \mathbf{x}_l^0 + \tau \cdot \mathbf{x}_u^0$  (if not known already) and  $\mathbf{x}_2^0 = (1-\tau) \cdot \mathbf{x}_u^0 + \tau \cdot \mathbf{x}_l^0$ 
    for i = 0 to  $N - 1$ 
        if  $f(\mathbf{x}_1^i) < f(\mathbf{x}_2^i)$  then
            // new upper bound found
             $\mathbf{x}_u^{i+1} \leftarrow \mathbf{x}_2^i$ ;  $\mathbf{x}_2^{i+1} \leftarrow \mathbf{x}_1^i$ ;
             $\mathbf{x}_1^{i+1} \leftarrow (1-\tau) \cdot \mathbf{x}_u^i + \tau \cdot \mathbf{x}_2^i$  // apply golden section ratio
        else
            // new lower bound found
             $\mathbf{x}_l^{i+1} \leftarrow \mathbf{x}_1^i$ ;  $\mathbf{x}_1^{i+1} \leftarrow \mathbf{x}_2^i$ 
             $\mathbf{x}_2^{i+1} \leftarrow (1-\tau) \cdot \mathbf{x}_l^i + \tau \cdot \mathbf{x}_1^i$  // apply golden section ratio
        end if
    next
    // [optional] final refinement of solution by cubic interpolation
    calculate  $\mathbf{x}^{k+1}$  based on cubic interpolation at the four points  $\mathbf{x}_l^i$ ,  $\mathbf{x}_1^i$ ,  $\mathbf{x}_2^i$ , and  $\mathbf{x}_u^i$ 

```



„Armijo line search“ (1)

- **Beobachtung:** Suche nach dem exakten Minimum benötigt (evtl. unnötig viel) Zeit
→ **Idee:** Finde mit begrenztem Aufwand eine „ausreichend gute Reduktion“ von $f(\mathbf{x})$ und nutze die Rechenleistung für eine neue (bessere) Schätzung der Suchrichtung
- **Abschätzung des Potentials zur Reduktion** von $f(\mathbf{x}_k)$ in Suchrichtung \mathbf{s}_k anhand Skalarprodukt mit Gradienten $\nabla f(\mathbf{x}_k)$: $m = \nabla f(\mathbf{x}_k)^T \cdot \mathbf{s}_k = \|\nabla f(\mathbf{x}_k)\| \cdot \|\mathbf{s}_k\| \cdot \cos \theta$
- Um Fortschritt zu erzielen, sollte gelten: $m < 0$, d.h. Richtung des steilsten Abstiegs von f an der Stelle \mathbf{x}_k unterscheidet sich um höchstens $\frac{\pi}{2}$ von der Suchrichtung \mathbf{s}_k
- $\nabla f^T \cdot \mathbf{s}_k$ ist Maß für „Reduktionspotential“ der 1-dim. Suche

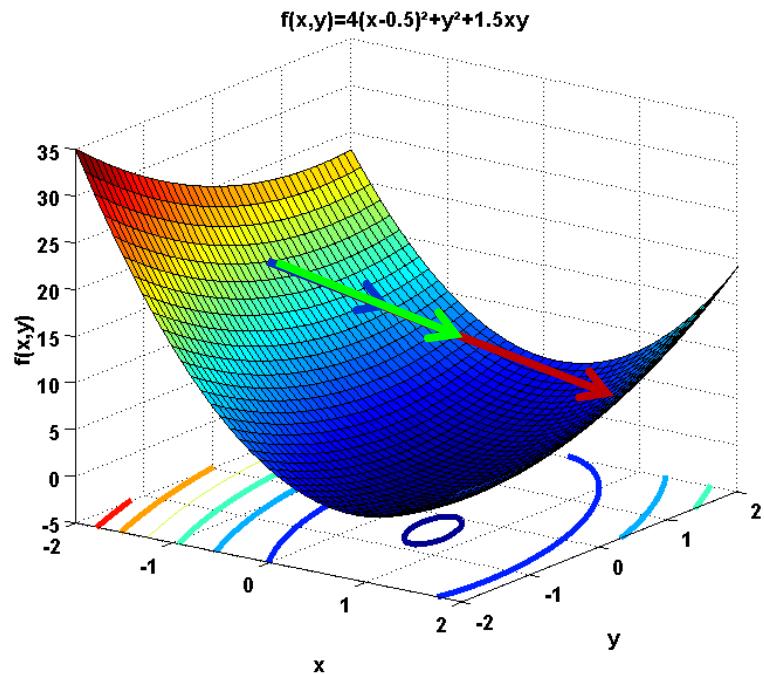
„Armijo line search“ (2)

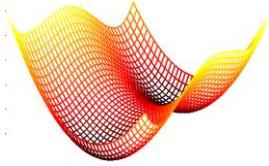
- **Armijo-Goldstein-Bedingung:**

Kriterium zur Beurteilung der Schrittwe. t : Genügend Fortschritt ist erzielt, wenn gilt:

$$f(\mathbf{x}_k + t \cdot \mathbf{s}_k) \leq f(\mathbf{x}_k) + t \cdot c_1 \cdot \nabla f(\mathbf{x}_k)^T \cdot \mathbf{s}_k; c_1 \in [0..1] \text{ z.B. } 0.5$$

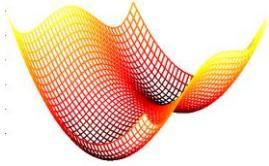
- **Ziel:** Vermeidung zu langer Schritte, die die Zielfunktion nur noch wenig verringern \rightarrow kleine t werden favorisiert!





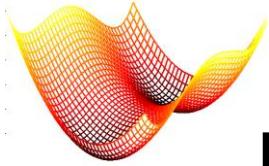
„Armijo line search“ (3)

- **Wolfe-Bedingungen:** zusätzlich zur Armijo-Goldstein-Bed.:
$$\mathbf{s}^T \cdot \nabla f(\mathbf{x}_k + t \cdot \mathbf{s}_k) \geq c_2 \cdot \mathbf{s}^T \cdot \nabla f(\mathbf{x}_k); c_2 \in [0..1], c_2 > c_1$$
- Vergleich des Skalarprodukts von Suchrichtung u. Gradient am Anfang und am Ende des Optimierungsschrittes
- Skalarprodukt negativ → **Betrag** muss **kleiner** werden
- Stellt sicher, dass keine zu kleine Schrittweite gewählt wird: spürbare Verringerung notwendig, z.B. weil
 - $\|\nabla f(\mathbf{x}_k + t \cdot \mathbf{s}_k)\|$ deutlich kleiner wurde oder
 - der Winkel zwischen \mathbf{s} und $-\nabla f$ deutlich größer wurde
- Krümmung von f beeinflusst sowohl Betrag von ∇f als auch dessen Richtung → „**Krümmungs-Bedingung**“



„Armijo line search“ (4)

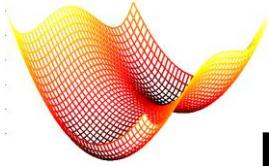
- Iterative Suche nach „optimalem“ t^* mittels „**backtracking**“:
Starte mit großem t_0 und reduziere die t_i um einen bestimmten Faktor, bis Armijo-Goldstein-Bedingung erfüllt ist: $t_{i+1} = \tau \cdot t_i$; z.B. $\tau = \frac{1}{2}$:
 1. Start: definiere $t_0 > 0$, setze $i = 0$
 2. Berechne $f(\mathbf{x}_k + t_i \cdot \mathbf{s}_k)$
 3. Armijo-Goldstein-Bedingung:
wenn $f(\mathbf{x}_k + t_i \cdot \mathbf{s}_k) \leq f(\mathbf{x}_k) + t_i c \cdot \nabla f(\mathbf{x}_k)^T \cdot \mathbf{s}_k$ setze
 $\mathbf{x}_{k+1} = \mathbf{x}_k + t_i \cdot \mathbf{s}_k$
 4. Setze $t_{i+1} = \tau \cdot t_i$ und gehe zu Schritt 2



Newton-Verfahren (1)

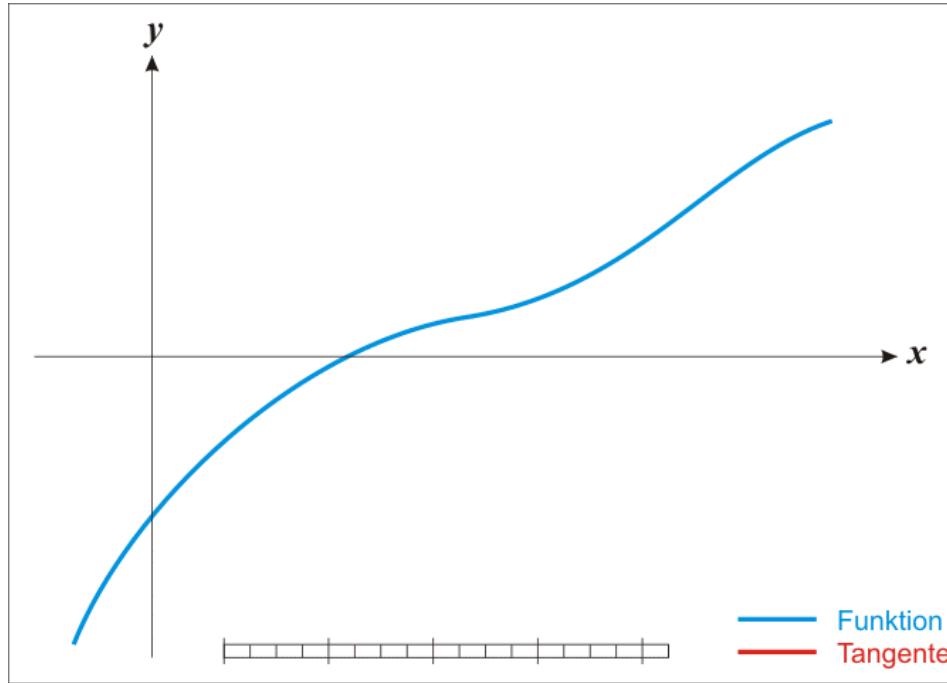
- Für **unimodale** Funktionen (~ Funktion f hat im untersuchten Bereich genau ein Optimum) gilt: Suche nach dem Optimum von $f(t)$ ist **äquivalent zur Suche der Nullstelle der ersten Ableitung $f'(t)$**
→ Anwendung des Newton-Verfahrens, welches die Nullstelle einer Funktion findet, auf $f'(t)$
- **Iterative Nullstellen-Suche:**
 - **Lineare Approximation** von $f'(t)$ an aktueller Pos. t_i :
$$f'(t_i + \Delta t) \approx f'(t_i) + \Delta t \cdot f''(t_i)$$
 - Neue Position ist **Nullstelle** der linearen Approximation:
$$f'(t_i) + \Delta t \cdot f''(t_i) = 0$$

$$\rightarrow t_{i+1} = t_i - f'(t_i)/f''(t_i)$$

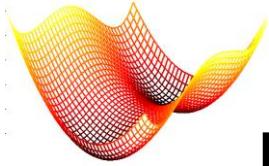


Newton-Verfahren (2)

- Iterativer Ablauf der Nullstellen-Suche:



© Wikipedia/Ralf Pfeifer



Newton-Verfahren (3)

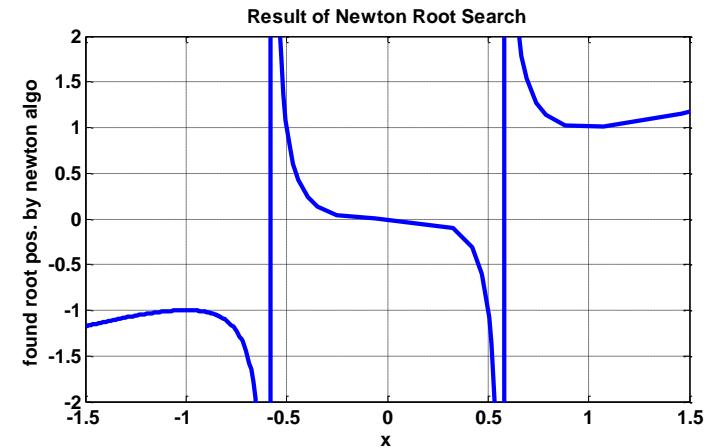
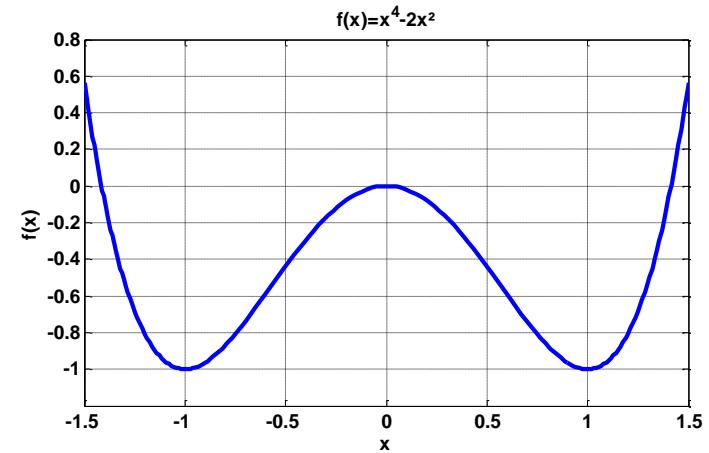
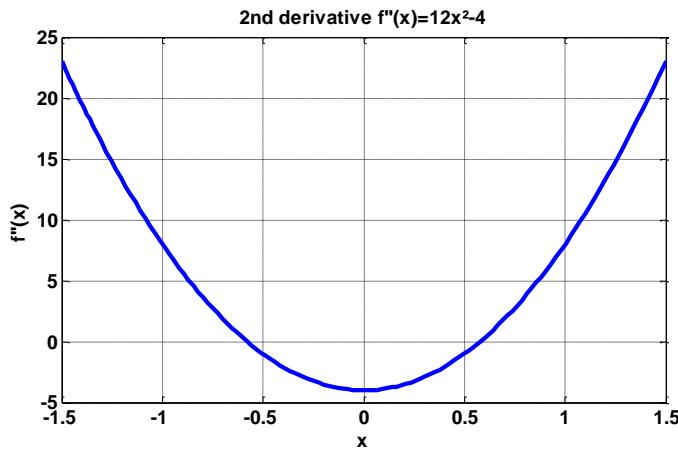
- Konvergenz z.B. wenn $|t_{i+1} - t_i| < \epsilon$ oder $f'(t_{i+1}) < \epsilon$
- **Achtung:** Verfahren divergiert, wenn $f''(t_i) = 0$
→ Notwendige Bedingung für Konvergenz: Unimodalität
(Funktion hätte Wendepunkt bei Nullstelle von f'')
- Bei multivariater Zielfunktion:
 - „Richtungsableitung“ in Suchrichtung ist Skalarprodukt von s_k und Gradient ∇f :
$$\partial f(\mathbf{x}_k + t \cdot s_k) / \partial t = \nabla f(\mathbf{x}_k + t \cdot s_k)^T \cdot s_k$$
 - Hier vorgestelltes (1-dimensionales) Newton-Verfahren ist Teil eines übergeordneten Verfahrens mit gleichzeitiger Wahl der Suchrichtung und 1-dimensionaler Optimierung pro Iteration

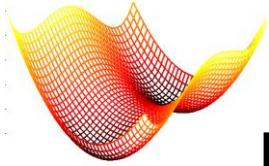
Newton-Verfahren (4)

Bsp.: Betrachte $f(x) = x^4 - 2x^2$

Nullstelle x_N der lin. Approx. der
1. Ableitung in Abh. vom
Startwert x_0 berechnet sich zu:

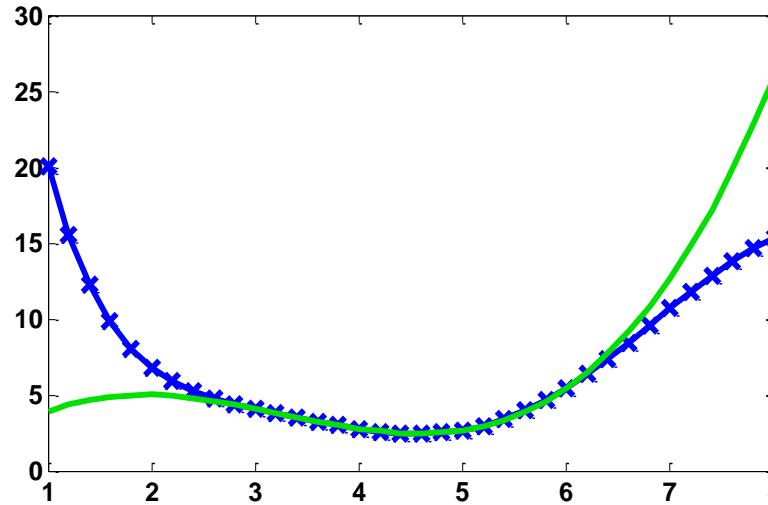
$$x_N = 2x_0^3/(3x_0^2 - 1)$$

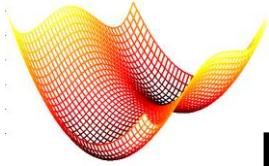




Polynom-Approximation (1)

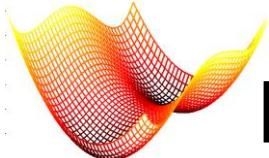
- Wenn mehrere Positionen + deren Funktionswerte bekannt:
Approximation mittels Polynom möglich, z.B.:
- 3 Pos.: Koeff. eines quadr. Polynoms eindeutig bestimmbar
- 4 Pos.: Polynom 3. Ordnung (kubisch)





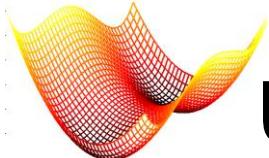
Polynom-Approximation (2)

- Wenn Koeffizienten bekannt, **kann Extremum des Polynoms in geschlossener Form ermittelt** werden
- **Iteratives** Vorgehen möglich:
 1. Polynom-Approximation
 2. Schätze Optimum t_i^* durch (analytische) Berechnung der Nullstelle der Ableitung des Polynoms
 3. Aktualisierung der Stützstellen-Positionen
- Eignung auch als letzter Schritt anderer line-Search-Verfahren, z.B. kubische Approximation bei Goldenem Schnitt (da dort der Funktionswert an 4 Positionen ohnehin bereits bekannt)



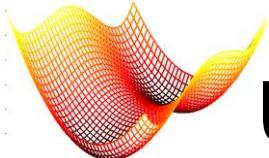
Polynom-Approximation: kubische Interpolation aus 4 Datenpunkten

```
// calc cubic coeffs from 4 data pts [x0..4,y0..4]
// cubic polynom: c0*x^3+c1*x^2+c2*x+c3
q1= x[2]*x[2]*x[2]*(x[1]-x[0]) - x[1]*x[1]*x[1]*(x[2]-x[0]) + x[0]*x[0]*x[0]*(x[2]-x[1]);
q2= x[3]*x[3]*x[3]*(x[1]-x[0]) - x[1]*x[1]*x[1]*(x[3]-x[0]) + x[0]*x[0]*x[0]*(x[3]-x[1]);
q3= (x[2]-x[1])*(x[1]-x[0])*(x[2]-x[0]);
q4= (x[3]-x[1])*(x[1]-x[0])*(x[3]-x[0]);
q5= y[2]*(x[1]-x[0]) - y[1]*(x[2]-x[0]) + y[0]*(x[2]-x[1]);
q6= y[3]*(x[1]-x[0]) - y[1]*(x[3]-x[0]) + y[0]*(x[3]-x[1]);
c0= (q3*q6-q4*q5)/(q2*q3-q1*q4); c1= (q5-c0*q1)/q3;
c2= (y[1]-y[0])/(x[1]-x[0]) - c1*(x[1]+x[0]) -
    c0*(x[1]*x[1]*x[1]-x[0]*x[0]*x[0])/(x[1]-x[0]);
c3= y[0] - c2*x[0] - c1*x[0]*x[0] - c0*x[0]*x[0]*x[0];
```



Univariate Optimierungsmethoden: Vergleich (1)

- **Polynom-Approximation:**
 - als letzter Schritt zum „Refinement“ geeignet, besonders wenn Funktionswerte bereits bekannt
- **Newton-Verfahren:**
 - Teil der Newton-Methode für multivariate Optimierung
 - Effizienz hängt vom Grad der Nichtlinearität ab!
- **Armijo Line Search:**
 - wenige Iterationen, aber inexakt
 - oft ein „guter“ Kompromiss bezüglich Laufzeit



Univariate Optimierungsmethoden: Vergleich (2)

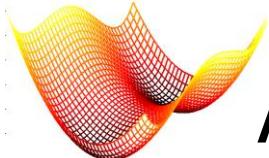
- **Intervall-Reduktion:**

- Konvergenz ist problem-unabhängig → allgemein einsetzbar
- Bisektion benötigt weniger Iterationen als Goldener Schnitt (50% vs. 38% Reduktion pro Iteration), aber: Berechnung der Ableitung notwendig (u.U. aufwändig)
→ Es hängt vom aktuellen Problem ab, welches der Verfahren effizienter ist!



Nicht-restringierte nichtlineare Optimierung

- **Verfahren erster Ordnung**
 - **Steilster Abstieg**
 - **Konjugierte Gradienten**
- **Verfahren zweiter Ordnung**
 - **Newton-Methode**
 - **Quasi-Newton-Verfahren**
- **Verfahren nullter Ordnung**
 - **Powell-Methode**
 - **Heuristik: Nelder-Mead**
- **Regressionsrechnung**
 - **Gauss-Newton-Verfahren**
 - **Levenberg-Marquardt-Methode**
 - **Stochastic Gradient Descent (SGD)**



Äquivalenz: Optimum quadratischer Fkt. mit Lösung eines linearen GS

- Multivariate quadratische Funktion:

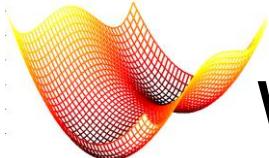
$$f_Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{b}^T \cdot \mathbf{x} + c$$

- Hat globales Minimum \mathbf{x}^* , falls \mathbf{A} positiv definit
- Es gilt: $\nabla f_Q(\mathbf{x}^*) = \mathbf{A} \cdot \mathbf{x}^* + \mathbf{b} = 0$, d.h. $\mathbf{A} \cdot \mathbf{x}^* = -\mathbf{b}$
- Daraus folgt:

Suche nach dem Minimum \mathbf{x}^* von $f_Q(\mathbf{x})$ (\mathbf{A} positiv definit) ist äquivalent zur Lösung des linearen Gleichungs-Systems

$$\mathbf{A} \cdot \mathbf{x} = -\mathbf{b}$$

- Multivariate quadratische Funktionen haben eine besondere Bedeutung für nichtlineare Optimierungsmethoden
- Optimierungs-Meth. auch zur Lösung linearer GS einsetzbar

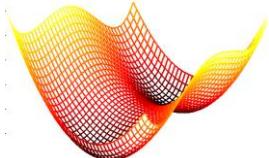


Wiederholung: Strategie multivariater Optimierung ohne NB

Grundlegende Optimierungsstrategie bei nichtlinearer, multivariater Optimierung ohne Nebenbedingungen: Iteratives, 2-stufiges Vorgehen:

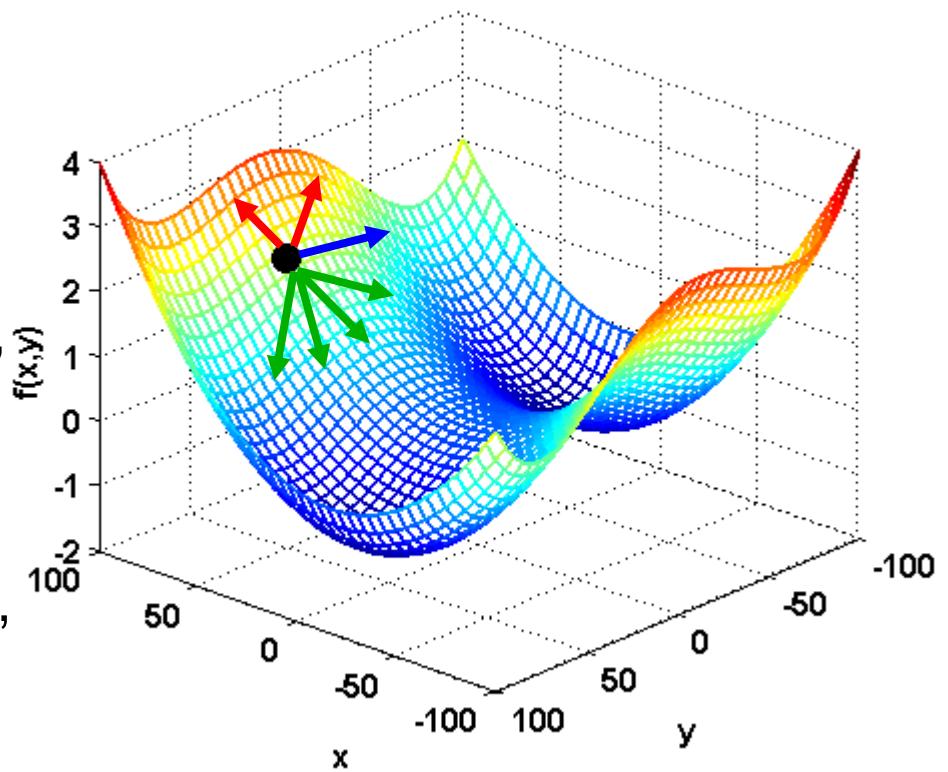
Wiederhole, bis Konvergenz (z.B. $f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) < \varepsilon$):

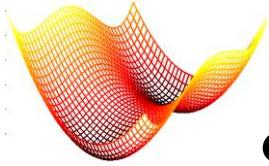
- 1) Finde „vielversprechende“ Suchrichtung s_k**
- 2) 1-dimensionale Optimierung entlang s_k : Finde „verbesserte“ Lösung \mathbf{x}_{k+1} entlang $\mathbf{x}_k + t \cdot s_k$, d.h. finde t_k^* , welches $f(\mathbf{x}_k + t \cdot s_k)$ optimiert**



„Günstige“ Suchrichtungen

- Allgemeine Regel:
Suchrichtung muss zu einer **Reduktion** des Funktionswertes führen, d.h. es muss gelten:
 $s_k \cdot \nabla f(\mathbf{x}_k) < 0$
- Konvergenzkriterium:
Gradient wird sehr klein, d.h.
 $\nabla f(\mathbf{x}_k) < \epsilon$





Suchrichtung: steilster Abstieg

- **Naheliegende Wahl** der Suchrichtung \mathbf{s}_k : negativer Gradient, d.h. Richtung des steilsten Abstiegs:

$$\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$$

- Pseudocode:

```
function optimizeSteepestDescent (in objective  $f(\mathbf{x})$ ,  
in start  $\mathbf{x}_0$ , in  $\epsilon$ , out solution  $\mathbf{x}^*$ )
```

repeat

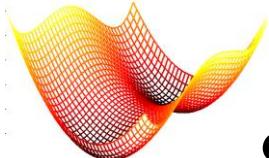
- 1.) calculate search direction $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$

- 2.) line search: $\mathbf{x}_{k+1} = \mathbf{x}_k + t^* \cdot \mathbf{s}_k$ with

$$t^* = \arg \min_t f(\mathbf{x}_{k+1} = \mathbf{x}_k + t \cdot \mathbf{s}_k)$$

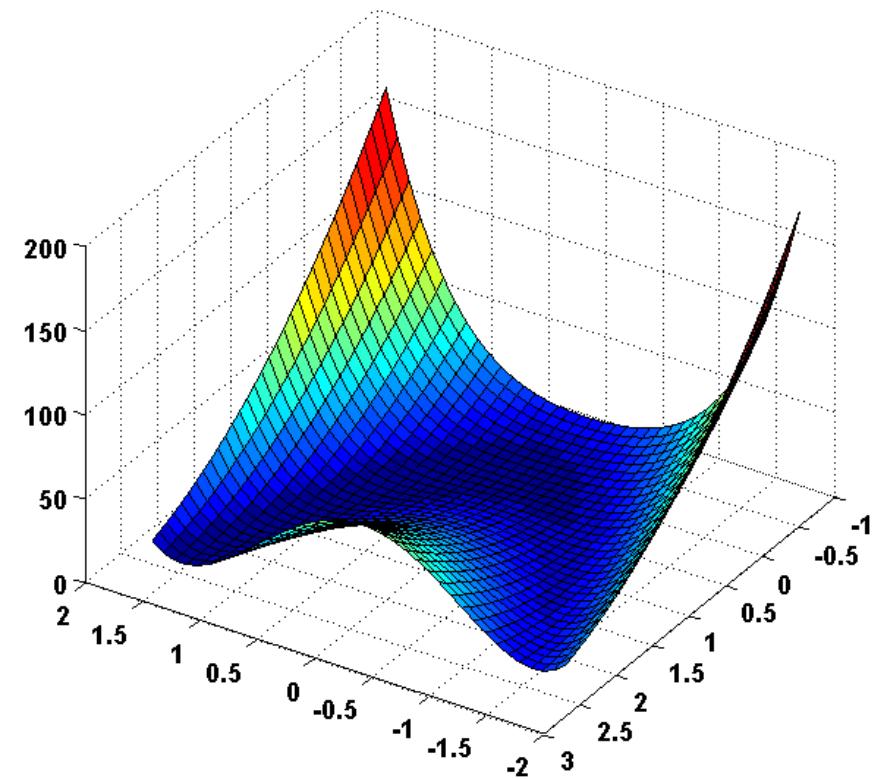
$k \leftarrow k + 1$

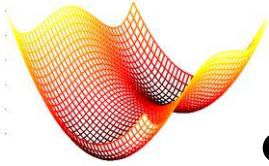
until $\|\nabla f(\mathbf{x}_k)\| < \epsilon$



Steilster Abstieg: Nachteile (1)

- **Problem 1:**
Konvergenzkriterium
basierend auf Gradient ist
nicht immer aussagekräftig
- Rosenbrock-Funktion:
 $f(x, y) = 10(y - x^2)^2 + (1 - x)^2$
- Schwierig, effizient zu
optimieren:
„langgezogenes“ Tal
entlang $y = x^2$



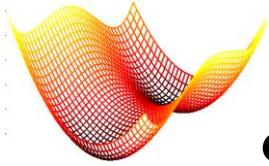


Steilster Abstieg: Nachteile (2)

- **Problem 2:** schlechte Konvergenz (nur linear), und v.a.: datenabhängig!
- Bei quadr. Fkt. $f_Q(\mathbf{x})$ mit $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ und exakter line search gilt:

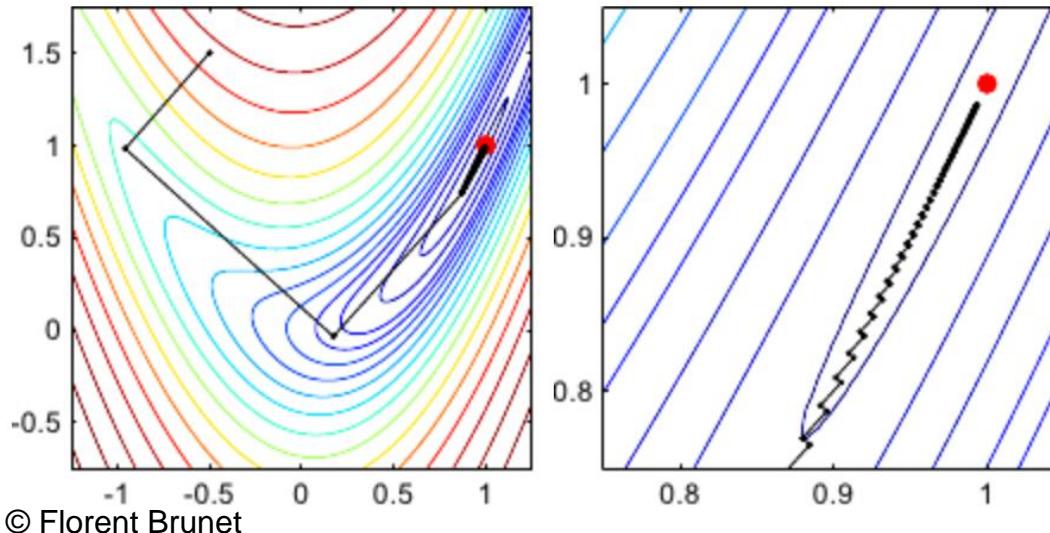
$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_A \leq \left(1 - \frac{2}{\kappa+1}\right) \cdot \|\mathbf{x}_k - \mathbf{x}^*\|_A$$

- A -Norm von \mathbf{v} : $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T \cdot A \cdot \mathbf{v}}$
- Konditionszahl κ : Quotient größter zu kleinster Eigenwert von $A \in \mathbb{R}^{N \times N}$: $\kappa = \mu_1 / \mu_N$
- Das heißt: **Konvergenz** hängt von A , also dem zu lösenden Problem, ab (**Daten-abhängig!**)
- **Lineare Konvergenz**; sehr langsam bei $\kappa \gg 1$

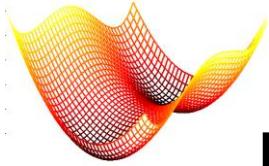


Steilster Abstieg: Nachteile (3)

- Suchrichtungen aufeinanderfolgender Iterationen sind bei exakter line search **treppenförmig**
- Am Beispiel Rosenbrock-Funktion: sehr schlechte Konvergenz im „gebogenen Tal“

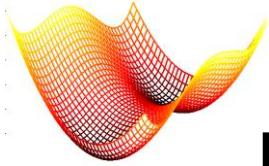


© Florent Brunet



Konjugiertes Gradientenverfahren (1)

- **Definition:** zwei Vektoren s_i und s_j sind zueinander **A-konjugiert**, wenn gilt:
$$s_i^T \cdot A \cdot s_j = 0$$
- Es lässt sich zeigen: werden zur Optimierung eines N -dimensionalen **quadratischen Optimierungsproblems** $f_Q(\mathbf{x})$ ausschließlich zueinander konjugierte Vektoren s_i verwendet, so wird bei exakter line search das **Minimum in (maximal) N Schritten erreicht**
- **Allerdings:** wegen numerischer Ungenauigkeiten wird Konvergenz tatsächlich in der Regel nicht in N Schritten erreicht → Implementierung als **iteratives Verfahren**

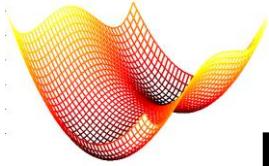


Konjugiertes Gradientenverfahren (2)

- **Gegeben:** quadratische Zielfkt. $f_Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{b}^T \cdot \mathbf{x}$
- Wie findet man iterativ konjugierte Vektoren, die anstelle des steilsten Abstiegs als Suchrichtungen in einem iterativen Verfahren verwendet werden können?
- **Start:** steilster Abstieg $\mathbf{s}_0 = -\nabla f(\mathbf{x}_0)$
- **Anschließend:** Berechne

$$\gamma_{k+1} = \frac{\nabla f_Q(\mathbf{x}_{k+1})^T \cdot \nabla f_Q(\mathbf{x}_{k+1})}{\nabla f_Q(\mathbf{x}_k)^T \cdot \nabla f_Q(\mathbf{x}_k)}$$
$$\mathbf{s}_{k+1} = -\nabla f_Q(\mathbf{x}_{k+1}) + \gamma_{k+1} \cdot \mathbf{s}_k$$

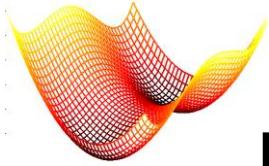
- Es lässt sich zeigen: alle derart berechneten \mathbf{s}_k sind zueinander **A-konjugiert**



Konjugiertes Gradientenverfahren (3)

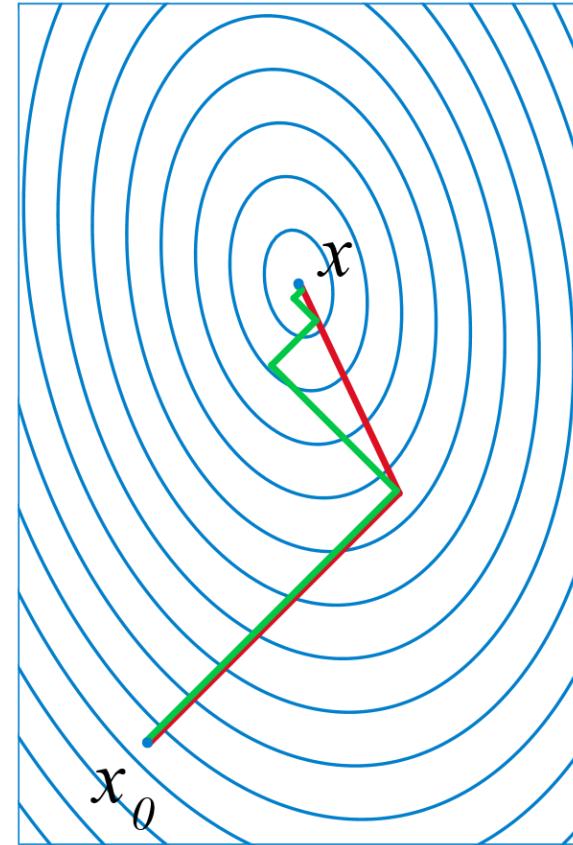
- Pseudocode für quadratische Funktionen:

```
function optimizeConjugateGradient(in objective  
fQ(x), in start x0, in ε, out solution x*)  
calculate initial search direction s0 = -∇fQ(x0)  
repeat  
    1.) line search: xk+1 = xk + t* · sk  
    2.) find next search dir. as conj. gradient:  
        a) γk+1 =  $\frac{\nabla f_Q(\mathbf{x}_{k+1})^T \cdot \nabla f_Q(\mathbf{x}_{k+1})}{\nabla f_Q(\mathbf{x}_k)^T \cdot \nabla f_Q(\mathbf{x}_k)}$   
        b) sk+1 = -∇fQ(xk+1) + γk+1 · sk  
    k ← k + 1  
until ∇f(xk) < ε
```

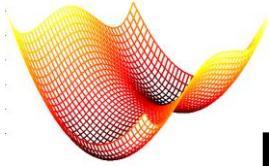


Konjugiertes Gradientenverfahren (4)

- Vergleich der Konvergenz von steilstem Abstieg (grün) mit konjugierten Gradienten (rot) bei quadratischen Funktionen $f_Q(\mathbf{x})$
- „Zick-Zack“ bei steilstem Abstieg wird vermieden!



© Wikipedia / Oleg Alexandrov

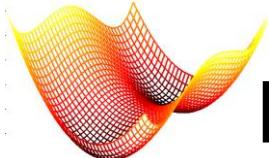


Konjugiertes Gradientenverfahren (5)

- **Konvergenzeigenschaften:** Bei quadr. Fkt. $f_Q(\mathbf{x})$ mit konjugierten Gradienten als s_k und exakter line search gilt:

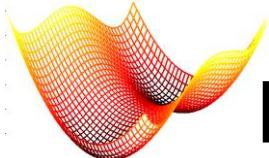
$$\|\mathbf{x}_k - \mathbf{x}^*\|_A \leq 2 \left(1 - \frac{2}{\sqrt{\kappa} + 1}\right)^k \cdot \|\mathbf{x}_0 - \mathbf{x}^*\|_A$$

- $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T \cdot A \cdot \mathbf{v}}$; $\kappa = \mu_1 / \mu_N$
- Wesentlich bessere Konvergenz für $\kappa \gg 1$, da dann $\sqrt{\kappa} \ll \kappa$
- Aber: **Konvergenz** ist immer noch (nur) **linear** und hängt immer noch von A ab!



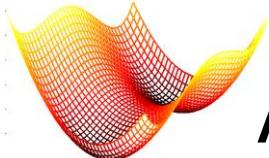
Konjugiertes Gradientenverfahren: Präkonditionierung

- **Idee:** Steigerung der Konvergenzgeschwindigkeit durch **Präkonditionierung:**
- Transformation so, dass die Matrix der transformierten quadratischen Zielfunktion eine wesentlich kleinere Konditionszahl κ hat
- Lineare Transformation der Zielvariablen $\mathbf{z} = \mathbf{L}^T \mathbf{x}$, d.h.:
$$\mathbf{x} = \mathbf{L}^{-T} \mathbf{z}$$
- Daraus folgt: $f_Q(\mathbf{z}) = \frac{1}{2} \mathbf{z}^T \cdot \mathbf{L}^{-1} \cdot \mathbf{A} \cdot \mathbf{L}^{-T} \cdot \mathbf{z} + (\mathbf{L}^{-1} \cdot \mathbf{b})^T \cdot \mathbf{z}$
- Vorteile, wenn Kondition von $\mathbf{L}^{-1} \cdot \mathbf{A} \cdot \mathbf{L}^{-T}$ wesentlich besser als Kondition von \mathbf{A}
- z.B.: $\mathbf{L} \cdot \mathbf{L}^T \approx \mathbf{A}$. Dann ist $\mathbf{L}^{-1} \cdot \mathbf{A} \cdot \mathbf{L}^{-T} \approx \mathbf{I}$



Konjugierte Gradientenverfahren: Varianten

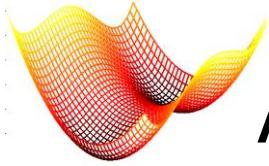
- **Idee:** wende das Verfahren auch auf „allgemeine“ Zielfunktionen (nicht nur quadratische) an (**Fletcher-Reeves**)
- Verbesserung der Konvergenz durch Modifikationen:
 - **Polak-Ribi  re:** $\gamma_{k+1} = \frac{\nabla f_Q(\mathbf{x}_{k+1})^T \cdot (\nabla f_Q(\mathbf{x}_{k+1}) - \nabla f_Q(\mathbf{x}_k))}{\nabla f_Q(\mathbf{x}_k)^T \cdot \nabla f_Q(\mathbf{x}_k)}$
 - **Hestenes-Stiefel:** $\gamma_{k+1} = \frac{\nabla f_Q(\mathbf{x}_{k+1})^T \cdot (\nabla f_Q(\mathbf{x}_{k+1}) - \nabla f_Q(\mathbf{x}_k))}{\mathbf{s}_k^T \cdot (\nabla f_Q(\mathbf{x}_{k+1}) - \nabla f_Q(\mathbf{x}_k))}$
 - „**Restart**“ mit negativem Gradienten nach einer festen Iterationsanzahl



Anwendungsbeispiel: BGA-Inspektion (1)

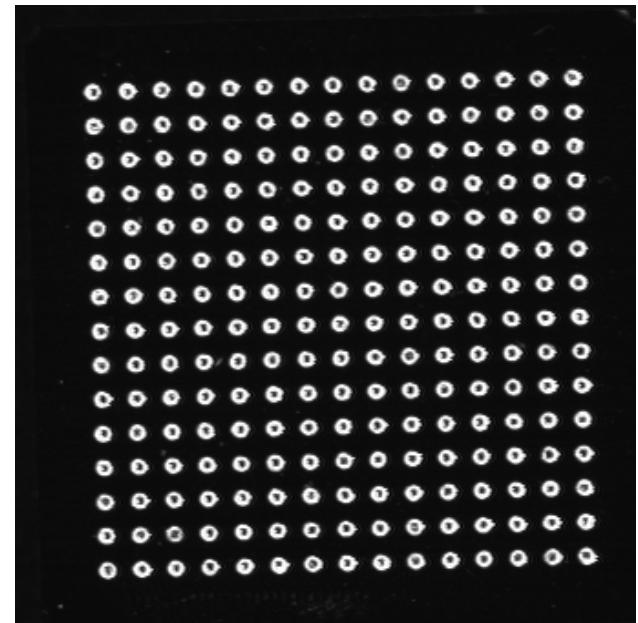
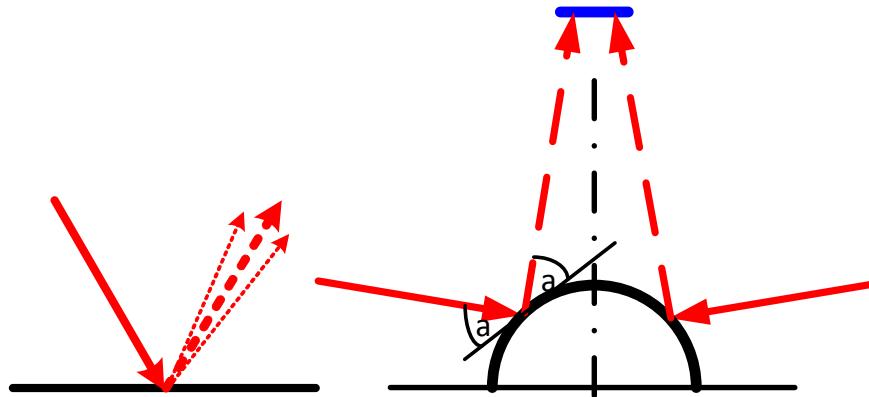
- Häufig vorkommendes SMD-Bauteil: **Ball Grid Array**, z.B. Prozessoren, Speicher, Chipsatz, etc.
- Balls als Anschlüsse auf der BE-Unterseite
- Müssen auf Defekte hin untersucht werden um sichere Funktion zu gewährleisten

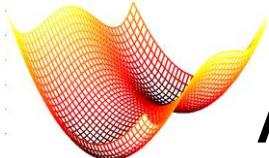




Anwendungsbeispiel: BGA-Inspektion (2)

- Bei flacher Beleuchtung bilden sich die Balls als helle Ringe ab
- Form der Ringe schwankt jedoch





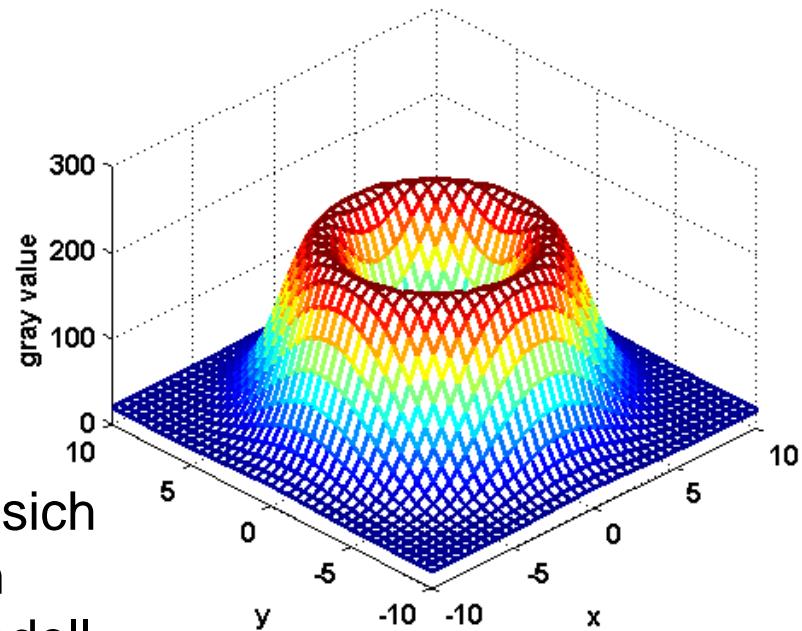
Anwendungsbeispiel: BGA-Inspektion (3)

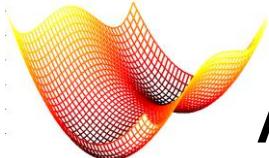
- Intakte Balls bilden sich jedoch immer als komplette Ringe ab, deren Form sich gut modellieren lässt:

$$M(x, y) = o + m \cdot \exp \left[- \left(\frac{\sqrt{(x - x_c)^2 + (y - y_c)^2} - r}{a} \right)^2 \right]$$

- Parameter des Modells lassen sich mittels Optimierung bestimmen (Minimiere Fehler zwischen Modell und abgebildeten Ball):

$$F(\mathbf{x}) = \sum_{x, y \in N} [I(x, y) - M(x, y, \mathbf{x})]^2$$





Anwendungsbeispiel: BGA-Inspektion (4)

- Gradient der Zielfunktion lässt sich analytisch berechnen → CG-Methode kann effizient eingesetzt werden

$$\frac{\partial F(\mathbf{x})}{\partial x_c} = \sum_{x,y \in N} \left[(2M(x,y) - 2I(x,y)) \cdot m \cdot e^k \cdot \left(\frac{2(x-x_c)}{a^2} \left(1 - \frac{r}{d}\right) \right) \right]$$

$$d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$$

$$\frac{\partial F(\mathbf{x})}{\partial y_c} = \sum_{x,y \in N} \left[(2M(x,y) - 2I(x,y)) \cdot m \cdot e^k \cdot \left(\frac{2(y-y_c)}{a^2} \left(1 - \frac{r}{d}\right) \right) \right]$$

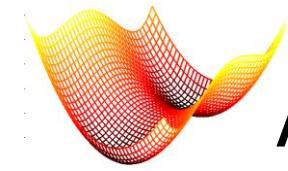
$$k = -((d - r)/a)^2$$

$$\frac{\partial F(\mathbf{x})}{\partial r} = \sum_{x,y \in N} \left[(2M(x,y) - 2I(x,y)) \cdot m \cdot e^k \cdot \left(-\frac{2r - 2d}{a^2} \right) \right]$$

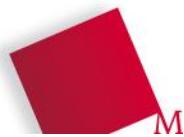
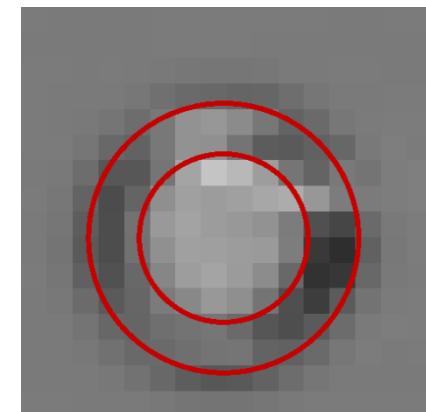
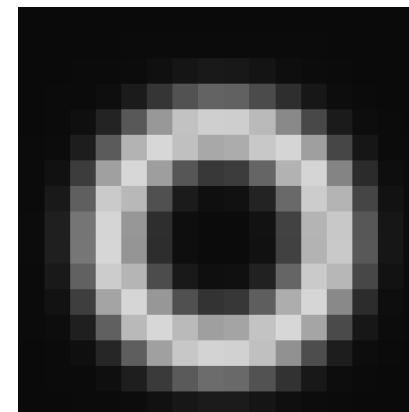
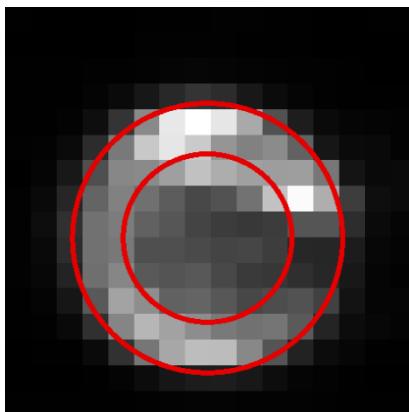
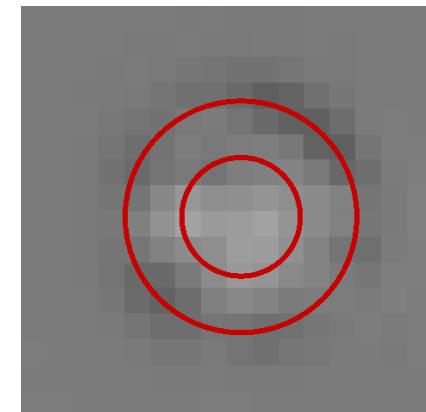
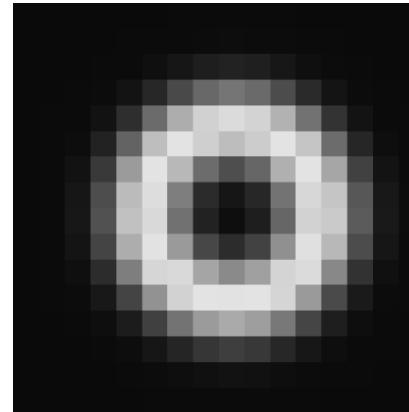
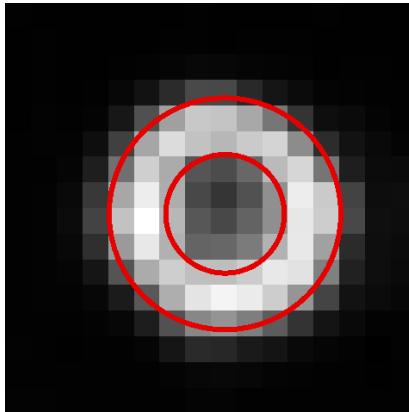
$$\frac{\partial F(\mathbf{x})}{\partial a} = \sum_{x,y \in N} \left[(2M(x,y) - 2I(x,y)) \cdot m \cdot e^k \cdot \left(-\frac{2(d-r)^2}{a^3} \right) \right]$$

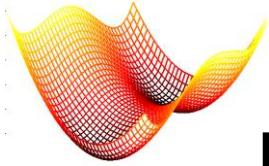
$$\frac{\partial F(\mathbf{x})}{\partial m} = \sum_{x,y \in N} [(2M(x,y) - 2I(x,y)) \cdot e^k]$$

$$\frac{\partial F(\mathbf{x})}{\partial o} = \sum_{x,y \in N} [2M(x,y) - 2I(x,y)]$$



Anwendungsbeispiel: BGA-Inspektion (5)



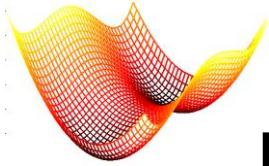


Newton-Verfahren (1)

- **Idee:** wende die Newton-Methode zur Nullstellen-Suche auf den (mehrdimensionalen) Gradienten $\nabla f(\mathbf{x}_k)$ an, d.h. finde in jeder Iteration die Nullstelle des Gradienten ($\nabla f(\mathbf{x}_k) = \mathbf{0}$) mittels Linearisierung von $\nabla f(\mathbf{x}_k)$:

$$\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k) \cdot \Delta \mathbf{x} \equiv \mathbf{0}$$

- Daraus ergibt sich:
$$\Delta \mathbf{x} = -\nabla^2 f(\mathbf{x}_k)^{-1} \cdot \nabla f(\mathbf{x}_k) = -\mathbf{H}(\mathbf{x}_k)^{-1} \cdot \nabla f(\mathbf{x}_k)$$
- Es wird also die Hesse-Matrix $\mathbf{H}(\mathbf{x}_k)$, d.h. die zweiten Ableitungen von $f(\mathbf{x}_k)$ benötigt → **Verfahren 2. Ordnung**
- Praktische Lösung mittels numerischer Lösung des Gleichungs-Systems $\mathbf{H}(\mathbf{x}_k) \cdot \Delta \mathbf{x} = -\nabla f(\mathbf{x}_k)$ vermeidet Matrix-Inversion

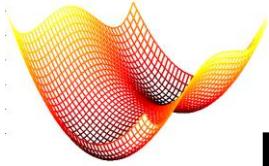


Newton-Verfahren (2)

- Startwert für die nächste Iteration: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}$
- Achtung: die Suchrichtung ist dabei nicht identisch mit dem steilsten Abstieg $-\nabla f(\mathbf{x}_k)!$
- Die Linearisierung von $\nabla f(\mathbf{x}_k)$ ist äquivalent zur **Approximation** von $f(\mathbf{x}_k)$ mittels **Taylor-Reihe** 2. Grades:
$$f(\mathbf{x}_k + \Delta\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k) \cdot \Delta\mathbf{x} + 1/2 \cdot \Delta\mathbf{x}^T \cdot \mathbf{H}(\mathbf{x}_k) \cdot \Delta\mathbf{x}$$

→ **Zweite Interpretation:** in jeder Iteration wird $f(\mathbf{x}_k)$ durch eine quadratische Form approximiert, welche analytisch gelöst wird
- **Variante:** gedämpftes Newton-Verfahren:

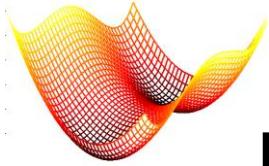
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \mathbf{H}(\mathbf{x}_k)^{-1} \cdot \nabla f(\mathbf{x}_k) \text{ mit } 0 < \lambda_k \leq 1$$



Newton-Verfahren (3)

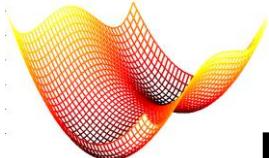
- Pseudocode:

```
function optimizeNewton(in objective  $f(\mathbf{x})$ , in start  
 $\mathbf{x}_0$ , in  $\epsilon$ , out solution  $\mathbf{x}^*$ )  
repeat  
    1.) calculate gradient  $\nabla f(\mathbf{x}_k)$   
    2.) calculate Hesse-Matrix  $\mathbf{H}(\mathbf{x}_k)$   
    3.) calculate displacement  $\Delta \mathbf{x}$  by solving the  
        linear equation system  $\mathbf{H}(\mathbf{x}_k) \cdot \Delta \mathbf{x} = -\nabla f(\mathbf{x}_k)$   
    4.) update current position:  $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}$   
     $k \leftarrow k + 1$   
until convergence, e.g.  $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \epsilon$ 
```

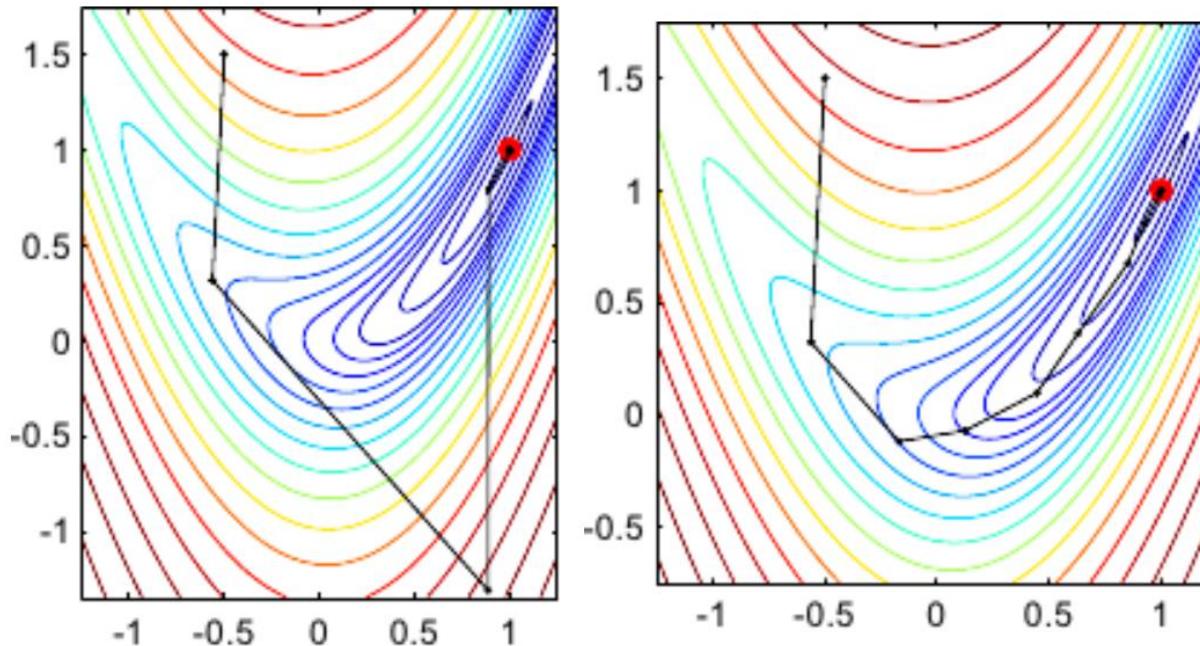


Newton-Verfahren: Eigenschaften

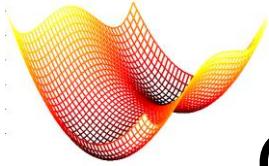
- **Sehr gute Konvergenzeigenschaften: quadratische Konvergenz** in lokaler Umgebung des gesuchten Minimums \mathbf{x}^* (Konvergenzbereich)
- **Affine Invarianz**, d.h. wird das Verfahren auf das linear transformierte Problem mit $f(\mathbf{A} \cdot \mathbf{z})$ mit $\mathbf{z} = \mathbf{A}^{-1} \cdot \mathbf{x}$ angewandt, so ist nicht nur die Lösung \mathbf{x}^* , sondern auch alle Iterierten \mathbf{x}_k identisch
- **Aber:** der **Konvergenzbereich** ist in der Regel **klein** → eignet sich als „letzter Schritt“
- Berechnung der Hesse-Matrizen in jeder Iteration teuer
→ **Wunsch: Approximation** von $\mathbf{H}(\mathbf{x}_k)$ durch einfacher zu berechnende Matrizen



Newton-Verfahren: Konvergenz



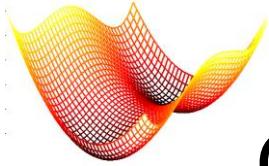
© Florent Brunet; links: Klassischer Newton-Algorithmus (6 Iterationen);
rechts: gedämpftes Newton-Verfahren (10 Iterationen, aber keine
„Überschwinger“ → größerer Konvergenzbereich)



Quasi-Newton-Verfahren (1)

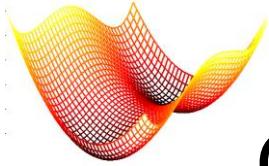
- **Idee:** Verfahren mit ähnlich guten Konvergenz-Eigenschaften wie bei Newton-Methode, aber Vermeidung der aufwändigen Berechnung von $\mathbf{H}(\mathbf{x}_k)$
- **Approximation** der $\mathbf{H}(\mathbf{x}_k)$ durch schneller zu berechnende $\mathbf{B}(\mathbf{x}_k)$ bzw. $\mathbf{R}(\mathbf{x}_k) = \mathbf{B}(\mathbf{x}_k)^{-1} \approx \mathbf{H}(\mathbf{x}_k)^{-1}$:
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \mathbf{R}(\mathbf{x}_k) \cdot \nabla f(\mathbf{x}_k)$$
- Notwendige Bedingung für superlineare Konvergenz:
$$\mathbf{R}(\mathbf{x}_k) \cdot (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) \approx \mathbf{x}_{k+1} - \mathbf{x}_k$$

mit $\mathbf{y}_k := \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$; $\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k$ gilt:
$$\mathbf{R}(\mathbf{x}_k) \cdot \mathbf{y}_k \approx \mathbf{s}_k$$



Quasi-Newton-Verfahren (2)

- **Problem:** y_k und s_k müssen erst mit Hilfe von $R(x_k)$ berechnet werden
- Man kann jedoch definieren: $R_{k+1} \cdot y_k = s_k$ (**Quasi-Newton-Bedingung**, die für superlineare Konvergenz erfüllt sein muss)
- **Iteratives Verfahren:** Näherung R_{k+1} der Hesse-Matrix für die nächste Iteration lässt sich von y_k und s_k ableiten
- **Weitere Bedingungen:**
 - H_k symmetrisch $\rightarrow R_k$ bzw. B_k sollen ebenfalls symmetrisch sein
 - R_{k+1} soll sich von R_k möglichst wenig unterscheiden

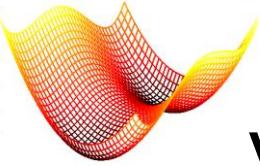


Quasi-Newton-Verfahren (3)

- **BFGS-Update-Formel** (Broyden, Fletcher, Goldfarb, Shanno):

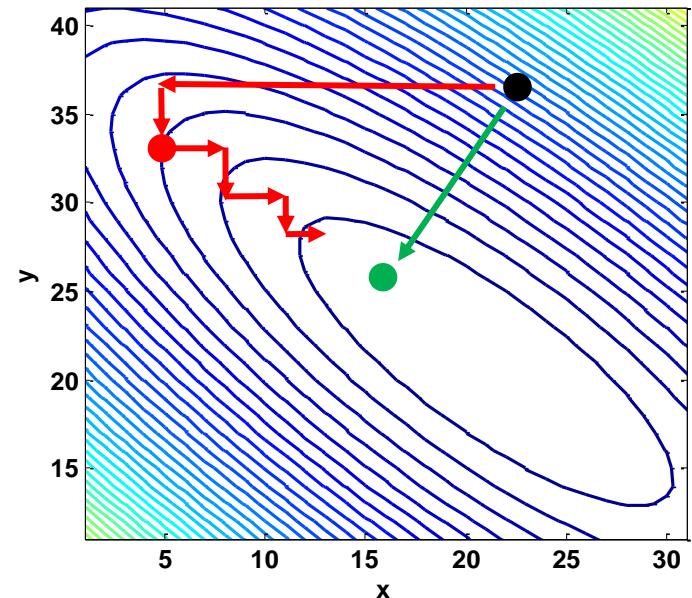
$$\mathbf{R}_{k+1} = \mathbf{R}_k + \frac{(\mathbf{s}_k - \mathbf{R}_k \mathbf{y}_k) \mathbf{s}_k^T + \mathbf{s}_k (\mathbf{s}_k - \mathbf{R}_k \mathbf{y}_k)^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{(\mathbf{s}_k - \mathbf{R}_k \mathbf{y}_k)^T \mathbf{y}_k}{(\mathbf{s}_k^T \mathbf{y}_k)^2}$$

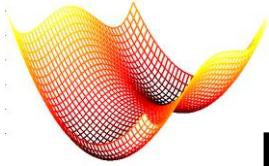
- Interessant vor allem bei voll besetzter Hesse-Matrix \mathbf{H}_k
- **Start:** $\mathbf{R}_0 = \mathbf{H}(\mathbf{x}_0)$ oder $\mathbf{R}_0 = \mathbf{I}$ (Identitäts-Matrix, dann äquivalent zum Konjugierten Gradientenverfahren)
- **Bedingung** für superlineare Konvergenz: $\mathbf{s}_k^T \mathbf{y}_k > 0$ für alle Iterierten. Ist diese erfüllt, so konvergiert das Verfahren **superlinear**



Verfahren ohne Ableitungen

- Informationen über die Ableitungen sind oft nur schwer und/oder aufwändig zu beschaffen → Gibt es Verfahren, welche ohne Ableitungen auskommen?
- Offensichtliche Möglichkeit: zyklisches Verwenden der Einheitsvektoren e_i als Suchrichtungen („taxi-cab-Methode“)
- Jedoch: **sehr schlechte Konvergenz**

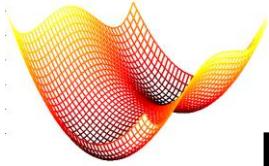




Powell-Methode (1)

- **Idee zur Verbesserung der Konvergenz:** Wieder-verwenden von Information aus vorherigen Iterationen zur Verbesserung der Suchrichtungsberechnung
- Mathematisch: versuche die in der Hesse-Matrix \mathbf{H} enthaltene Info durch diese gesammelte Information in Matrix $\tilde{\mathbf{H}}$ zu **approximieren**
- **Start:** Taxi-Cab-Methode für jede der N Dimensionen
- Eine Iteration der Powell-Methode umfasst zunächst N 1-dim. Optimierungen mit Schrittweiten α^i
- **Erste Approximation** der „Hesse-Info“ gemäß der ermittelten α^i :

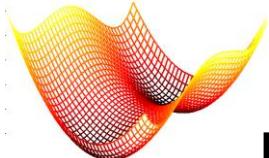
$$\tilde{\mathbf{H}}^0 = \begin{bmatrix} \alpha^1 & & & \mathbf{0} \\ & \alpha^2 & & \\ & & \ddots & \\ \mathbf{0} & & & \alpha^N \end{bmatrix}$$



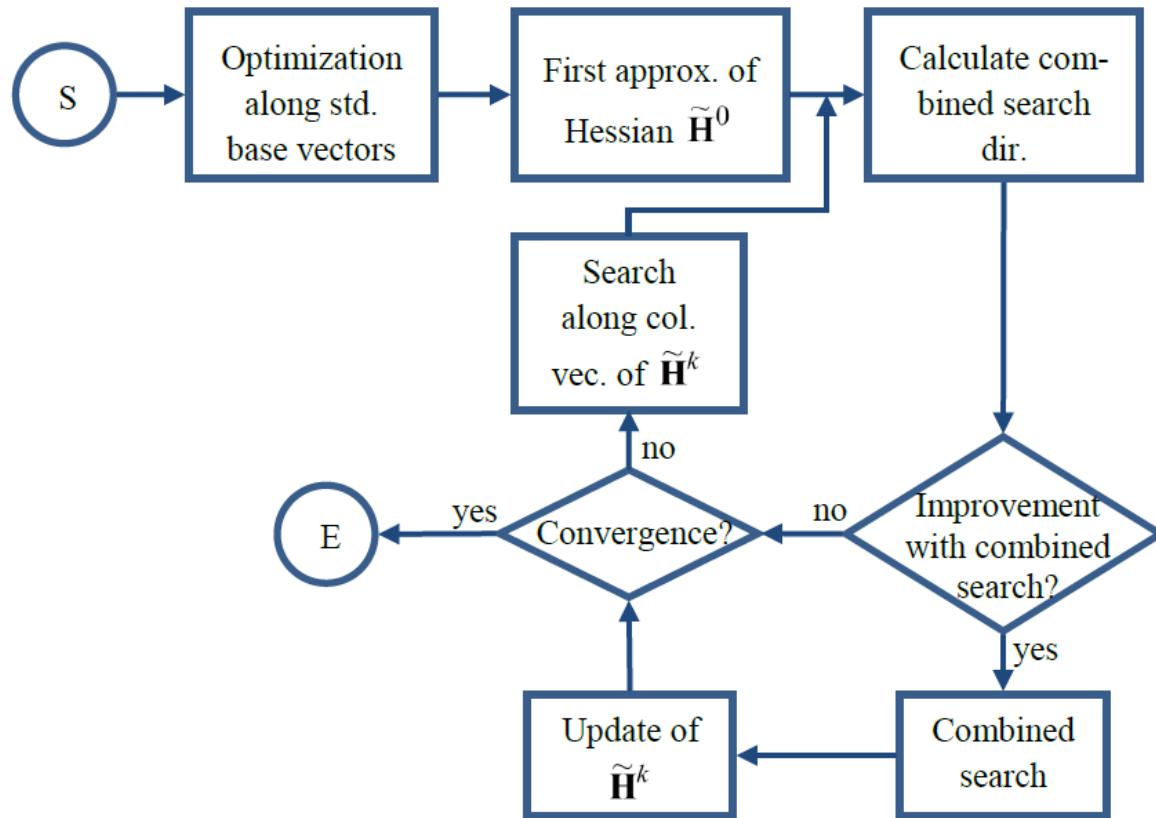
Powell-Methode (2)

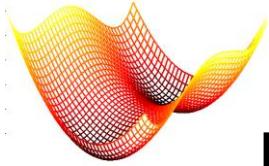
- Am Ende jeder Iteration: berechne **kombinierten Suchvektor** $s_k^{N+1} = \sum_{i=1}^N \alpha^i s_k^i$ und führe kombinierte Suche entlang dieser Suchrichtung durch
- In der ersten Iteration ist dies $s_1^{N+1} = \sum_{i=1}^N \alpha^i e_i$
- **Update der „Hesse-Info“:** ersetze eine Spalte von \widetilde{H}_k durch diese kombinierte Suchrichtung s_k^{N+1}
- Wiederhole diese Schritte, bis Konvergenz, z.B. $|x_{k+1} - x_k| \leq \epsilon$
- Spalten i von \widetilde{H}_k definieren die Suchrichtungen der eindimensionalen Suchen:

$$s_k^i = [h_{1,i}^k \quad \cdots \quad h_{N,i}^k]^T$$



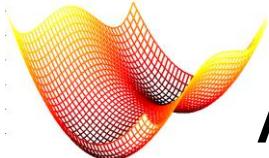
Powell-Methode (3)





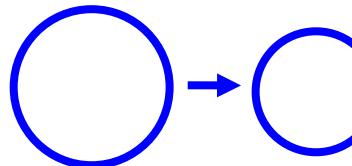
Powell-Methode (4)

- **Offene Frage:** welche Spalte von $\widetilde{\mathbf{H}}_k$ soll durch kombinierte Suchrichtung \mathbf{s}_k^{N+1} ersetzt werden?
- **Modifizierte Powell-Methode:** wähle diejenige Spalte i_r von $\widetilde{\mathbf{H}}_k$, bei deren eindimensionaler Suche sich die **größte Reduktion** des Funktionswertes erzielen ließ, d.h.
$$r = \operatorname{argmax}_r (f_k^{r-1} - f_k^r)$$
- **Hintergrund:** diese Spalte beeinflusst die kombinierte Suchrichtung maßgeblich und sollte daher zur Vermeidung linearer Abhängigkeit der Spalten von $\widetilde{\mathbf{H}}_k$ eliminiert werden.
- **Konvergenz:** Für quadratische Zielfkt. $f_Q(\mathbf{x})$ konvergieren die Suchrichtungen zu zueinander konjugierten Vektoren

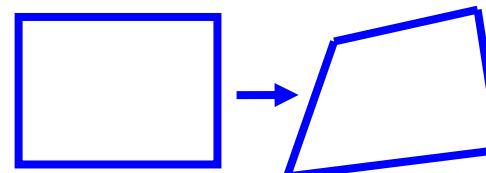
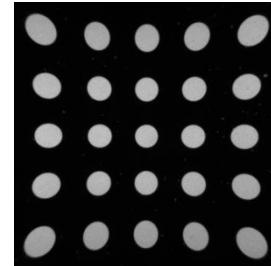


Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (1)

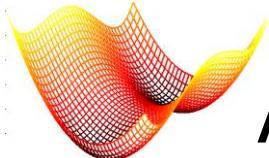
- **Kamera als Mess-System:** Bilder sollten frei von Verzeichnung sein
- Einsatz teurer Optiken irgendwann nicht mehr wirtschaftlich
→ Bestimmen der Verzeichnung + Korrektur per Software
- Zwei Komponenten: **Perspektive + radiale Verzeichnung**



$$d' = \frac{d}{1 + \kappa \cdot |d|^2}$$



$$\begin{bmatrix} w \cdot u \\ w \cdot v \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (2)

- **Mathematische Formulierung** des Problems:
- Jedes Pixel $[x_i, y_i]$ im unverzerrten Bild wird auf das Pixel $[u_i, v_i]$ im verzerrten Bild abgebildet:
 $[u_i, v_i] = T(\mathbf{t}, [x_i, y_i])$
- Parametervektor \mathbf{t} hat 11 Elemente (3 für radiale Verzeichnung + 8 für Perspektive)

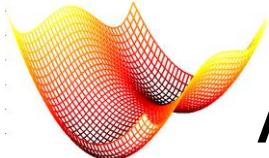
$$u_i(\mathbf{t}) = \cos \left[\arctan \left(\frac{p_2(\mathbf{t})}{p_1(\mathbf{t})} \right) \right] \cdot d(\mathbf{t}) + x_c$$

$$v_i(\mathbf{t}) = \sin \left[\arctan \left(\frac{p_2(\mathbf{t})}{p_1(\mathbf{t})} \right) \right] \cdot d(\mathbf{t}) + y_c$$

$$d(\mathbf{t}) = \frac{\sqrt{p_1(\mathbf{t})^2 + p_2(\mathbf{t})^2}}{1 + \kappa \cdot (p_1(\mathbf{t})^2 + p_2(\mathbf{t})^2)} \quad \text{with}$$

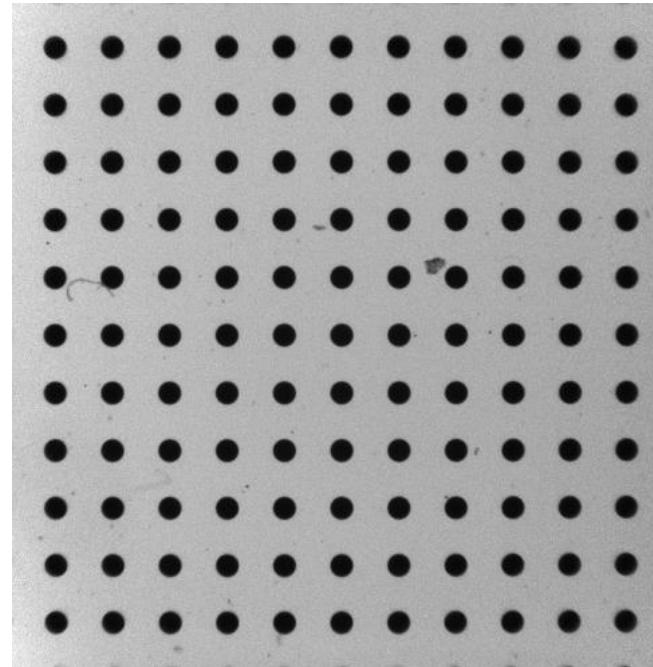
$$p_1(\mathbf{t}) = \frac{a_{11}x_i + a_{12}y_i + a_{13}}{a_{31}x_i + a_{32}y_i + 1} - x_c$$

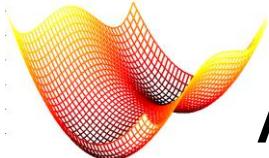
$$p_2(\mathbf{t}) = \frac{a_{21}x_i + a_{22}y_i + a_{23}}{a_{31}x_i + a_{32}y_i + 1} - y_c$$



Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (3)

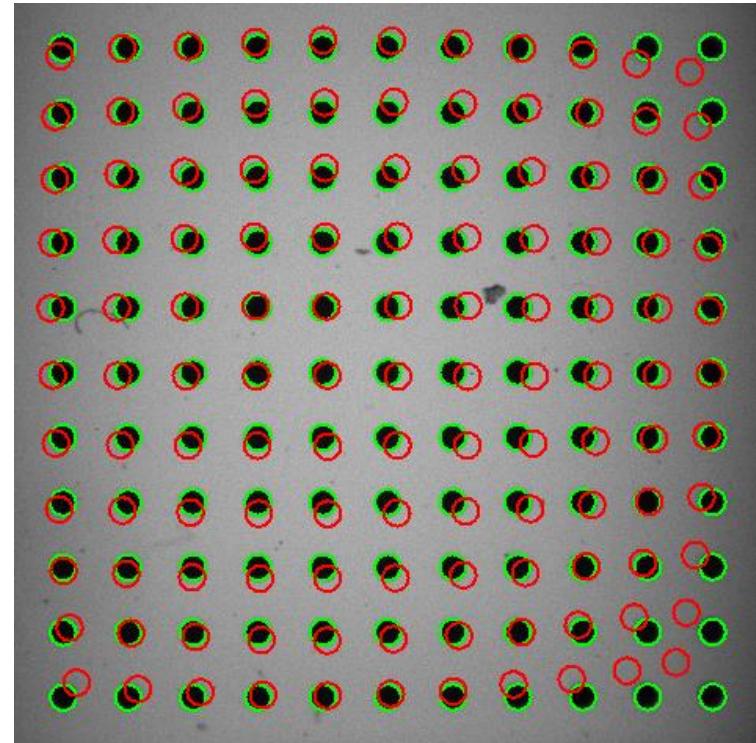
- Bestimmen der Koeffizienten t durch Vermessen eines (hochgenauen) Prüfnormals

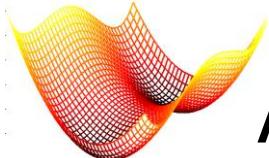




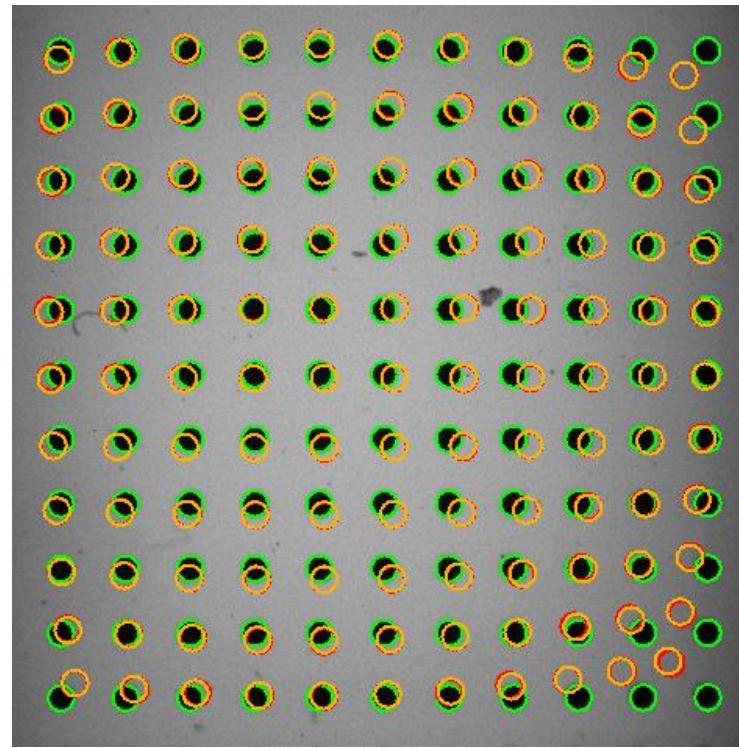
Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (4)

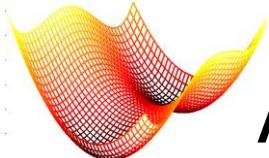
- Zu jeder Soll-Position $[x_i, y_i]$ (grün) wird eine Ist-Position 1.) berechnet ($T(\mathbf{t}, [x_i, y_i])$) sowie 2.) durch Messung ermittelt ($[u_i, v_i]$) (rot; Fehler vervielfacht)
- **Zielfunktion:** Es wird die Summe der Fehlerquadrate (Fehler = Differenz $T(\mathbf{t}, [x_i, y_i]) - [u_i, v_i]$, d.h. Abweichung zwischen Modell und Messung) an allen Messpunkten minimiert





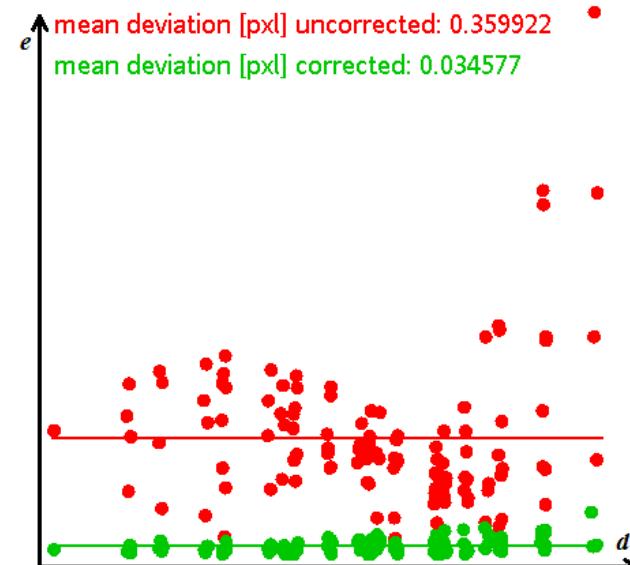
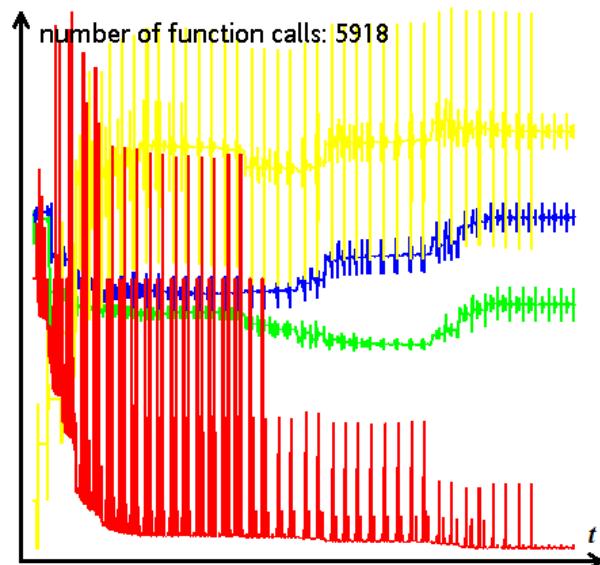
Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (5)

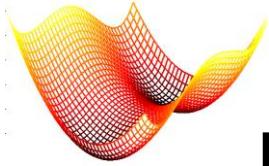




Anwendungsbeispiel Powell-Methode: Verzeichnungskorrektur Kamera (6)

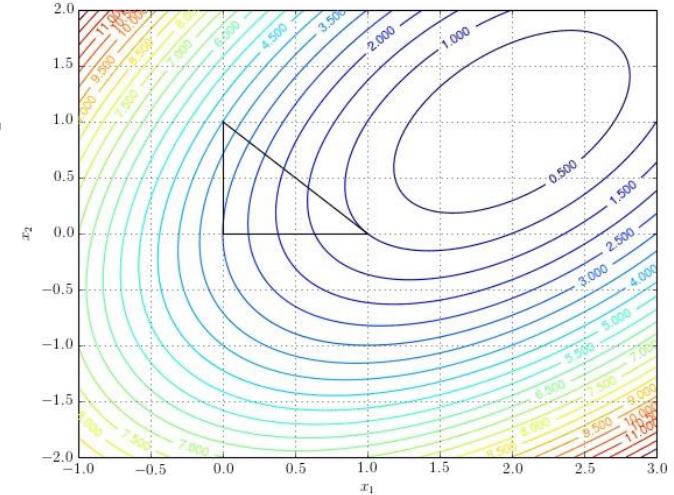
- Verlauf einiger Parameter während der Optimierung (links, rot: Fehlersumme, gelb/grün/blau: ausgewählte Parameter)
- Rechts: Fehler vor (rot) und nach der Optimierung (grün)



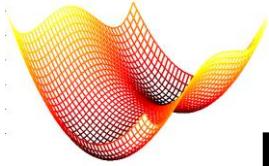


Nelder-Mead-Methode (1)

- **Heuristik**, welche **ohne Ableitungen** auskommt
- **Basiert auf $N + 1$ -Polytopen**
„Simplex“: geometrisches Gebilde, welches im N -dimensionalen Raum aus $N + 1$ Ecken besteht, z.B. Dreieck bei $N = 2$, Tetraeder bei $N = 3$
- Betrachtet werden die **Funktionswerte in den Ecken des Simplex**, d.h. $f(\mathbf{x}_{\min}) = f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \dots \leq f(\mathbf{x}_{N+1}) = f(\mathbf{x}_{\max})$

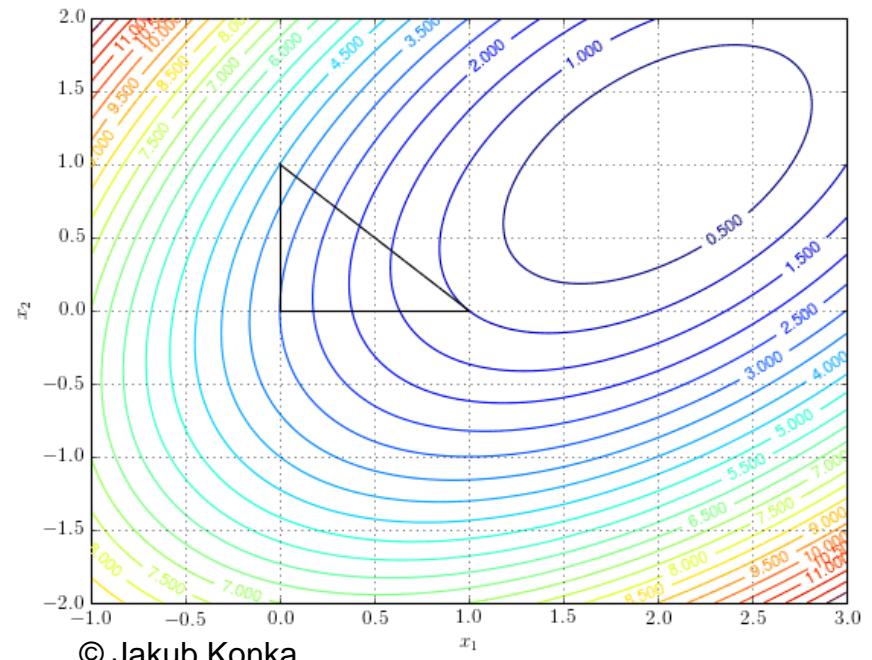


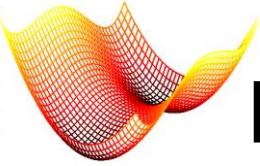
© Jakub Konka



Nelder-Mead-Methode (2)

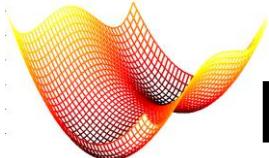
- **Idee:** ersetze in jeder Iteration die „schlechteste Ecke“ x_{\max} (mit dem größten Wert der Zielfunktion) durch eine „bessere“ Position mit kleinerem Funktionswert
- Auf **beliebige Funktionen** $f(\mathbf{x})$ anwendbar
- Wird in MATLAB-Funktion **fminsearch** verwendet





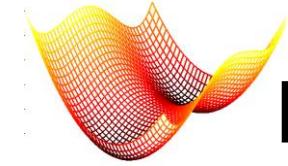
Nelder-Mead-Methode: Eckentausch (1)

- Berechne Zentrum \mathbf{x}_0 aller Eckpunkte ohne \mathbf{x}_{\max}
- Spiegle \mathbf{x}_{\max} an \mathbf{x}_0 , d.h. berechne $\mathbf{x}_r = \mathbf{x}_0 + \alpha \cdot (\mathbf{x}_0 - \mathbf{x}_{\max})$ mit $\alpha > 0$
- **Idee:** „auf der anderen Seite“ des Simplex ist die Situation vermutlich besser als bei \mathbf{x}_{\max}
- Wenn $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_{\max})$: Situation konnte verbessert werden → tausche \mathbf{x}_{\max} mit \mathbf{x}_r („**Reflexion**“)
- Wenn sogar $f(\mathbf{x}_r) < f(\mathbf{x}_{\min})$: evtl. weitere Verbesserung möglich: berechne $\mathbf{x}_e = \mathbf{x}_0 + \beta \cdot (\mathbf{x}_0 - \mathbf{x}_{\max})$ mit $\beta > \alpha$, um Simplex zu vergrößern und größere Schritte zu realisieren
- Wenn $f(\mathbf{x}_e) < f(\mathbf{x}_r)$: tausche \mathbf{x}_{\max} mit \mathbf{x}_e („**Expansion**“)

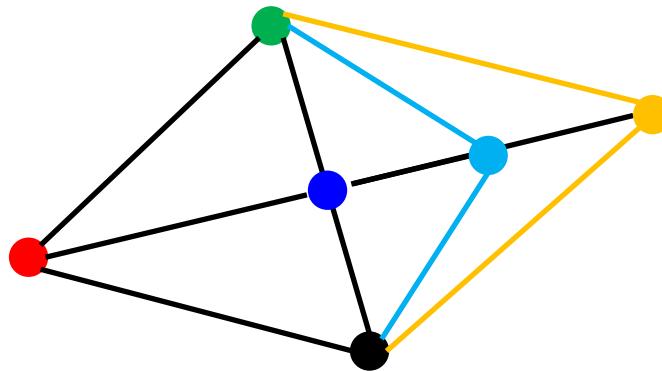


Nelder-Mead-Methode: Eckentausch (2)

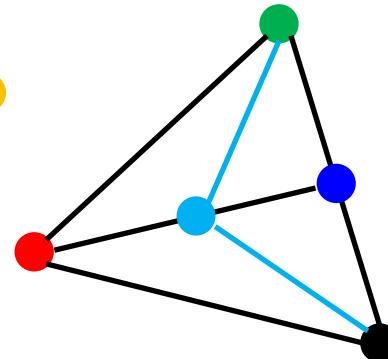
- Wenn $f(\mathbf{x}_r) \geq f(\mathbf{x}_{\max})$: evtl. Verbesserung im Inneren des Simplex, d.h. berechne $\mathbf{x}_c = \mathbf{x}_0 + \gamma \cdot (\mathbf{x}_0 - \mathbf{x}_{\max})$ mit $-1 < \gamma < 0$, d.h. \mathbf{x}_c liegt „zwischen“ \mathbf{x}_0 und \mathbf{x}_{\max}
- Wenn $f(\mathbf{x}_c) < f(\mathbf{x}_{\max})$: Situation konnte verbessert werden
→ tausche \mathbf{x}_{\max} mit \mathbf{x}_c („**Kontraktion**“, da Simplex kleiner wird)
- Sonst: schrumpfe Simplex „hin zu \mathbf{x}_{\min} “, d.h. ersetze alle Punkte außer \mathbf{x}_{\min} durch $\mathbf{x}'_i = \mathbf{x}_{\min} + \delta \cdot (\mathbf{x}_i - \mathbf{x}_{\min})$ („**komprimiere**“ **Simplex**)
- Typische Werte: $\alpha = 1$, $\beta = 2$, $\gamma = -1/2$, $\delta = 1/2$



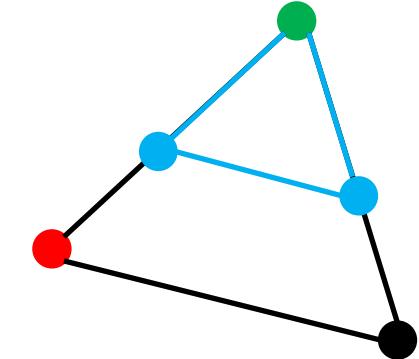
Nelder-Mead-Methode: Eckentausch (3)



Reflexion/Expansion



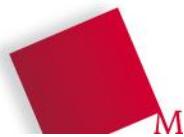
Kontraktion

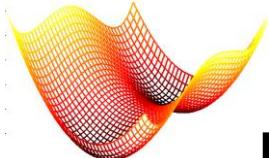


Komprimierung

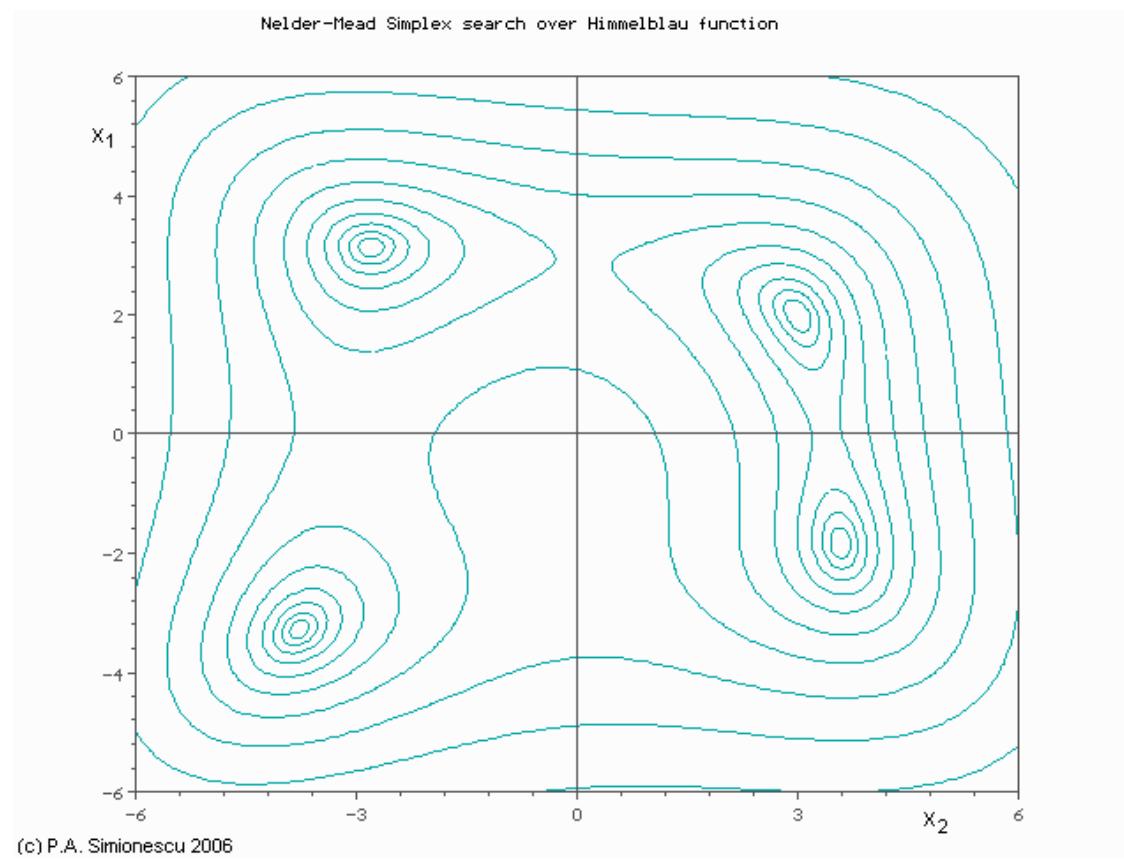
Eigenschaften:

- Relativ robust, benötigt aber viele Iterationen und muss nicht unbedingt gegen ein Minimum konvergieren
- Performance hängt stark vom Start-Simplex ab!

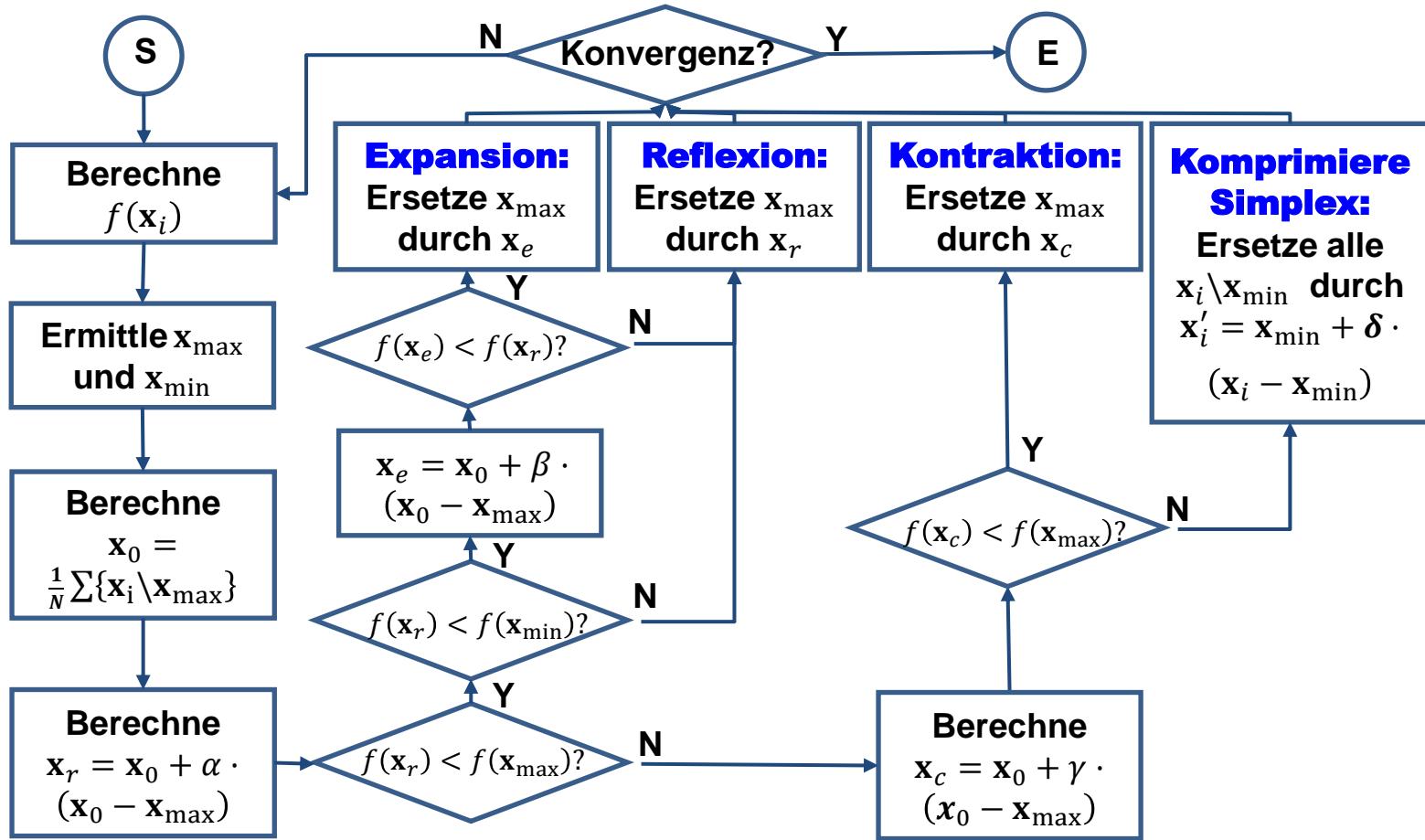


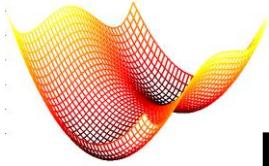


Nelder-Mead-Methode: Beispiel



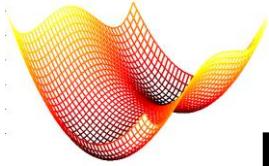
Nelder-Mead-Methode: Flowchart





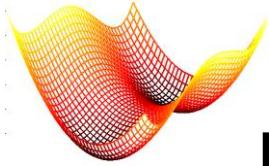
Nichtlineare Ausgleichsrechnung (1)

- **Gegeben:** $f: \mathbb{R}^N \rightarrow \mathbb{R}^M$: $\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_M(\mathbf{x}) \end{bmatrix}$
- **Gesucht:** \mathbf{x} , welches $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ möglichst gut erfüllt
- In der Regel $M > N \rightarrow$ keine exakte Lösung
- Deshalb ist die Aufgabe: **Minimiere das Fehlerquadrat**
$$\phi(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^M f_i(\mathbf{x})^2$$
- **Lösung** des Problems ist dann $\mathbf{x}^* = \operatorname{argmin}(\phi(\mathbf{x}))$



Nichtlineare Ausgleichsrechnung (2)

- $f_i(\mathbf{x})$ haben häufig die Form $f_i(\mathbf{x}) = m_i(\mathbf{x}) - y_i$
- **Interpretation:**
 - i : Versuchsindex
 - y_i : Messwerte
 - $m_i(\mathbf{x})$: Modellfunktion, welche die y_i möglichst gut approximieren soll
 - \mathbf{x} : (gesuchte) Parameter der Modellfunktion
 - $f_i(\mathbf{x})$: Messfehler
- Aufgabe ist dann: finde einen Modellparameter \mathbf{x}^* so, dass die Messfehler „in Summe“ möglichst klein sind



Nichtlineare Ausgleichsrechnung (3)

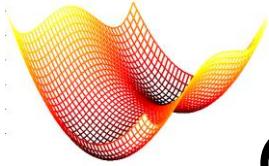
- Wie sehen die **Ableitungen** bei dieser speziellen Struktur von $\phi(\mathbf{x})$ aus?
- **Kettenregel** liefert z.B. für $\frac{\partial \phi(\mathbf{x})}{\partial x_1}$:

$$\frac{\partial \phi(\mathbf{x})}{\partial x_1} = \frac{1}{2} \sum_i 2 \cdot f_i(\mathbf{x}) \cdot \frac{\partial f_i(\mathbf{x})}{\partial x_1}$$

- Daraus ergibt sich für den **Gradient** $\nabla \phi(\mathbf{x})$:

$$\nabla \phi(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_M(\mathbf{x})}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_N} & \dots & \frac{\partial f_M(\mathbf{x})}{\partial x_N} \end{bmatrix} \cdot \mathbf{f}(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T \cdot \mathbf{f}(\mathbf{x})$$

- **$\mathbf{J}(\mathbf{x})$: Jacobi-Matrix von $\mathbf{f}(\mathbf{x})$ (Matrix der Ableitungen)**



Gauss-Newton-Verfahren (1)

- 2. Ableitung: Hesse-Matrix:

$$\mathbf{H}(\phi(\mathbf{x})) = \mathbf{J}(\mathbf{x})^T \cdot \mathbf{J}(\mathbf{x}) + \mathbf{B}(\mathbf{x}) \text{ mit } \mathbf{B}(\mathbf{x}) = \sum_i f_i(\mathbf{x}) \cdot \nabla^2 f_i(\mathbf{x})$$

- **Optimierung mit Newton-Methode**, d.h.

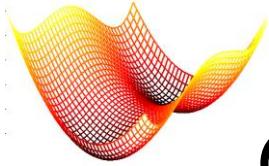
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \mathbf{H}(\phi(\mathbf{x}_k))^{-1} \cdot \nabla \phi(\mathbf{x}_k)$$

- Jedoch: Berechnung von $\mathbf{B}(\mathbf{x}_k)$ oft sehr aufwändig

→ **Näherung**: Weglassen von $\mathbf{B}(\mathbf{x}_k)$

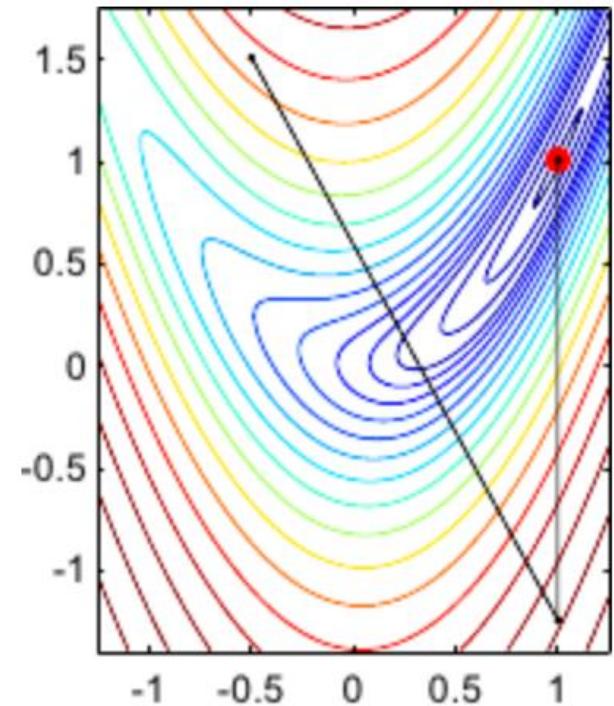
→ **Gauss-Newton-Verfahren**:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot [\mathbf{J}(\mathbf{x})^T \cdot \mathbf{J}(\mathbf{x})]^{-1} \cdot \mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{f}(\mathbf{x}_k)$$

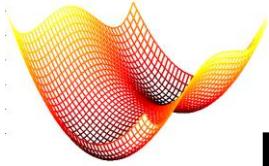


Gauss-Newton-Verfahren (2)

- **Konvergenz ähnlich gut zu Newton-Verfahren, wenn „Messfehler“ $f_i(x)$ klein** (dann sind die Elemente von $B(x^*)$ klein).
- Beispiel: Rosenbrock-Funktion
 $f(x, y) = 10(y - x^2)^2 + (x - 1)^2$
- Kann zu Fehlerquadratsumme umgeformt werden mit $f_1(x) = x - 1$ und $f_2(x, y) = \sqrt{10} \cdot (y - x^2)$
- Jedoch: **schlechte Konvergenz bei verrauschten Messwerten**



© Florent Brunet

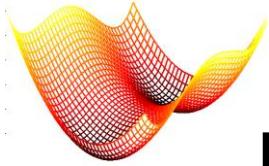


Lineare Ausgleichsrechnung

- **Spezialfall:** alle $f_i(\mathbf{x})$ linear, d.h. $\mathbf{f}_L(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x} - \mathbf{b}$
- Dann gilt: $\mathbf{J}(\mathbf{f}_L(\mathbf{x})) = \mathbf{A}$
- Aus $\nabla \phi_L(\mathbf{x}) = \mathbf{0}$ ergibt sich:

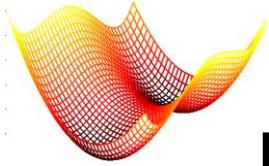
$$\begin{aligned}\mathbf{A}^T \cdot (\mathbf{A} \cdot \mathbf{x} - \mathbf{b}) &= \mathbf{0} \\ \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{x} &= \mathbf{A}^T \cdot \mathbf{b}\end{aligned}$$

- Lösung ist eindeutig, falls Spalten von \mathbf{A} unabhängig, dann ist $\mathbf{A}^T \cdot \mathbf{A}$ positiv definit



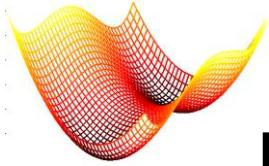
Levenberg-Marquardt-Algorithmus (1)

- **Ziel:** Verbesserung der Konvergenz bei verrauschten Messwerten
- Suchrichtung bei Gauss-Newton-Verfahren:
$$\mathbf{s}_k = -[\mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{J}(\mathbf{x}_k)]^{-1} \cdot \mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{f}(\mathbf{x}_k)$$
- **Idee** von Levenberg:
$$\mathbf{s}_k = -[\mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{J}(\mathbf{x}_k) + \lambda \mathbf{I}]^{-1} \cdot \mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{f}(\mathbf{x}_k)$$
- **Dämpfungsfaktor** λ bestimmt Verhalten des Verfahrens
- **Große** λ : $\mathbf{s}_k \approx -[\lambda \mathbf{I}]^{-1} \cdot \mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{f}(\mathbf{x}_k) = 1/\lambda \cdot \nabla \phi(\mathbf{x}_k)$
- \mathbf{s}_k entspricht bei großen λ ca. dem steilsten Abstieg
- Gute Wahl wenn weit vom Minimum entfernt, da dann garantierte Verbesserung



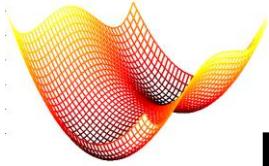
Levenberg-Marquardt-Algorithmus (2)

- **Kleine** λ : Suchrichtung entspricht im Wesentlichen dem Gauss-Newton-Verfahren
- Gute Wahl wenn nah am Minimum, da dann gute Konvergenz des Gauss-Newton-Verfahrens
- **Aktualisiere** λ_k in jeder Iteration für besten Kompromiss:
 - Wenn $\phi(\mathbf{x}_{k+1}) < \phi(\mathbf{x}_k)$: $\lambda_{k+1} = 1/\nu \cdot \lambda_k$, z.B. $\nu = 2$
 - Sonst: $\lambda_{k+1} = \nu \cdot \lambda_k$, d.h. vergrößere λ und kein Update von \mathbf{x}



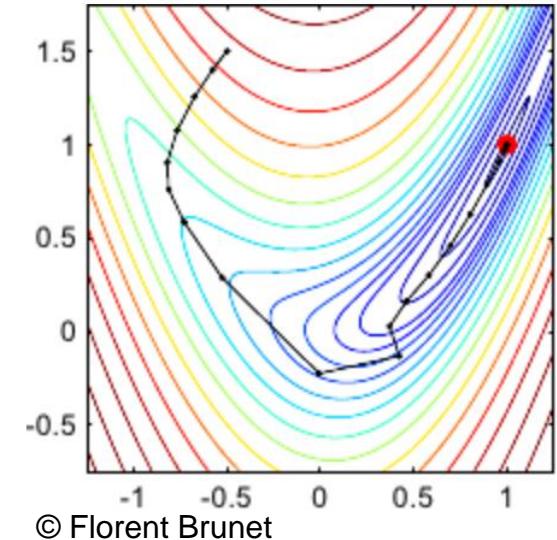
Levenberg-Marquardt-Algorithmus (3)

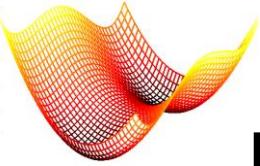
- **Nachteil** des Levenberg-Algorithmus: für große λ_k wird Krümmung nicht berücksichtigt (keine 2. Ableitungen)
→ **Idee** von Marquardt: Ersetze \mathbf{I} durch Matrix, welche 2. Ableitungen approximiert, d.h. verwende $\mathbf{J}(\mathbf{x}_k)^T \mathbf{J}(\mathbf{x}_k)$:
$$\mathbf{s}_k = -[\mathbf{J}(\mathbf{x}_k)^T \mathbf{J}(\mathbf{x}_k) + \lambda \cdot \text{diag}[\mathbf{J}(\mathbf{x}_k)^T \mathbf{J}(\mathbf{x}_k)]]^{-1} \cdot \mathbf{J}(\mathbf{x}_k)^T \cdot \mathbf{f}(\mathbf{x}_k)$$
- Weitere **Modifikation**: verbessertes Update von λ_k :
 - Berechne pro It. zwei Lösungs-Kandidaten mit λ_k und λ_k/ν und betrachte $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k}$ und $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k/\nu}$
 - $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k} < \mathbf{f}(\mathbf{x}_k)$ und $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k/\nu} < \mathbf{f}(\mathbf{x}_k)$: Verminderung von λ_k verbessert Situation →
$$\lambda_{k+1} = \lambda_k / \nu$$



Levenberg-Marquardt-Algorithmus (4)

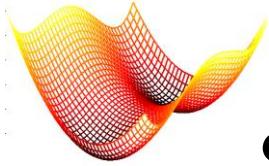
- $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k} < \mathbf{f}(\mathbf{x}_k)$ und $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k/\nu} > \mathbf{f}(\mathbf{x}_k)$:
Verbesserung nur bei Betonung auf
Abstieg $\rightarrow \lambda_{k+1} = \lambda_k$
- $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k} > \mathbf{f}(\mathbf{x}_k)$ und $\mathbf{f}(\mathbf{x}_{k+1})_{\lambda_k/\nu} > \mathbf{f}(\mathbf{x}_k)$:
keine Verbesserung \rightarrow
„Sicherheitsmodus“ $\lambda_{k+1} = \lambda_k \cdot \nu$
(eventuell iterativ)
- **Konvergenzrate:** Im **Idealfall** ähnlich
(exaktem) Newton-Verfahren
quadratisch, in **ungünstigen Fällen**
aber nur **linear** (abh. vom Fehler bei der
Näherung für \mathbf{H})





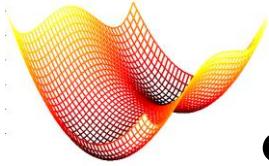
Levenberg-Marquardt-Algorithmus (5)

```
function optimizeLevenbergMarquardt (in objective function  $f(\mathbf{x}) = \sum_{i=1}^n r_i(\mathbf{x})^2$ , in  
initial solution  $\mathbf{x}^0$ , in parameters  $\lambda_0$ ,  $\nu$ , and  $\varepsilon$ , out final solution  $\mathbf{x}^*$ )  
  
// main iterative multidimensional search loop  
 $k \leftarrow 0$   
repeat  
    calculate the Jacobi matrix  $J_r(\mathbf{x}^k)$  at the current position  $\mathbf{x}^k$   
    update the solution twice: with  $\lambda$  yielding  $\mathbf{x}_{\lambda}^{k+1}$  and  $f(\mathbf{x})_{\lambda}^{k+1}$ , and with  $\lambda/\nu$   
    yielding  $\mathbf{x}_{\lambda/\nu}^{k+1}$  and  $f(\mathbf{x})_{\lambda/\nu}^{k+1}$   
    // update weighting of the components (lambda parameter)  
    if  $f(\mathbf{x})_{\lambda/\nu}^{k+1} < f(\mathbf{x})^k$  then  
         $\lambda \leftarrow \lambda/\nu$   
         $\mathbf{x}^{k+1} \leftarrow \mathbf{x}_{\lambda/\nu}^{k+1}$   
    else  
        if  $f(\mathbf{x})_{\lambda}^{k+1} < f(\mathbf{x})^k$  then  
             $\mathbf{x}^{k+1} \leftarrow \mathbf{x}_{\lambda}^{k+1}$  // lambda remains unchanged  
        else  
            // successive update of lambda  
            repeat  
                 $\lambda \leftarrow \lambda \cdot \nu$   
                until  $f(\mathbf{x})_{\lambda/\nu}^{k+1} < f(\mathbf{x})^k$   
            end if  
        end if  
         $k \leftarrow k+1$   
    until convergence:  $|f(\mathbf{x}^{k+1}) - f(\mathbf{x}^k)| \leq \varepsilon \cdot (|f(\mathbf{x}^{k+1})| + |f(\mathbf{x}^k)|)$   
     $\mathbf{x}^* \leftarrow \mathbf{x}^k$ 
```



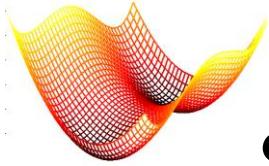
Stochastic Gradient Descent (1)

- **Gegeben:** Zielfunktion als Summe: $\phi(\mathbf{x}) = \sum_{i=1}^M f_i(\mathbf{x})$
- **Lösung** des Problems ist wieder $\mathbf{x}^* = \operatorname{argmin}(\phi(\mathbf{x}))$
- Anwenden von Methode des steilsten Abstiegs (Gradient Descent):
 - Suchrichtung in Iteration k : $\mathbf{s}_k = -\sum_{i=1}^M \nabla f_i(\mathbf{x}_k)$
 - Update: $\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \sum_{i=1}^M \nabla f_i(\mathbf{x}_k)$
- **Problem:** hoher Rechenaufwand bei großen M
- **Beispiel:** Training von Modellen mit vielen Trainingsdaten (für jedes Trainingsbeispiel gibt es ein separates $f_i(\mathbf{x})$, z.B. „deep learning“ bei Convolutional Neural Networks (CNN))
- De-facto-Standard beim Training von Neuronalen Netzen



Stochastic Gradient Descent (2)

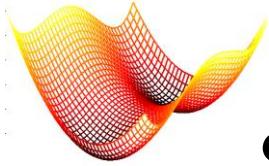
- **Idee:** verwende für jedes Update nur ein (zufällig ausgewähltes) $f_i(\mathbf{x})$
- Update: $\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \nabla f_i(\mathbf{x}_k)$; i zufällig
 - Jetzt: jede Iteration ist einfach zu berechnen, aber viele Iterationen notwendig
 - **Problematisch:** Wahl der Schrittweite λ_k
 - **Günstig:** Starte mit großem λ_k , reduziere λ_k sukzessive
 - Es lässt sich zeigen: bei geeigneter Wahl von λ_k (fast sichere) Konvergenz in ein (lokales) Minimum von $\phi(\mathbf{x})$



Stochastic Gradient Descent (3)

- Pseudocode:

```
function optStochasticGradientDescent (in objective  
f(x), in start x0, in  $\epsilon$ , in  $\lambda_0$ , out solution x*)  
repeat  
    randomly shuffle "the  $i$ 's" ( $f_i(\mathbf{x})$ )  
    for i=0 to M-1  
        1.) calc. search direction  $\mathbf{s}_k = -\nabla f_i(\mathbf{x}_k)$   
        2.) update:  $\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \nabla f_i(\mathbf{x}_k)$   
         $k \leftarrow k + 1$   
    next  
    adjust stepsize  $\lambda_k$   
until  $\|\nabla f(\mathbf{x}_k)\| < \epsilon$ 
```



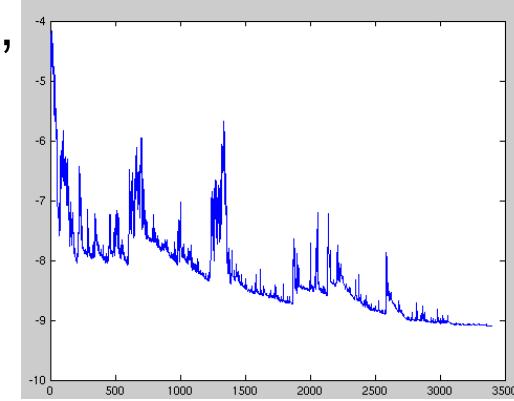
Stochastic Gradient Descent (4)

- Typischer Verlauf: Zielfunktion sinkt generell, aber auch immer wieder (starke) Anstiege möglich

→ Variationen:

- Berücksichtige mehrere $f_i(\mathbf{x})$ in jeder It.:
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \sum_{l=1}^L \nabla f_l(\mathbf{x}_k); L \ll M$$
- „Momentum Method“:
 - „Merke“ Update in jeder Iteration
 - Aktuelles Update ist Linearkombination aus Gradient und vorherigem Update:

$$\Delta \mathbf{x}_k = -\lambda_k \cdot \nabla f_i(\mathbf{x}_k) + \tau \cdot \Delta \mathbf{x}_{k-1}; \mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$$

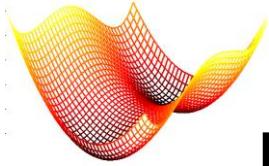


© Wikipedia / Joe Pharos



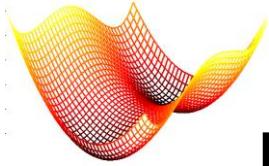
Numerische Aspekte

- **Konvergenzeigenschaften**
- **Kondition**
- **Skalierung**
 - **Skalierung mit MATLAB**
- **Numerische Differenzierung**
- **Numerische „Fallen“**



Konvergenzrate (1)

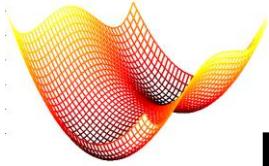
- Betrachte Fortschritt pro Iteration, d.h. Entwicklung des Fehlers $e_k = \mathbf{x}_k - \mathbf{x}^*$:
$$\lim_{k \rightarrow \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = \lim_{k \rightarrow \infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^r}$$
- $k \rightarrow \infty$: Verhalten **am Ende** des Verfahrens
- Falls $\lim_{k \rightarrow \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} := C$: Fehler verändert sich um **Faktor** C : Konvergenz mit Rate r
- $r = 1$: **lineare Konvergenz**
 - Nur Konvergenz bei $C < 1$ (sonst divergiert das Verfahren)
 - Sehr langsame Konvergenz wenn $C \rightarrow 1$



Konvergenzrate (2)

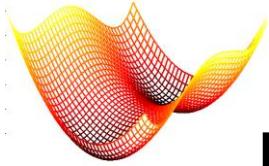
- $r = 1$ und $C = 0$: **superlineare** Konvergenz
- $r \geq 2$: **quadratische** Konvergenz: $\lim_{k \rightarrow \infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^2} = C$
- Je größer r , umso schnellere Konvergenz
- Jede Konvergenz mit Rate $r > 1$ ist auch superlinear:
$$\lim_{k \rightarrow \infty} \frac{\|e_{k+1}\|}{\|e_k\|} = \lim_{k \rightarrow \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} \|e_k\|^{r-1} = C \lim_{k \rightarrow \infty} \|e_k\|^{r-1} = 0$$

- **Konvergenzbereich**: Menge M aller Startpunkte \mathbf{x}_0 , für die das Verfahren gegen das gesuchte Optimum \mathbf{x}^* konvergiert.
 $M = S$ (gesamter Definitionsbereich) \rightarrow **global** konvergent
 \rightarrow Ziel: Konvergenzbereich M sollte **möglichst groß** sein



Kondition eines Problems (1)

- **Kondition:** Abhangigkeit (Empfindlichkeit) der Losung eines Problems von der „Storung“ der Eingangsdaten
 - Mathematisches Ma: **Konditionszahl** κ
 - Bei Matrizen: κ ist Quotient groter zu kleinster Eigenwert von $A \in \mathbb{R}^{N \times N}$: $\kappa := \mu_1 / \mu_N$ (falls existent)
 - Bei Funktionen:
$$\frac{\|f(\mathbf{x}) - f(\mathbf{x} + \boldsymbol{\varepsilon})\|}{\|f(\mathbf{x})\|} \leq \kappa_{rel} \cdot \frac{\|\boldsymbol{\varepsilon}\|}{\|\mathbf{x}\|} \rightarrow \kappa_{rel} := \frac{\|\nabla f(\mathbf{x})\| \cdot \|\mathbf{x}\|}{\|f(\mathbf{x})\|}$$
- $\kappa \geq 1$: gut konditioniertes Problem („**well-conditioned**“)
 - $\kappa \gg 1$: schlecht konditioniertes Problem („**ill-conditioned**“)
 - $\kappa \rightarrow \infty$: schlecht gestelltes Problem; Losung unmoglich zu berechnen („**ill-posed**“)

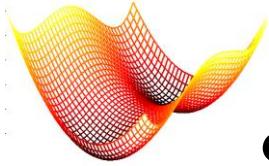


Kondition eines Problems (2)

- **Praxisrelevanz:** (kleinere) Störungen der Eingangsdaten sollen Lösung nicht (wesentlich) verfälschen
- **Numerik:** Eingangsdaten sind immer unexakt, allein schon wegen Quantisierung (auch bei Gleitkomma-Zahlen!)

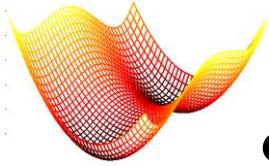
→ Es ist stets auf gute Konditionierung zu achten!

- Auch Lösungsalgorithmus hat Einfluss!
- **Maßnahmen zur Verbesserung der Kondition:**
Vorkonditionierung des Problems:
 - (lineare) Transformation bei quadratischen Programmen mit Ziel κ der Matrix A zu reduzieren
 - Skalierung der Eingangsvariablen



Skalierung (1)

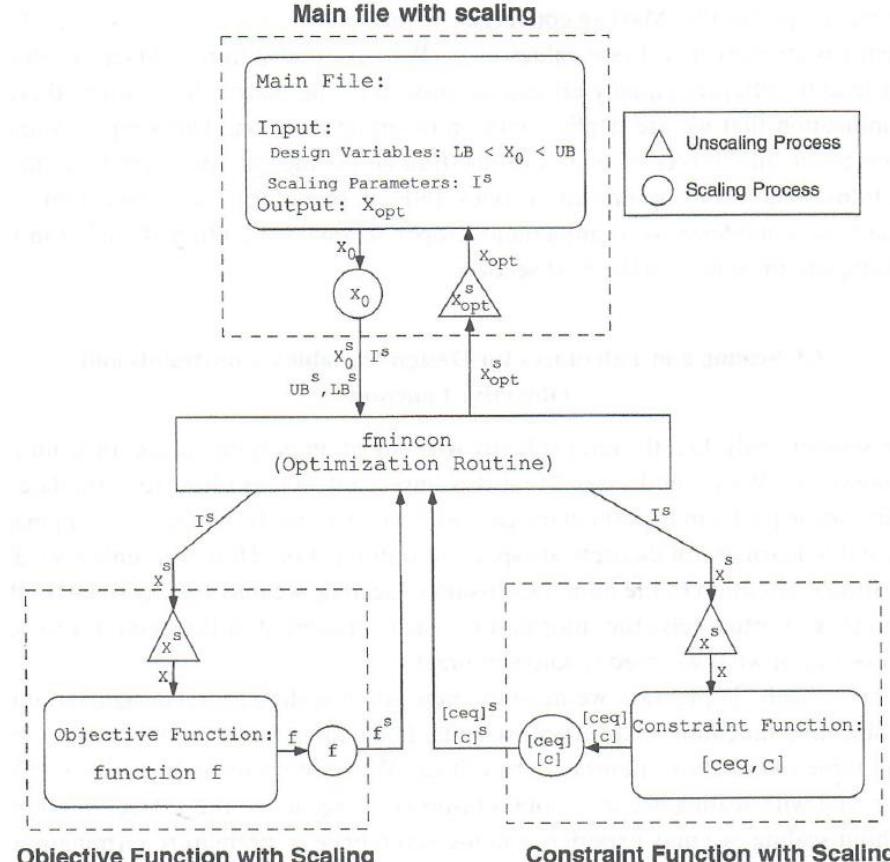
- Numerische Repräsentation reeller Zahlen im Computer nur mit **endlich vielen Dezimalstellen** möglich
- z.B. in MATLAB: Standard-Einstellung für Optimierung ist 10^{-6} , d.h. 6 Nachkomma-Stellen ($n_{def} = 7$)
 - Probleme, wenn Zielvariablen sehr kleine Werte annehmen
 - Lösung: **Skalierung**: Multiplikation mit Faktoren α_i :
- Jetzt ist Genauigkeit für jede Zielvariable einstellbar ($n_{desired}$ gültige Ziffern):
$$\alpha_i = 10^{n_{desired}-n_{def}} / x_{i,typical}$$
- Zu berücksichtigen sind **typische Werte** $x_{i,typical}$



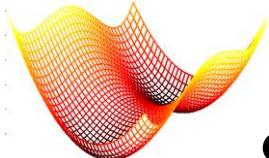
Skalierung (2)

- **Skalierung ist auch zu empfehlen, wenn die Zielvariablen stark unterschiedliche Wertebereiche aufweisen**
- **Daumen-Regel:** Angleichung der Wertebereiche
- Analog zur Skalierung der Zielvariablen:
 - Skalierung der Zielfunktion(en): $f(\mathbf{x})_i^s = \beta_i \cdot f(\mathbf{x})_i$
 - Skalierung der Nebenbedingungen: $g(\mathbf{x})_i^s = \gamma_i \cdot g(\mathbf{x})_i$

Skalierung (3): Umsetzung



© Messac / Cambridge University Press



Skalierung in MATLAB

- Ändern der Default-Werte für numerische Genauigkeit mittels Struktur **optimset**:

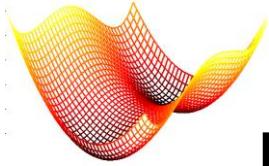
```
options = optimset('TolFun', 1e-10, 'TolX', 1e-10,  
'TolCon', 1e-10)
```

- Einbinden der Skalierungsfaktoren:

```
[xopt_s, fopt_s] = fmincon('objfun', x0_s, A, b,  
Aeq, beq, xl, xu, 'nlconstrfun', options, alphas,  
betas, gammas)
```

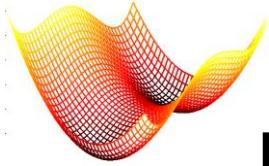
```
[neq_constr_s, eq_constr_s] = nlconstrfun(x_s,  
alphas, betas, gammas)
```

```
funcval_s = objfun(x_s, alphas, betas, gammas)
```



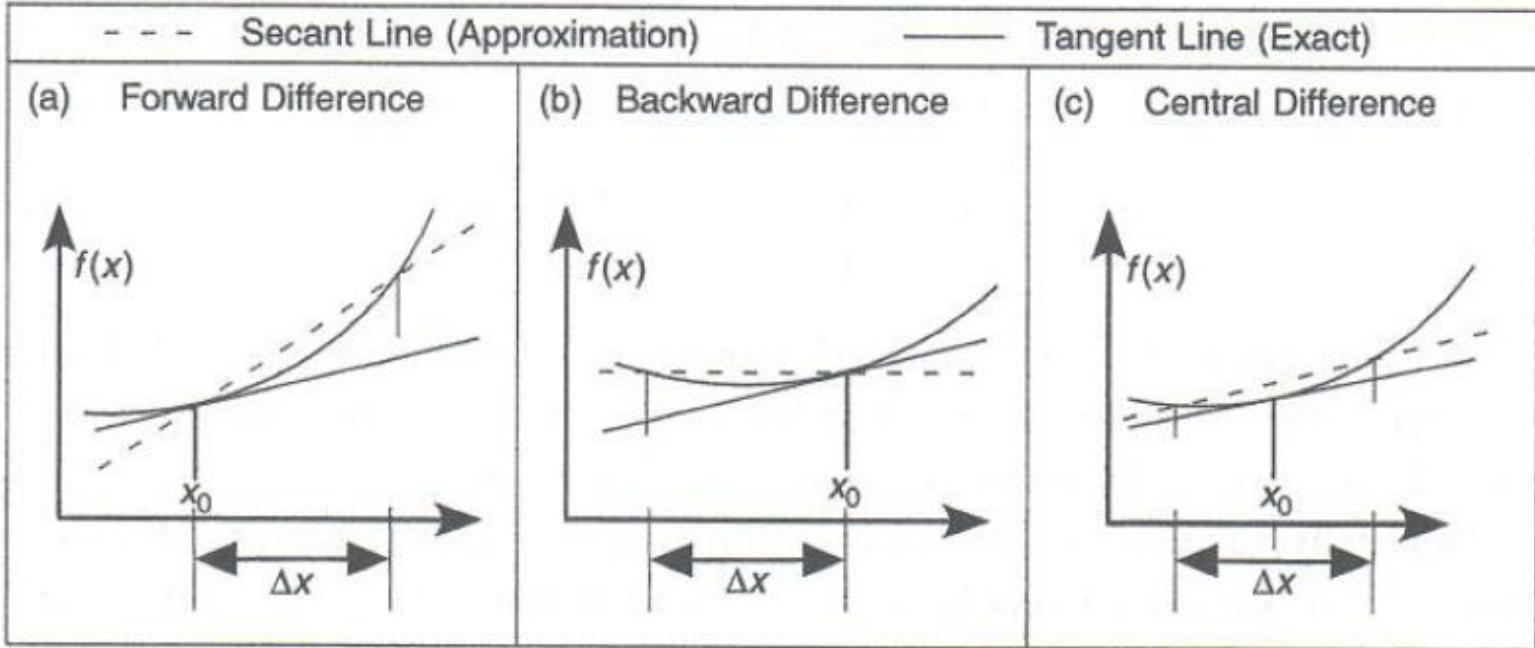
Numerische Differenzierung (1)

- Wie bestimmt man den Gradienten $\nabla f(\mathbf{x}_k)$ der Zielfunktion?
 - Variante 1: **analytisches Berechnen** in geschlossener Form: ist oft komplex oder gar nicht möglich
 - Variante 2: **numerische Approximation** durch Differenzbildung
- **Rückwärts-Differenz:**
$$\frac{\partial f(\mathbf{x}_k)}{\partial x_i} \approx \frac{\Delta f(\mathbf{x}_k)}{\Delta x_i} = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k - \Delta x_i)}{\Delta x_i}$$
- **Vorwärts-Differenz:**
$$\frac{\partial f(\mathbf{x}_k)}{\partial x_i} \approx \frac{\Delta f(\mathbf{x}_k)}{\Delta x_i} = \frac{f(\mathbf{x}_k + \Delta x_i) - f(\mathbf{x}_k)}{\Delta x_i}$$
- **Zentrale Differenz:**
$$\frac{\partial f(\mathbf{x}_k)}{\partial x_i} \approx \frac{\Delta f(\mathbf{x}_k)}{\Delta x_i} = \frac{f\left(\mathbf{x}_k + \frac{\Delta x_i}{2}\right) - f\left(\mathbf{x}_k - \frac{\Delta x_i}{2}\right)}{\Delta x_i}$$

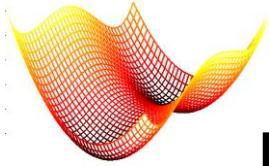


Numerische Differenzierung (2)

- Finite Differenzen sind **lineare Approximation**:



© Messac / Cambridge University Press

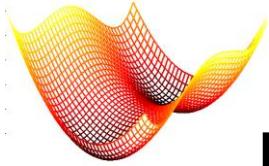


Numerische Differenzierung (3)

- Für jede Zielvariable **getrennte Berechnung**, d.h.

$$\nabla f(\mathbf{x}_k) \approx \left[\frac{\Delta f(\mathbf{x}_k)}{\Delta x_1}, \frac{\Delta f(\mathbf{x}_k)}{\Delta x_2}, \dots, \frac{\Delta f(\mathbf{x}_k)}{\Delta x_N} \right]^T$$

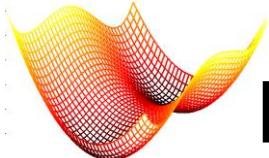
- **Genauigkeit** hängt ab von:
 - **Nichtlinearität der Funktion:** je größer Krümmung, desto ungenauer
 - **Größe der Δx_i :** je größer die Δx_i , desto stärker beeinflusst Nichtlinearität, je kleiner die Δx_i , desto stärker wirkt sich endliche numerische Genauigkeit aus!
 - **Skalierung der x_i :** stark variierende Wertebereiche sind ungünstig!



Numerische Differenzierung (4)

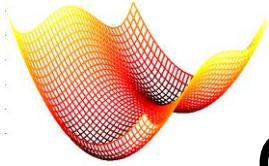
- Zentrale Differenz ist am genauesten, benötigt aber auch die meisten Funktionsberechnungen: $2N$ (vs. $N + 1$ bei Vorwärts-/Rückwärts-Differenz)
 - Viele Funktionsberechnungen für den Wert des Gradienten an einer einzigen Stelle
 - Optimierungsverfahren nullter Ordnung haben auch ihre Daseinsberechtigung!
- MATLAB:

```
options = optimset('FinDiffType','central')
```



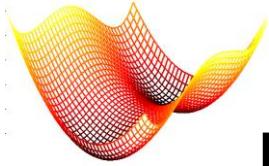
Numerische Differenzierung: Vor- bzw. Nachteile

Analytische Berechnung	Finite Differenzmethode
Schnell (Laufzeit)	zeitaufwändig
Exakt → schnelle Konvergenz	Inexakt → langsame Konvergenz
Gute numerische Stabilität	Gefahr: numerische Instabilität
Berechnung der Ableitungen schwierig oder nicht möglich	Einfach zu implementieren
Muss vom Anwender bereitgestellt werden	Integraler Bestandteil vieler Optimierungsbibliotheken



Optimierungsergebnisse: Tests

- Ergebnis der Optimierung immer **auf physikalische Plausibilität prüfen!**
 - Bei unplausiblen Ergebnissen: Modellierung überprüfen bzw. überdenken!
- **Empfindlichkeitsanalyse:** Mehrmaliges Durchführen der Optimierung mit leicht anderen Startwerten und/oder Optimierungsparametern (z.B. Koeffizienten von Termen)
 - Weichen die Ergebnisse (zu) stark voneinander ab, ist das ein Hinweis auf numerische Probleme
- „**Daumenregel**“: je höherdimensional ein Problem (große Anz. an Zielvar.), umso wahrscheinlicher sind numerische Probl. → Modellierung auf Vereinfachungen untersuchen!



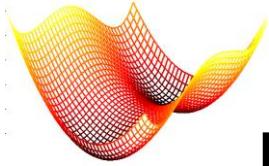
Numerische „Fallen“

- Schlechte Konditionierung des Problems
- „Abbruch“ der Optimierung (z.B. Max. Anzahl von Iterationen erreicht)
- Division durch sehr kleine Werte
- Wurzel von negativen Werten
- Singuläre Matrizen aufgrund von endlicher Genauigkeit der einzelnen Elemente
- Startpunkt der Optimierung ist zu nahe am gesuchten Optimum bzw. zu weit entfernt (außerhalb Konvergenzbereich)



Lineare Optimierung

- **Problemformulierung**
- **Rolle der Nebenbedingungen**
- **Standardform eines Linearen Programms**
- **Simplex-Methode**
 - **Kanonische Form**
 - **Simplex-Tableau**
- **Dualitätsbegriff**
- **Sensitivitätsanalyse**
- **Anwendung: Transportproblem**



Lineare Programme

- Lineares Programm (LP):

$$\min_{\mathbf{x} \in S} f_L(\mathbf{x}) = \min_{\mathbf{x} \in S} \mathbf{c}^T \cdot \mathbf{x}$$

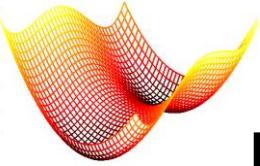
Nebenbedingungen:

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$$

$$\mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}$$

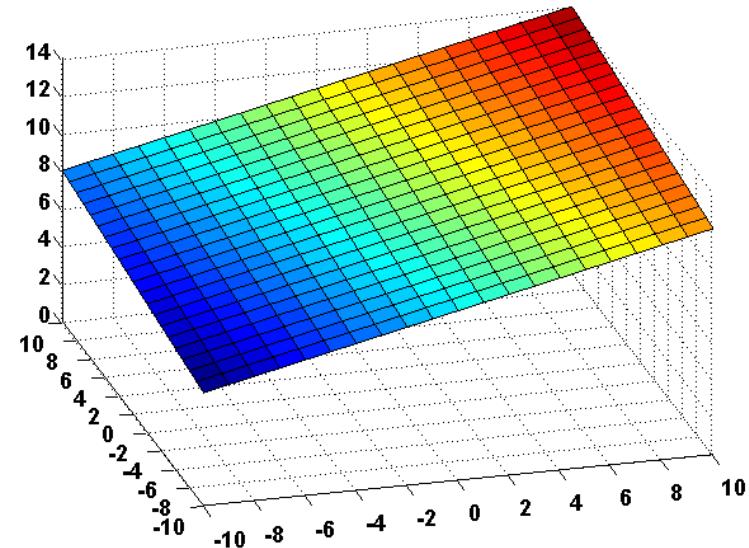
$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$

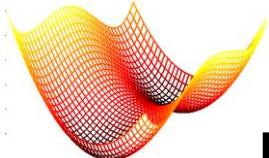
- **Besonderheit:** Zielfunktion $f_L(\mathbf{x}) = \mathbf{c}^T \cdot \mathbf{x}$ und alle Nebenbedingungen sind **linear** in \mathbf{x}
→ **Vereinfachung** des Problems



Lineare Programme: Rolle der NB (1)

- **Abstiegsrichtung von $f_L(\mathbf{x})$ ändert sich nicht:**
 $\nabla f_L(\mathbf{x}) = \text{const}$
- Ohne Restriktionen liegt Lösung des LP im Unendlichen
- **Restriktionen sind notwendig** zur Beschränkung des Lösungsraumes S
- **Lösung liegt „am Rand“ von S**
- **Vorteil:** Lösung muss auch nur am Rand gesucht werden



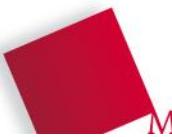
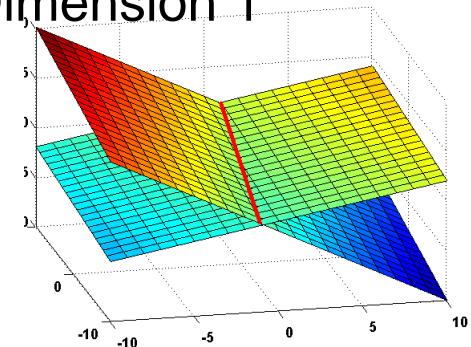
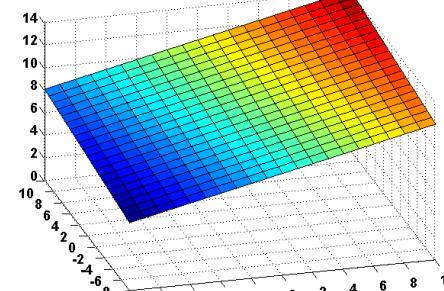
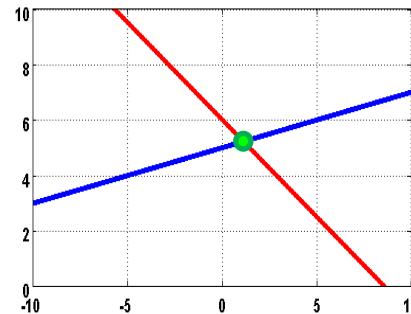
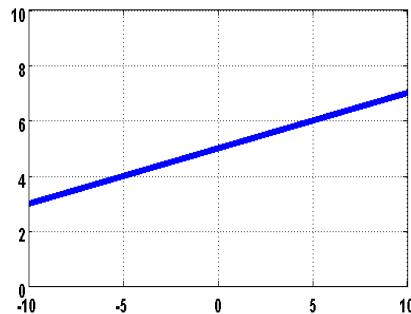


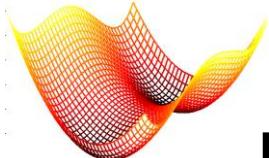
Lineare Programme: Rolle der NB (2)

- **Lineare Gleichungen als NB:**

$$a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_N \cdot x_N = b$$

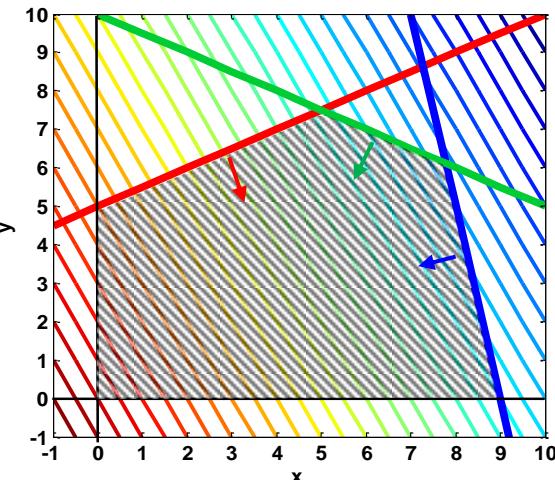
- \mathbb{R}^2 : $a_1x_1 + a_2x_2 = b \rightarrow x_2 = -a_1/a_2 \cdot x_1 + b/a_2 \rightarrow$ Gerade
- \mathbb{R}^3 : $a_1x_1 + a_2x_2 + a_3x_3 = b \rightarrow$ Ebene
- **Jede Gleichung reduziert Dimension von S um 1**
- \mathbb{R}^2 : Schnittpunkt zweier Geraden \rightarrow Punkt: Dimension 0
- \mathbb{R}^3 : Schnittpunkt zweier Ebenen \rightarrow Gerade: Dimension 1





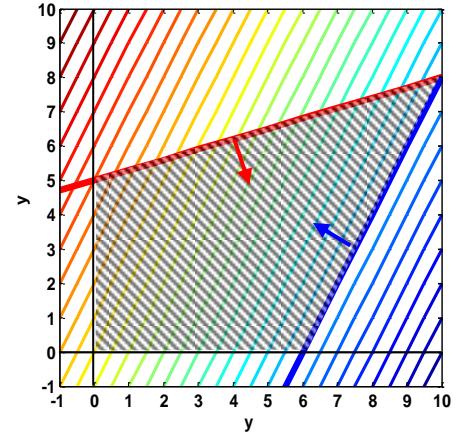
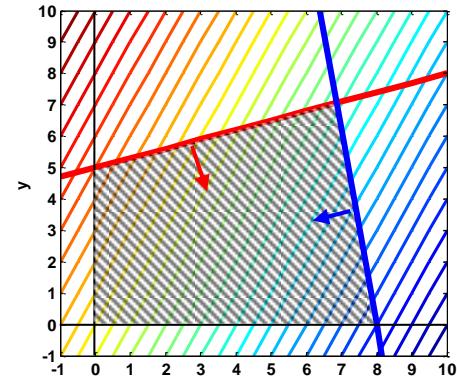
Lineare Programme: Rolle der NB (3)

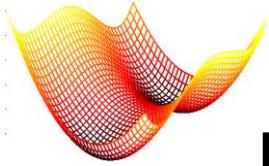
- **Lineare Ungleichungen als NB:**
$$a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_N \cdot x_N \leq b$$
- **Teilen Lösungsraum in zwei Halbräume**
- **Trennfläche hat Dimension $N - 1$ (s. Gl.)**
- An **Trennfläche** gilt das **Gleichheitszeichen** → Ungleichung ist **aktiv**
- Mehrere Ungleichungen beschränken Lösungsraum S typischerweise auf einer **Polytop** mit Dimension $\leq N$
- Lösung am „Rand“ → Lösung des LP wird durch eine „Ecke“ oder „Seitenfläche“ des Polytops definiert



Lineare Programme: Lösungsarten (1)

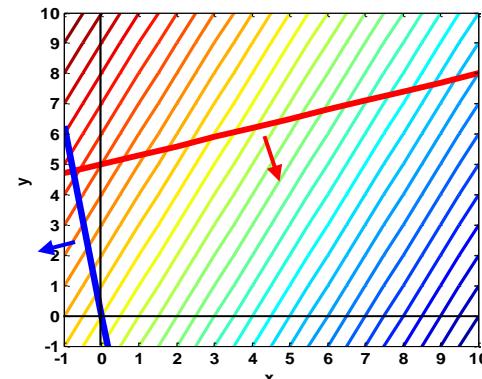
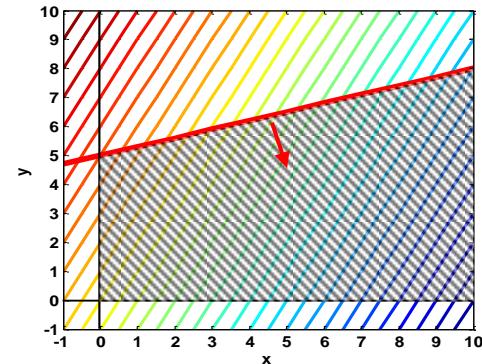
- **Eindeutig:** Lösung liegt in einer „Ecke“ des Polyeders („Normalfall“) (Lösungsraum S symbolisiert durch schraffierte Fläche, nur positive Werte für alle x_i erlaubt)
- **Nicht eindeutig:** komplette „Seitenfläche“ des Polyeders löst das LP, d.h. die Ableitung der Zielfunktion entlang dieser Seitenfläche ist gleich 0
- Hinweis: Es soll stets gelten: $x, y > 0$

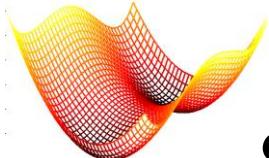




Lineare Programme: Lösungsarten (2)

- **Unbegrenzt:** Lösungsraum ist in Abstiegsrichtung der Zielfunktion unbegrenzt → Lösung „liegt im Unendlichen“
- **Keine Lösung:** Nebenbedingungen beschränken Lösungsraum auf leere Menge ($S = \{ \}$), .d.h. kein Punkt im \mathbb{R}^N kann alle NB gleichzeitig erfüllen





Simplex-Algorithmus: Grundidee (1)

- **Situation:** Lösungsraum wird durch die NB „beschnitten“ und hat die Form eines Polytops (\approx “Vieleck“; \approx “**Simplex**“)
- Lösung ist stets in den “Ecken” des Simplex zu suchen
- **Simplex-Methode: Suche in jeder Iteration eine benachbarte Ecke des Simplex, in der die Zielfunktion einen niedrigeren Wert hat**
- “Ecke” hat Dimension 0
 - bei N -dim. Probl. brauchen wir N Gleichungen als NB
 - M Gleichungs-NB bewirken immer Reduktion um M
 - Es gibt stets $K = N - M$ aktive Ungleichungen
- **In jeder Simplex-Iteration wird eine aktive Ungleichung durch eine andere ersetzt**

Simplex-Algorithmus: Grundidee (2)

- Veranschaulichung Funktionsprinzip Simplex-Methode:
- 1.lt.: schwarze Pos. → blau; 2.lt. Blau → türkis (Optimum)
- Beispiel:

$$f_L(\mathbf{x}) = -3x_1 - 1.5x_2 + 20$$

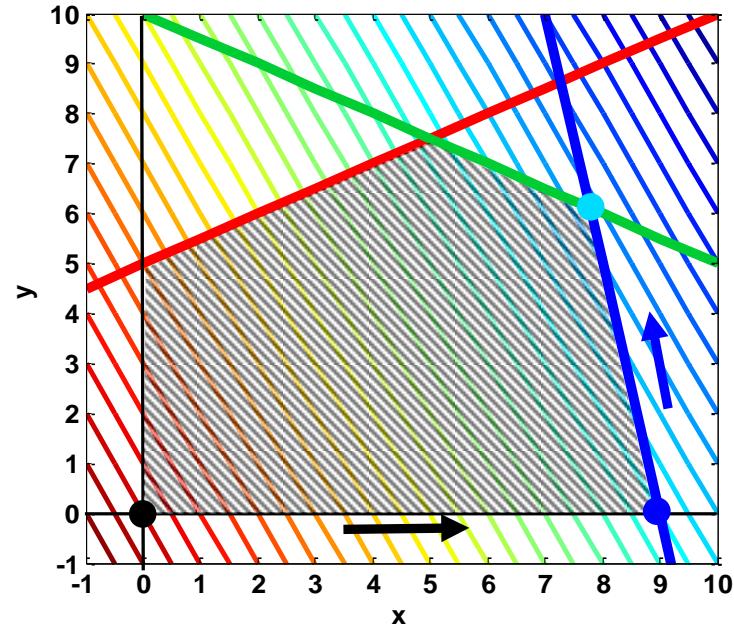
NB:

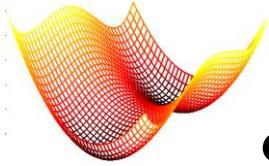
$$-0.5x_1 + x_2 \leq 5 \text{ (rot)}$$

$$5x_1 + x_2 \leq 45 \text{ (blau)}$$

$$0.5x_1 + x_2 \leq 10 \text{ (grün)}$$

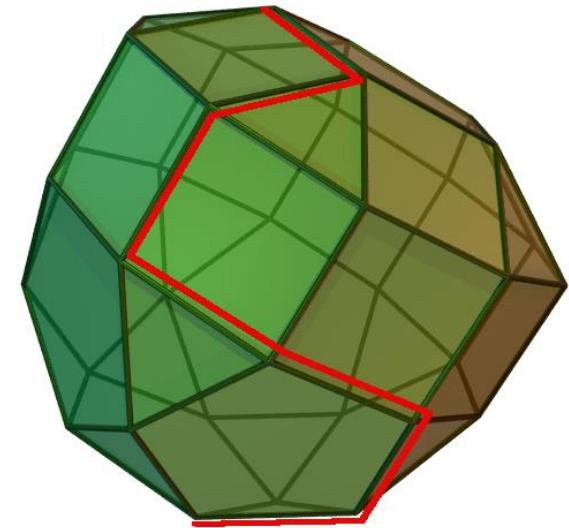
$$x_{1/2} \geq 0$$



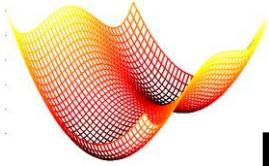


Simplex-Algorithmus: Grundidee (3)

- **Start:** beliebige Ecke des Polyeders (mit M Gleichungen und $K = N - M$ aktiven Ungleichungen als NB)
- Untersuche die „benachbarten“ Ecken (unterscheiden sich in einer aktiven Ungleichung)
- Gehe entlang „absteigender Kante“, d.h. $f_L(\mathbf{x}) = \mathbf{c}^T \cdot \mathbf{x}$ wird kleiner zur neuen aktuellen Position
- Wiederholung, bis es keine absteigende Kante mehr gibt



© Wikipedia / Sdo



Lineare Programme: Standardform (1)

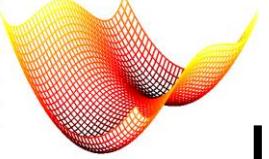
- Notwendig für automatisierte Abarbeitung der Simplex-Methode: (mathematischer) Formalismus
- Hierzu: **Standardform** des linearen Programms:

$$\min_{\mathbf{x} \in S} f_L(\mathbf{x}) = \min_{\mathbf{x} \in S} \mathbf{c}^T \cdot \mathbf{x}$$

Nebenbedingungen:

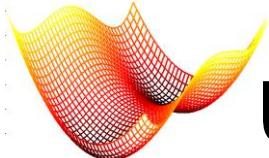
$$\begin{aligned}\mathbf{A} \cdot \mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}\end{aligned}$$

- Ausschließlich Gleichungen als “allg.” NB
- Weitere NB: alle Zielvariablen müssen ≥ 0 sein
→ Die einzigen Ungleichungs-NB sind $\mathbf{x} \geq \mathbf{0}$



Lineare Programme: Standardform (2)

- **Jedes LP ist in Standardform transformierbar:**
- Hierzu: **Einführung zusätzlicher Variablen**
- Unbeschränkte Variablen können durch Differenz zweier nicht-negativer Variablen ersetzt werden, z.B.:
 $x_u = x_{+,1} - x_{+,2}$
- Ungleichungen können durch Einführen von sog. **Schlupfvariablen** x_s in Gleichungen umgeformt werden
 - $a_1x_1 + a_2x_2 \leq b \rightarrow a_1x_1 + a_2x_2 + x_{s,1} = b$
 - $a_1x_1 + a_2x_2 \geq b \rightarrow a_1x_1 + a_2x_2 - x_{s,2} = b$
- Auflösung beliebig beschränkter Variablen durch Kombination dieser Methoden



Umformung eines LP in Standardform: Beispiel

- Nebenbedingungen des Beispiels:

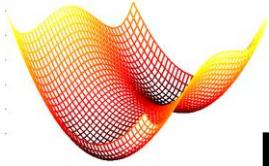
$$-0.5x_1 + x_2 \leq 5 \quad \rightarrow -0.5x_1 + x_2 + x_3 = 5$$

$$5x_1 + x_2 \leq 45 \quad \rightarrow 5x_1 + x_2 + x_4 = 45$$

$$0.5x_1 + x_2 \leq 10 \quad \rightarrow 0.5x_1 + x_2 + x_5 = 10$$

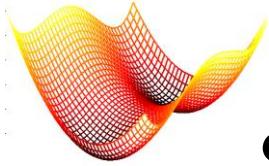
- NB als Gleichungssystem der Standardform:

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \end{bmatrix}$$



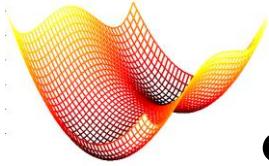
Lineare Programme: Standardform (3)

- Warum ist Standardform vorteilhaft?
- $\mathbf{A} : M \times N$ –Matrix (M Gleichungen, N Unbekannte)
- Wenn $M = N$: Lösung ist durch Gleichungssystem der NB eindeutig definiert.
- In der Regel aber gilt: $M < N \rightarrow$ Es müssen noch $K = N - M$ Ungleichungen aktiv sein
- Ugl. der Std-Form: $\mathbf{x} \geq \mathbf{0} \rightarrow K$ Zielvariablen sind gleich 0
 \rightarrow (maximal) M Zielvariablen ungleich 0, d.h. aktiv
- Alle **aktiven Zielvariablen** (d.h. ungleich 0) definieren eine sog. **Basislösung** des Problems
- Es existieren $C = \frac{N!}{M!(N-M)!}$ Basislösungen



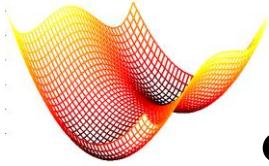
Simplex-Algorithmus (1)

- Aufgabe bei LP in Standardform:
 - Welche Zielvariablen sind inaktiv (d.h. gleich 0)?
 - Welchen Wert nehmen die aktiven Zielvariablen an?
- **Grundprinzip Simplex-Algorithmus:** Gehe von Basislösung zu Basislösung
- Bei gültiger Lösung darf sich Anzahl der aktiven Zielvariablen nicht verändern
 - In jedem Simplex-Schritt wird eine aktive Zielvariable x_{out} inaktiv (d.h. gleich 0 gesetzt), und dafür eine andere (bis dato inaktive) Zielvariable x_{in} aktiv, d.h. ungleich 0
- Noch zu klären: wie findet man x_{out} und x_{in} ?



Simplex-Algorithmus (2)

- Identifikation von x_{out} und x_{in} mittels Umwandlung des LP in **kanonische Form**
- **Kanonische Form des NB-Gleichungssystems:**
 - Alle aktiven Variablen x_j (mit Wert $\neq 0$) beeinflussen nur eine Zeile i des GS
 - zugehörige Spalte j hat genau ein Element $a_{ij} \neq 0$
 - Jede Zeile wird von exakt einer akt. Variable beeinflusst
 - Zugehöriges Matrix-Element a_{ij} soll den Wert 1 haben
 - Damit müssen alle $b_i > 0$ sein: auf der linken Seite sind ausschließlich nichtnegative Summanden!

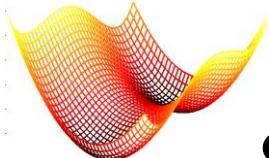


Simplex-Algorithmus (3)

- Wenn ursprüngliches LP ausschließlich Ungleichungen mit „≤“ als NB hat ist Standardform auch automatisch in kanonischer Form
- Sonst: **Umformung** der Standardform in kanonische Form nötig
- **Vorteil: Basislösung unmittelbar ablesbar**, z.B.:

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \end{bmatrix}$$

liefert
 $x_1 = x_2 = 0;$
 $x_3 = 5; x_4 = 45; x_5 = 10$



Simplex-Algorithmus (4)

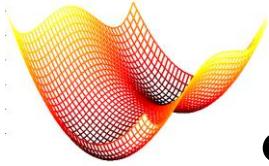
- „Arbeitsmittel“ **Simplex-Tableau**: Um die Zielfunktion erweitertes „Nebenbedingungs-Gleichungssystem“
- Dient dem effizienten Durchführen der Iterationen

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2N} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} & 0 & 0 & \cdots & 1 \\ c_1 & c_2 & \cdots & c_N & 0 & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ x_{N+1} \\ x_{N+2} \\ \vdots \\ x_{N+M} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \\ f(\mathbf{x}) - f(\mathbf{x}^k) \end{bmatrix}$$

Allg. Formulierung

Beispiel

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \\ -3 & -1.5 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \\ f(\mathbf{x}) - 20 \end{bmatrix}$$



Simplex-Algorithmus (5)

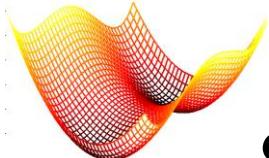
Arbeitsschritte in jeder Iteration des Simplex-Algorithmus:

1. Identifizieren von x_{in} :

- Betrachte letzte Zeile (Zielfkt.) des Simplex-Tableaus
- x_{in} wird aktiv, d.h. sein Wert wird größer → für eine Reduktion der Zielfunktion muss zugehöriges c_{in} negativ sein
- **Deshalb: x_{in} wird durch den kleinsten negativen Koeffizienten in der letzten Zeile bestimmt**
- Sind **alle Koeffizienten ≥ 0** ist keine Verbesserung mehr möglich → **Konvergenz erreicht**

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \\ -3 & -1.5 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \\ f(\mathbf{x}) - 20 \end{bmatrix}$$

The matrix has 4 rows and 5 columns. The first row has entries -0.5, 1, 1, 0, 0. The second row has entries 5, 1, 0, 1, 0. The third row has entries 0.5, 1, 0, 0, 1. The fourth row has entries -3, -1.5, 0, 0, 0. The vector on the right has entries 5, 45, 10, and $f(\mathbf{x}) - 20$. The variable x_1 is circled in green in the first column of the matrix, and the constant term 5 is circled in green in the vector on the right.



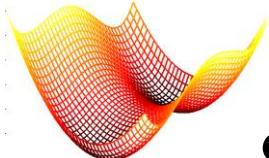
Simplex-Algorithmus (6)

2. Identifizieren von x_{out} :

- Hierzu: Betrachte Spalte \mathbf{a}_{in}
- Reduktion von x_{out} (war >0 , wird $=0$) bewirkt immer Reduktion eines Summanden in der linken Seite des GS → muss durch $a_{j,in} \cdot x_{in}$ ausgeglichen werden → es kommen nur die positiven Elemente von \mathbf{a}_{in} in Frage
- Bestimme unter allen pos. $a_{j,in}$ dasjenige j^* , wo $a_{j,in}/b_j$ maximal (sonst würden sich im nächsten Schritt (Wiederherstellen kanon. Form) negative b_j ergeben)
- $a_{j^*,in}$: **Pivotelement**
- x_{out} ist die aktive Variable der Zeile j^* (gibt genau eine)
- Alle Elemente von \mathbf{a}_{in} negativ → LP nicht lösbar!

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \\ -3 & -1.5 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \\ f(\mathbf{x}) - 20 \end{bmatrix}$$

The matrix has a blue circle around the first column.

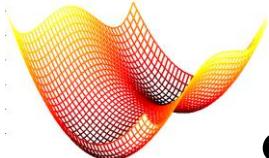


Simplex-Algorithmus (7)

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \\ -3 & -1.5 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \\ f(\mathbf{x}) - 20 \end{bmatrix}$$

3. Wiederherstellen der kanonischen Form:

- Notwendig, damit in nächster Iteration gleicher Mechanismus angewandt werden kann
- Hierzu: **Benutze Pivotelement** $a_{j^*,in}^{*}$
- Dividiere Zeile j^* durch $a_{j^*,in}^{*} \rightarrow a_{j^*,in}^{k+1}$ wird 1
- Alle anderen $a_{(.),in}^{k+1}$ sollen 0 werden (aktives x_{in} darf nur eine Gleichung beeinflussen)
- Hierzu für alle $j \neq j^*$: multipliziere Zeile j^* (bereits geändert) mit $a_{j,in}$ und subtrahiere dieses Ergebnis von der Zeile j
- Beachte: Schritt 1-3 ist **programmierbarer Formalismus!**



Simplex-Algorithmus: Beispiel

$$\begin{bmatrix} -0.5 & 1 & 1 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 \\ 0.5 & 1 & 0 & 0 & 1 \\ -3 & -1.5 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 45 \\ 10 \\ f(\mathbf{x}) - 20 \end{bmatrix}$$

- x_{in}
- Pivotelement
- x_{out}
- Lösung

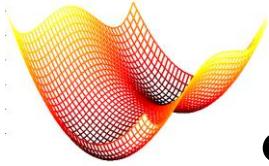
$$\begin{bmatrix} 0 & 1.1 & 1 & 0.1 & 0 \\ 1 & 0.2 & 0 & 0.2 & 0 \\ 0 & 0.9 & 0 & -0.1 & 1 \\ 0 & -0.9 & 0 & 0.6 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 9.5 \\ 9 \\ 5.5 \\ f(\mathbf{x}) + 7 \end{bmatrix}$$

Lösung:

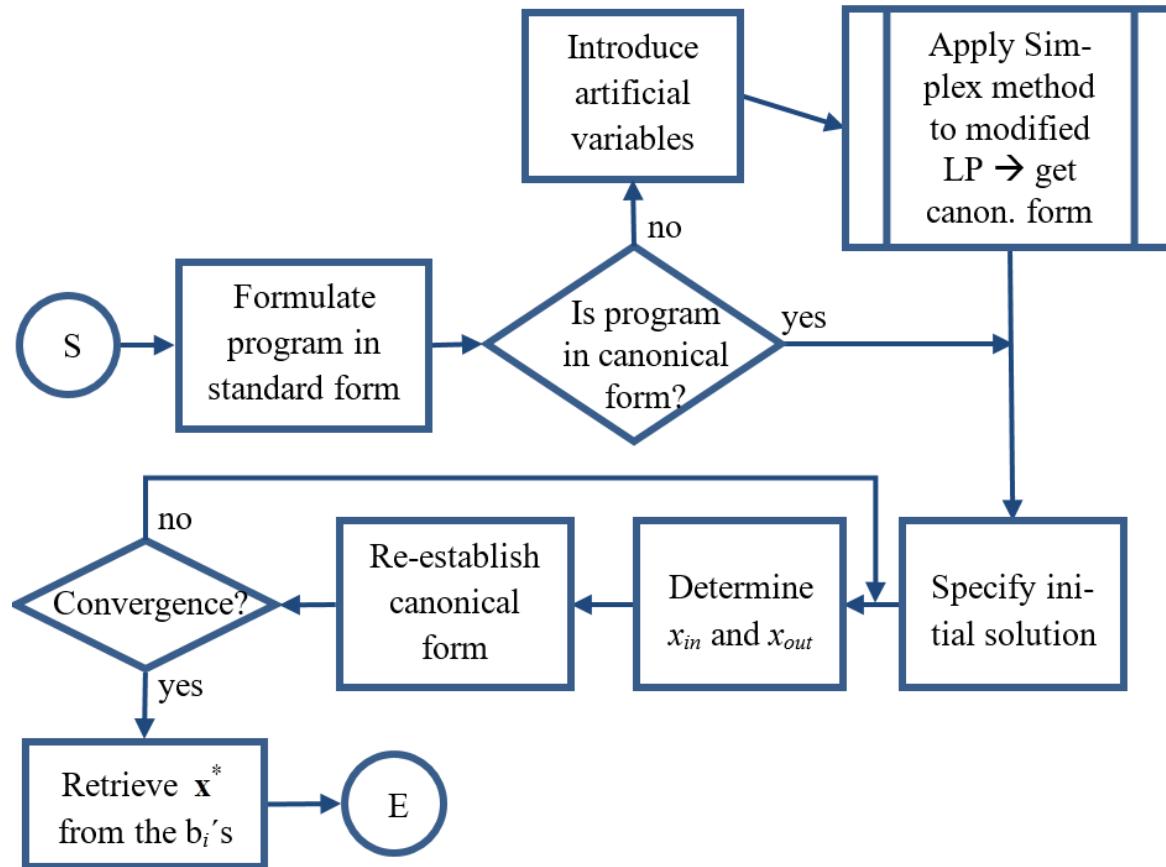
$$\begin{bmatrix} 0 & 0 & 1 & \frac{2}{9} & -\frac{11}{9} \\ 1 & 0 & 0 & \frac{2}{9} & -\frac{2}{9} \\ 0 & 1 & 0 & -\frac{1}{9} & \frac{10}{9} \\ 0 & 0 & 0 & 0.5 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{25}{9} \\ \frac{70}{9} \\ \frac{55}{9} \\ f(\mathbf{x}) + 12.5 \end{bmatrix}$$

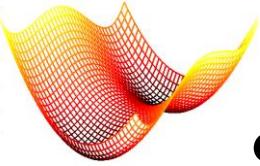
$$x_1 = 70/9 ; x_2 = 55/9 \\ [x_3 = 25/9 ; x_4 = x_5 = 0] \\ f(\mathbf{x}^*) = -12.5$$





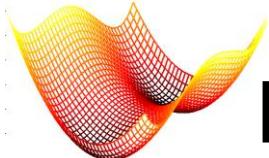
Simplex-Algorithmus: Flussdiagramm





Simplex-Algorithmus: Pseudocode

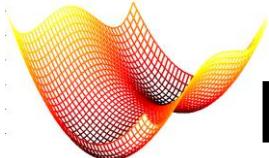
```
function solveLinearProgramSimplex (in objective function  $f(\mathbf{x})$ , in constraints specified by  $\mathbf{A}$  and  $\mathbf{b}$ , out solution  $\mathbf{x}^*$ )  
    // conversion of linear program to standard form (done „by hand“)  
    1. introduce a slack variable for each inequality constraint in order to convert it into an equality constraint  
    2. split each non-restricted design variable into the difference of two design variables restricted to non-negative values  
    3. create the simplex tableau  
  
    // conversion of simplex tableau into canonical form, if necessary (phase I)  
    if linear program is not canonical then  
        multiply each row  $j$  where  $b_j < 0$  by -1  
        call simplex phase II with modified program or rearrange rows of simplex tableau by linear combinations of other rows in order to  
        introduce zero's  
    end if  
  
    // solve the linear program (phase II of the simplex method)  
    specify initial basic feasible solution (contains only slack variables)  
    // iterative search loop (moves from vertex to vertex)  
    repeat  
        // replace design variable  $x_{out}$  by  $x_{in}$   
        find  $c_{in}$  which has the most negative value (yields  $x_{in}$ )  
        find the largest ratio  $a_{j,in}/b_j$  for all  $j \in [1..M]$  among all  $a_{j,in} > 0$  (yields row  $j^*$  and Pivot Element  $a_{j*,in}$ )  
         $x_{out}$  is the active variable of row  $j^*$  (abort if all  $a_{j,in} \leq 0$ )  
        // pivot on  $a_{j*,in}$ : re-establish canonical form  
        divide row  $j^*$  by  $a_{j*,in}$  by -->  $a_{j*,in}$  becomes 1 after this operation  
        for all other rows  $j$   
            multiply the updated row  $j^*$  by  $a_{j,in}$   
            subtract the result from row  $j$  -->  $a_{j,in}$  becomes 0 after this operation  
        next  
    until convergence (all  $c_i$ 's are  $\geq 0$ )  
    retrieve solution  $\mathbf{x}^*$  from the  $b_i$ 's and  $f(\mathbf{x}^*)$  from the bottom line of the current simplex tableau
```



Beispiel: optimaler Futtermittel-Einsatz (1)

- Bauer verfüttet Klee und Kraftfutter (mit bestimmten Kosten und Nährwerten) an Kuh (mit bestimmten Nährwert-Bedarf)
- Gesucht: kostengünstigster Futtermittel-Mix, der den Nährwert-Bedarf deckt.

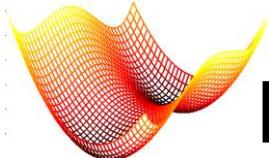
Eigenschaft	Kohlenh. / E	Proteine / E	Vitamine / E	Kosten / E
1 E Kraftfutter (x_1)	20	15	5	10 €
1 E Klee (x_2)	20	3	10	7 €
Bedarf	600	150	200	



Beispiel: optimaler Futtermittel-Einsatz (2)

- Mathematische Formulierung:
 - Minimiere Zielfunktion $f_L(x_1, x_2) = 10x_1 + 7x_2$
 - NB: $20x_1 + 20x_2 \geq 600$
 $15x_1 + 3x_2 \geq 150$
 $5x_1 + 10x_2 \geq 200; x_1, x_2 \geq 0$
- Simplex-Tableau der Standardform:

$$\begin{bmatrix} 20 & 20 & -1 & 0 & 0 \\ 15 & 3 & 0 & -1 & 0 \\ 5 & 10 & 0 & 0 & -1 \\ 10 & 7 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 600 \\ 150 \\ 200 \\ f_L(\mathbf{x}) \end{bmatrix}$$

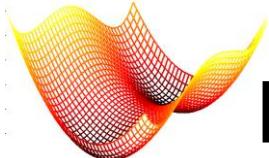


Beispiel: optimaler Futtermittel-Einsatz (3)

- Umformung in kanonische Form ergibt:

$$\begin{bmatrix} 0 & 20 & 1 & 0 & -4 \\ 0 & 27 & 0 & 1 & -3 \\ 1 & 2 & 0 & 0 & -1/5 \\ 0 & -13 & 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 200 \\ 450 \\ 40 \\ f_L(\mathbf{x}) - 400 \end{bmatrix}$$

- Momentane Lösung (aktive Variablen): $x_1 = 40$; $x_3 = 200$; $x_4 = 450$; $f_L(\mathbf{x}) = 400$
- x_{in} : x_2 ; x_{out} : x_3 ; Pivotelement: $a_{1,2}$

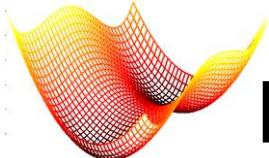


Beispiel: optimaler Futtermittel-Einsatz (4)

- Simplex-Tableau bei Lösung:

$$\begin{bmatrix} 0 & 1 & 13/80 & 1/12 & 0 \\ 0 & 0 & 9/16 & 5/12 & 1 \\ 1 & 0 & -17/80 & -1/12 & 0 \\ 0 & 0 & 79/80 & 1/4 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 25 \\ 75 \\ 5 \\ f_L(\mathbf{x}) - 225 \end{bmatrix}$$

- Optimale Lösung (aktive Variablen): $x_1 = 5$; $x_2 = 25$; $x_5 = 75$; $f_L(\mathbf{x}) = 225$ €

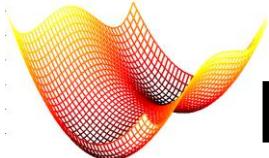


Lösen Linearer Programme mit MATLAB

- Lösen von LP in MATLAB mittels Kommando **linprog**:

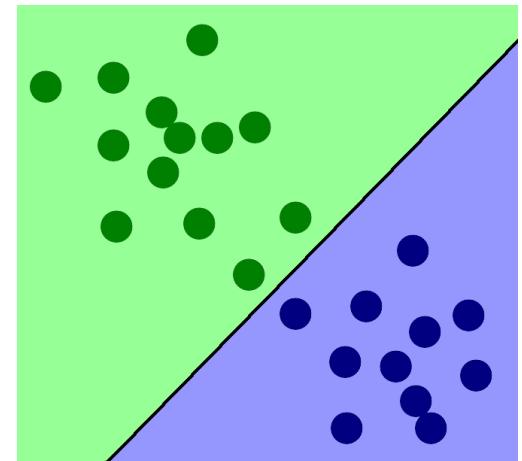
```
xopt = linprog(c, A, b, Aeq, beq, xl, xu, x0)
```

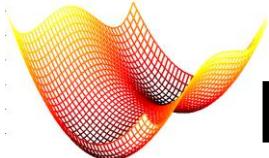
- x_{opt} : Returnwert: gesuchte Optimalposition \mathbf{x}^*
- c : (Vektor der) Koeffizienten der Zielfunktion
- A, b : Koeffizienten der linearen Ungleichungen als NB
- $A_{\text{eq}}, b_{\text{eq}}$: Koeffizienten der linearen Gleichungen als NB
- x_l, x_u : untere (\mathbf{x}_l) bzw. obere Begrenzung (\mathbf{x}_u) des Lösungsraumes
- x_0 : Startposition der Optimierung



Beispiel Lineares Programm: Linearer Binärer Klassifikator (1)

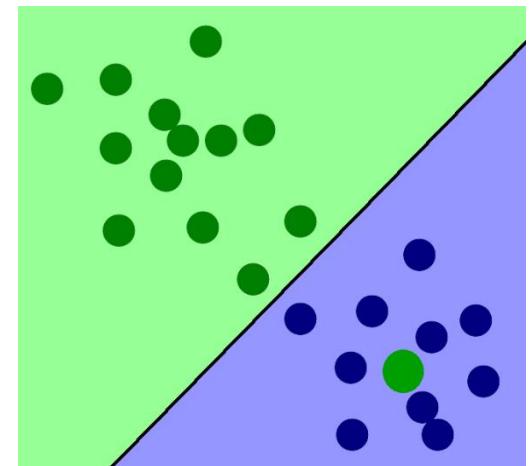
- **Aufgabe:** Zwei Mengen von Datenpunkten \mathbf{x}_i im \mathbb{R}^N sollen **möglichst ideal separiert** werden. (Zwei Klassen $y_i \in \{-1,1\}$ → **Binärer Klassifikator**)
- Separierung anhand einer **Trennfläche** im \mathbb{R}^N mit Dimension $N - 1$
- Trennfläche kann als Höhenlinie einer Funktion aufgefasst werden
- **Linearer Klassifikator:** Funktion ist linear, z.B. $\mathbf{a}^T \cdot \mathbf{x} + \mathbf{b}$
- **Klassifikationsregel:** Schätzung $\hat{y} = \text{sign}(\mathbf{a}^T \cdot \mathbf{x} + \mathbf{b})$
- **Optimierung:** Finde „optimale“ \mathbf{a}, \mathbf{b} anhand von Trainingsdaten $\{y_i, \mathbf{x}_i\}$

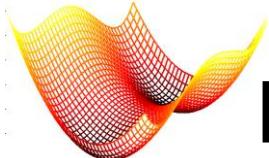




Beispiel Lineares Programm: Linearer Binärer Klassifikator (2)

- **Separierbare** Daten: es gilt stets $y_i \cdot (\mathbf{a}^T \cdot \mathbf{x}_i + \mathbf{b}) > 0$
- Geschickte Skalierung von \mathbf{a}, \mathbf{b} erlaubt stets Umformung zu $y_i \cdot (\mathbf{a}^T \cdot \mathbf{x}_i + \mathbf{b}) > 1$
- **Nicht-Separierbare** Daten: keine „fehlerfreie“ lineare Trennung möglich
- Dann gilt: $y_i \cdot (\mathbf{a}^T \cdot \mathbf{x}_i + \mathbf{b}) \geq 1 - \nu_i$ mit $\nu_i \geq 0 \rightarrow$ je größer ν_i , umso schlechtere Separierung
- **Beobachtung:** j -tes Element a_j von \mathbf{a} ist Gewichtungsfaktor des j -ten Merkmals x_j von \mathbf{x}



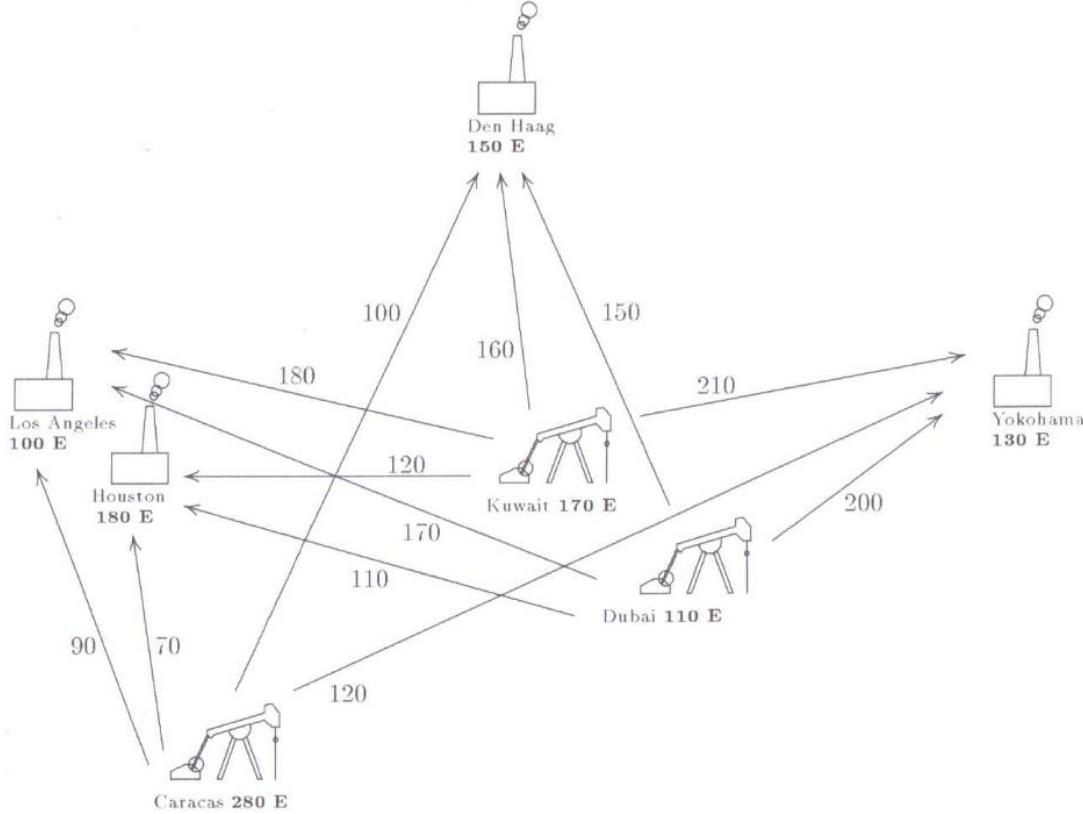


Beispiel Lineares Programm: Linearer Binärer Klassifikator (3)

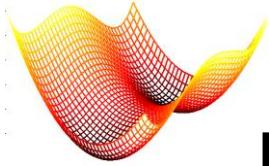
Mögliche **Optimierungskriterien** beim Training eines linearen binären Klassifikators:

- **Bestmögliche Trennung** bei nicht-separierbaren Trainingsdaten:
 - Ermittle $\min_{\mathbf{a}, \mathbf{b}, \mathbf{v}} \sum_{i=1}^M v_i$
 - NB: $v_i > 0; y_i \cdot (\mathbf{a}^T \cdot \mathbf{x}_i + \mathbf{b}) \geq 1 - v_i$
- **Möglichst „dünn besetzter“ Klassifikator**, d.h. möglichst viele $a_j \cong 0$. \rightarrow mögl. wenige Merkm. x_j relevant (Beschl.):
 - Ermittle $\min_{\mathbf{a}, \mathbf{b}} \sum_{j=1}^N a_j$ (sog. L1-Norm von \mathbf{a})
 - NB: $y_i \cdot (\mathbf{a}^T \cdot \mathbf{x}_i + \mathbf{b}) \geq 1$

Beispiel: Transportproblem (1)

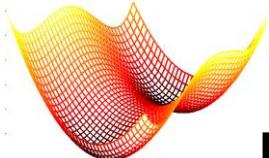


© Messac / Cambridge University Press



Beispiel: Transportproblem (2)

- Ursprüngliche Motivation von Dantzig zur Entwicklung der Simplex-Methode
- **Gegeben:**
 - An den Orten $i; 1 \leq i \leq N$ wird ein Gut produziert (in Menge s_i)
 - An den Orten $j; 1 \leq j \leq M$ wird ein Gut verbraucht (in Menge d_j)
 - Transportkosten c_{ij} pro Einh. von i nach j sind bekannt
- **Gesucht:** Transportmengen x_{ij} so, dass globale Kosten minimiert werden



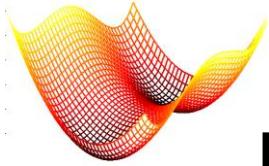
Beispiel: Transportproblem (3)

- **Mathematische Formulierung:** minimiere Zielfunktion

$$\sum_{i=1}^N \sum_{j=1}^M c_{ij} \cdot x_{ij}$$

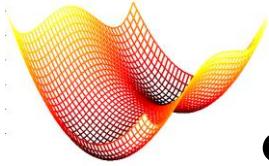
- **Nebenbedingungen:**

- Es kann in jedem Produktionsort nicht mehr abtransportiert werden als produziert wird: $\sum_{j=1}^M x_{ij} \leq s_i$
- Es muss zu jedem Verbrauchsortort mindestens so viel transportiert werden wie verbraucht wird: $\sum_{i=1}^N x_{ij} \geq d_j$
- Transportmengen sind positiv: $x_{ij} \geq 0$
- Keine Lösung, wenn $\sum_{i=1}^N s_i < \sum_{j=1}^M d_j$



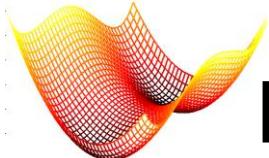
Beispiel: Transportproblem (4)

- In der Regel wird mehr produziert als verbraucht, d.h.
 $\sum_{i=1}^N s_i > \sum_{j=1}^M d_j$
- Überschuss kann durch Einführen eines zusätzlichen Verbrauchers $M + 1$ („Müllhalde“) mit $d_{M+1} = \sum_{i=1}^N s_i - \sum_{j=1}^M d_j$ aufgefangen werden ($c_{iM+1} = 0$)
- Damit werden alle NB zu Gleichungen (außer $x_{ij} \geq 0$) → Problem liegt in Standard-Form vor und kann mit Simplex-Methode gelöst werden
- **Startlösung: Nord-West-Eckenregel:** transportiere Produktion s_1 nach d_1 , dann nach d_2 , ... bis sie aufgebraucht ist, dann Abtransport von s_2 , etc.



Sensitivitätsanalyse (1)

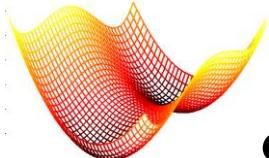
- Wie wirkt sich das **Einführen einer neuen Variable** x_{new} aus (z.B. Lohnt sich ein neu angebotenes Kraftfutter)?
 - Einführen von x_{new} resultiert in **neuer Spalte** a_{new}
 - Betrachte **Teilmatrix** A_{J^*} (in Standardform), welche aus allen zu den aktiven Variablen der bisherigen Optimallösung (d.h. $x_j^* > 0$) gehörenden Spalten besteht und erweitere dies um eine Spalte a_+ für ZF
 - J^* ist Basislösung $\rightarrow A_{J^*}$ ist invertierbar
 - Auswirkung auf die Gesamtkosten ist durch das unterste Element des Produktes $A_{J^*}^{-1} \cdot a_{new}$ abschätzbar: wenn < 0 , dann sinkt die Kostenfunktion, wenn x_{new} aktiv (d.h. > 0) wird!



Beispiel: optimaler Futtermittel-Einsatz (Wdh.)

- Mathematische Formulierung:
 - Minimiere Zielfunktion $f_L(x_1, x_2) = 10x_1 + 7x_2$
 - NB: $20x_1 + 20x_2 \geq 600$
 $15x_1 + 3x_2 \geq 150$
 $5x_1 + 10x_2 \geq 200; x_1, x_2 \geq 0$
- Simplex-Tableau der Standardform:

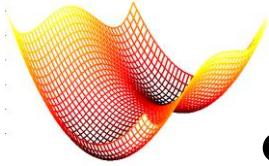
$$\begin{bmatrix} 20 & 20 & -1 & 0 & 0 \\ 15 & 3 & 0 & -1 & 0 \\ 5 & 10 & 0 & 0 & -1 \\ 10 & 7 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 600 \\ 150 \\ 200 \\ f_L(\mathbf{x}) \end{bmatrix}$$



Sensitivitätsanalyse (2)

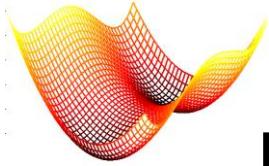
- **Beispiel:** lohnt sich neues, biologisches Kraftfutter (30 KH/E, 10 Pr./E, 10 Vit/E) mit Kosten 12 €/E
→ $\mathbf{a}_{new} = [30 \quad 10 \quad 10 \quad 12]^T$
- Zusätzliche Spalte in \mathbf{A}_{J^*} ist $\mathbf{a}_+ = [0 \quad 0 \quad 0 \quad 1]^T$

- $\mathbf{A}_{J^*} = \begin{bmatrix} 20 & 20 & 0 & 0 \\ 15 & 3 & 0 & 0 \\ 5 & 10 & -1 & 0 \\ 10 & 7 & 0 & 1 \end{bmatrix} \rightarrow \mathbf{A}_{J^*}^{-1} = \frac{1}{48} \cdot \begin{bmatrix} -0.6 & 4 & 0 & 0 \\ 3 & -4 & 0 & 0 \\ 27 & -20 & -48 & 0 \\ -15 & -12 & 0 & 48 \end{bmatrix}$
- $\mathbf{A}_{J^*}^{-1} \cdot \mathbf{a}_{new} = [* \quad * \quad * \quad 1/8]^T$: letzter Wert ist >0 , daher lohnt sich der Einsatz dieses Futters nicht!



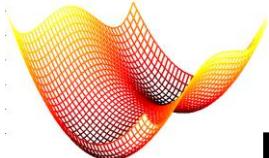
Sensitivitätsanalyse (3)

- Wie teuer darf das neue Futter maximal sein, damit sich der Einsatz lohnt?
 - Hierzu: ersetze Kosten durch Variable p in \mathbf{a}_{new} : $\mathbf{a}_{new,p}$
 - Unterstes Element in $\mathbf{A}_{J^*}^{-1} \cdot \mathbf{a}_{new,p}$ hängt jetzt von p ab
 - Finde p , mit dem dieser Term negativ wird („Grenzkosten“)
 - Im Beispiel:
 - Letzter Wert setzt sich zusammen aus:
 - $-\frac{15 \cdot 30}{48} - \frac{12 \cdot 10}{48} + p = 0 \rightarrow p = 11\frac{7}{8}$
- Das neue Futtermittel lohnt sich, wenn es billiger als 11,88€ ist



Duale Programme (1)

- **Ziel:** ermittle für ein LP in Standardform eine **untere Schranke**, d.h. eine Funktion $f_L(\mathbf{y}) = f_L(\mathbf{d}(\mathbf{y}) \cdot \mathbf{x})$, die stets $\leq f_L(\mathbf{x})$ ist.
- Bei Standardform gilt stets: $\mathbf{x} \geq \mathbf{0}$
 - Jedes Elem. i des neuen Param.-vektors \mathbf{d} muss $\leq c_i$ sein
- Betrachte hierzu $\mathbf{y}^T \cdot \mathbf{A}$: Wenn man in $f_L(\mathbf{x})$ den Parametervektor \mathbf{c} durch $\mathbf{y}^T \cdot \mathbf{A}$ ersetzt, muss daher gelten:
$$\mathbf{y}^T \cdot \mathbf{A} \leq \mathbf{c}^T$$
- Neue Zielvariablen $\mathbf{y} \in \mathbb{R}^M$; M : Anzahl Gleichungs-NB im ursprünglichen Problem
- Für die untere Schranke gilt dann: $f_L(\mathbf{y}) = \mathbf{y}^T \cdot \mathbf{A} \cdot \mathbf{x} = \mathbf{y}^T \cdot \mathbf{b}$



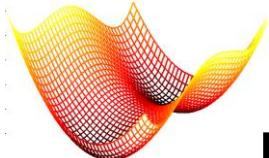
Duale Programme (2)

- Will man eine **optimale (größte)** untere Schranke ermitteln, so ergibt sich ein **neues Optimierungsproblem**, welches mit dem ursprünglichen verknüpft ist (**duales Programm**):

$$\max_{\mathbf{y}} f_L(\mathbf{y}) = \max_{\mathbf{y}} f_L(\mathbf{y}^T \cdot \mathbf{b})$$

NB: $\mathbf{y}^T \cdot \mathbf{A} \leq \mathbf{c}^T$ (Sicherstellen untere Schranke)

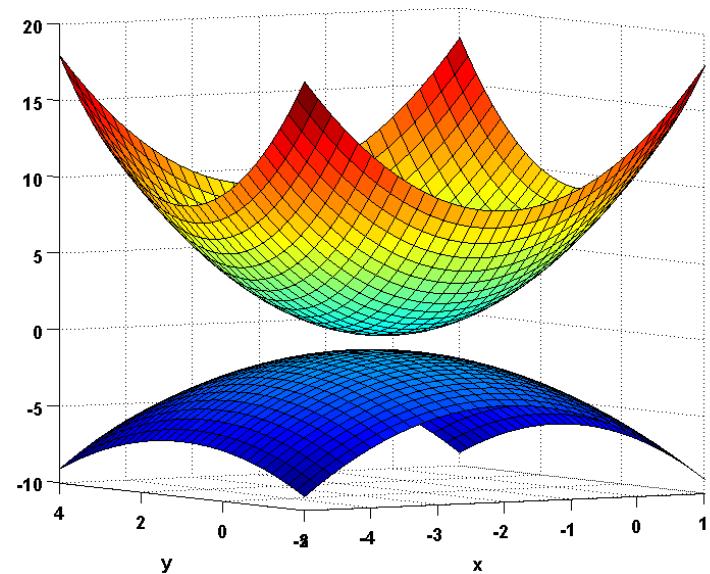
- Ursprüngl. Progr. wird als **primales Programm** bezeichnet
- **Beobachtungen:**
 - Anzahl M der Gleichungs-NB im primalen Programm entspricht Anzahl Zielvariablen im dualen Programm
 - Anzahl N der Ungleichungs-NB im dualen Programm entspricht Anzahl Zielvariablen im primalen Programm

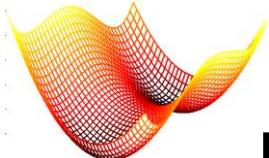


Duale Programme (3)

- Allgemein gelten folgende **Beziehungen** zwischen primalem und dualem Programm:

Primales Programm	Duales Programm
Nichtneg. Variable	Ungleichung
Nicht beschränkte Variable	Gleichung
Ungleichung	Nichtneg. Variable
Gleichung	Nicht beschränkte Variable





Duale Programme (4)

- **Schwache Dualität:** $f_L(\mathbf{y})$ ist nie größer als $f_L(\mathbf{x})$ (untere Schranke):

$$\mathbf{c}^T \cdot \mathbf{x} \geq \mathbf{y}^T \cdot \mathbf{A} \cdot \mathbf{x} = \mathbf{y}^T \cdot \mathbf{b}$$

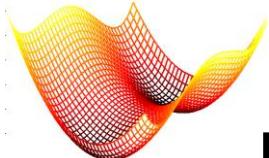
- **Starke Dualität:** vergleicht man die Optimallösungen \mathbf{x}^* und \mathbf{y}^* , so gilt:

$$f_L(\mathbf{y}^*) = f_L(\mathbf{x}^*)$$

- Besitzt ein LP in Standardform eine Optimallösung \mathbf{x}^* , so ist auch das zugehörige duale Programm lösbar (\mathbf{y}^* existiert) und es gilt die starke Dualität, d.h.:

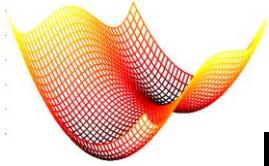
$$\mathbf{y}^{*T} \cdot \mathbf{b} = \mathbf{c}^T \cdot \mathbf{x}^*$$

- **Vorteil:** DP kann bei der Berechnung der Optimallösung des primalen LP vorteilhaft eingesetzt werden



Duale Programme (5)

- Formulierung der NB des dualen Programms in Standardform durch Einführung von **Schlupfvariablen**
 $\mathbf{s} \in \mathbb{R}^{N+}: \mathbf{y}^T \cdot \mathbf{A} + \mathbf{s}^T = \mathbf{c}^T; \mathbf{s} \geq \mathbf{0}$
 - Lösungsraum ist dann $\{\mathbf{y}, \mathbf{s}\} \in \mathbb{R}^{M+N}$
 - Aus starker Dualität folgt $\mathbf{s}^T \cdot \mathbf{x} = 0$:
 $0 = \mathbf{c}^T \mathbf{x} - \mathbf{y}^T \mathbf{b} = \mathbf{c}^T \mathbf{x} - \mathbf{y}^T \cdot \mathbf{A} \cdot \mathbf{x} = (\mathbf{c}^T - \mathbf{y}^T \cdot \mathbf{A}) \cdot \mathbf{x} = \mathbf{s}^T \cdot \mathbf{x}$
 - Für alle i gilt stets: $x_i \geq 0$ und $s_i \geq 0$, d.h. alle Summanden von $\mathbf{s}^T \cdot \mathbf{x}$ sind ≥ 0
- **Komplementärer Schlupf**: wenn $x_i > 0$, dann muss $s_i = 0$ sein (und umgekehrt), d.h. die aktiven Variablen x_i im primalen Programm entsprechen genau den inaktiven Variablen s_i im dualen Programm (und umgekehrt)

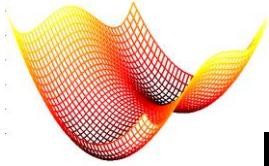


Innere-Punkte-Methode (1)

- **Alternativer Lösungsansatz:** betrachte primales und duales Programm gemeinsam
- Formuliere aus allen NB (Gleichungen) und Komplementaritätsbedingung $\mathbf{s}^T \cdot \mathbf{x} = 0$ das Gl.-Syst.

$$\psi(\mathbf{x}, \mathbf{y}, \mathbf{s}) = \begin{bmatrix} \mathbf{A} \cdot \mathbf{x} - \mathbf{b} \\ \mathbf{A}^T \cdot \mathbf{y} + \mathbf{s} - \mathbf{c} \\ \mathbf{x} \cdot \mathbf{s} \end{bmatrix} = \mathbf{0}$$

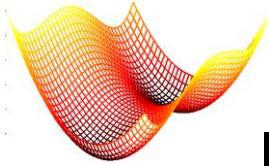
- Es gilt: $\mathbf{X} = \text{diag}(\mathbf{x})$ und $\mathbf{S} = \text{diag}(\mathbf{s})$
- Dieses GS hat $2N + M$ Gleichungen und genauso viele Unbekannte ($\mathbf{y} \in \mathbb{R}^M; \mathbf{x}, \mathbf{s} \in \mathbb{R}^N$) \rightarrow Wenn Lösung existiert ist sie eindeutig + kann d. Lösen des GS berechnet werden
- Allerdings: wegen bilinearer Terme $\mathbf{X} \cdot \mathbf{s}$ **kein lineares GS!**



Innere-Punkte-Methode (2)

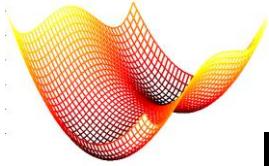
- Zum Vergleich: Simplex-Tableau ist **unterbestimmt**, d.h. hat viele (Basis-)Lösungen, wo man in jeder Iteration eine bessere Basis-Lösung sucht
- Hier: **eindeutige Lösung mittels Newton-Methode** zur Nullstellen-Suche von $\psi(\mathbf{x}, \mathbf{y}, \mathbf{s})$ möglich
- Definiere $[\mathbf{x}_k, \mathbf{y}_k, \mathbf{s}_k]^T = \mathbf{z}_k$
- Newton: Linearisierung von $\psi(\mathbf{z})$ an der Stelle \mathbf{z}_k
- Hierzu: **betrachte Jacobi-Matrix** von $\psi(\mathbf{x}, \mathbf{y}, \mathbf{s})$:

$$\mathbf{J}_\psi(\mathbf{x}, \mathbf{y}, \mathbf{s}) = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^T & \mathbf{I} \\ \mathbf{S} & \mathbf{0} & \mathbf{X} \end{bmatrix}$$



Innere-Punkte-Methode (3)

- Lineare Approx.: $\psi(\mathbf{z}_k) + \mathbf{J}_{\psi,k} \cdot \Delta \mathbf{z} := \mathbf{0} \rightarrow \mathbf{J}_{\psi,k} \cdot \Delta \mathbf{z} = -\psi(\mathbf{z}_k)$
→ Iteratives Verfahren:
- **Start:** $[\mathbf{x}_0, \mathbf{y}_0, \mathbf{s}_0]^T = \mathbf{z}_0$;
- löse das (lineare) GS $\mathbf{J}_{\psi,k} \cdot \Delta \mathbf{z} = -\psi(\mathbf{z}_k)$
- Update $\mathbf{z}_{k+1} = \mathbf{z}_k + \Delta \mathbf{z}$
- Wiederholung bis Konvergenz
- **Problem:** keine Garantie, dass $\mathbf{x}, \mathbf{s} \geq \mathbf{0}$!
- Das bedeutet, das im Lauf der Iteration der gültige Lösungsbereich verlassen werden kann. Zudem ist Nichtsingularität von \mathbf{J}_{ψ} nur bei $\mathbf{x}, \mathbf{s} > \mathbf{0}$ garantiert → Gefahr von undefiniertem Verhalten!

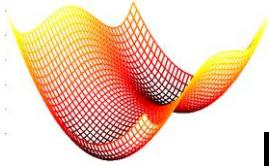


Innere-Punkte-Methode (4)

- Sicherstellen von $\mathbf{x}, \mathbf{s} > \mathbf{0}$ durch **Einführen von $\mu > 0$** :

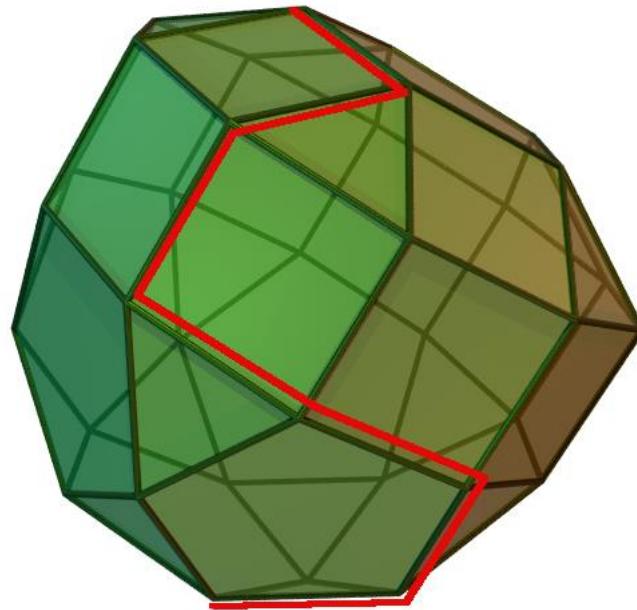
$$\psi_\mu(\mathbf{x}, \mathbf{y}, \mathbf{s}) = \begin{bmatrix} \mathbf{A} \cdot \mathbf{x} - \mathbf{b} \\ \mathbf{A}^T \cdot \mathbf{y} + \mathbf{s} - \mathbf{c} \\ \mathbf{X} \cdot \mathbf{s} - \mu \end{bmatrix}$$

- Je größer μ , umso „sicherer“ ist $\mathbf{X} \cdot \mathbf{s} > \mathbf{0}$ erfüllt, d.h. umso sicherer befinden wir uns im Inneren des Lösungsbereichs. Lösung ist aber wg. Komplementarität ($\mathbf{s}^T \cdot \mathbf{x} = 0$) am Rand!
- Iterative **Innere-Punkte-Methode**: starte mit großem μ und reduziere μ in späteren Iterationen immer weiter
- **Vorteil** gg. Simplex-Methode: Konvergenzgeschwindigkeit theoretisch abschätzbar. **Allerdings**: In der Praxis hängt es vom konkreten Problem ab, welches Verfahren schneller ist!

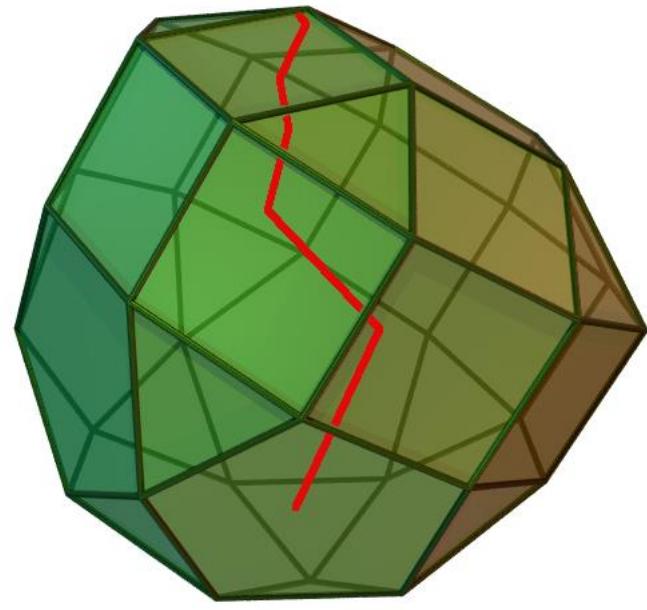


Innere-Punkte-Methode (5)

- Veranschaulichung Funktionsweise Simplex-Verfahren (links) und Innere-Punkte-Methode (rechts):



© Wikipedia / Sdo

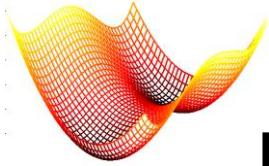


© Wikipedia / Sdo



Restringierte nichtlineare Optimierung

- **Mathematische Formulierung**
- **Projektionsverfahren**
- **Eliminationsverfahren**
- **Lagrange-Methode**
- **Penalty-Methoden**
- **Barriere-Methoden**
- **SQP-Verfahren**



Mathematische Formulierung

- Allgemeine Formulierung eines restriktierten nichtlinearen Programms CP:

$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

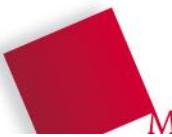
Nebenbedingungen:

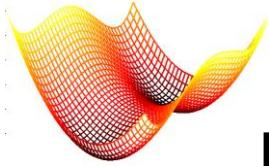
$$\mathbf{g}(\mathbf{x}) \leq 0$$

$$\mathbf{h}(\mathbf{x}) = 0$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$

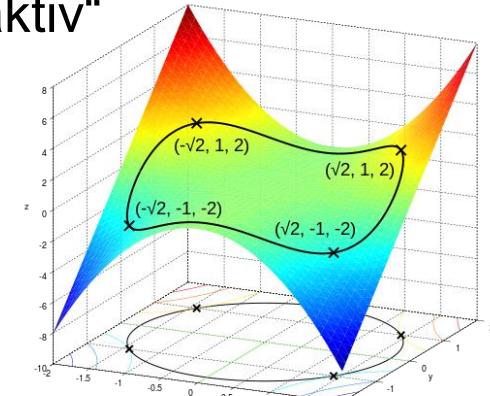
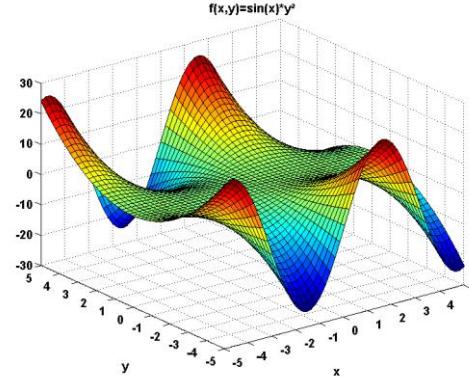
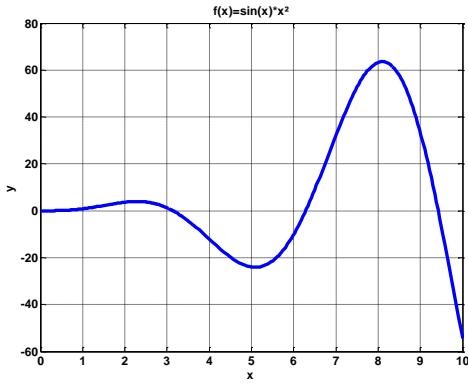
- Beliebige Funktionen $\mathbf{g}(\mathbf{x})$ und $\mathbf{h}(\mathbf{x})$ möglich
- Separate Behandlung der „side constraints“ $\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$ ermöglicht numerisch effizientere Verfahren
- Schwieriger zu behandeln als nichtrestriktierte Probleme

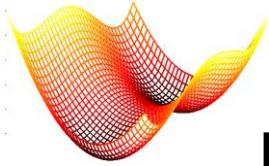




Einfluss der Nebenbedingungen

- **Rolle** der nichtlinearen NB analog zu linearen NB:
 - **Gleichungen** reduzieren Dimension des Lösungsraums um 1, z.B.: \mathbb{R}^2 : Kurve; \mathbb{R}^3 : Ebene
 - **Ungleichungen** beschränken Lösungsraum auf einen Halbraum. An Trennfläche („Hyperebene“) gilt Gleichheitszeichen → Ungleichung ist „aktiv“

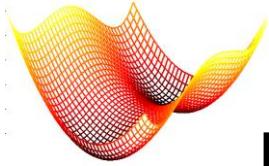




Projektionsverfahren (1)

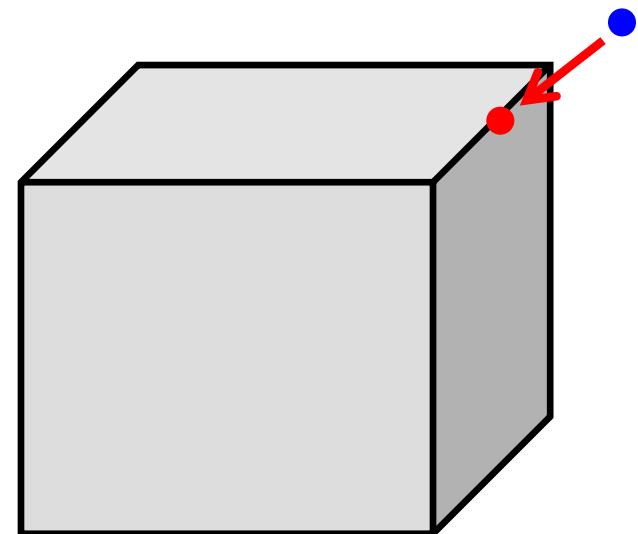
- **Grundidee:**
 - wende zur Verbesserung einer Lösung $\mathbf{x}_k \rightarrow \mathbf{x}_{k+1}$ zunächst Optimierungsverfahren für die nicht-restringierte Zielfunktion an
 - Wenn \mathbf{x}_{k+1} außerhalb Lösungsraum S : Finde dasjenige $\mathbf{x}_{k+1}^p \in S$, welches „am nächsten“ an \mathbf{x}_{k+1} liegt → Ermittle **Projektion** von \mathbf{x}_{k+1} auf S
- Effektiv v.a. bei der Behandlung von „**side/box constraints**“ $\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$, weil dann jedes Element x_i **separat** behandelt werden kann:

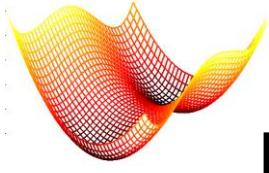
$$x_{k+1,i}^p = \min(x_{u,i}, \max(x_{k+1,i}, x_{l,i}))$$



Projektionsverfahren (2)

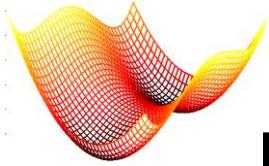
- Evtl. auch anwendbar in jedem Schritt der line search
- **Vorteil: für box constraints einfach zu berechnen**
- **Nachteil (1):** allgem. Lösung des Projektionsproblems (bei „komplexeren“ NB als box constraints) u.U. aufwändig
- **Nachteil (2):** Konvergenzeigenschaften unklar, insb. wenn Projektion die Position stark verändert





Eliminationsverfahren

- Wenn **ausschließlich Gleichungen als NB**: Betrachte M Gleichungen als Gleichungssystem
- $N - M$ Variablen können als konstant angesehen werden, dann kann GS gelöst werden und das Ergebnis in die Zielfunktion eingesetzt werden
 - **Elimination** von M Variablen möglich
 - Restproblem hat Dimension $N - M$ und ist nicht-restringiert
- **Vorteil**: „2 Fliegen“ (Vereinfachung und Elimination der NB)
- **Nachteile**: Lösung des (i.d.R. nichtlinearen) GS verursacht Aufwand und ist nicht gut automatisierbar (z.B. „übriggebliebene“ Variablen müssen als Konstanten behandelt werden)



Umwandlung restringierter Programme

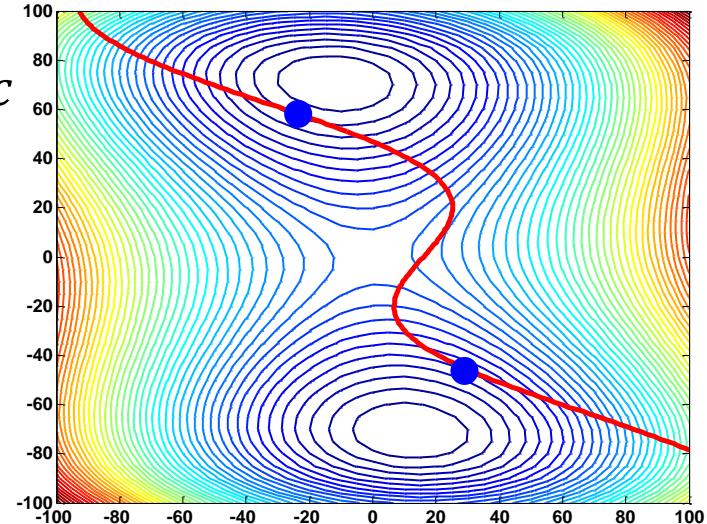
Allg. Ansatz: Umformung des restringierten Programms

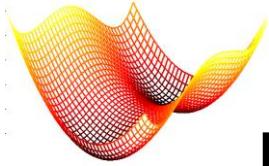
1. in ein nicht-restringiertes Programm, welches die gleiche Lösung \mathbf{x}^* besitzt.
2. In eine Folge nichtrestringierter Programme, deren Lösungen gegen \mathbf{x}^* konvergieren

- Hierzu: **Modifikation der Zielfunktion** $f_R(\mathbf{x})$ in $f_E(\mathbf{x})$
- $f_E(\mathbf{x})$ kann dann mit einer der bekannten Methoden gelöst werden
- Evtl. **iteratives Anpassen** von $f_E(\mathbf{x})$, d.h. mehrmaliges Durchführen der nichtrestringierten Optimierung mit jeweils anders modifizierter Zielfunktion

Lagrange-Funktion (1)

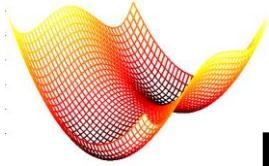
- **Beispiel:** 2-dim. Zielfkt. $f(x, y)$ mit einer Gleichung als NB: $h(x, y) = c$
- NB ist Kurve im \mathbb{R}^2 , z.B. $x^2 + y^2 = c$ definiert Kreis in xy-Ebene
- Lösung \mathbf{x}^* des restriktiven Problems muss auf Kurve liegen
- **Beobachtung:** bei Lsg. \mathbf{x}^* muss $h(x, y) = c$ **tangential** zu einer Höhenlinie $f(x, y) = \text{const}$ sein.
- **Grund:** nur dort verschwindet $\partial f(x, y)$ „in Kurvenrichtung“ (**Es gilt stets:** Gradient $\nabla f(\mathbf{x})$ ist senkrecht zu Höhenlinien $f(\mathbf{x}) = \text{const}$)





Lagrange-Funktion (2)

- NB $h(\mathbf{x}) = c$ kann auch als Höhenlinie der Funktion $h(\mathbf{x})$ aufgefasst werden: Fkt. $h(\mathbf{x})$ hat stets konstanten Wert c
 - Damit ist $\nabla h(\mathbf{x})$ ebenfalls stets senkrecht zu Höhenlinie $h(\mathbf{x}) = c$
- Im Optimum \mathbf{x}^* gilt: Gradient $\nabla f(\mathbf{x}^*)$ ist „parallel“ zu Gradient $\nabla h(\mathbf{x}^*)$, d.h. $\nabla f(\mathbf{x}^*) = -\lambda \cdot \nabla h(\mathbf{x}^*)$
- Dies kann als notwendige Bedingung für eine Lösung \mathbf{x}^* aufgefasst werden
- Definition **Lagrange-Funktion**:
- $$L(x, y, \lambda) = f(x, y) + \lambda \cdot [h(x, y) - c]$$
- Betrachten der Situation, in der der Gradient $\nabla L(x, y, \lambda)$ gleich Null wird:



Lagrange-Funktion (3)

- Wenn $\nabla L(x, y, \lambda) = \mathbf{0}$, dann ist:

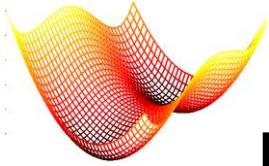
$$1. \begin{bmatrix} \partial L(x, y, \lambda) / \partial x \\ \partial L(x, y, \lambda) / \partial y \end{bmatrix} = \begin{bmatrix} \partial f(x, y) / \partial x + \lambda \cdot \partial h(x, y) / \partial x \\ \partial f(x, y) / \partial y + \lambda \cdot \partial h(x, y) / \partial y \end{bmatrix} = \mathbf{0}$$

Das ist äquivalent zu $\nabla f(\mathbf{x}) = -\lambda \cdot \nabla h(\mathbf{x})$, d.h. **Nullstelle von ∇L erfüllt die Bedingung für Optimum \mathbf{x}^* unter Berücksichtigung der NB**

$$2. \frac{\partial L(x, y, \lambda)}{\partial \lambda} = h(x, y) - c = 0$$

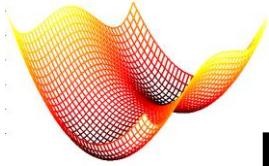
Das ist **äquivalent zur NB des Optimierungsproblems**, d.h. die NB sind jetzt in die Lagrange-Funktion quasi „integriert“!

- **Beachte:** $L(x, y, \lambda)$ ist nicht-restringiert, erfüllt aber in seinen stationären Punkten trotzdem die NB $h(x, y) = c$



Lagrange-Funktion (4)

- Punkte \mathbf{x}^*, λ^* , bei denen $\nabla L(\mathbf{x}^*, \lambda^*) = \mathbf{0}$ gilt, garantieren:
 - Ausgehend von \mathbf{x}^* kann f in „gültige“ \mathbf{x} -Richtungen (NB erfüllt) nur ansteigen (bei Minimierungsproblemen)
 - NB wird eingehalten
- Der Punkt $(\mathbf{x}^*, \lambda^*)$, welcher $\nabla L(\mathbf{x}^*, \lambda^*) = \mathbf{0}$ erfüllt liefert mit \mathbf{x}^* einen **Kandidaten** für die Lösung des restr. Programms
- **Kriterium** zum Auffinden von \mathbf{x}^* : Suche Stellen, an denen $\nabla L = \mathbf{0}$. Jetzt sind keine NB mehr nötig! (Analogie zur nicht-restringierten Optimierung: suche Stellen mit $\nabla f = \mathbf{0}$)
- **Jedoch:** $\nabla L = \mathbf{0}$ i.d.R. nur **notwendige Bedingung**, allerdings hinreichend bei konvexen Zielfunktionen
- **Außerdem:** zusätzliche Variable λ erhöht Komplexität

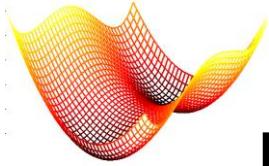


Lagrange-Funktion (5)

- **Allg. Ansatz** für L Gleichungen als NB:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot h_l(\mathbf{x})$$

- **Dann gilt:** $\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_i} = 0$ ist äquivalent zu $\frac{\partial f(\mathbf{x})}{\partial x_i} = - \sum_{l=1}^L \lambda_l \cdot \frac{\partial h_l(\mathbf{x})}{\partial x_i}$, d.h. $\nabla f(\mathbf{x}) = - \sum_{l=1}^L \lambda_l \cdot \nabla h_l(\mathbf{x})$
- Das bedeutet: wenn die Abl. der Lagr.-Fkt. gleich **0** ist, ist der Gradient der Zielfkt. f gleich der gewichteten Summe (gemäß $\boldsymbol{\lambda}$) der Gradienten der Gleichungs-Fkt. h_l
- Stellt Optimalitätskriterium unter Einhaltung mehrerer Gleichungs-NB sicher (**Verallg. „parallele Gradienten“**)
- $\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_l} = 0$ stellt Einhaltung der NB $h_l(\mathbf{x}) = 0$ sicher



Lagrange-Funktion (6)

- **Verallgemeinerung** auf Programm mit Gleichungen und Ungleichungen als NB:

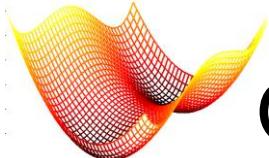
$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

NB: $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$; $\mathbf{h}(\mathbf{x}) = \mathbf{0}$

- Lagrange-Funktion ist jetzt:

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{m=1}^M \mu_m \cdot g_m(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot h_l(\mathbf{x})$$

- Ungleichungs-NB $\mathbf{g}(\mathbf{x})$ **verkomplizieren** die Situation
- Es sind **weitere Bedingungen** bzw. Einschränkungen **notwendig**, damit die Suche nach $\nabla L = \mathbf{0}$ weiterhin Kandidaten für das gesuchte Optimum liefert



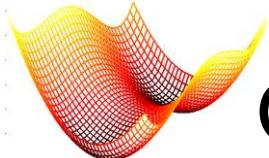
Optimalitätsbedingungen nach Karush-Kuhn-Tucker (1)

- Jetzt folgt aus den $\frac{\partial L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})}{\partial x_i} = 0$:

$$\nabla f(\mathbf{x}) + \sum_{m=1}^M \mu_m \cdot \nabla g_m(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot \nabla h_l(\mathbf{x}) = \mathbf{0}$$

(Verallgemeinerung notw. Bed. „parallele Gradienten“)

- **Zwei Fälle** für Ungleichungs-NB $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$:
 - $\mathbf{g}(\mathbf{x}) = \mathbf{0}$: Kein Problem, wird wie Gl.-NB behandelt
 - $\mathbf{g}(\mathbf{x}) < \mathbf{0}$: darf auf obige Bedingung keinen Einfluss haben → wird durch zusätzliche Forderung $\mu_m \cdot g_m(\mathbf{x}) = 0$ erreicht (**Komplementarität**)
- Man kann weiter zeigen: Es muss auch $\mu_m \geq 0$ gelten



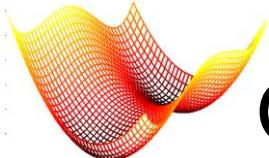
Optimalitätsbedingungen nach Karush-Kuhn-Tucker (2)

Wenn folgende **Bedingungen** erfüllt sind:

1. $\nabla f(\mathbf{x}^*) + \sum_{m=1}^M \mu_m^* \cdot \nabla g_m(\mathbf{x}^*) + \sum_{l=1}^L \lambda_l^* \cdot \nabla h_l(\mathbf{x}^*) = \mathbf{0}$
2. $\mathbf{g}(\mathbf{x}^*) \leq \mathbf{0}$ und $\mathbf{h}(\mathbf{x}^*) = \mathbf{0}$
3. $\mu_m^* \geq 0$ für $m = 1, \dots, M$
4. $\mu_m^* \cdot g_m(\mathbf{x}^*) = 0$ für $m = 1, \dots, M$ (Komplementarität)

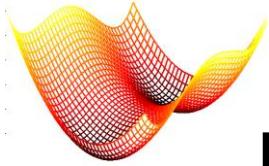
Dann ist $[\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*]$ ein **Karush-Kuhn-Tucker (KKT)-Punkt** des Optimierungsproblems, d.h. ein **Kandidat für die gesuchte Lösung**

Die KKT-Bedingungen sind **notwendige** Optimalitätsbedingungen, jedoch hinreichend für konvexe f, \mathbf{g} und affine \mathbf{h}



Optimalitätsbedingungen nach Karush-Kuhn-Tucker (3)

- Alle **aktiven** Ungleichungs-NB sowie die Gleichungs-NB definieren das sog. „**active set**“
- Ein KKT-Punkt $[x^*, \mu^*, \lambda^*]$ existiert jedoch nur, wenn bestimmte **Regularitätsbedingungen** („constraint qualifications“) erfüllt sind
- **LICQ** („linear independence constraint qualification“): Die Gradienten $\nabla g_m(x^*)$ der aktiven Ungleichungs-NB sowie aller Gleichungs-NB ($\nabla h_l(x^*)$) sind **linear unabhängig**. Daraus folgt: **KKT-Punkt existiert und ist eindeutig**
- Wenn die KKT-Bed. ein eindeutig bestimmtes Gleichungs-System definieren ist u.U. eine **direkte Lösung** der Optimierungsaufgabe durch Lösen dieses GS möglich

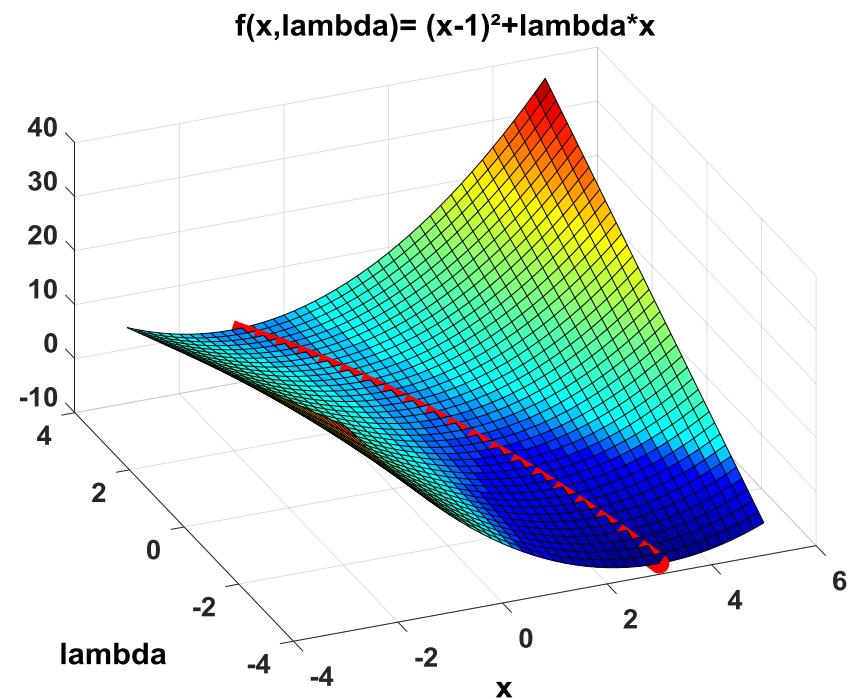


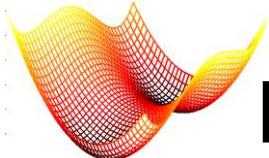
Lagrange-Dualität (1)

- Betrachte Lagrange-Fkt. L bei „festem“ $\mu = \mu'$, $\lambda = \lambda'$. Dann ist L nur noch von den Variablen x abhängig
- Dann ist $x'' = \underset{x \in S}{\operatorname{argmin}} L(x, \mu = \mu', \lambda = \lambda')$ dasjenige x , welches L bei „festem“ μ', λ' minimiert.
- Ermittlung von x'' bei verschiedenen Werten für μ', λ'
→ D ist Menge aller x'' , welche L für konstante μ', λ' minimieren.
- Dann ist $\max_{\mu, \lambda} q(\mu, \lambda)$ mit $q(\mu, \lambda) = \underset{x \in D}{L}(x, \mu, \lambda)$ das **duale Problem** zum ursprünglichen Optimierungsproblem PN
- **Beachte:** $q(\mu, \lambda)$ ist stets **konkav** (hat immer Maximum)
- L hängt sowohl von primalen als auch dualen Variablen ab!
- Im **KKT-Pkt** $[x^*, \mu^*, \lambda^*]$ gilt **starke Dualität**: $q(\mu^*, \lambda^*) = f(x^*)$

Lagrange-Dualität (2): Beispiel

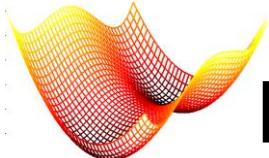
- **Beispiel:** Betrachte $f(x) = (x - 1)^2$; NB: $x = 0$
- Das ergibt: $L(x, \lambda) = (x - 1)^2 + \lambda \cdot x$
- $\partial L(x, \lambda) / \partial x = 2(x - 1) + \lambda$
- Minimum bei festem λ' :
 $2(x - 1) + \lambda' = 0 \rightarrow x = -\frac{\lambda'}{2} + 1$
- D.h. $q(\lambda) = \lambda - \lambda^2/4 \rightarrow$ konkave Funktion, hat Max.!
- Starke Dualität: $q(\lambda^* = 2) = f(x^* = 0) = 1$





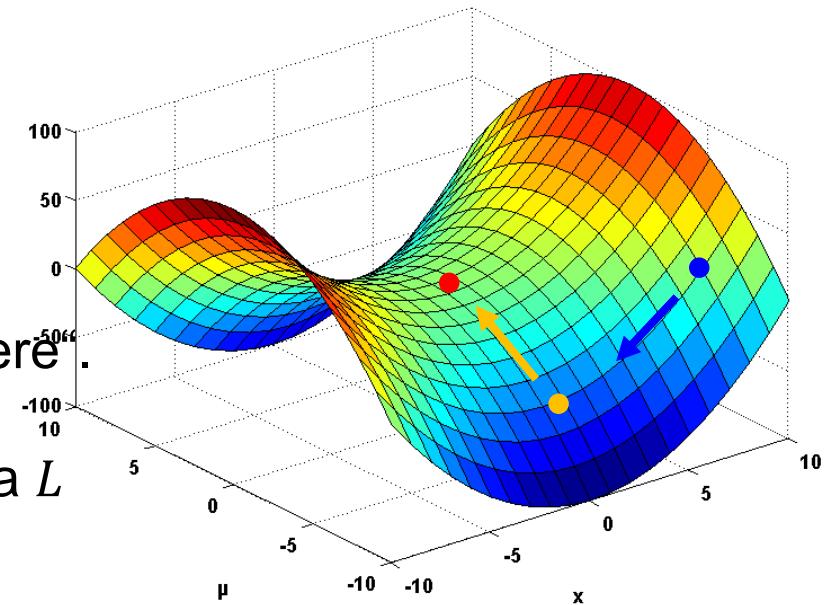
Lagrange-Funktion: Sattelpunkt-Optimierung (1)

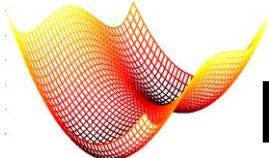
- Bei „festem“ μ', λ' hat $L(\mathbf{x}, \mu', \lambda')$ Minimum in \mathbf{x} -Richtung
 - Beschränkt man sich auf alle diese Minima (d.h. $\mathbf{x} \in D$), so ist der KKT-Punkt das Maximum von $\max_{\mathbf{x} \in D} L(\mathbf{x}, \mu, \lambda)$
- KKT-Punkt $[\mathbf{x}^*, \mu^*, \lambda^*]$ ist **Sattelpunkt** der Lagr.-Fkt. $L(\mathbf{x}, \mu, \lambda)$!
- KKT-Punkt $[\mathbf{x}^*, \mu^*, \lambda^*]$ kann „direkt“ durch „Optimierung“ der Lagrange-Funktion $L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \sum_{m=1}^M \mu_m \cdot g_m(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot h_l(\mathbf{x})$ ermittelt werden (Suche des Sattelpunktes)
 - Alternierende Suche mittels
 - Minimierung von L bei „festem“ μ', λ'
 - Trick: Näherung für die Maximums-Suche: halte \mathbf{x}' konstant und ändere μ, λ so, dass sich L vergrößert



Lagrange-Funktion: Sattelpunkt-Optimierung (2)

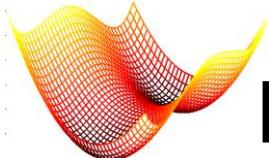
- **Vorgehen: Alternierende Optimierung** von $L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$:
 1. Setze $\boldsymbol{\mu}, \boldsymbol{\lambda} = \text{const}$ und finde $\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ (Minimierung in \mathbf{x} -Richtung)
 2. Setze $\mathbf{x} = \text{const}$ und „maximiere“ L in $\boldsymbol{\mu}, \boldsymbol{\lambda}$ -Richtung: **Gradient Ascent** mit fester Schrittwe., da L linear in $\boldsymbol{\mu}, \boldsymbol{\lambda}$
 3. Wiederhole 1. und 2. bis Konvergenz (gegen $[\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*]$)





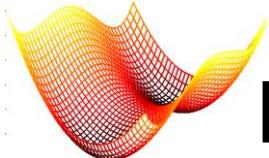
Lagrange-Funktion: Sattelpunkt-Optimierung (3)

- **Beachte:** keine Garantie, dass
 1. Verfahren gegen einen Sattelpunkt konvergiert
 2. Sattelpunkt von L das gesuchte Minimum des restringierten Problems liefert!
- **Allgemein** gilt lediglich: wenn die notw. Bedingungen für „Optimum“ des modifizierten Problems, d.h. $\nabla L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathbf{0}$ erfüllt sind, dann sind auch die notw. Optimalitätsbed. des Ursprungsproblems (zusammen mit den NB) erfüllt
 - Sattelpunkt-Optimierung liefert lediglich einen „**kritischen Punkt**“, d.h. Kandidaten für das Optimum des Urspr.-Probl.
 - Praktikabilität hängt vom konkreten Problem ab (Beschaffenheit Zielfunktion, „Güte“ des Startpunktes, etc.)



Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (1)

- **Beobachtung:** digitale Bilder $I(x, y)$ enthalten immer ein (unerwünschtes) **Rauschen**, welches näherungsweise als gauß-verteilt angenommen werden kann.
 - **Aufgabe:** Schätze das rekonstruierte Bild $\hat{R}(x, y)$, welches das Rauschen bestmöglich unterdrückt
 - **Mathematisch:** Minimiere Kostenfunktion E
- **Sinnvolle Modellierung:** Minimiere „Unterschied“ zwischen I und \hat{R} , d.h. $E = \sum_x \sum_y (\hat{R}(x, y) - I(x, y))^2$
- Problem ist **schlecht gestellt („ill-posed“)**, da die offensichtliche Lösung $\hat{R}(x, y) = I(x, y)$ nicht weiter hilft.

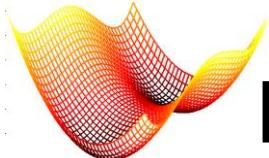


Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (2)

- **Beobachtung:** Pixel-weises Rauschen führt zu Helligkeitsunterschieden zwischen benachbarten Pixeln
→ **Regularisierung:** Favorisiere Lösungen, bei denen $\hat{R}(x, y)$ in Summe möglichst wenig lokale Intensitäts-Schwankungen aufweist:

$$E_{TV} = \frac{1}{2\lambda} \sum_x \sum_y (\hat{R}(x, y) - I(x, y))^2 + \sum_x \sum_y |\nabla \hat{R}(x, y)|$$

- E_{TV} enthält „**Total Variation**“ Regularisierungsterm (**kann als NB aufgefasst werden**)
- **Vorteil:** „ursprüngl.“ Intensitäts-Sprünge bleiben erhalten
- **Nachteil:** nicht überall differenzierbar → Anwendung von Abstiegs-Methoden so nicht möglich



Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (3)

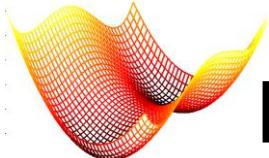
- **Umformung:** Führe Vektorfeld $\mathbf{p}(x, y) = [p_1(x, y), p_2(x, y)]$ als duale Variable ein
- Mit Hilfe von \mathbf{p} lässt sich $|\mathbf{v}|$ als ein (differenzierbares) Skalarprodukt beschreiben: $|\mathbf{v}| = \max_{|\mathbf{p}| \leq 1} \langle \mathbf{v}, \mathbf{p} \rangle$

$$E_{TV} = \max_{|\mathbf{p}| \leq 1} \left[\frac{1}{2\lambda} \sum_x \sum_y (\hat{R}(x, y) - I(x, y))^2 + \sum_x \sum_y \langle \nabla \hat{R}(x, y), \mathbf{p}(x, y) \rangle \right]$$

- Daraus ergibt sich für die Lösung R^* :

$$R^* = \min_R \max_{|\mathbf{p}| \leq 1} \left[\frac{1}{2\lambda} \sum_x \sum_y (\hat{R}(x, y) - I(x, y))^2 + \langle \nabla \hat{R}(x, y), \mathbf{p}(x, y) \rangle \right]$$

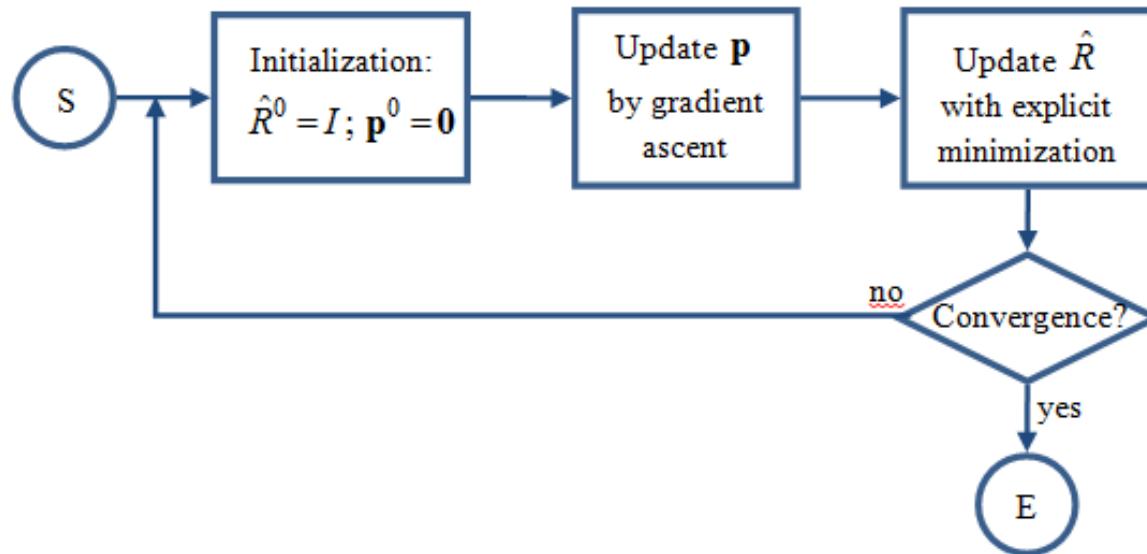
→ Aufgabe ist (primale-duale) Sattelpunkt-Optimierung von $E_{TV}(\hat{R}, \mathbf{p})$

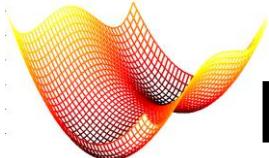


Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (4)

- **Iteratives Vorgehen:**

1. Update von \mathbf{p} , wobei $\hat{\mathcal{R}}$ konstant gehalten wird
2. Update von $\hat{\mathcal{R}}$, wobei \mathbf{p} konstant gehalten wird





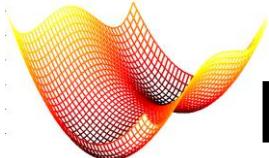
Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (5)

- **Numerische Implementierung** der Optimierungsschritte:
- $\partial E_{TV}(\hat{R}, \mathbf{p}) / \partial \mathbf{p} = \nabla \hat{R}$ → **Gradienten-Anstiegs-Verfahren**:

$$\mathbf{p}^{n+1}(x, y) = \frac{\mathbf{p}^n(x, y) + \tau \cdot \nabla \hat{R}^n(x, y)}{\max\left(1, \left|\mathbf{p}^n(x, y) + \tau \cdot \nabla \hat{R}^n(x, y)\right|\right)}$$

- Nenner dient **Rückprojektion** auf gültige Lsg. (NB $|\mathbf{p}| \leq 1$)
- **Umformung**: $\sum_x \sum_y \langle \nabla \hat{R}, \mathbf{p} \rangle = - \sum_x \sum_y \hat{R} \cdot \text{div}(\mathbf{p})$
- $\partial E_{TV}(\hat{R}, \mathbf{p}) / \partial \hat{R} = -\text{div}(\mathbf{p}) + 1/\lambda \cdot (\hat{R} - I)$ mit $\text{div}(\mathbf{p}) = \partial p_1 / \partial x + \partial p_2 / \partial y$ → explizite Minimierung möglich:

$$\hat{R}^{n+1}(x, y) = \hat{R}^n(x, y) + \lambda \cdot \text{div}[\mathbf{p}^{n+1}(x, y)]$$

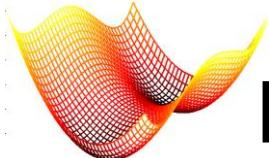


Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (6)

- Anzahl der Unbekannten in \hat{R} und \mathbf{p} ist $W \times H$ (Anzahl der pixel des Bildes)! \rightarrow sehr effiziente Berechnung notwendig!
 \rightarrow Bei \hat{R} forward difference, bei $\text{div}(\mathbf{p})$ „backward difference“

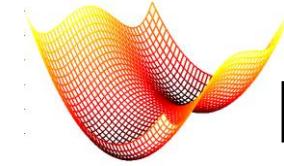
$$\nabla \hat{R}(x, y) = \begin{bmatrix} \hat{R}_x \\ \hat{R}_y \end{bmatrix} \quad \hat{R}_x = \begin{cases} \hat{R}(x+1, y) - \hat{R}(x, y) & \text{if } 0 < x < W \\ 0 & x = W \end{cases}$$
$$\hat{R}_y = \begin{cases} \hat{R}(x, y+1) - \hat{R}(x, y) & \text{if } 0 < y < H \\ 0 & y = H \end{cases}$$

$$\text{div}[\mathbf{p}(x, y)] = \begin{cases} p_1(x, y) - p_1(x-1, y) & \text{if } 1 < x \leq W \\ p_1(x, y) & x = 1 \end{cases}$$
$$+ \begin{cases} p_2(x, y) - p_2(x, y-1) & \text{if } 1 < y \leq H \\ p_2(x, y) & y = 1 \end{cases}$$

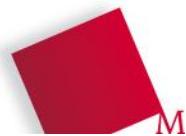
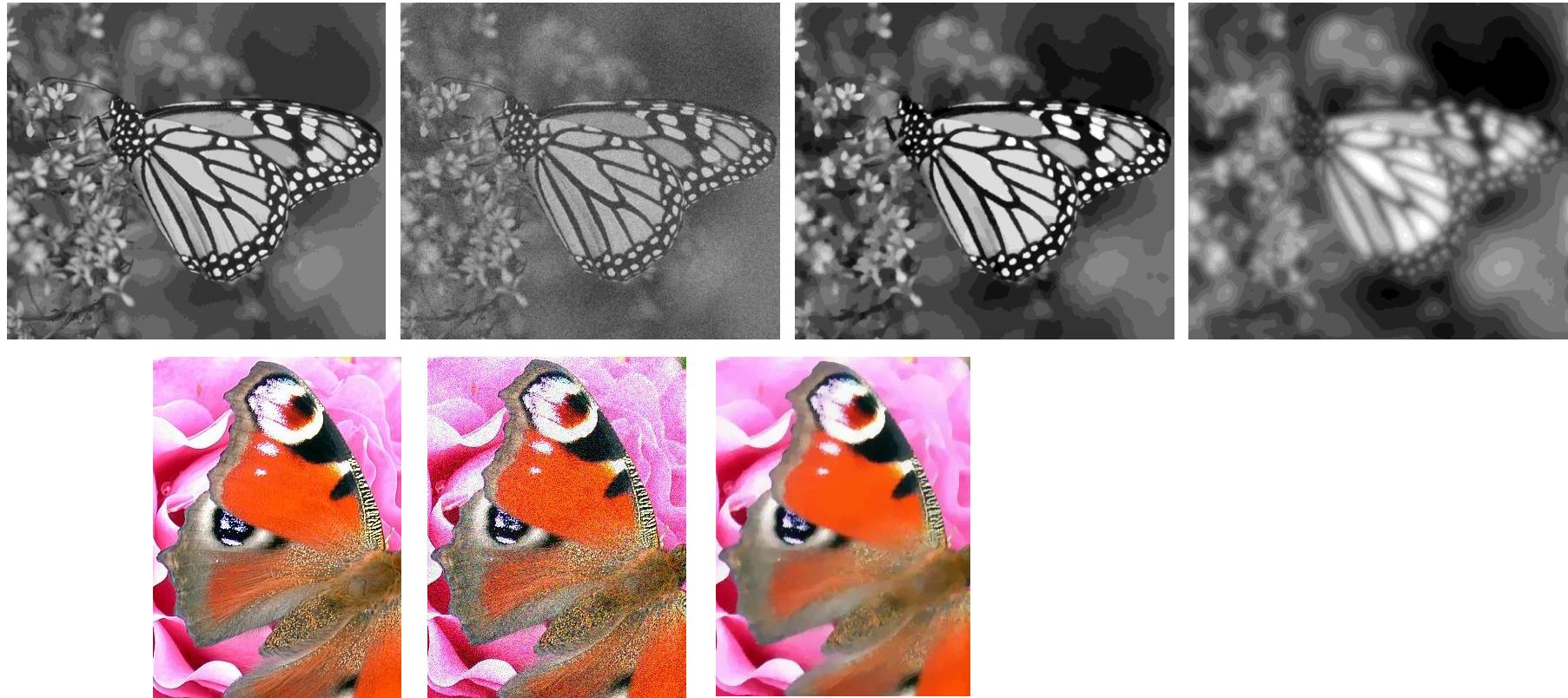


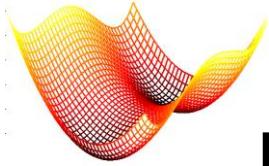
Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (7)

```
function denoiseTotalVariation (in Image I, in parameters  $\tau$ 
and  $\lambda$ , in convergence crit.  $\varepsilon$ , out reconstructed image  $\hat{R}^*$ )
// initialization
n ← 0;  $\hat{R}^0 ← I$ ; p0 ← 0
// main optimization loop
repeat
    // gradient ascend in p-direction ( $\hat{R}$  fixed)
    for y = 0 to H-1
        for x = 0 to W-1
            calculate  $\nabla \hat{R}^n(x, y)$ 
             $p^{n+1}(x, y) = \frac{p^n(x, y) + \tau \cdot \nabla \hat{R}^n(x, y)}{\max(1, |p^n(x, y) + \tau \cdot \nabla \hat{R}^n(x, y)|)}$  // update p
    // explicit minimization in  $\hat{R}$ -direction (p fixed)
    for y = 0 to H-1
        for x = 0 to W-1
            calculate  $\text{div}[p^{n+1}(x, y)]$ 
             $\hat{R}^{n+1}(x, y) = I(x, y) + \lambda \cdot \text{div}[p^{n+1}(x, y)]$  // update  $\hat{R}$ 
    calculate energy  $E_{TV}^{n+1}$ 
    n ← n+1
until  $E_{TV}^n - E_{TV}^{n-1} < \varepsilon$  // convergence criterion
 $R^* ← \hat{R}^n$ 
```



Beispiel Sattelpunkt-Optimierung: Total Variation Denoising (8)





Least Squares mit linearen NB (1)

- **Betrachte:** quadratisches Programm QPN mit linearen NB:

$$\min_{\mathbf{x} \in S} f_Q(\mathbf{x}) = \min_{\mathbf{x} \in S} \mathbf{x}^T \cdot \mathbf{H} \cdot \mathbf{x} + \mathbf{c}^T \cdot \mathbf{x}$$

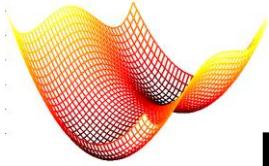
Nebenbedingungen: $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}; \mathbf{x} \geq \mathbf{0}$

- **Optimierung:** Formuliere Lagrange-Funktion

$$L(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \mathbf{x}^T \cdot \mathbf{H} \cdot \mathbf{x} + \mathbf{c}^T \cdot \mathbf{x} + \mathbf{u}^T \cdot (\mathbf{A} \cdot \mathbf{x} - \mathbf{b}) - \mathbf{v}^T \cdot \mathbf{x}$$

- Least Squares mit lin. NB eignet sich zur Optimierung mittels **Analyse der KKT-Bedingungen**

- **Strategie:** Fasse alle Gleichungen der KKT-Bedingungen als Gleichungs-System auf. Der \mathbf{x} -Anteil der Lösung dieses GS liefert dann Kandidat für das Optimum von QPN (bei QPN sind KKT-Bedingungen hinreichend)



Least Squares mit linearen NB (2)

- Die **KKT-Bedingungen** sind bei QPN:

$$\nabla L(\mathbf{x}, \mathbf{u}, \mathbf{v}) = 2\mathbf{H} \cdot \mathbf{x} + \mathbf{c} + \mathbf{A}^T \cdot \mathbf{u} - \mathbf{v} = \mathbf{0}$$

$$\Rightarrow -2\mathbf{H} \cdot \mathbf{x} - \mathbf{A}^T \cdot \mathbf{u} + \mathbf{v} = \mathbf{c} \quad (i)$$

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}; \mathbf{x} \geq \mathbf{0}$$

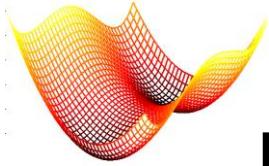
$\mathbf{u} \geq \mathbf{0}; \mathbf{v} \geq \mathbf{0}$ (Lagrange-Multiplikator bei Ungleichungen)

$$\Rightarrow \mathbf{A} \cdot \mathbf{x} + \mathbf{s} = \mathbf{b}; \text{ (Einführen von Schlupf-Var.)} \quad (ii)$$

$$\mathbf{v}^T \cdot \mathbf{x} = 0; \mathbf{u}^T \cdot \mathbf{s} = 0 \text{ (Komplementarität)} \quad (iii)$$

$$\mathbf{x} \geq \mathbf{0}; \mathbf{u} \geq \mathbf{0}; \mathbf{v} \geq \mathbf{0}; \mathbf{s} \geq \mathbf{0} \quad (iv)$$

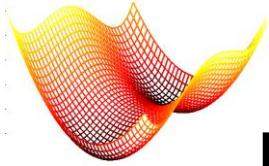
- Beobachtung:** Ähnlichkeit der KKT-Bed. (i)-(iv) zu NB bei linearen Programmen in Standardform
- Allerdings:** GS ist **nichtlinear** wg. Komplementarität



Least Squares mit linearen NB (3)

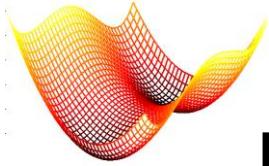
- Durch Umformung von \mathbf{A} ist stets $\mathbf{b} \geq \mathbf{0}$ möglich: Jede Gleichung kann mit beliebigem Faktor multipliziert bzw. mit Linearkombination der anderen Gl. modifiziert werden
 - Wenn $\mathbf{c} \geq \mathbf{0}$, dann Annahme: $\mathbf{x} = \mathbf{0}; \mathbf{u} = \mathbf{0}$
 - Dann folgt aus (i) $\mathbf{v} = \mathbf{c}$ und aus (ii) $\mathbf{s} = \mathbf{b}$
 - **triviale Lsg.**: $\mathbf{x} = \mathbf{0}; \mathbf{u} = \mathbf{0}; \mathbf{v} = \mathbf{c}; \mathbf{s} = \mathbf{b}$ liefert KKT-Punkt
 - Jedoch: Bei realen Problemen sind i.d.R. einzelne $c_i < 0$
 - Dann Umformung von (i) durch Einführen neuer Variablen $z_i > 0$ für alle $c_i < 0$:

$$-2\mathbf{H} \cdot \mathbf{x} - \mathbf{A}^T \cdot \mathbf{u} + \mathbf{v} + \sum_{c_i < 0} e_i z_i = \mathbf{c}_+ \quad (i')$$



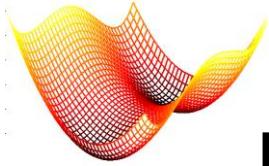
Least Squares mit linearen NB (4)

- e_i : i -ter Einheitsvektor
- **Jetzt gilt stets $c_+ \geq 0$**
- Jetzt kann **Lösung effizient** berechnet werden:
 - Start: $v_i = c_i$ falls $c_i \geq 0$, sonst $v_i = 0$ und $z_i = -c_i$
 - Dann Minimieren der linearen Zielfkt. $\sum_{c_i < 0} z_i$ unter den NB (i), (ii), (iii) und (iv) mittels **Simplex-Algorithmus**
- Wegen der nichtlinearen (iii) wird Simplex-Algorithmus geringfügig bei der Auswahl von „ x_{in} “ **modifiziert**:
 - Ist s_i Teil der Basislösung, darf u_i nicht hinein (und umgekehrt)
 - Ist x_i Teil der Basis, darf v_i nicht hinein (und umgekehrt)



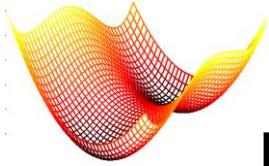
Beispiel: Portfolio-Optimierung (1)

- **Aufgabe:** Investiere ein bestimmtes Kapital K in N verschiedene Investments möglichst optimal. D.h. gesucht ist die „Aufteilung“ von K in $\mathbf{x} = [x_1 \quad \dots \quad x_N]$ so, dass
 - Das **Risiko** möglichst **minimiert** wird
 - Der **Ertrag** E möglichst **groß** ist
- Betrachte zur Risiko-Minimierung die in der **Vergangenheit erzielten Renditen** \mathbf{r}_l ; $1 \leq l \leq L$ der Investments in den vergangenen L Perioden.
- Daraus lässt sich berechnen:
 - **Mittlere Rendite** $\hat{\mathbf{r}} = 1/L \cdot \sum_{l=1}^L \mathbf{r}_l$
 - **Kovarianzmatrix** $\mathbf{C} = 1/L \cdot \sum_{l=1}^L (\mathbf{r}_l - \hat{\mathbf{r}}) \cdot (\mathbf{r}_l - \hat{\mathbf{r}})^T$



Beispiel: Portfolio-Optimierung (2)

- Aus den Kenngrößen der Vergangenheit lässt sich für eine angenommene Aufteilung von K in \mathbf{x} folgendes abschätzen:
 - **Erwartungswert des Ertrags:** $\hat{E} = \hat{\mathbf{r}}^T \cdot \mathbf{x}$
 - **Varianz des Ertrags:** $\sigma(\mathbf{x})^2 = 1/L \cdot \sum_{l=1}^L (\mathbf{r}_l^T \mathbf{x} - \hat{\mathbf{r}}^T \mathbf{x})^2 = \mathbf{x}^T \cdot \mathbf{C} \cdot \mathbf{x}$ (\mathbf{C} : Kovarianzmatrix)
- Mögliches **Optimierungskriterium:** Minimiere Risiko, d.h. Varianz des Ertrags unter der Bedingung, dass der erwartete Ertrag eine bestimmte Schwelle μ überschreitet:
 - **Minimierungsaufgabe:** finde $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} \mathbf{x}^T \cdot \mathbf{C} \cdot \mathbf{x}$
 - Nebenbed.: $\hat{\mathbf{r}}^T \cdot \mathbf{x} \geq \mu$; $\sum_{i=1}^N x_i \leq K$; $\mathbf{x} \geq \mathbf{0}$
 - Allerdings: noch Modifikationen nötig! (v.a. wg. „ \geq -NB“)



Penalty-Verfahren (1)

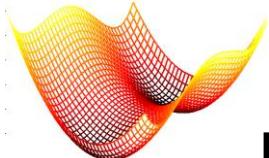
- Betrachte wieder allg. restringiertes Programm (PR)

$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

$$\text{NB: } \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \mathbf{h}(\mathbf{x}) = \mathbf{0}$$

- **Grundidee:** Finde Lösung \mathbf{x}^* durch iteratives Berechnen von (einfacher zu lösenden) nicht-restringierten Problemen, die (PR) immer besser approximieren
- **Hierzu:** Einführen von **Straf-Funktionen** $c(\mathbf{x}): \mathbb{R}^N \rightarrow \mathbb{R}_+$, deren Funktionswert umso größer wird, je weiter \mathbf{x} von S entfernt liegt, d.h.:

$$\begin{aligned} c(\mathbf{x}) &> 0 \quad \forall \mathbf{x} \notin S \\ c(\mathbf{x}) &= 0 \quad \forall \mathbf{x} \in S \end{aligned}$$



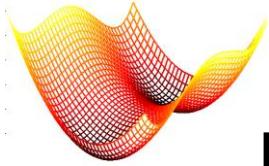
Penalty-Verfahren (2)

- Beispiel für $c(\mathbf{x})$:

$$c(\mathbf{x}) = \sum_l |h_l(\mathbf{x})|^\alpha + \sum_m g_m^+(\mathbf{x})^\alpha$$

Mit $g_m^+(\mathbf{x}) := \max\{0, g_m(\mathbf{x})\} \rightarrow$ Es gilt stets $c(\mathbf{x}) \geq 0$

- **Neue Zielfunktion** unter Berücksichtigung der **Strafterme**:
 $p(\mathbf{x}, r) = f(\mathbf{x}) + r \cdot c(\mathbf{x}); \quad r > 0$
- Für **kleine** r unterscheidet sich $\underset{\mathbf{x}}{\operatorname{argmin}} p(\mathbf{x}, r)$ kaum von der Lösung des nichtrestringierten Problems $\underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$
- Für **große** r hingegen dürfte Lösung $\underset{\mathbf{x}}{\operatorname{argmin}} p(\mathbf{x}, r)$ die Einhaltung der NB sicher stellen, d.h. eine gültige Lösung von (PR) liefern



Penalty-Verfahren (3)

→ Ansatz für iterative Strategie:

start with $\mathbf{x}_0 \in \mathbb{R}^2$ and small $r_0 > 0$

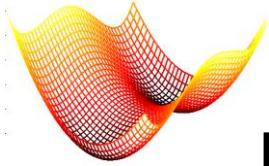
repeat

- 1.) calculate $\mathbf{x}_{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} p(\mathbf{x}_k, r)$

- 2.) $r_{k+1} = \beta \cdot r_k; \quad \beta \geq 2$
 $k \leftarrow k + 1$

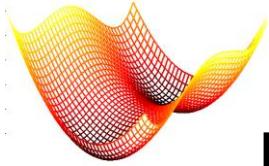
until convergence (e.g. $\mathbf{x}_k \in S$)

- Annäherung an S bzw. \mathbf{x}^* „**von außen**“: zuerst Suche des „besten“ Minimierers, dann Absichern d. Einhaltung der NB
- r wird im Lauf der Iteration immer größer
- **Achtung:** Konvergenz gegen Minimum \mathbf{x}^* nicht garantiert!
 - $p(\mathbf{x}, r)$ z.B. wg. „Knick“ in $g_m^+(\mathbf{x})$ nicht überall differenzierbar



Penalty-Verfahren (4)

- Wünschenswertes Verhalten:
 1. Verwendung von Abstiegsverfahren $\rightarrow p(\mathbf{x}, r)$ sollte **überall differenzierbar** sein
 2. **Schnelle Konvergenz** bereits bei möglichst kleinen r
 - Diese beiden Wünsche sind in der Regel **unvereinbar**:
 - Für $\alpha = 1$ lässt sich zeigen: Verfahren konvergiert gegen die Optimallösung \mathbf{x}^* , wenn gilt: $r > \max\{\mu_i^*\}$
 - Bei $\alpha = 2$ ist $c(\mathbf{x})$ zwar überall differenzierbar, dann ist Konvergenz aber nicht mehr garantiert
- Modifiziere $p(\mathbf{x}, r)$ so, dass beide Forderungen gelten:
erweiterte Lagrange-Funktion („augmented Lagrangian“)

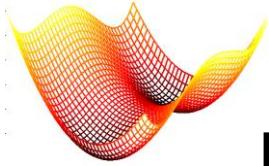


Penalty-Verfahren (5)

- **Definition** der erweiterten Lagrange-Funktion $\Lambda(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \mathbf{r})$:

$$\begin{aligned}\Lambda(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \mathbf{r}) = & f(\mathbf{x}) + \sum_{m=1}^M \frac{r_m}{2} \left[\left(g_m(\mathbf{x}) + \frac{\mu_m}{r_m} \right)^+ \right]^2 + \\ & \sum_{l=1}^L \frac{r_l}{2} \left[h_l(\mathbf{x}) + \frac{\lambda_l}{r_l} \right]^2 - \frac{1}{2} \sum_{m=1}^M \frac{\mu_m^2}{r_m} - \frac{1}{2} \sum_{l=1}^L \frac{\lambda_l^2}{r_l}\end{aligned}$$

- Es lässt sich zeigen:
 - $\Lambda(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \mathbf{r})$ entspricht Summe aus „normaler“ Lagrange-Funktion $L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ und speziellem Strafterm $c(\mathbf{x}, \mathbf{r})$
 - $\Lambda(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \mathbf{r})$ ist differenzierbar
 - Sattelpunkt $[\mathbf{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*]$ von Λ enthält Lösung \mathbf{x}^* von (PR)



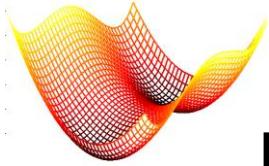
Penalty-Verfahren (6)

- **Lösungsstrategie** ähnlich wie zuvor: iteratives **alternierendes Update**: minimiere \mathbf{x} bei festem $\boldsymbol{\mu}, \boldsymbol{\lambda}$, maximiere dann $\boldsymbol{\mu}, \boldsymbol{\lambda}$ bei festem \mathbf{x}
- Bei **Minimierung** in \mathbf{x} -Richtung verwende Gradienten

$$\nabla_{\mathbf{x}} \Lambda(\mathbf{x}, \mathbf{r}) = \nabla f(\mathbf{x}) + \sum_{m=1}^M r_m \left(g_m(\mathbf{x}) + \frac{\mu_m}{r_m} \right)^+ \nabla g_m(\mathbf{x}) + \sum_{l=1}^L r_l \left(h_l(\mathbf{x}) + \frac{\lambda_l}{r_l} \right) \nabla h_l(\mathbf{x})$$

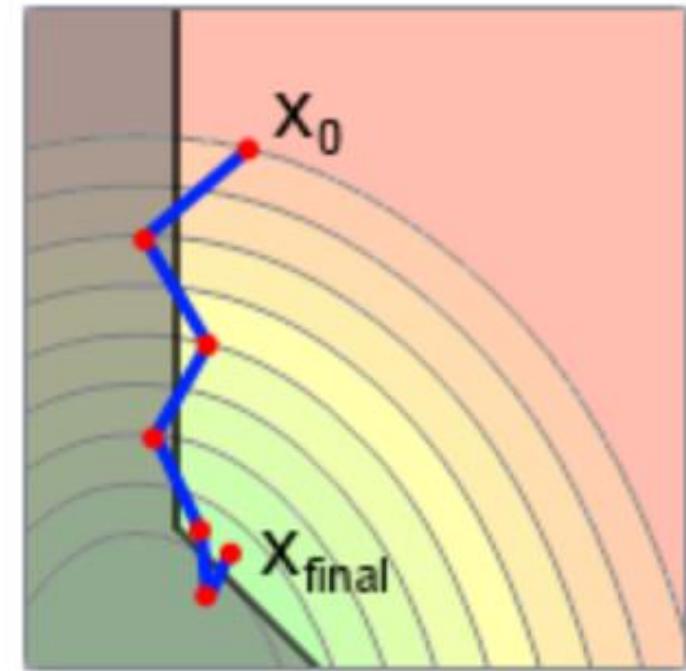
- **Maximierung** in $\boldsymbol{\mu}, \boldsymbol{\lambda}$ -Richtung: „punkt-weises“ Update:

$$\mu_m^{k+1} := r_m \left(g_m(\mathbf{x}) + \frac{\mu_m}{r_m} \right)^+ ; \lambda_l^{k+1} := r_l \left(h_l(\mathbf{x}) + \frac{\lambda_l}{r_l} \right)$$

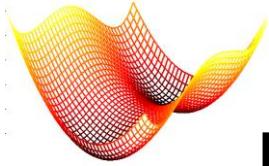


Penalty-Verfahren (7)

- **Prinzipielles Vorgehen:**
sukzessives Erhöhen des Gewichtungsfaktors r für die Strafterme
- Wegen der zusätzlich zum Strafterm vorhandenen Lagrange-Multiplikatoren ist Wahl von r unkritisch: oft bleibt r klein und konstant, trotzdem rasche Konvergenz
- Beispiel:



© ALGLIB



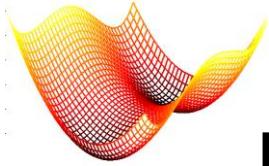
Barriere-Methode (1)

- Betrachte restr. Programm (PR) mit Ungleichungen als NB

$$\min_{\mathbf{x} \in S} f(\mathbf{x}); \text{ NB: } g(\mathbf{x}) \leq 0$$

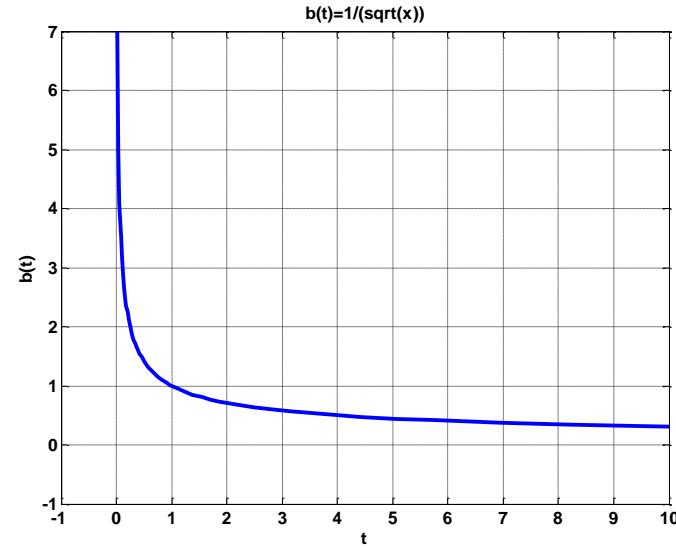
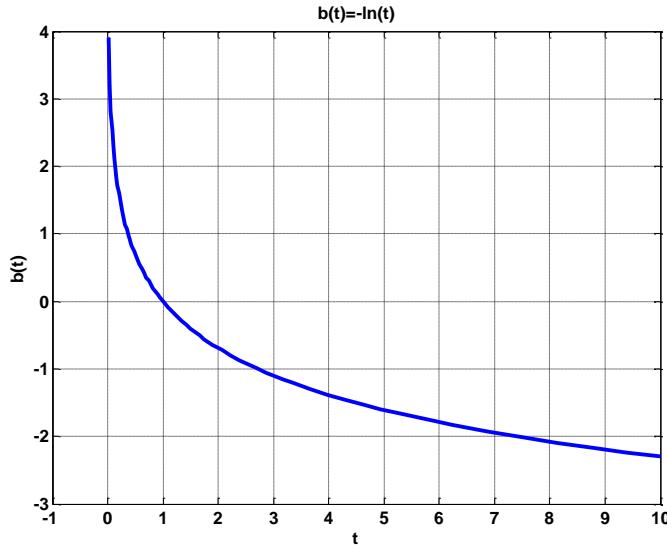
- **Grundidee:** wenn $\mathbf{x}_0 \in S$, dann stelle sicher, dass alle \mathbf{x}_k in S bleiben \rightarrow **Strafterme** $b(t)$ mit $t = -g_m(\mathbf{x})$, welche zum Rand von S hin ansteigen und außerhalb S unendlich sind.
 \rightarrow **Innere-Punkte-Methode:** Strafterme bewirken „Abstoßen“ vom Rand von S ; Annäherung an \mathbf{x}^* „von innen“
- **Neue Zielfunktion** unter Berücksichtigung der **Strafterme**:

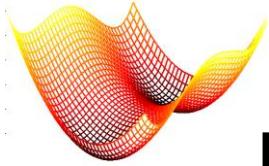
$$p(\mathbf{x}, \mu) = f(\mathbf{x}) + \mu \cdot \sum_{m=1}^M b[-g_m(\mathbf{x})]; \mu > 0$$



Barriere-Methode (2)

- **Beispiele** für Straf-Funktionen:
 - Für Ungl. als NB: $b(t) = -\ln(t)$; $b(t) = 1/t^\alpha$; $\alpha > 0$
 - Für Gleichungen als NB: $b(t) = t^2$



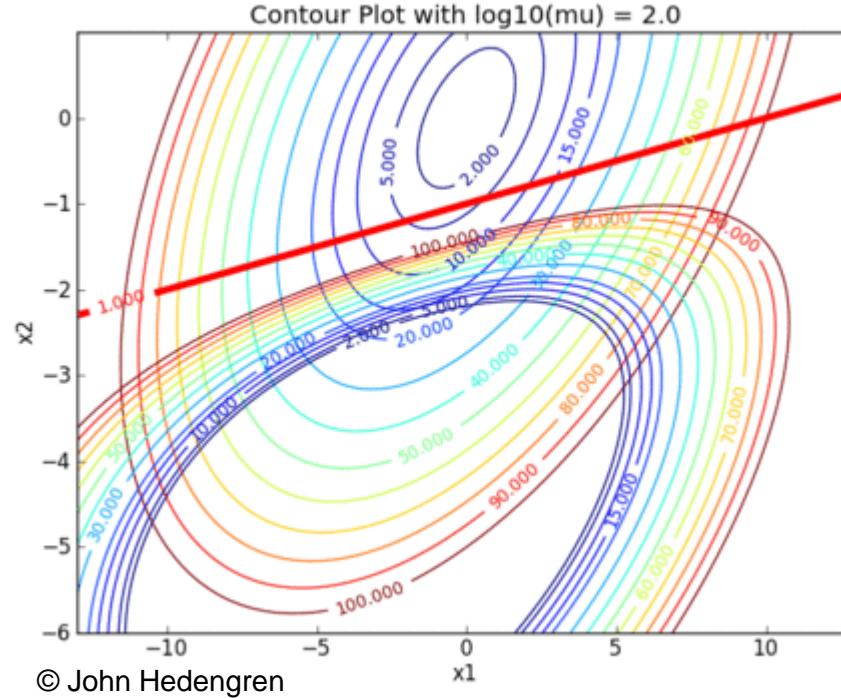


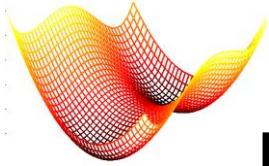
Barriere-Methode (3)

- **Iteratives Vorgehen:** große μ stellen sicher, dass wir innerhalb S bleiben, starte daher mit großem μ
- dann: **verkleinere** μ sukzessive, so dass die Lösung von $p(\mathbf{x}, \mu)$ „von innen“ gegen das gesuchte \mathbf{x}^* konvergiert.
- In jeder Iteration kann Optimum von $p(\mathbf{x}, \mu)$ mit einem geeigneten nichtrestringierten Optimierungs-Verfahren ermittelt werden
- **Problem:** in der Praxis oft schlechte Konditionierung bei kleinen $\mu \rightarrow$ Verfahren ist so nicht einsetzbar!

Barriere-Methode (4)

- Veranschaulichung schematisches Vorgehen bei Barriere-Methode:





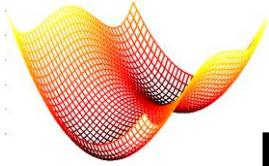
Barriere-Methode (5)

- **Ansatz** zur Verbesserung der Konvergenz: Fasse den Gewichtungsfaktor μ als Teil eines erweiterten Problems auf, der ebenfalls optimiert wird (analog Lagrange)
- Wähle $b(t) = -\ln(t)$. Dann ist Zielfunktion wie folgt:

$$p(\mathbf{x}, \mu) = f(\mathbf{x}) - \mu \cdot \sum_{m=1}^M \ln(-g_m(\mathbf{x}))$$

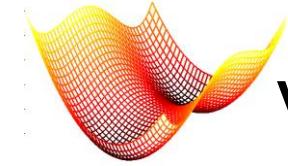
- Notwendige Optimalitätsbedingung: $\nabla_{\mathbf{x}} p(\mathbf{x}, \mu) = \mathbf{0}$, d.h.
$$\nabla_{\mathbf{x}} p(\mathbf{x}, \mu) = \nabla_{\mathbf{x}} f(\mathbf{x}) - \mu \cdot \sum_{m=1}^M \frac{1}{g_m(\mathbf{x})} \cdot \nabla g_m(\mathbf{x}) := \mathbf{0}$$
- Einführen einer **dualen Variable** $\boldsymbol{\lambda} \in \mathbb{R}^M$ mit $g_m(\mathbf{x}) \cdot \lambda_m = \mu$:

$$d(\mathbf{x}, \boldsymbol{\lambda}) := \nabla_{\mathbf{x}} p(\mathbf{x}, \boldsymbol{\lambda}) = \nabla_{\mathbf{x}} f(\mathbf{x}) - \sum_{m=1}^M \lambda_m \cdot \nabla g_m(\mathbf{x}) = \mathbf{0}$$

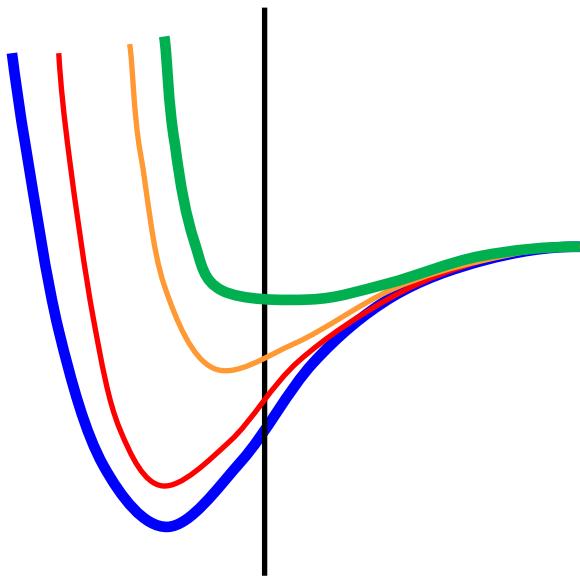


Barriere-Methode (6)

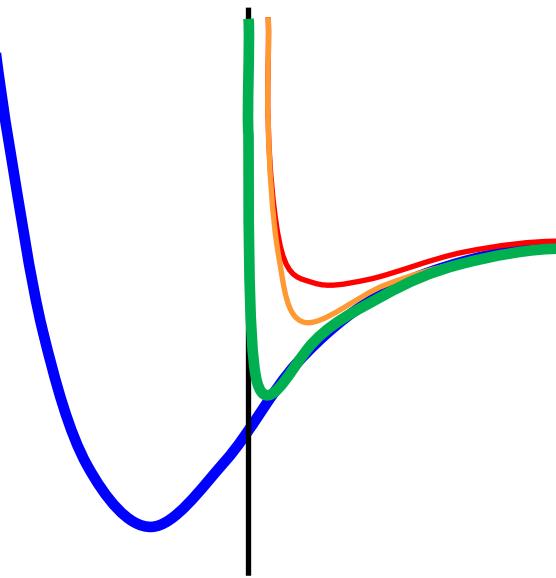
- **Lösungsansatz:** wende **Newton-Methode** an:
 - Linearisiere $d(\mathbf{x}, \boldsymbol{\lambda})$ an der Stelle $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$
 - Finde $(\mathbf{x}_{k+1}, \boldsymbol{\lambda}_{k+1})$ durch Nullstellen-Suche bei der linearisierten Funktion $d(\mathbf{x}, \boldsymbol{\lambda})$
- Linearisierung:
$$d(\mathbf{x}_k + \Delta\mathbf{x}, \boldsymbol{\lambda}_k + \Delta\boldsymbol{\lambda}) \approx d(\mathbf{x}_k, \boldsymbol{\lambda}_k) + \nabla_{\mathbf{x}}d(\mathbf{x}_k, \boldsymbol{\lambda}_k) \cdot \Delta\mathbf{x} + \nabla_{\boldsymbol{\lambda}}d(\mathbf{x}_k, \boldsymbol{\lambda}_k) \cdot \Delta\boldsymbol{\lambda}$$
- Nullstellen-Suche: $d(\mathbf{x}_k + \Delta\mathbf{x}, \boldsymbol{\lambda}_k + \Delta\boldsymbol{\lambda}) := 0$, d.h. Ermittlung von $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda})$
- Update der Lösung: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}$, $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \Delta\boldsymbol{\lambda}$
- Liefert aber nur N Gleichungen für $N + M$ Unbekannte → Lösung mittels weiterer Linearisierung der $g_m(\mathbf{x}) \cdot \lambda_m = \mu$



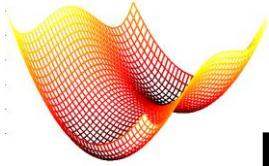
Vergleich: Penalty- und Barriere-Methode



Penalty-Methode



Barriere-Methode



Lagrange-Newton-Iteration (1)

- Betrachte gleichungsrestriktionsbeschränktes Programm (PGR)

$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

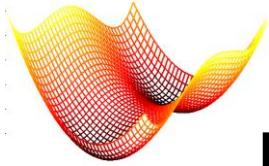
$$\text{NB: } \mathbf{h}(\mathbf{x}) = 0$$

- **Grundidee: Lösung lässt sich direkt aus den KKT-Bedingungen ableiten**
- Lagrange-Funktion ist hier: $L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot h_l(\mathbf{x})$
- Die **KKT-Bedingungen** sind dann:

$$\nabla f(\mathbf{x}^*) + \sum_{l=1}^L \lambda_l \cdot \nabla h_l(\mathbf{x}^*) = \mathbf{0}$$

$$\mathbf{h}(\mathbf{x}) = \mathbf{0}$$

(keine Komplementarität)

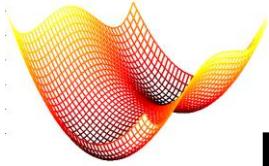


Lagrange-Newton-Iteration (2)

- **Grundidee:** Fasse die KKT-Bedingungen als (i.A. nichtlineares) Gleichungssystem auf und löse dieses:

$$\nabla L(\mathbf{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla_{\mathbf{x}} f(\mathbf{x}) + \sum_{l=1}^L \lambda_l \cdot \nabla_{\mathbf{x}} h_l(\mathbf{x}) \\ \mathbf{h}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

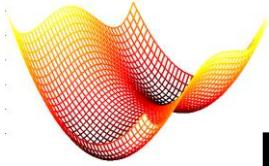
- Gradientenbildung von L bezüglich \mathbf{x} und $\boldsymbol{\lambda}$
- **Lösung** des (nichtlinearen) GS mittels **Newton-Methode**:
 1. Linearisiere $\nabla L(\mathbf{x}, \boldsymbol{\lambda})$ an aktueller Position $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$:
$$\nabla L(\mathbf{x}_k + \Delta \mathbf{x}, \boldsymbol{\lambda}_k + \Delta \boldsymbol{\lambda}) \approx \nabla L(\mathbf{x}_k, \boldsymbol{\lambda}_k) + \nabla^2 L(\mathbf{x}_k, \boldsymbol{\lambda}_k) \cdot (\Delta \mathbf{x}, \Delta \boldsymbol{\lambda})^T$$
Damit wird $\nabla L(\mathbf{x}_k + \Delta \mathbf{x}, \boldsymbol{\lambda}_k + \Delta \boldsymbol{\lambda}) = \mathbf{0}$ ein **lineares** GS
 2. Löse dieses lin. Gleichungssystem → ermittle $(\Delta \mathbf{x}, \Delta \boldsymbol{\lambda})$
 3. Update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}; \boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \Delta \boldsymbol{\lambda}$



Lagrange-Newton-Iteration (3)

- $\nabla L(\mathbf{x}_k, \boldsymbol{\lambda}_k) + \nabla^2 L(\mathbf{x}_k, \boldsymbol{\lambda}_k) \cdot (\Delta \mathbf{x}, \Delta \boldsymbol{\lambda})^T := \mathbf{0}$ ergibt:
$$\begin{bmatrix} \mathbf{Q}(\mathbf{x}_k) & \mathbf{J}(\mathbf{h}(\mathbf{x}_k))^T \\ \mathbf{J}(\mathbf{h}(\mathbf{x}_k)) & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \Delta \mathbf{x} \\ \boldsymbol{\lambda}_k + \Delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla f(\mathbf{x}_k) \\ \mathbf{h}(\mathbf{x}_k) \end{bmatrix}$$
- $\mathbf{J}(\mathbf{h}(\mathbf{x}_k))$: Jacobi-Matrix der Gleichungs-NB
- $\mathbf{Q}(\mathbf{x}_k, \boldsymbol{\lambda}_k)$: Matrix der zweiten Ableitungen von $L(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ nach \mathbf{x} , d.h. $\mathbf{Q}(\mathbf{x}_k, \boldsymbol{\lambda}_k) = \nabla^2 f(\mathbf{x}_k) + \sum_{l=1}^L \lambda_{l,k} \cdot \nabla^2 h_l(\mathbf{x}_k)$
- Dies ist ein lineares System von $N + M$ Gleichungen
- Lösung des lin. GS liefert Updates \mathbf{x}_{k+1} und $\boldsymbol{\lambda}_{k+1}$
- Konvergenz ist erreicht, wenn $\nabla L(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ sehr klein werden

und $\mathbf{h}(\mathbf{x}) \approx \mathbf{0}$, d.h. $d := \left\| \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}(\mathbf{h}(\mathbf{x}_k))^T \cdot \boldsymbol{\lambda}_k \\ \mathbf{h}(\mathbf{x}_k) \end{bmatrix} \right\| < \epsilon$



Lagrange-Newton-Iteration (4)

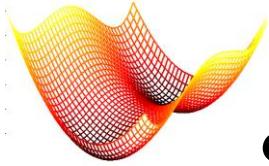
- Pseudo-Code Lagrange-Newton-Iteration:

```
function optimizeLagrangeNewton (in  $f(\mathbf{x})$ ,  $\mathbf{h}(\mathbf{x})$ , in  
 $(\mathbf{x}_0, \boldsymbol{\lambda}_0)$ , in  $\epsilon$ , out solution  $\mathbf{x}^*$ )
```

```
repeat
```

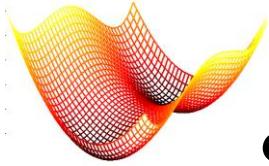
- 1.) calculate $\nabla f(\mathbf{x}_k)$, $\mathbf{J}(\mathbf{h}(\mathbf{x}_k))$, $\mathbf{Q}(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ and d
 - 2.) **if** $d < \epsilon$ **exit**
 - 3.) solve linear GS yielding $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda})$
 - 4.) update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}$; $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \Delta\boldsymbol{\lambda}$
- $$k \leftarrow k + 1$$

- Im Prinzip ist Lagrange-Newton-Iteration eine Erweiterung des Newton-Verfahrens auf gleichungsrestring. Probleme
→ Quadratische Konvergenz, aber kleiner Konverg.-Bereich



SQP-Verfahren (1)

- **Beobachtung:** Lösung des lin. GS in jeder Iteration des Lagrange-Newton-Verfahrens ist äquivalent zur Lösung eines quadratischen Programms mit linearen Gleichungen als NB (so wie Newton-Methode äquivalent zur Lösung eines nichtrestr. Quadratischen Programms ist)
- In diesem Fall (PGR):
 - Minim. $q(\Delta \mathbf{x}) = \frac{1}{2} \Delta \mathbf{x}^T \mathbf{Q}(\mathbf{x}_k, \boldsymbol{\lambda}_k) \Delta \mathbf{x} + \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x} + \nabla f(\mathbf{x}_k)$
 - NB: $\mathbf{J}(\mathbf{h}(\mathbf{x}_k)) \Delta \mathbf{x} + \mathbf{h}(\mathbf{x}_k) = \mathbf{0}$
- Lagrange-Newton-It. ist daher äquivalent zu iterativem Lösen einer Folge quadratischer (Hilfs-) Programme
→ „**Sequential Quadratic Programming**“ (SQP)



SQP-Verfahren (2)

- **Verallgemeinerung** auf allg. restriktives Programm (PR):

$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

NB: $\mathbf{g}(\mathbf{x}) \leq 0, \mathbf{h}(\mathbf{x}) = 0$

- **Quadratische Hilfsprogr.** (PQ) sind wie folgt definiert:

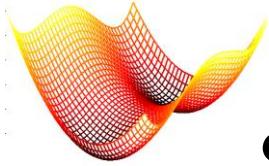
$$\min_{\Delta \mathbf{x}} q(\Delta \mathbf{x}) = \frac{1}{2} \Delta \mathbf{x}^T \mathbf{Q}(\mathbf{x}_k, \boldsymbol{\mu}_k, \boldsymbol{\lambda}_k) \Delta \mathbf{x} + \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x} + \nabla f(\mathbf{x}_k)$$

NB: $\mathbf{J}(\mathbf{g}(\mathbf{x}_k)) \Delta \mathbf{x} + \mathbf{g}(\mathbf{x}_k) \leq \mathbf{0}$

$\mathbf{J}(\mathbf{h}(\mathbf{x}_k)) \Delta \mathbf{x} + \mathbf{h}(\mathbf{x}_k) = \mathbf{0}$

- **Zusammenhang** zwischen PQ und PR:

- Approximation von $f(\mathbf{x})$ mit **quadratischer Form** $q(\Delta \mathbf{x})$
- Approx. von $\mathbf{g}(\mathbf{x}), \mathbf{h}(\mathbf{x})$ durch **Linearisierung** an \mathbf{x}_k

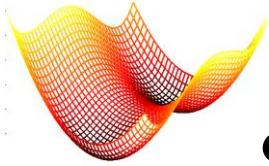


SQP-Verfahren (3)

- $\mathbf{Q}(\mathbf{x}_k, \boldsymbol{\mu}_k, \boldsymbol{\lambda}_k)$: Matrix der 2. Ableitungen von $L(\mathbf{x}_k, \boldsymbol{\mu}_k, \boldsymbol{\lambda}_k)$ nach \mathbf{x} , wobei jetzt auch Ungl. $\mathbf{g}(\mathbf{x}_k)$ in L enthalten sind
 $\rightarrow \mathbf{Q}(\mathbf{x}_k, \boldsymbol{\mu}_k, \boldsymbol{\lambda}_k) = \nabla^2 f(\mathbf{x}_k) + \sum_{m=1}^M \mu_{m,k} \cdot \nabla^2 g_m(\mathbf{x}_k) + \sum_{l=1}^L \lambda_{l,k} \cdot \nabla^2 h_l(\mathbf{x}_k)$
- **Grundidee:** Approx. durch Serie von einfacher zu lösenden Hilfsprogrammen (quadratische Zielfkt. und lineare NB)
- **Maß für die Konvergenz:** Betrag eines Vektors d , welcher aus den Abl. von L nach \mathbf{x} sowie der Gl.-NB besteht:

$$d := \left\| \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}(\mathbf{g}(\mathbf{x}_k))^T \cdot \boldsymbol{\mu}_k + \mathbf{J}(\mathbf{h}(\mathbf{x}_k))^T \cdot \boldsymbol{\lambda}_k \\ \mathbf{h}(\mathbf{x}_k) \end{bmatrix} \right\|$$

- **Active Set:** Es wird in der Regel nur ein Teil der Ungleichungs-NB aktiv sein, d.h. einige $\mu_{m,k} = 0$



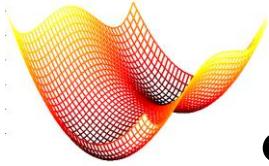
SQP-Verfahren (4)

- Pseudo-Code SQP für allg. (PR):

```
function optimizeSQP (in  $f(\mathbf{x})$ ,  $\mathbf{g}(\mathbf{x})$ ,  $\mathbf{h}(\mathbf{x})$ , in  
 $(\mathbf{x}_0, \boldsymbol{\mu}_0, \boldsymbol{\lambda}_0)$ , in  $\epsilon$ , out solution  $\mathbf{x}^*$ )
```

repeat

- 1.) calculate $\nabla f(\mathbf{x}_k)$, $\mathbf{J}(\mathbf{h}(\mathbf{x}_k))$, $\mathbf{J}(\mathbf{g}(\mathbf{x}_k))$ and d
 - 2.) **if** $d < \epsilon$ **exit**
 - 3.) calculate $\mathbf{Q}(\mathbf{x}_k, \boldsymbol{\mu}_k, \boldsymbol{\lambda}_k)$
 - 4.) linearize $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ at \mathbf{x}_k
 - 4.) solve quadratic program (QP) $\rightarrow (\Delta \mathbf{x}, \Delta \boldsymbol{\mu}, \Delta \boldsymbol{\lambda})$
 - 5.) update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}$; $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \Delta \boldsymbol{\lambda}$;
 $\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \Delta \boldsymbol{\mu}$
- $$k \leftarrow k + 1$$



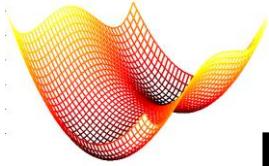
SQP-Verfahren (5)

- **Hilfsprogramme** (QP) können z.B. mit **Innere-Punkte-Methoden** effizient gelöst werden
- **Eigenschaften:** wie bei Newton-Verfahren quadratische Konvergenz, aber u.U. nur kleiner Konvergenzbereich. Außerdem wird in der Regel keine Folge von zulässigen Punkten erzeugt
 - Maßnahmen zur **Vergrößerung des Konvergenzbereichs**, wie z.B. Schrittweiten-Steuerung bzw. um Zulässigkeit zu garantieren
- SQP zählt aktuell zu den effizientesten und beliebtesten Sovern für allg. restringierte Optimierungsaufgaben
- Wird auch in MATLAB bei **fmincon** eingesetzt



Diskrete Optimierung

- **Arten von diskreten Problemen**
- **Relaxation**
- **Erschöpfende Suche**
- **Zuordnungsprobleme**
- **Heuristik: Search Tree**
- **RANdom SAmple Consensus (RANSAC)**
- **„Branch and Bound“-Verfahren**
- **Heuristik: Vogel-Approximation**
- **Reihenfolgeplanung: Johnson-Algorithmus**



Mathematische Formulierung

- Allgemeine Formulierung eines diskreten Programms:

$$\min_{\mathbf{x}, \mathbf{y}, \mathbf{z}} f(\mathbf{x}, \mathbf{y}, \mathbf{z})$$

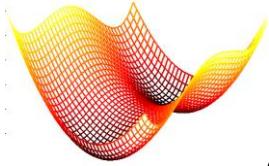
Nebenbedingungen:

$$g(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq 0$$

$$h(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0$$

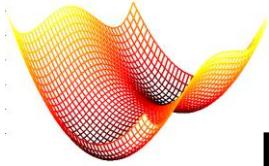
$$\mathbf{x} \in \mathbb{Z}^M; \mathbf{y} \in \mathbb{R}^N; \mathbf{z} \in A$$

- Verschiedene Wertebereiche für Zielvariablen $\mathbf{x}, \mathbf{y}, \mathbf{z}$:
 - $\mathbf{x} \in \mathbb{Z}^M$: nur Integer-Werte zulässig
 - $\mathbf{y} \in \mathbb{R}^N$ (reelle Zahlen): auch Gleitkomma-Zahlen möglich
 - $\mathbf{z} \in A$: A bezeichnet (endliche) Menge beliebiger Elemente



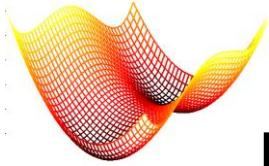
Arten von diskreten Programmen

- **Integer Programme:** nur x existiert, d.h. alle Zielvariablen können ausschließlich (diskrete) Integer-Werte annehmen
- **Gemischte Programme:** x und y existieren, d.h. einige Zielvariablen können Gleitkomma-Werte annehmen
- **Diskrete „nicht-Integer-Programme“:** nur z existiert, wobei A aus einer endlichen Menge von diskreten Zahlen besteht
- **Binäre Programme:** nur x existiert; jede Zielvariable ist zudem binär, d.h. kann nur zwei Werte 0 und 1 annehmen
- **Kombinatorische Programme:** nur z existiert, wobei die Elemente von A je eine Kombination/Zuordnung definieren (z.B. Zuordnung detektiertes Objekt zu passendem Modell)



Probleme bei diskreten Programmen

- „merkwürdige“ **Erkenntnis**: das Vorhandensein von diskreten Variablen **x** und/oder **z** **erschwert** die Lösungsfindung erheblich. Gründe u.a.:
 - **Iterierte** der bisherigen Lösungsverfahren sind **in der Regel nicht gültig**, d.h. außerhalb des Lösungsraumes
 - **Weniger Info**: Zielfunktion ist nur an den diskreten Stützstellen bekannt
 - Nichtlin. Opt.: Ableitungen lassen sich nur ungenau numerisch berechnen (i.d.R. zu großer Abstand zwischen den Stützstellen)
 - Lin. Opt.: Polyeder des Lösungsraumes nicht genau bekannt („Schnittpunkte“ der NB i.d.R. nicht gültig)

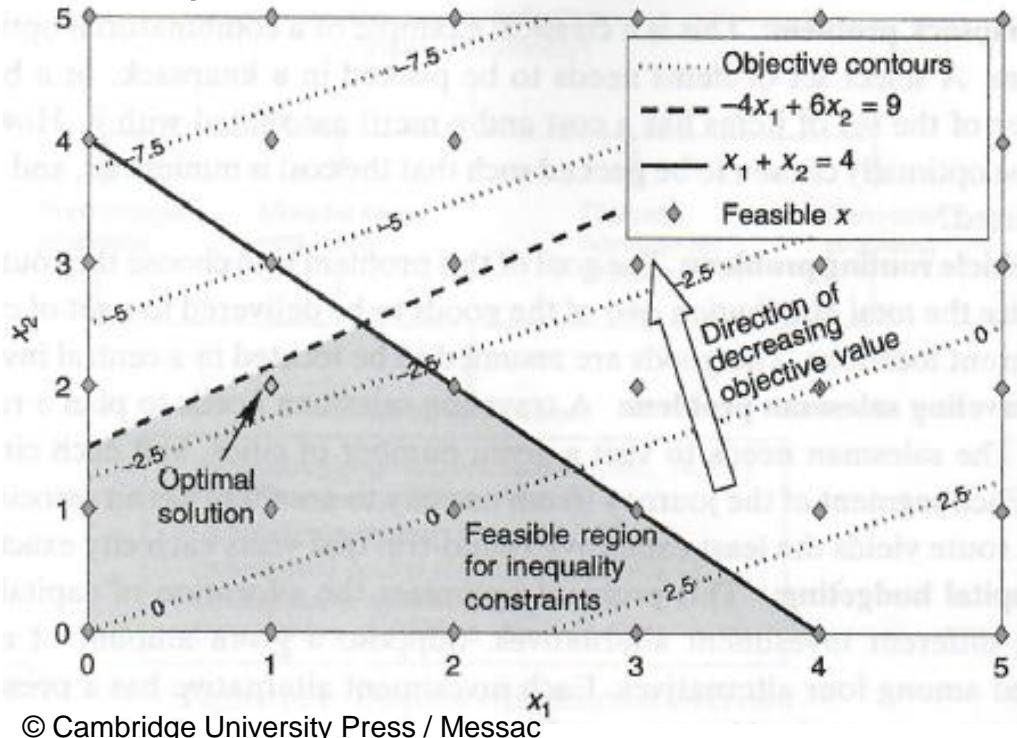


Relaxation (1)

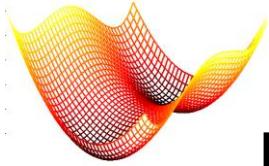
- Allg. Definition **Relaxations-Methode**: Finde Näherungslösung durch Umwandeln in ein leichter zu lösendes Problem
- Bei Integer-Programmen: **behandle die x , als ob sie auch Float-Werte annehmen dürften**, d.h. lasse $x' \in \mathbb{R}^M$ zu.
- Lösung in zwei Schritten:
 - Löse das Programm mit $[x', y] \in \mathbb{R}^{M+N}$ mit Standard-Verfahren (z.B. Newton-Methode)
 - Die Lösung x^* ergibt sich, indem die $x^{*,i}$ auf Integer-Werte gerundet werden: $x_i^* = \text{round}(x_i^{*,i})$
- **Problem: Lösung ist u.U. nicht gültig**, weil NB verletzt (keine Kontrolle der Einhaltung der NB beim Runden!)

Relaxation (2)

- Beispiel für Probleme beim Runden

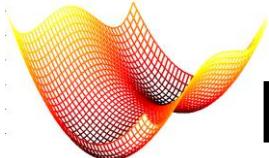


© Cambridge University Press / Messac



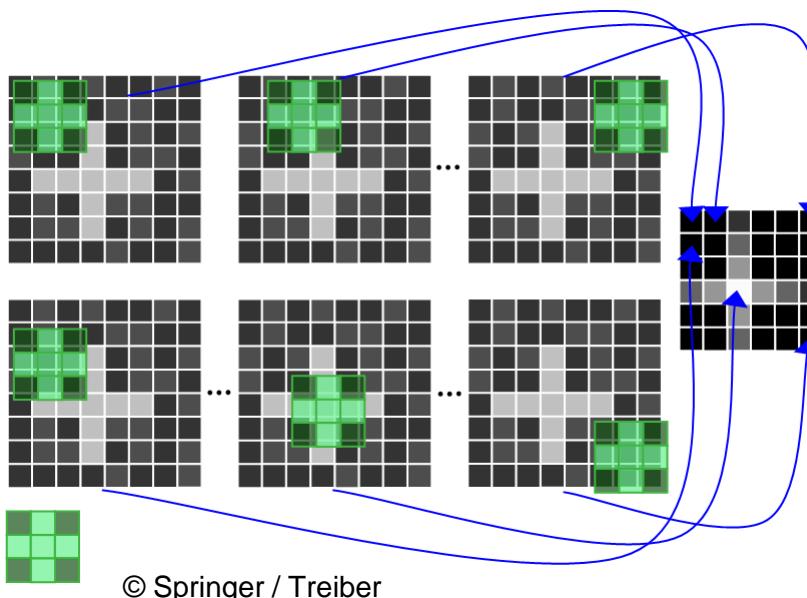
Erschöpfende Suche

- **Vorteil** bei Lösungsmengen endlicher Mächtigkeit (wenn alle Zielvariablen diskret sind): man **kann alle Lösungskandidaten überprüfen**
- **Erschöpfende Suche („brute force“): Berechne Zielfunktionswert für alle möglichen (diskreten) Lösungen.** Optimum ergibt sich aus dem Vergleich dieser Zielfunktionswerte
- Vorteil: **globales Optimum wird garantiert gefunden**
- Nachteil: in der Regel **nur für kleine M praktikabel** (sonst zu viele Berechnungen der Zielfunktion notwendig): Anzahl der Zielfunktions-Berechnungen ist **Produkt** der Kandidatenanzahl aller Zielvariablen!



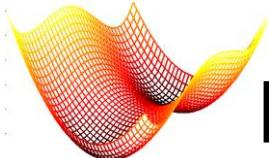
Erschöpfende Suche: Beispiel Template Matching (1)

- **Aufgabe:** finde alle Objekte in einem Bild anhand eines gelernten Musters (Templates)
- Erschöpfende Suche: „**Abrastern**“ aller pixel des Bildes



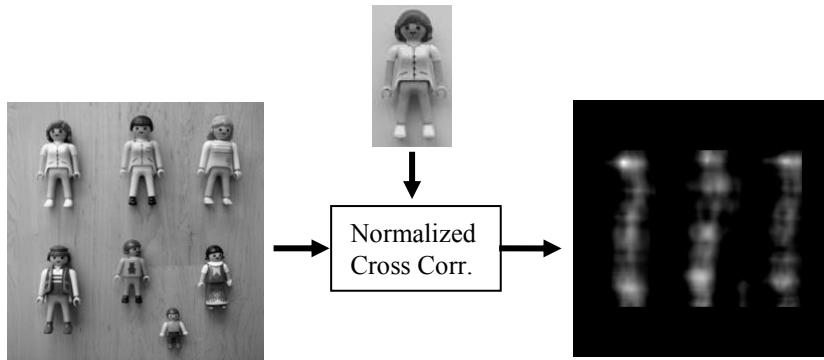
- Bei jedem Pixel $[a, b]$: Berechne Kreuzkorrelations-Koeffizient $\rho(a, b)$ zwischen Template I_T und korrespondierendem Bildbereich I_S

$$\rho(a, b) = \frac{\sum_{x=0}^W \sum_{y=0}^H (I_S(x+a, y+b) - \bar{I}_S) \cdot (I_T(x, y) - \bar{I}_T)}{\sqrt{\sum_{x=0}^W \sum_{y=0}^H (I_S(x+a, y+b) - \bar{I}_S)^2 \cdot \sum_{x=0}^W \sum_{y=0}^H (I_T(x, y) - \bar{I}_T)^2}}$$

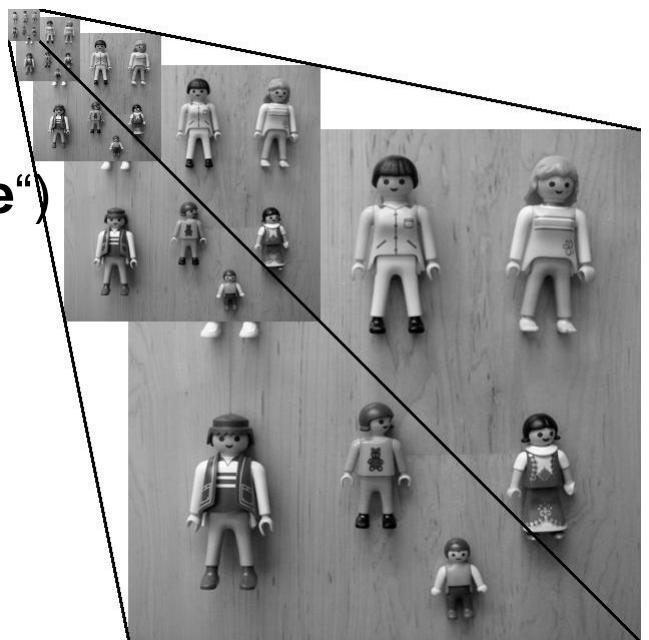


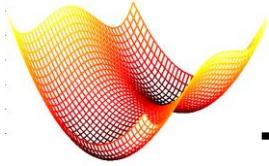
Erschöpfende Suche: Beispiel Template Matching (2)

- Gut durchführbar 2-dim. Lösungsraum (x,y-Position), kritisch bei 4 Dim. (x,y,phi,scale), nicht durchführbar bei Berücksichtigung der Perspektive
- Verbesserung: „coarse-to-fine“: erschöpfende Suche nur bei niedriger Auflösung („**Bildpyramide**“)



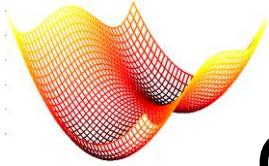
© Springer / Treiber





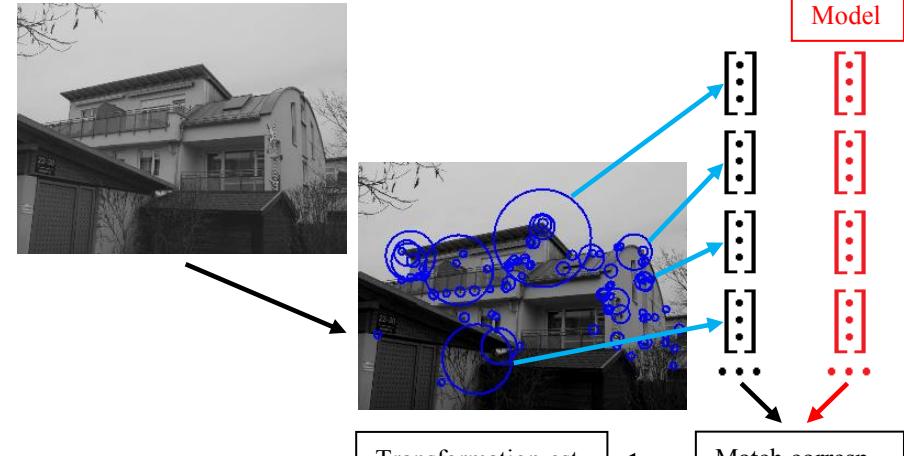
Zuordnungsprobleme (1)

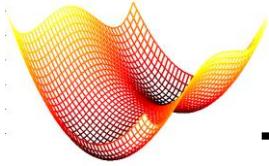
- **Aufgabe:** Finde die optimale Zuordnung ϕ der Elemente p_i einer Menge P zu den Elementen q_j der Menge Q , d.h. finde Funktion $\phi: i = \phi(j)$
- **Oft wird 1:1-Zuordnung gesucht**, d.h. jedes Element von P wird genau einem Element von Q zugeordnet. (s. Beispiel: Zuordnung Modell-Deskriptoren zu den im Bild gefundenen)
- Wenn die Anzahl der Elemente von P und Q unterschiedlich: einzelne Elemente können unberücksichtigt bleiben.
- Beispiel für **kombinatorisches Programm** (nur z existiert)
→ Lösungsmenge ist endlich, jedoch „**combinatorial explosion**“: Anzahl d. Lösungen wächst exponentiell: wenn $\|P\| = \|Q\| = K$, ist Lösungs-Anzahl $\|A\| = K!$ (Fakultät)



Objekterkennung mittels Deskriptoren

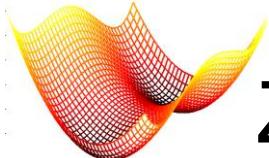
- Umgehen der erschöpfenden Suche bei Objekterkennung
1. Finde markante Stellen im Bild (sog. „interest points“ IP)
 2. Beschreibe Umgebung eines jeden IP's durch einen sog. Deskriptor
 3. Ordne den gefundenen Deskriptoren Modell-Deskr. zu
 4. Bestimme Lage aus dieser Zuordnung





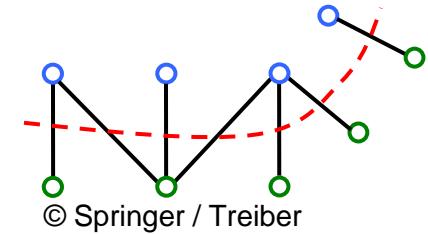
Zuordnungsprobleme: Beispiele

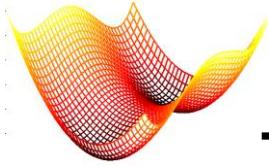
- **Objekterkennung** mittels Deskriptoren
- Partnerbörse („**Heiratsproblem**“): Mengen P und Q sind Frauen bzw. Männer. Aufgabe: Identifiziere möglichst viele Paare, so dass Summe der „Sympathie“ möglichst hoch
- **Produktionsplanung**: verteile Aufträge auf Mitarbeiter bzw. Produktionsmittel (Maschinen) möglichst „effizient“ (z.B. hinsichtlich Kosten, Umsatz, etc.)
- **Dienstplanung** Nahverkehr: verteile verfügbare U-Bahnen und/oder Fahrer möglichst „optimal“ auf die vorhandenen Linien (z.B. maximiere Anzahl der Fahrgäste, etc.)
- **Mobilfunk**: gesucht ist eine bestmögliche Zuteilung der zur Verfügung stehenden Frequenzen auf Basis-Antennen



Zuordnungsprobleme: mathematische Modellierung

- Zwei Arten der Modellierung:
- **Bipartiter Graph:**
 - Jedes Element von P und Q wird durch einen Knoten des Graphen modelliert
 - Verbindungen („Kanten“) zwischen zwei Knoten nur zulässig, wenn ein Knoten $\in P$, der andere $\in Q$
- **Adjazenzmatrizen A:** Zeilen $a_{p,*}$ entsprechen den Elementen von P , Spalten $a_{*,q}$ den Elementen von Q . Jedes Element $a_{p,q}$ definiert die Kosten einer Zuordnung
- Jede Kante des bipartiten Graphen entspricht einem Element von A



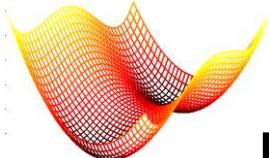


Zuordnungsprobleme: Heutistik

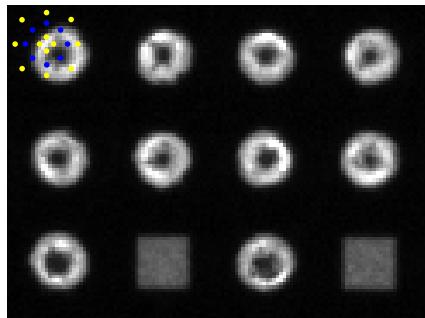
- Erschöpfende Suche selbst für kleine Kardinalitäten K nicht anwendbar (Anzahl der Lösungsmöglichkeiten $\|A\| = K!$):

K	3	5	10	20	50	100
$K!$	6	120	$3.6 \cdot 10^6$	$2.4 \cdot 10^{18}$	$3.0 \cdot 10^{64}$	$9.3 \cdot 10^{157}$

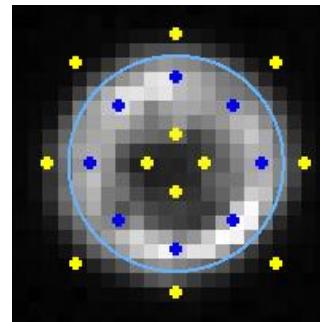
- **Beobachtung:** i.d.R. führen konkrete Zuordnungen einzelner Elemente dazu, dass bestimmte Zuordnungen anderer Elemente extrem unwahrscheinlich sind
→ **Idee:** Anwenden einer Heuristik zur Reduktion der „sinnvollen“ Lösungsmöglichkeiten
- Beispiel: Bestimme Lage eines Objektes mittels „**Search Tree**“



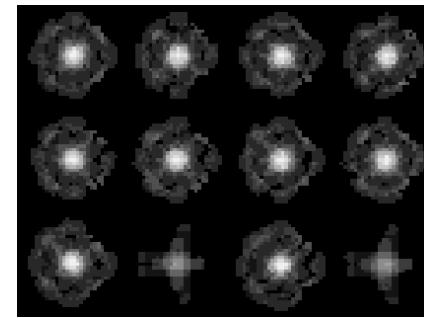
Heuristik Search Tree (1)



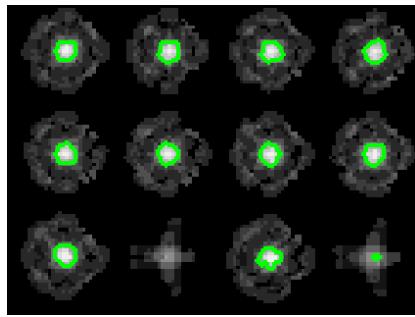
BGA Component



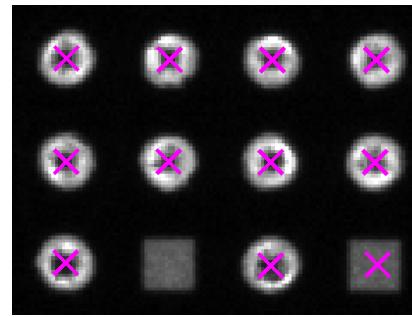
Filter for Balls



Matching Function

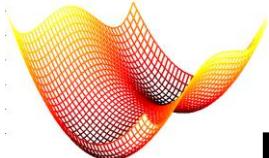


Labeled Matching



Found Features

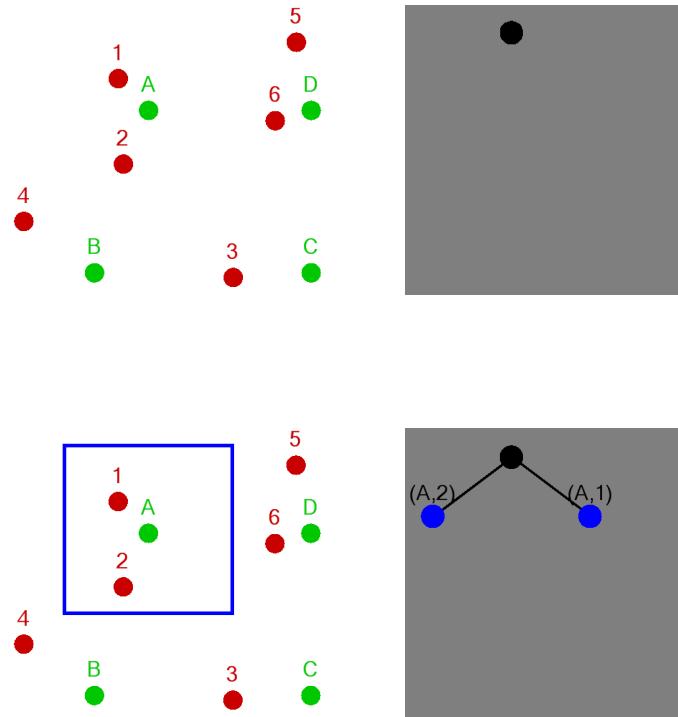
© Springer / Treiber



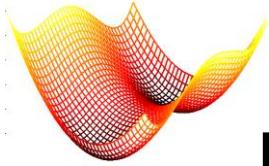
Heuristik Search Tree (2)

A-D: model
features

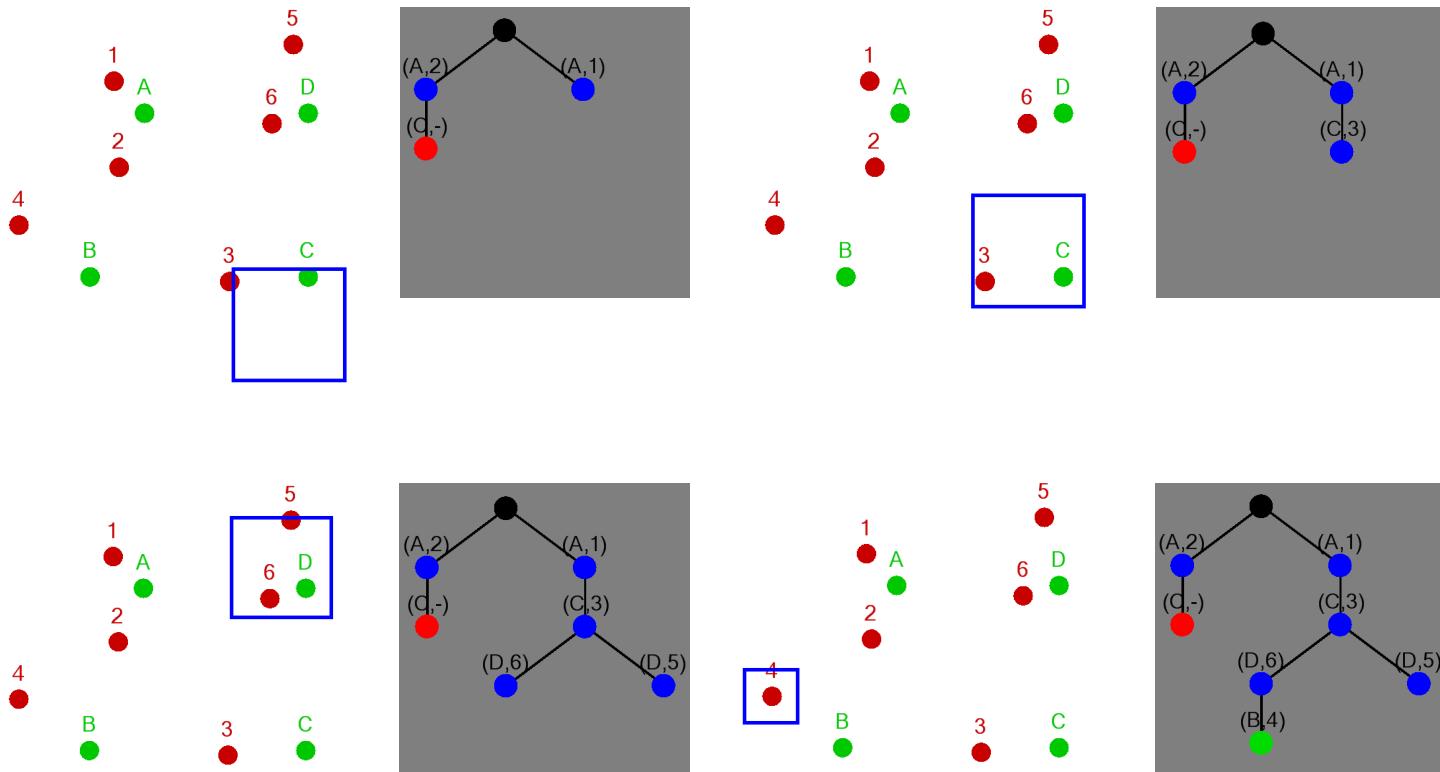
1-6: features
found in
query image



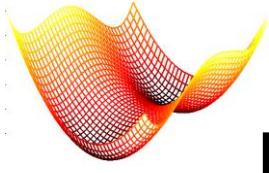
© Springer / Treiber



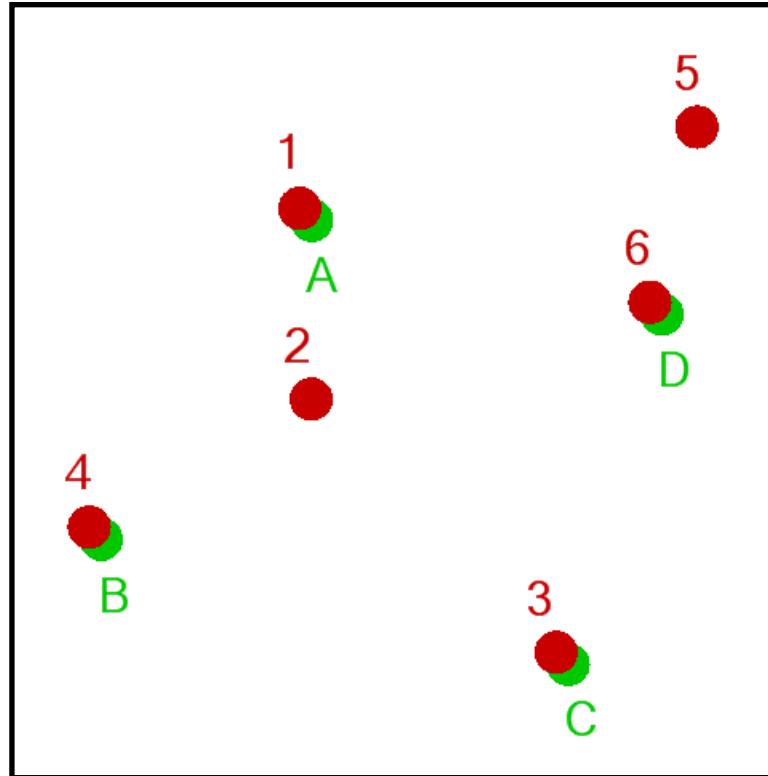
Heuristik Search Tree (3)



© Springer / Treiber



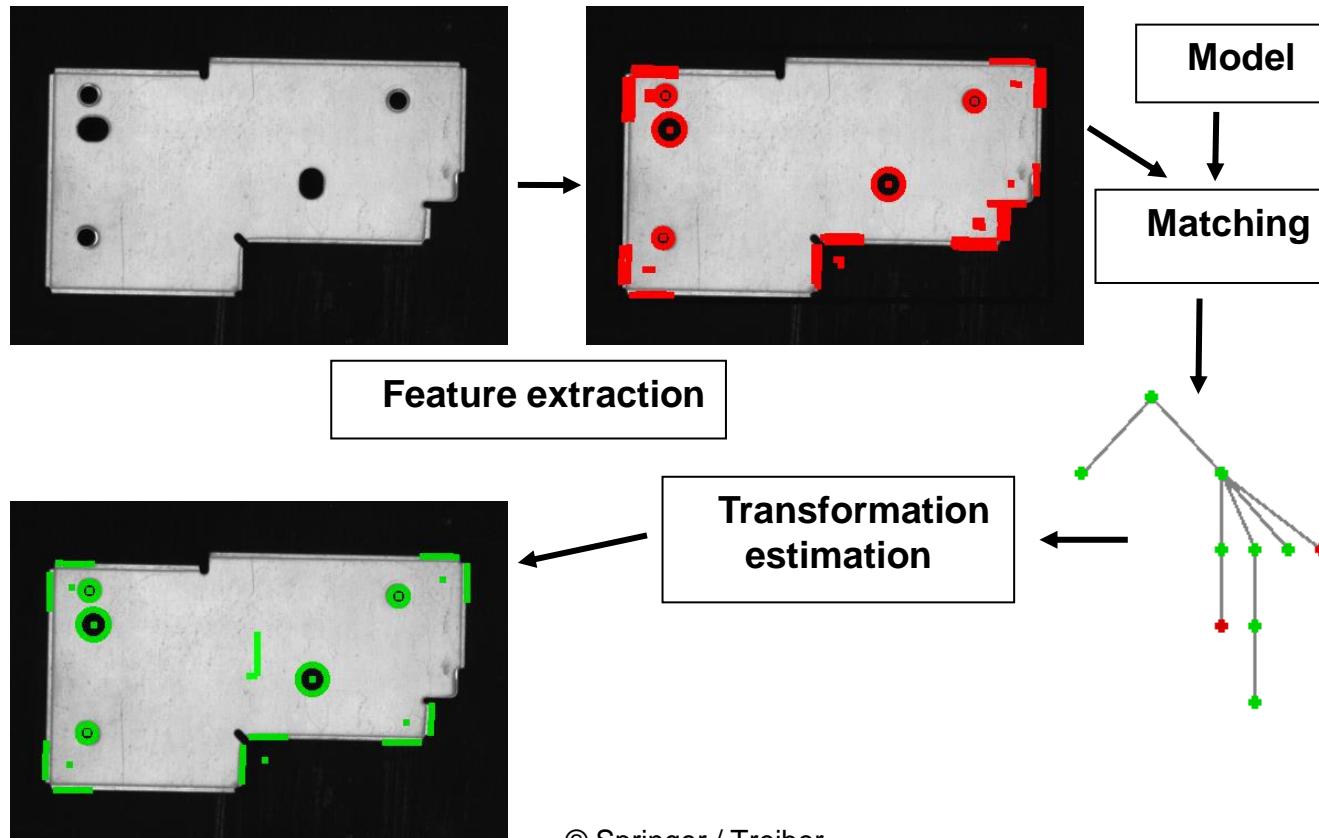
Heuristik Search Tree (3)



© Springer / Treiber



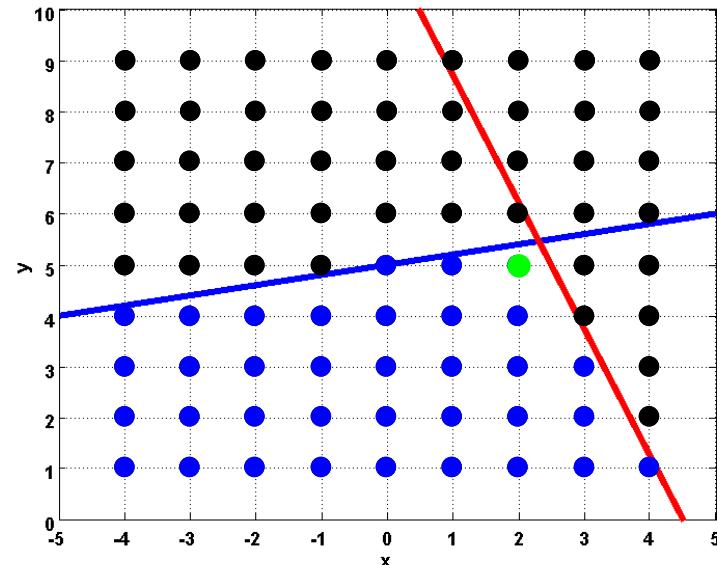
Heuristik Search Tree (4)

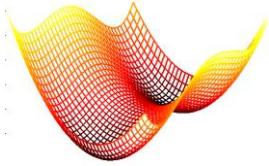


© Springer / Treiber

„Branch and Bound“-Methode (1)

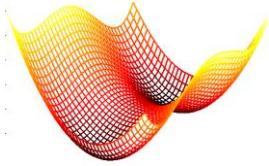
- Betrachte lineares Integer-Programm $\min f_I(\mathbf{x}) = \min_{\mathbf{x} \in \mathbb{Z}^N} \mathbf{c}^T \cdot \mathbf{x}$ mit den NB $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$; $\mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}$; $\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$
- **Problem: Simplex-Algorithmus so nicht anwendbar**, da die „Ecken“ des Simplex nicht genau bekannt (Problem: Schnittpunkte der „NB-Gleichungen“ sind i.d.R. $\notin \mathbb{Z}^N$)





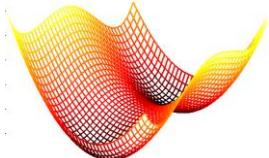
„Branch and Bound“-Methode (2)

- **Vorgehensweise:** „Branch and Bound“:
- **löse zunächst relaxiertes Integer-Programm** mit Standard-Verfahren (z.B. Simplex-Methode)
- **Betrachte dessen Lösung \mathbf{x}_r^***
 - $\mathbf{x}_r^* \in \mathbb{Z}^N$: gültige Lösung gefunden
 - $\mathbf{x}_r^* \notin \mathbb{Z}^N$: betrachte eine (bel.) **Zielvariable** $x_i^* \notin \mathbb{Z}$ der **Lösung**: Formuliere 2 neue Progr. (P1), (P2) (**branch**)
- Jedes dieser neuen Programme erhält **eine zusätzliche Ungleichung als NB** (damit erhält x_i eine Integer-Lsg.):
 - Für (P1): $x_i \leq \text{floor}(x_i^*)$
 - Für (P2): $x_i \geq \text{ceil}(x_i^*)$ (**bound**)

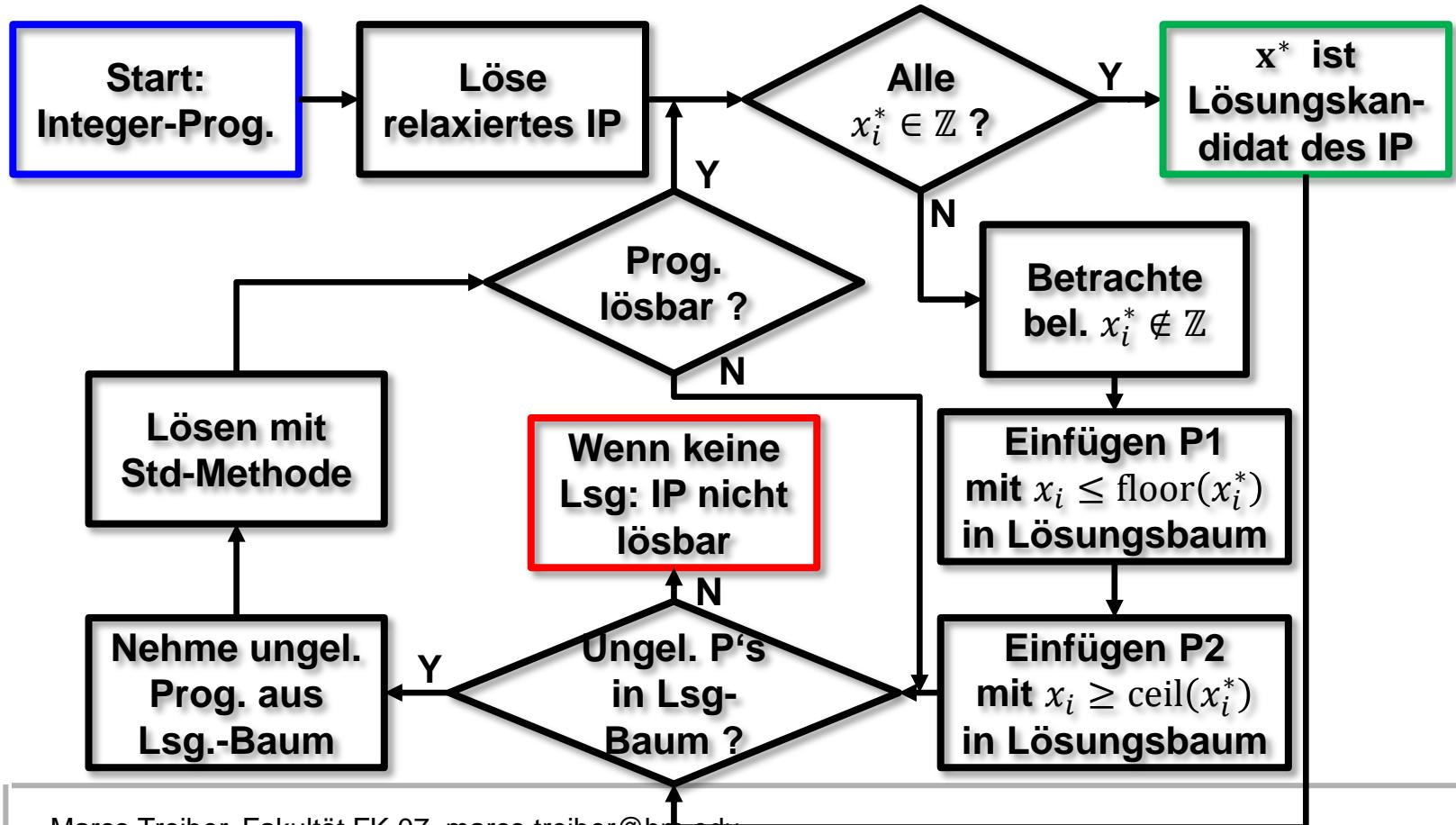


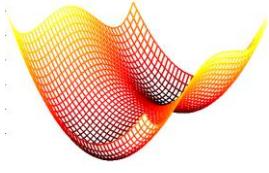
„Branch and Bound“-Methode (3)

- ergibt „**Lösungs-Baum**“ (da jeweils zwei mögliche Lsg.)
- Dann: **Rekursive Lösung dieser neuen Probleme**, bis
 - $x_r^* \in \mathbb{Z}^N$: (neue) gültige Lösung des ursprünglichen IP gefunden
 - Programm aufgrund der hinzugefügten NB mehr nicht lösbar
- Analyse des **kompletten Baumes** für globales Opt. nötig
- Idee zur **Beschleunigung**: Wenn bereits (mindestens) eine Integer-Lösung bekannt, braucht ein Zweig, dessen Optimallsg. einen höheren Wert als die bereits bekannte Integer-Lösung aufweist, nicht weiter analysiert zu werden. Grund: durch neue NB kann Lösungswert nur ansteigen!

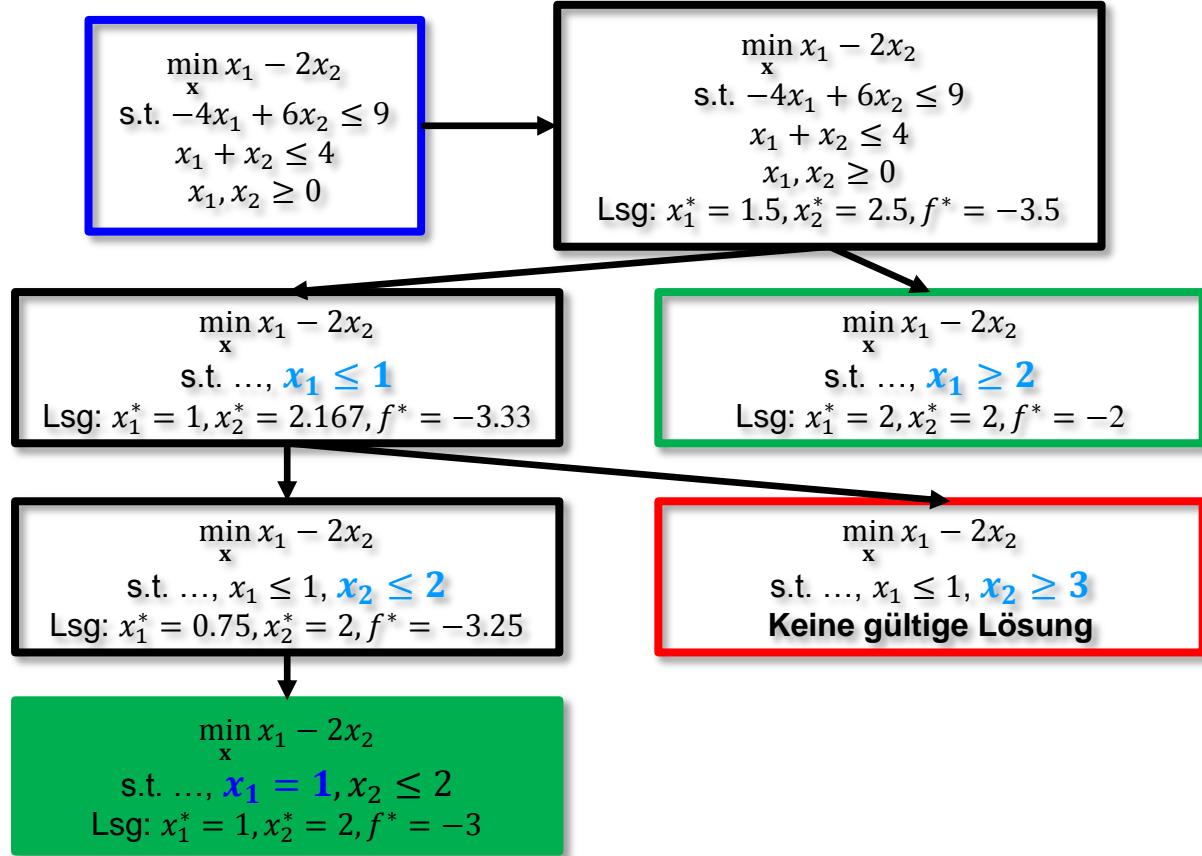


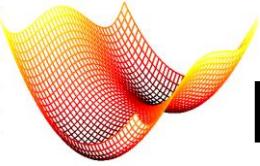
„Branch and Bound“-Methode (4)





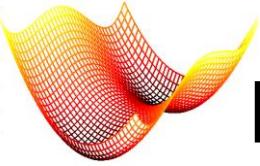
„Branch and Bound“-Methode (5)





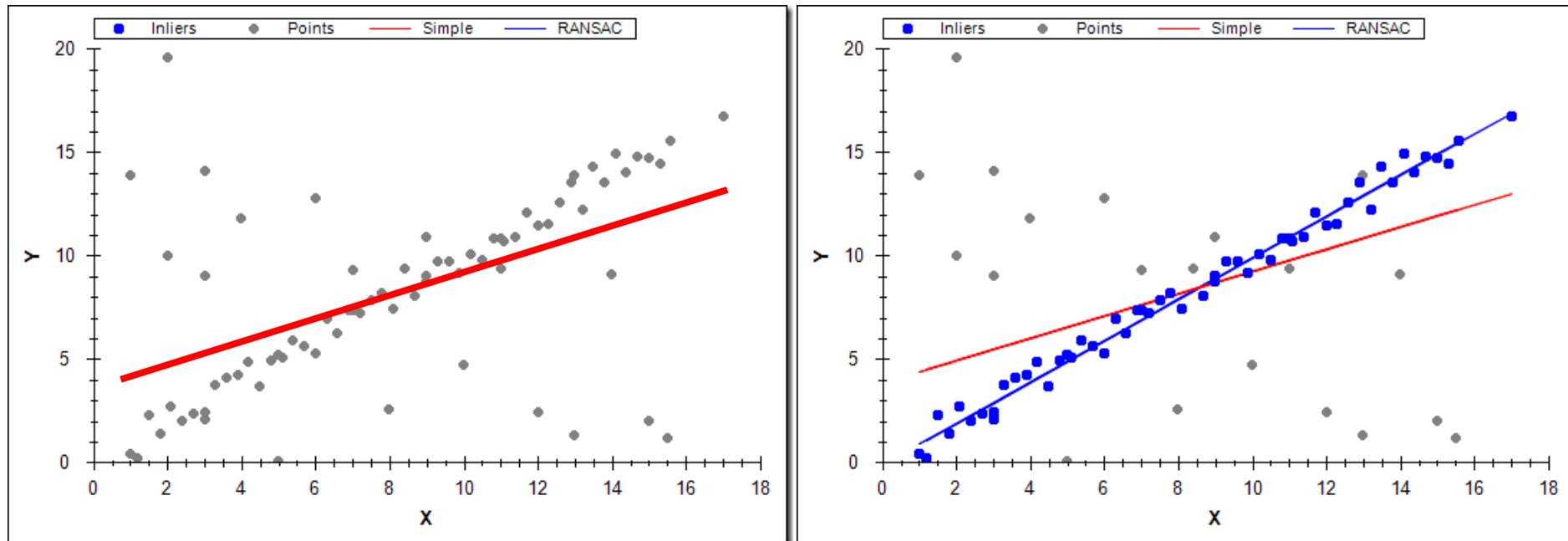
Robustes Matching: Random Sample Consensus (RANSAC) (1)

- Oft wird eine gefundene Zuordnung ϕ für weitere Berechnungen genutzt, z.B. bei Objekterkennung für Lage des Objektes (Berechnung der Parameter t einer Transformation T aus den Positionen der korresp. Deskriptoren)
- **Problem:** Falsche Zuordnungen führen in der Regel zu total faschen Lösungen!
- **Beispiel: Regressionsrechnung** mittels Minimierung des Fehlerquadrats
- **Ursache: Fehler sind nicht normalverteilt** (das ist jedoch Grundannahme bei Minimierung des Fehlerquadrats)
- **Messfehler** („Rauschen“) vs. **Klassifikationsfehler** („Ausreißer“)



Robustes Matching: Random Sample Consensus (RANSAC) (2)

- Beispiel für Probleme mit „gross errors“ bei Regression:



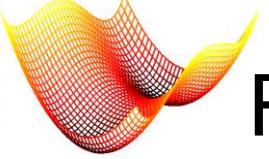
© Satish Singh

Marco Treiber, Fakultät FK 07, marco.treiber@hm.edu

05.10.2017

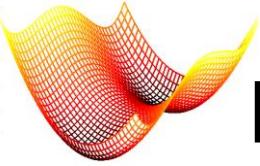
300





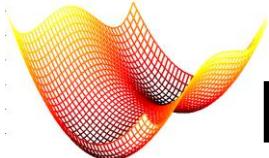
Robustes Matching: Random Sample Consensus (RANSAC) (3)

- Bei der Regression/Fehlerquadratsminimierung trägt **jedes** Datum zum Ergebnis bei. **Ausreißer** erhalten durch Quadratsbildung **besonders hohes Gewicht**
 - Neue Herangehensweise: versuche, **Ausreißer** zu **eliminieren** anstelle von „Herausmitteln“
 - Prinzip: **Hypothesen-Test-Verfahren**:
 1. Zufällige Auswahl der **minimalen** Menge an Daten (z.B. zwei Messpunkte für Regressionsgerade), um eine Hypothese aufzustellen („**Random Samples**“)
 2. Finde alle Daten, die in Einklang mit aktueller Hypothese sind (diese Daten bilden „**Consensus Set**“)



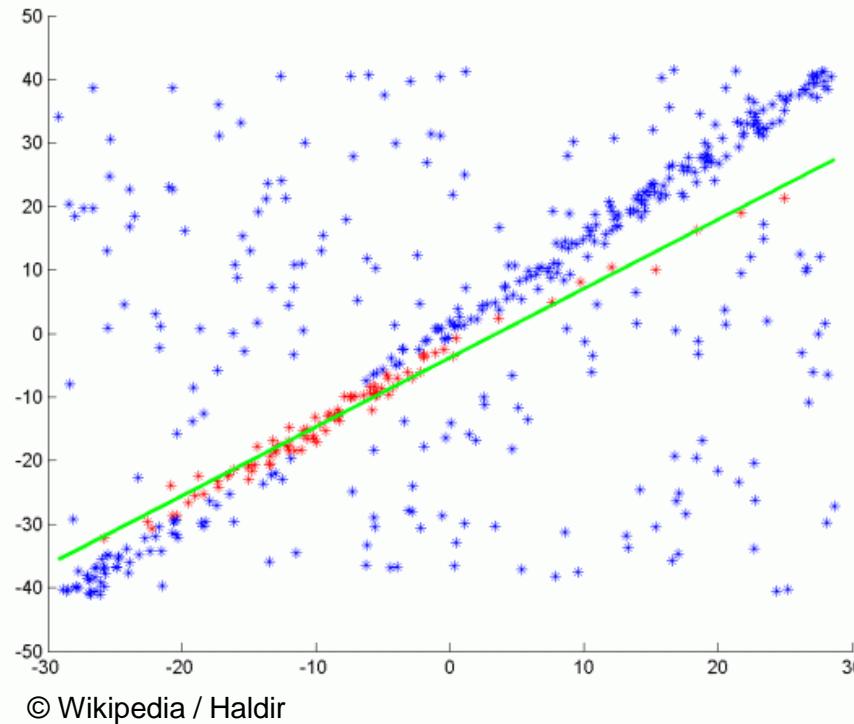
Robustes Matching: Random Sample Consensus (RANSAC) (4)

- Wiederholtes Durchführen dieser Prozedur führt zu immer besseren Consensus Sets
- **Am Schluss:** Berechnen der Lösung aus dem größten Consensus Set mit „herkömmlicher“ Regression
- Jetzt ist Regressionsrechnung mittels Minimierung des Fehlerquadrats abwendbar: Im Idealfall enthält Consensus Set ausschließlich „Inlier“!
- **Anzahl der Wiederholungen** wird durch Größe der minimalen Daten sowie des Anteils der Ausreißer bestimmt
- Minimale Wiederholungsanzahl ist theoretisch abschätzbar
→ feste Anzahl von Iterationen

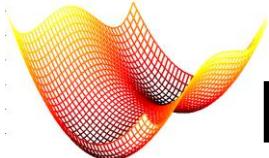


Robustes Matching: Random Sample Consensus (RANSAC) (5)

- Beispiel Ablauf RANSAC für linearen Fit:

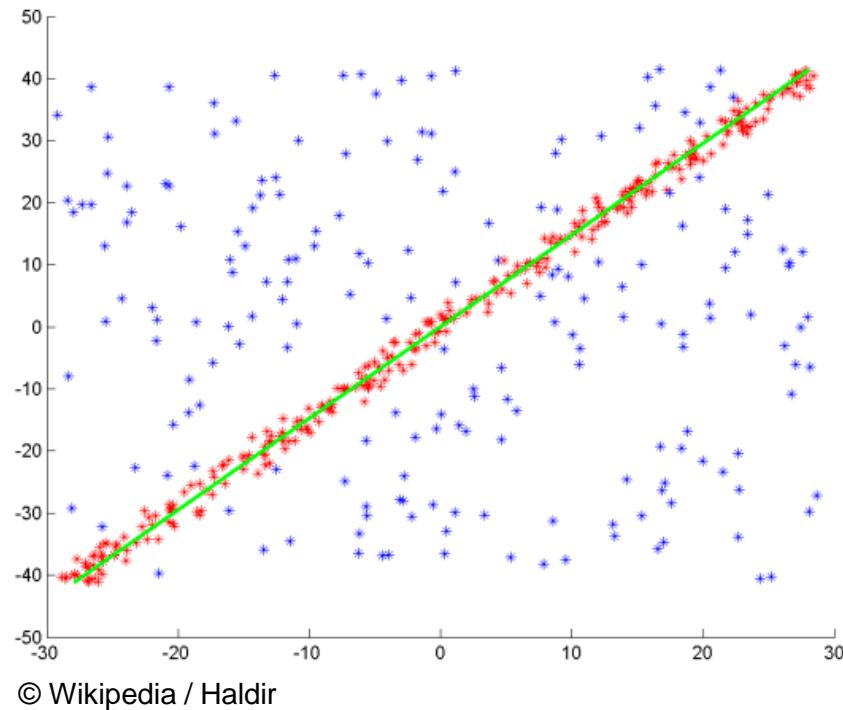


© Wikipedia / Haldir

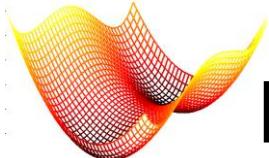


Robustes Matching: Random Sample Consensus (RANSAC) (6)

- **Ergebnis** (größter „Consensus-Set“):

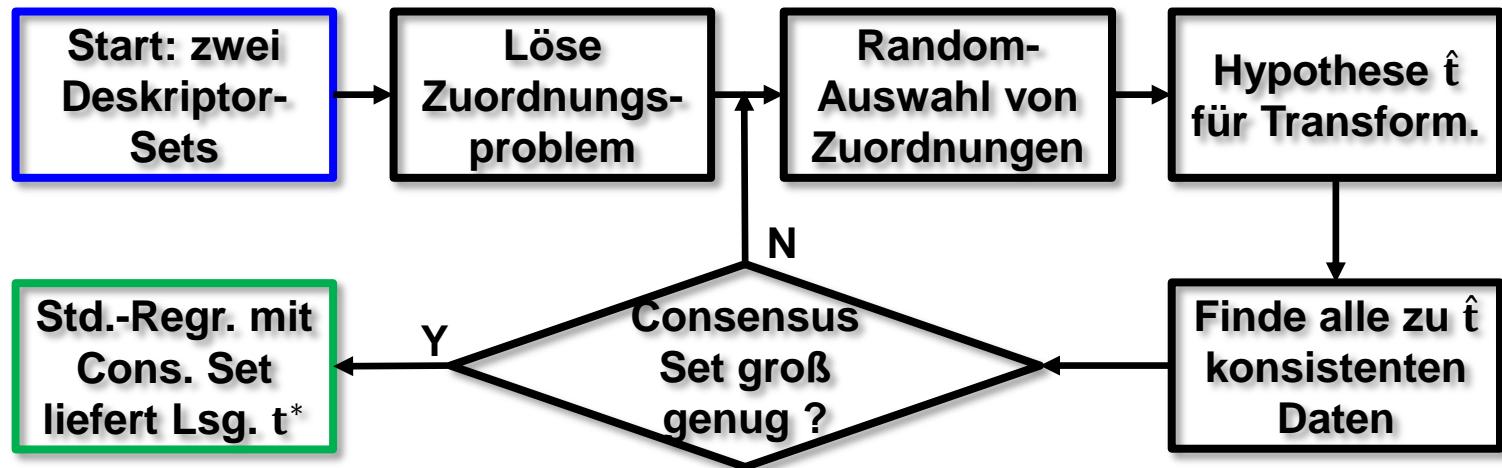


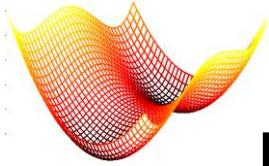
© Wikipedia / Haldir



Robustes Matching: Random Sample Consensus (RANSAC) (7)

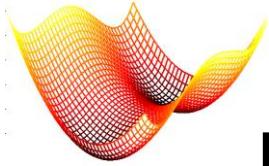
- Ablauf bei der Objekterkennung (finde optimale Transformation zur Bestimmung der Lage eines Objektes)





Heuristik: Vogel-Approximation (1)

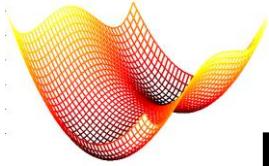
- Die Vogel-Approximation eignet sich vor allem für **Verteilungs- und Transportprobleme**
- **Grundidee** ist die Problemlösung unter Beachtung von **Opportunitätskosten**
- Es wird nicht die Lösung gewählt, die die geringsten Kosten verursacht, sondern die Lösung, die die höchste **Kostensteigerung** verursacht, wenn man sie **nicht** wählen würde
- **Approximation (Heuristik)**: keine Garantie für Optimallsg., aber „sehr gut“, d.h. vermutl. „sehr nahe“ an Optimallösung
- Eignet sich für die **Ermittlung einer Start-Lösung** für exaktes Verfahren (z.B. anst. NW-Eckenr. bei Transportpr.)



Heuristik: Vogel-Approximation (2)

- **Beispiel:**
Drei Personen A, B, und C sollen zu drei Standorten I, II und III transportiert werden. Die jeweiligen Entfernungen zwischen Person und Standort lassen sich mit einer Kreuztabelle darstellen:
- Optimierungsziel ist, die Transportstrecke insgesamt so gering wie möglich zu halten

	I	II	III
A	2	4	7
B	6	10	12
C	9	8	3

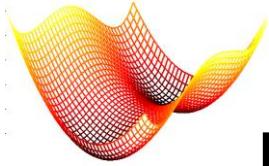


Heuristik: Vogel-Approximation (3)

- Separate Betrachtung: wähle z.B. für Person A den kürzesten Weg (A-I mit 2km)
- Jedoch: Weg I wäre auch für B optimal. Es müsste dann ein Weg gewählt werden, der mindestens 4 km länger ist (Opportunitätskosten) → wechselseitige Abhängigkeit!
- Wie ersichtlich ist, ist es günstiger, für A die nicht optimale Zuordnung A-II zu wählen, da hier der Nachteil kleiner ist

	I	II	III
A	2	4	7
B	6	10	12
C	9	8	3

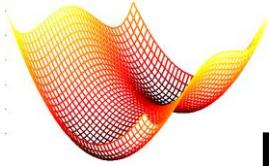
Diagramm zur Vogel-Approximation: Eine 3x4-Matrix mit Werten für Personen A, B und C über drei Alternativen I, II und III. Rote Pfeile zeigen die Differenzen zwischen den Werten von Spalte I und Spalte II für jede Zeile.



Heuristik: Vogel-Approximation (4)

- Berechnung Opportunitätskosten für jede „scheinbar“ optimale Auswahlmöglichkeit: Suche in jeder Zeile/Spalte nach dem kleinsten Wert und errechnet die Differenz zum zweitkleinsten (Alternativ-)Wert
- Max aller Opportunitäts-Werte → Suche in dieser Zeile bzw. Spalte nach den kleinsten Kosten → wähle C-III

	I	II	III	
A	2	4	7	2
B	6	10	12	4
C	9	8	3	5
	4	4	4	

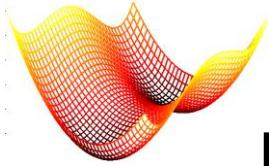


Heuristik: Vogel-Approximation (5)

- Da die Zuordnung C-III getroffen wurde, fallen automatisch alle Kombinationsmöglichkeiten mit C und III weg
 - verbleibenden Möglichkeiten: wieder Opportunitätskosten
 - Die Opportunitätskosten sind für A-II am höchsten → wähle A-II → alle Kombinationen mit A oder II fallen weg
 - Damit verbleibt als letzte Kombination B-I
- Gesamte Strecke: C-III: 3 + A-II: 4 + B-I: 6 = 13 km

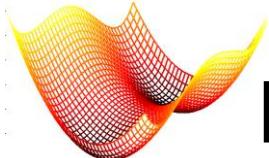
	I	II	III
A	2	4	7
B	6	10	12
C	9	8	3

	I	II	
A	2	4	2
B	6	10	4
	4	6	



Reihenfolgeplanung

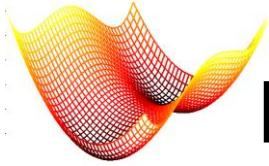
- Ein häufiges Optimierungsproblem ist abgesehen von der Auswahl nutzenmaximierender Projekte die **nutzenmaximierende Planung von Projekten**
- Problemstellung allgemein: **X Tätigkeiten müssen auf Y Stationen durchgeführt werden**
- **Fragestellung:** In welcher Reihenfolge werden welche Tätigkeiten auf welchen Stationen durchgeführt?
- Einsatzmöglichkeiten (z.B.):
 - Produktionsplanung
 - Projektplanung
 - usw.



Reihenfolgeplanung: Johnson-Algorithmus (1)

- **Beispiel:** Fünf Aufträge X_{1-5} sollen auf zwei Maschinen A_1 und A_2 gefertigt werden
- Voraussetzung: jeder Auftrag zunächst auf A_1 , dann auf A_2
- Ziel: **Minimierung der Gesamtarbeitszeit**
- Bearbeitungszeiten p für jeden Auftrag auf den jeweiligen Maschinen: s. Tabelle
- **Johnson-Algorithmus:** opt. Lsg. für zwei Maschinen

Maschinen	Aufträge				
	X_1	X_2	X_3	X_4	X_5
A_1	3	7	4	5	7
A_2	6	2	7	3	4



Reihenfolgeplanung: Johnson-Algorithmus (2)

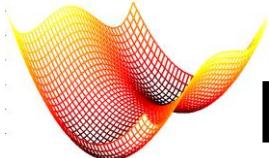
Entscheidungsregel für die Auftragsorganisation:

1. Suche die minimale Bearbeitungszeit p (im Beispiel: 2)
Sollten mehrere gleiche Werte existieren, so wähle einen beliebigen aus
2. Ist dieser Wert bei Maschine A1, dann wird dieser Auftrag als **erstes** bearbeitet.
Ist er bei Maschine A2, so wird er als **letzter** bearbeitet
(im Beispiel: da der Wert 2 zu Maschine A2 gehört, wird dieser Auftrag nach hinten gestellt)
3. Streiche den so zugeordneten Auftrag und gehe zu 1.
 - Im Beispiel ergibt sich folgende Auftragsreihenfolge:
 $X_1 - X_3 - X_5 - X_4 - X_2$



Multikriterielle und globale Optimierung

- **Multikriterielle Optimierung**
 - Pareto-Menge
 - Lösungsstrategien
- **Globale Optimierung**
 - Mehrere Startpunkte
 - Evolutionäre Verfahren
- **Surrogate Modellierung**



Multikriterielle Optimierung (1): Definition

- In der Praxis sind oft **mehrere Zielgrößen gleichzeitig zu optimieren**, z.B.
 - Motor: minimiere Verbrauch + minimiere Herstellkosten
 - Tragwerk: maximiere Tragfähigkeit + minimiere Masse
- In der Regel **konkurrierende Ziele → Kompromiss** nötig („trade-off“)

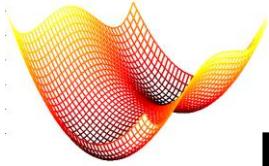
$$\min_{\mathbf{x}} \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}) \quad f_2(\mathbf{x}) \quad \dots \quad f_Z(\mathbf{x})]^T; z \in [1, 2, \dots, Z]$$

Nebenbedingungen:

$$\mathbf{g}(\mathbf{x}) \leq 0$$

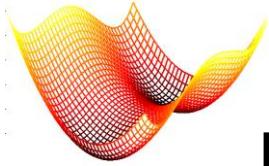
$$\mathbf{h}(\mathbf{x}) = 0$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$$



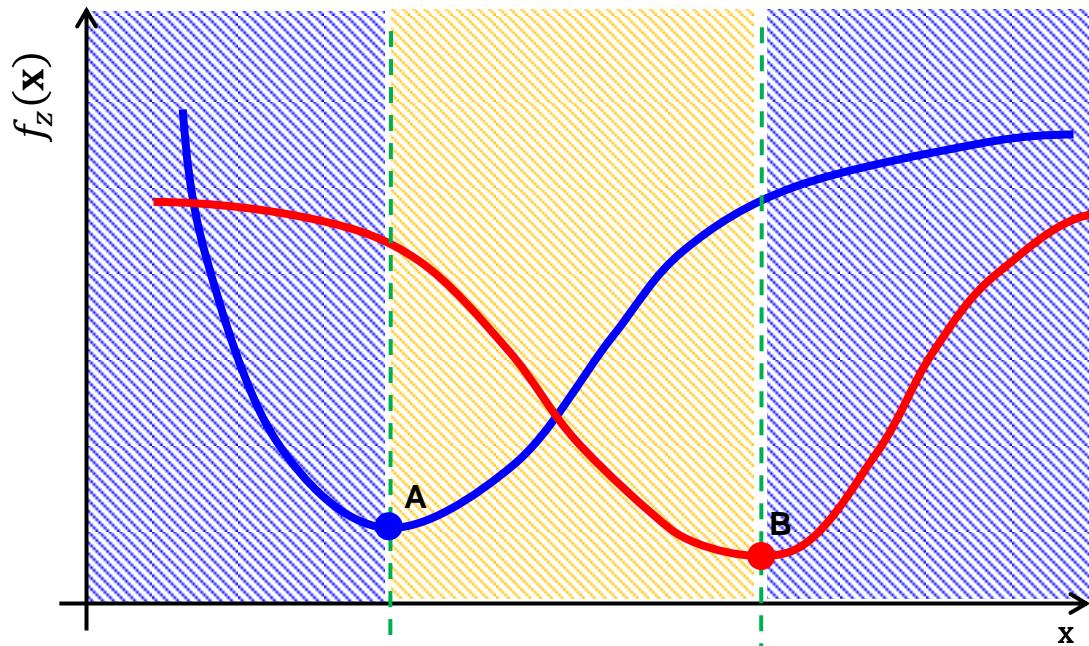
Multikriterielle Optimierung (2)

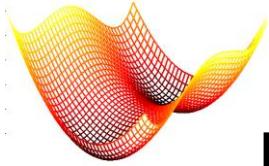
- Multikriterielle Probleme können sich auch ergeben, wenn (zunächst) nur eine Größe zu optimieren ist:
 - **Nicht vorhersehbare Eingangsgrößen**: Bsp.: gesucht ist Gewinn-Maximum, aber Parameter wie Nachfrage oder erzielbarer Preis sind unklar → Entwickle mehrere Szenarien mit unterschiedl. Werten dieser Param. und finde diejenige Lösung, welche **in Summe optimal** ist.
 - **Stochastische Optimierungsgrößen**: Bsp.: Portfolio-Optimierung (maximiere Ertrag): Finde diejenige Lsg., welche Mittelwert μ maximiert und Varianz σ minimiert
- **Lösungsansatz**: Optimierte nur eine dieser Größen, führe für die anderen Schranken ein (Bsp.: Max μ ; NB: $\sigma \leq t$)



Pareto-Optimum (1)

- Aufgrund konkurrierender Ziele gilt es im Allg. **keine eindeutige Lösung** x^* , die alle $f_z(x)$ gleichzeitig optimiert



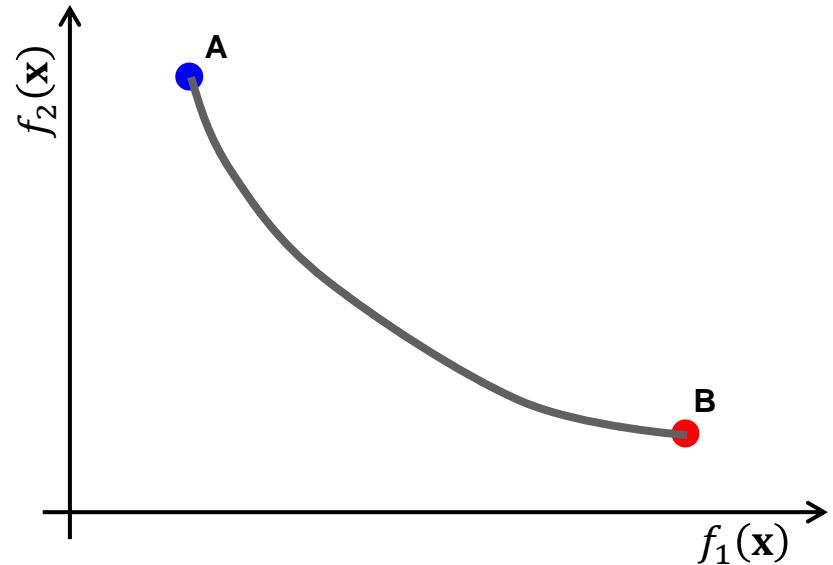


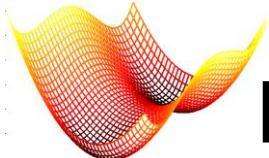
Pareto-Optimum (2)

- Aufgrund konkurrierender Ziele gilt es im Allg. **keine eindeutige Lösung** \mathbf{x}^* , die alle $f_z(\mathbf{x})$ gleichzeitig optimiert
- Stattdessen: mehrere Punkte $\hat{\mathbf{x}}$, in deren Umgebung ein Fortschritt in einer Zielfunktion $f_i(\mathbf{x})$ eine Verschlechterung in mindestens einer anderen $f_j(\mathbf{x})$ bedeutet (solche $\hat{\mathbf{x}}$ werden als „**Pareto-Optimum**“ bezeichnet)
- **Pareto-Menge**: Menge aller (möglichen) Pareto-Optima
- **Ziel der multikriteriellen Optimierung: Finde Pareto-Optima und idealerweise die komplette Pareto-Menge**

Pareto-Optimum (3)

- Pareto-Menge im Z -dimensionalen Funktionsraum dargestellt ergibt eine spezielle Form, die sog. „**Pareto-Front**“:
- Berechne für ein $\hat{\mathbf{x}}$ alle $f_z(\hat{\mathbf{x}}) \rightarrow$ ein Punkt im Funktionsraum
- Pareto-Front ist Menge dieser Punkte für alle $\hat{\mathbf{x}}$
- Z.B. bei 2 Krit.: 1-dim. Kurve
- Pareto-Front nicht notwendigerweise zusammenhängend
- Verlauf der Pareto-Front hängt vom Verhältnis der Funktionswerte ab



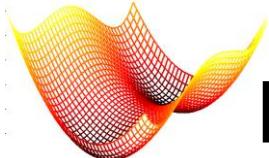


Lösungsstrategie: gewichtete Summe (1)

- **Grundidee** zur Ermittlung von Pareto-Optima: **Umformung zu einer uni-kriteriellen (d.h. skalaren) Zielfunktion $J(\mathbf{x})$**
- Hierzu: bilde **gewichtete Summe der einzelnen Krit.** $f_z(\mathbf{x})$

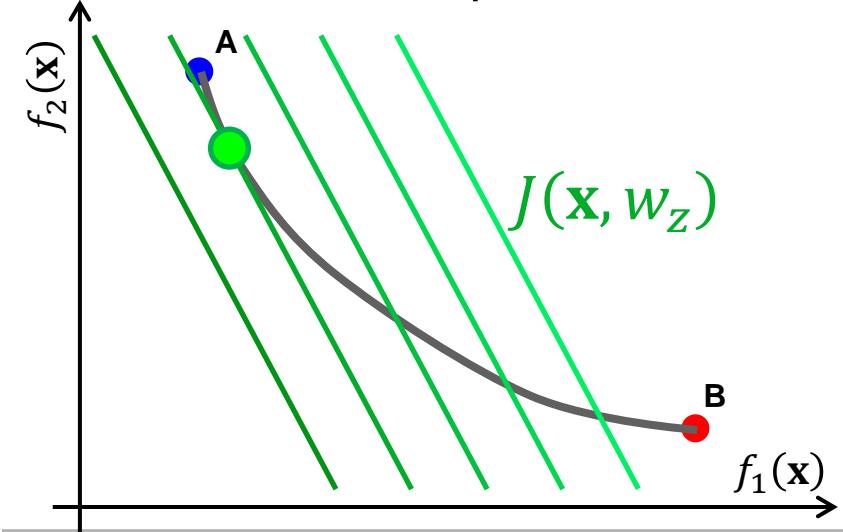
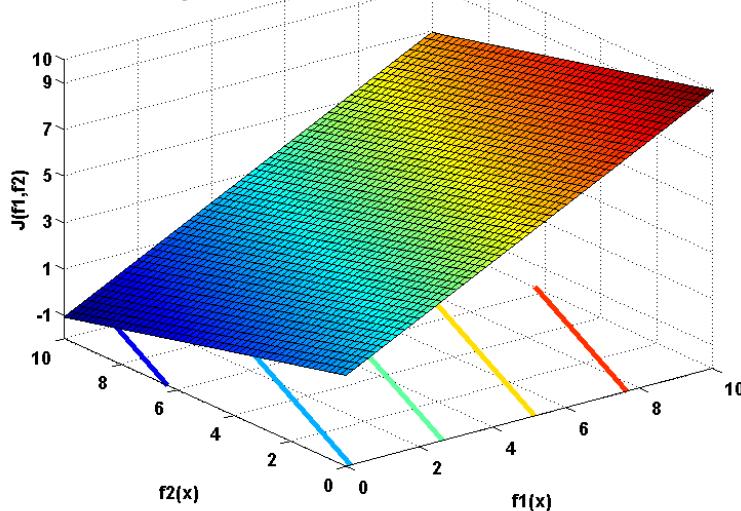
$$J(\mathbf{x}) = \sum_{z=1}^Z w_z \cdot f_z(\mathbf{x})$$

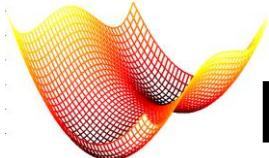
- **Trade-off** wird durch die **Gewichtungsfaktoren w_z** gesteuert
- Verschiedene w_z führen zu unterschiedlichen Pareto-Optima
- $J(\mathbf{x})$ ist **linear** in w_z und $f_z(\mathbf{x})$
- Höhenlinien von $J(\mathbf{x})$ im Funktionsraum sind Geraden,
Steigung der Geraden hängt von den w_z ab
- Pareto-Opt. $\hat{\mathbf{x}}$ ist Schnittp. e. Höhenl. von $J(\mathbf{x})$ mit Pareto-Front



Lösungsstrategie: gewichtete Summe (2)

- Beispiel für das Festlegen eines Pareto-Optimums auf der Pareto-Front durch eine gewichtete Summe für $Z = 2$:
- Variation der w_z führt zu unterschiedl. geneigten Höhenlinien von $J(\mathbf{x})$ und damit zu unterschiedlichen Pareto-Optima



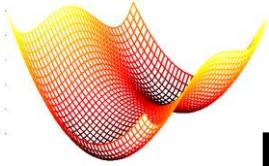


Lösungsstrategie: gewichtete Summe (3)

- **Problem:** Pareto-Optima, welche in konkaven Stellen der Pareto-Front liegen, können mit der gewichteten Summe nicht berechnet werden (es existiert kein „passendes“ w_z für diese Stellen)
- **Ausweg:** Erweiterung auf **nichtlineare** $J'(\mathbf{x})$, z.B.:

$$J'(\mathbf{x}) = \sum_{z=1}^Z w_z \cdot f_z(\mathbf{x})^n$$

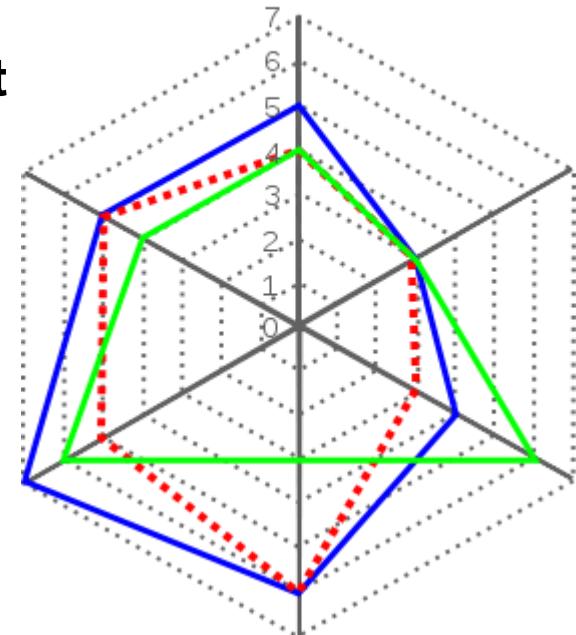
- In der Regel wird n als kleiner Integer-Wert > 1 gewählt
- Bei $n > 1$ sind jetzt gekrümmte Höhenlinien möglich
- Damit können auch konkave Stellen der Pareto-Front „erreicht“ werden



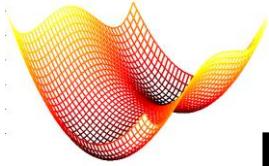
Netzdiagramme

- **Graphische Visualisierung, wie sich ein ermitteltes Pareto-Optimum \hat{x} auf die einzelnen Kriterien auswirkt**
- Jedes Kriterium bekommt eine Achse
- Sternförmige Anordnung der Achsen
- Verbinden der Funktionswerte $f_z(\hat{x})$ ergibt Polygon für jede Lösung \hat{x}
- „Güte“ der Lösung \hat{x} auf einen Blick erkennbar, auch bei mehreren Kriterien
- Einfacher Vergleich mehrerer Lsg.

Netzdiagramm

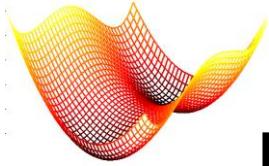


© Wikipedia / Tubas



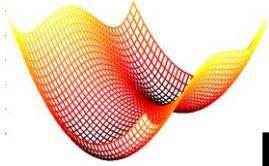
Multikriterielle Optimierung

- „**Goal-Programming**“: Bestimme $\hat{\mathbf{x}}$ so, dass die $f_z(\mathbf{x})$ möglichst exakt bestimmte vorab definierte Zielwerte t_z erreichen
- Daraus ergibt sich (n gerade):
$$J_G(\mathbf{x}) = \sum_{z=1}^Z w_z \cdot (f_z(\mathbf{x}) - t_z)^n$$
- oder: minimiere max. Abweichung: $J_{GL}(\mathbf{x}) \approx \max_z |f_z(\mathbf{x}) - t_z|$
(Achtung: Betragsbildung unschön → andere Umsetzung)
- Funktionen in MATLAB:
 - **fgoalattain**: für goal programming (minimiert $J_G(\mathbf{x})$)
 - **fminimax**: minimiert den „worst-case-Wert“ (min. $J_{GL}(\mathbf{x})$)



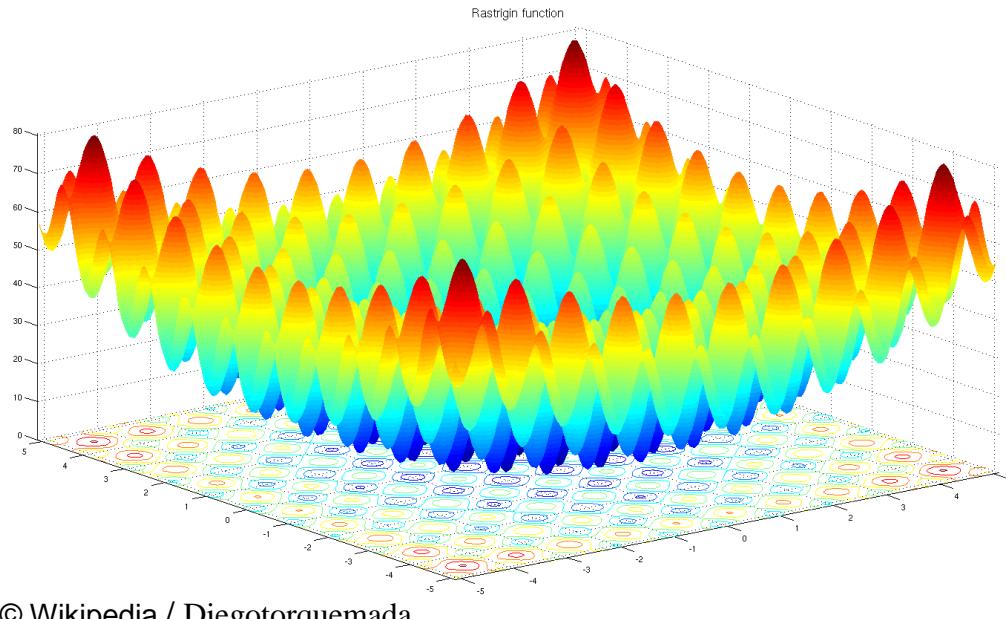
Multimodale Zielfunktionen (1)

- In der Praxis sind Zielfunktionen (mehr oder weniger stark) nichtlinear
- Tendenz zu mehr als einem (lokalen) Optimum (sog. **multimodale** Funktionen)
- Problem: **Abstiegs-Methoden finden lediglich lokale Optima**
- Am Ende der (lokalen) Optimierung folgende offenen Fragen:
 - Ist gefundenes Optimum auch das globale Minimum?
 - Gibt es weitere lokale Optima? Wenn ja, wie viele?
- Antwort auf diese Fragen ist entscheidend für die Beurteilung des Optimierungs-Ergebnisses, aber systematisch sehr, sehr schwer zu geben

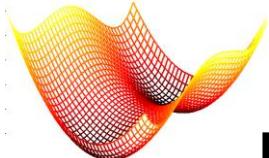


Multimodale Zielfunktionen (2)

- Beispiel: **Rastrigin-Funktion** $f(\mathbf{x}) = A n + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$
- Wird zum Testen von globalen Optimierungsverfahren verwendet

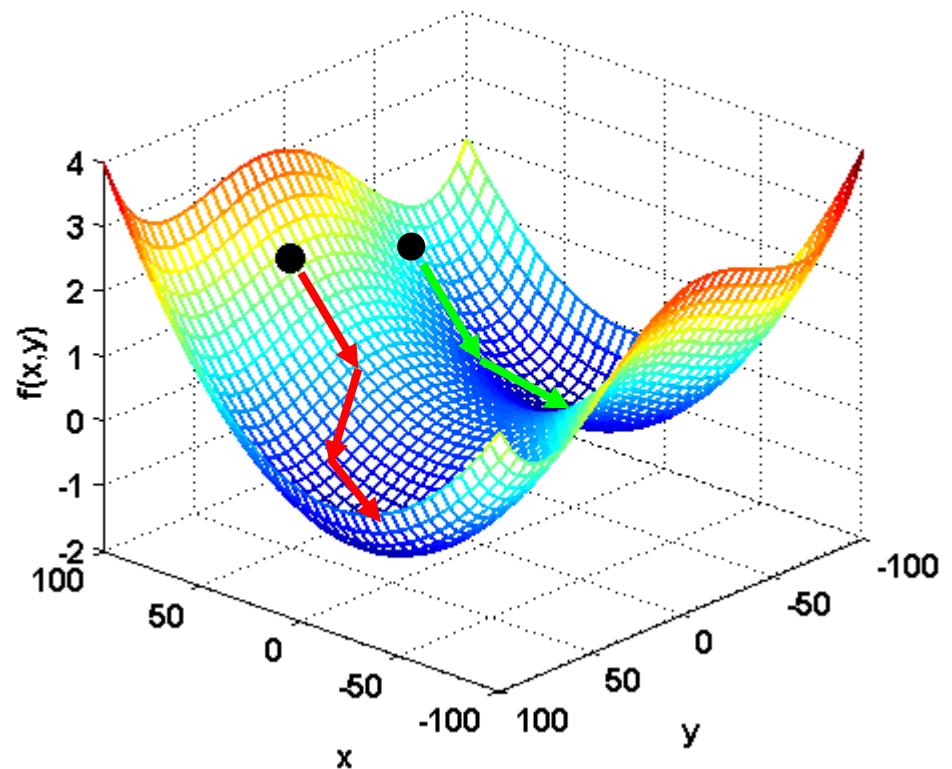


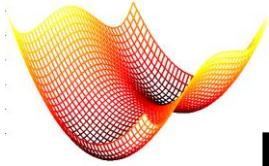
© Wikipedia / Diegotorquemada



Multimodale Zielfunktionen (3)

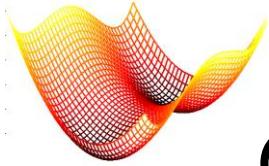
- **Abstiegsverfahren:** Konvergenz gegen **lokales** Minimum
- **Beobachtung** bei mehreren lok. Minima: es hängt vom **Startpunkt** ab, gegen welches Minimum das Verfahren konvergiert!





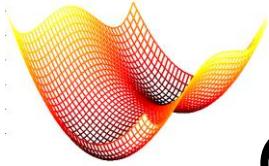
Multimodale Zielfunktionen (4)

- **Lösungsansatz:** Führe Q (unabhängige) **Optimierungen mit verschiedenen Startpunkten** \mathbf{x}_0^q durch $\rightarrow Q$ Lsg. $\mathbf{x}^{*,q}$
- Wahl der Startpunkte z.B. mittels **äquidistantem Rastern** des gesamten Lösungsraumes
- **Endergebnis** ist dasjenige $\mathbf{x}^{*,q}$, welches den **geringsten Zielfunktionswert** hat: $\mathbf{x}^* = \min_q f(\mathbf{x}^{*,q})$
- **Einfach zu implementierende Strategie**, aber:
 - Q -facher **Rechenaufwand**
 - Immer noch **keine Garantie**, dass globales Minimum auch wirklich gefunden wurde
- Jedoch: Optimierungen unabhängig \rightarrow **gut parallelisierbar**



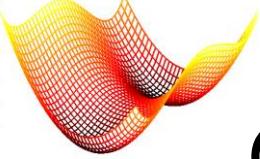
Genetische Optimierung (1)

- Probleme mathematisch motivierter Algorithmen bei sehr komplexen Aufgaben:
 - Anzahl N der Zielvariablen sehr groß → Rechenaufwand
 - Tendenz zu „ungünstigen“ Situationen (starke Nichtlinearitäten, Diskontinuitäten, etc.), d.h. Verletzung der Konvergenzkriterien / Optimalitätsbedingungen
- Alternativer Ansatz: **biologisch motivierte Optimierungsstrategien**
 - Natur dient als Vorbild
 - **Metaheuristik:** \approx Heuristik, deren prinzipieller Ablauf allgemein anwendbar ist → Suche nach einer **guten** Lsg.



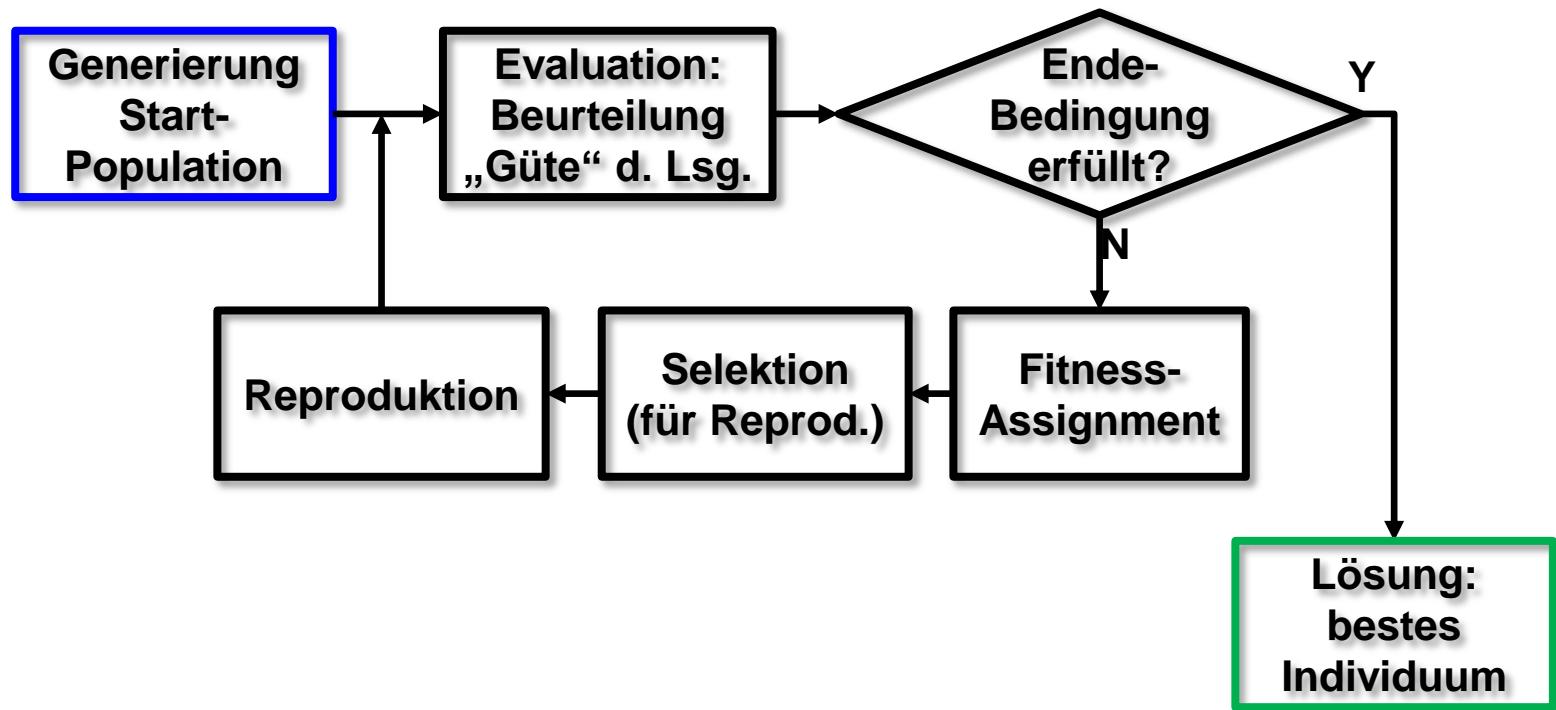
Genetische Optimierung (2)

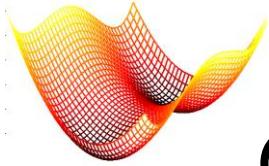
- **Evolutionäre Algorithmen:** **Vorbild Evolution:** Strategie zur optimalen Anpassung an Lebensbedingungen, um Überlebenschancen zu maximieren
- **Genetische Algorithm.**: „Mutation und Selektion“ der Gene
- **Grundidee:**
 - Start mit einer Menge von Lösungs-Kandidaten („**Population**“); jeder Kandidat ist „Individuum“
 - Ablauf: **Simulation der Evolution** durch gezielte, d.h. mögl. vorteilhafte iterative Modifikation der Population
 - Ergebnis: **Lösung** wird durch das **beste („fitteste“) Individuum** bestimmt



Genetische Optimierung (3)

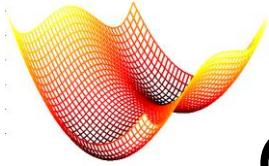
- Schematischer Ablauf:





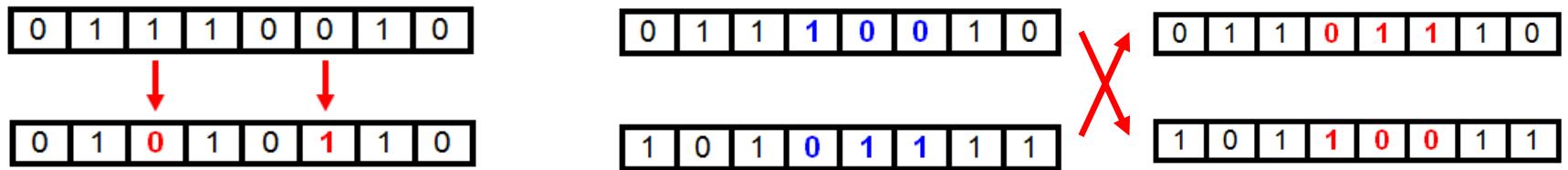
Genetische Optimierung (4)

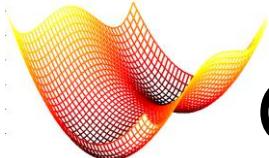
- **Start-Population:** Lösungskandidaten (Individuen) werden oft durch Zufalls-Prozesse ermittelt
- **Repräsentation einer Lösung durch Individuum :**
 - **binärer String**, d.h. 0/1-Folgen fester Länge
 - Z.B.: codiere jede Zielvariable x_i in L bits (Diskretisierung des Wertebereichs) und „staple“ sie im String
- **Evaluation:** berechne Zielfunktionswert für jedes Individuum → Beurteilung/Vergleich der Individuen möglich
- **Ende-Bedingungen** (Wann stoppt Evolution?): Rechenzeit, Anzahl der Reproduktions-Schritte, „Güte“ des besten Individuums



Genetische Optimierung (5)

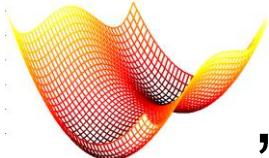
- **Fitness-Assignment:** Sortieren der Indiv. nach Zielfkt.-werten
- **Selektion:** nur die „fittesten“ Individuen dienen als „Ausgangsmaterial“ für den nächsten Reproduktionsschritt
- **Reproduktionsstrategien:**
 - **Elitismus:** fitteste Individuen kommen unverändert in die nächste Generation
 - **Mutation:** (zufälliges) „Toggeln“ einzelner Bits des Strings
 - **Kreuzung:** (zufäll.) Quertausch von Teilstrings bei 2 Indiv.





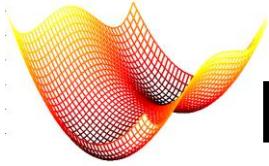
Genetische Optimierung: Eigenschaften

„Traditioneller“ (mathem.) Ansatz	Genetische Optimierung
Immer ein Lösungskandidat	Viele Kandidaten gleichzeitig + mehr Flexibilität
Deterministisches Vorgehen	Enthält Zufalls -Ereignisse + Ausweg in kniffligen Situationen - Nicht reproduzierbar
Bedingungen: Zielfkt. differenzierbar, stetig, ... Optimalitätsbedingungen	„ Black-box-Optimierung “: + Keine Anforderung an Zielfunktion, sehr allgemein
+ sehr gute Performance wenn bestimmte Bedingungen erfüllt sind	- In der Regel mehr Iterationen nötig + Robuster bei multimodalen Zielfkt.
in „günstigen“ Situationen: + Optimum garantiert + garantierte Konvergenz-Geschw.	Im wesentlichen Heuristik : - keine Garantie, dass Opt. gefunden - keine garantierte Konvergenz-Rate



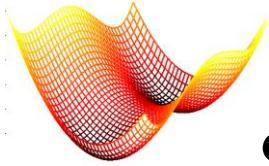
„Herausforderungen“ bei vielen praktischen Optimierungsaufgaben

- Viele praktische Probleme sind komplex. Folgende Eigenschaften sind charakteristisch und problematisch:
 1. **Große Anzahl N an Ziel-Variablen.** Bsp: $N \approx 1000$ bei der Planung eines großen Windparks zur Energie-Erzeugung
 2. **Hoher Rechenaufwand**
 3. **Zusammenhänge** zwischen Zielvariablen und Optimierungskriterium nicht bekannt oder zumindest **unklar**
- **Strategien:**
- Für 1.): Reduktion von N durch „design variable linking“ (Elimination einiger Zielvariablen x_i)
 - Für 2.) und 3.): **Surrogate Modellierung**



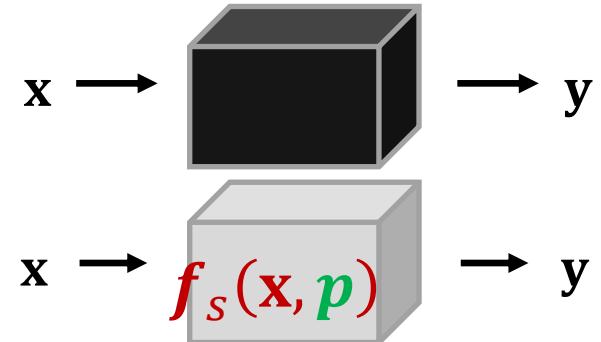
Dimensionsreduktion durch Variablenelimination

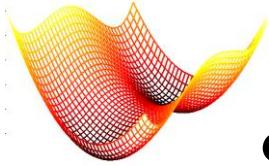
- Ausgangspunkt: $\mathbf{x} \in \mathbb{R}^N$; N sehr groß
- **Ziel: Reduktion von N durch Elimination einiger Zielvariablen x_i**
- Hierzu: Versuche, einige x_i durch andere x_j zu beschreiben und zu ersetzen, d.h. formuliere Zusammenhänge $x_i \stackrel{\text{def}}{=} r_i(\mathbf{x}_j)$
- **Beispiele:**
 - r_i aus Gleichungen als NB $\mathbf{h}(\mathbf{x})$ ableitbar
 - Starke Korrelationen werden als **feste Zusammenhänge** modelliert (Preis abhängig von Menge, Kosten abh. von Arbeitsstunden, etc.)



Surrogate Modellierung (1)

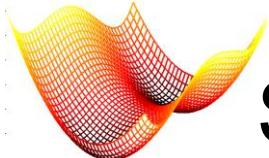
- **Ausgangspunkt:** Unklarer Zusammenhang zwischen zu optimierenden Größen y und x , d.h. Zielfunktion unbekannt!
- **Approximation:** $y := f_s(x, p)$
- Struktur von $f_s(x, p)$ wird vorgegeben (z.B. Polynom k-ten Grades), Parametervektor p wird anhand Trainingsdaten gelernt →
Adaption auf konkretes Problem
- Struktur von $f_s(x, p)$ wird so gewählt, dass Komplexität (Laufzeit) beherrschbar
- Oft: **Wähle p so, dass Test-Daten möglichst gut approximiert → eigenständiges Optimierungsproblem**





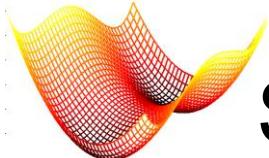
Surrogate Modellierung (2)

- **Schritte der Modellerstellung:**
 1. Wähle Struktur der $f_s(\mathbf{x}, \mathbf{p})$, z.B. Polynome
 2. Generierung Trainingsdaten, z.B. Messwerte $\mathbf{y}_i(\mathbf{x}_i)$
 3. Training des Modells: Ermittlung der Parameter \mathbf{p} , z.B. durch Minimierung der Fehler $\sum |\mathbf{y}_i(\mathbf{x}_i) - f_s(\mathbf{x}_i, \mathbf{p})|$
 4. Modell-Validierung: Berechnung der Fehler von Test-Samples $|\mathbf{y}_t(\mathbf{x}_t) - f_s(\mathbf{x}_t, \mathbf{p})|$
- Nur anwendbar, wenn Trainingsdaten verfügbar!
- „Güte“ der Optimierung von der Qualität der Trainingsdaten abhängig (Anzahl, Messrauschen, etc.)



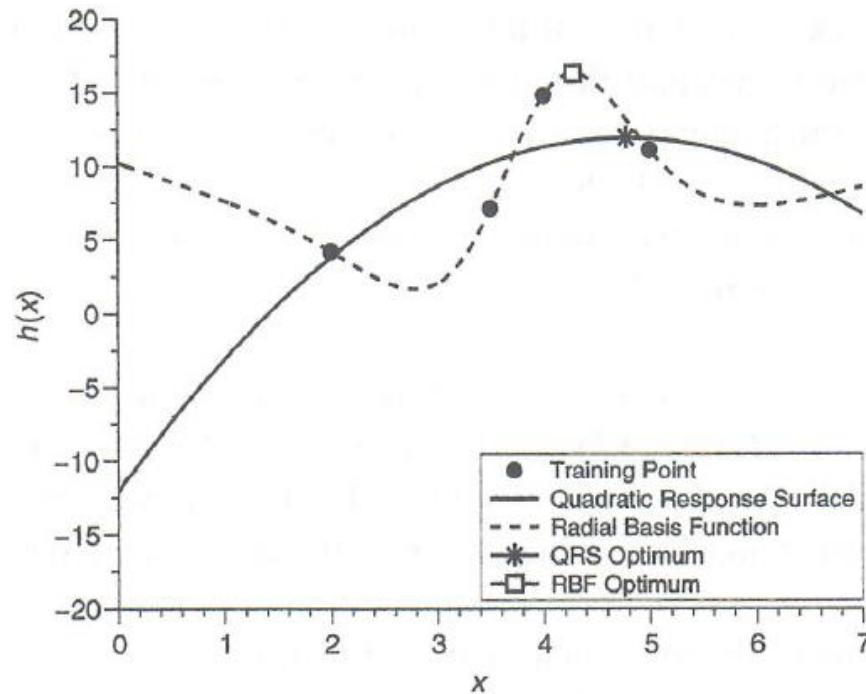
Surrogate Modellierung: Wahl der Funktionen (1)

- Häufig benutzte „Klassen“ von Funktionen für $f_s(\mathbf{x}, \mathbf{p})$:
- **Polynome**, d.h. $f_s(\mathbf{x}, \mathbf{p}) = p_0 + \sum_{i=1}^N p_i x_i + \sum_{i=1}^N \sum_{j=i}^N p_{ij} x_i x_j + \sum_{i=1}^N \sum_{j=i}^N \sum_{k=j}^N p_{ijk} x_i x_j x_k + \dots$
 - „glätten“ Messdaten (i.d.R. $f_s(\mathbf{x}_i, \mathbf{p}) \neq \mathbf{y}_i$)
 - Gefahr: „Overfitting“ bei zu hohem Grad
- **Radial Basis Functions (RBF)**, d.h.
 $f_s(\mathbf{x}, \mathbf{p}) = \sum_{i=1}^M w_i \psi(\|\mathbf{x} - \mathbf{x}_i\|)$ mit $\psi(r) = r$ (linear) oder
 $\psi(r) = r^2 \ln r$ (thin plate spline) od. $\psi(r) = e^{-r^2/k}$ (Gauß), etc.
 - Interpolieren zwischen den Messwerten, d.h. $f_s(\mathbf{x}_i, \mathbf{p}) = \mathbf{y}_i$
 - Gut bei starken Nichtlinearitäten, aber nicht gut definiert bei \mathbf{x} außerhalb des Trainingsbereichs

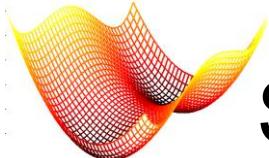


Surrogate Modellierung: Wahl der Funktionen (2)

- Beispiel: Vergleich von Polynom 2. Grades mit linearer RBF:

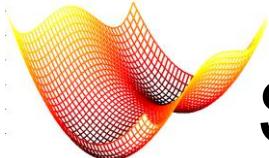


© Cambridge University Press / Messac



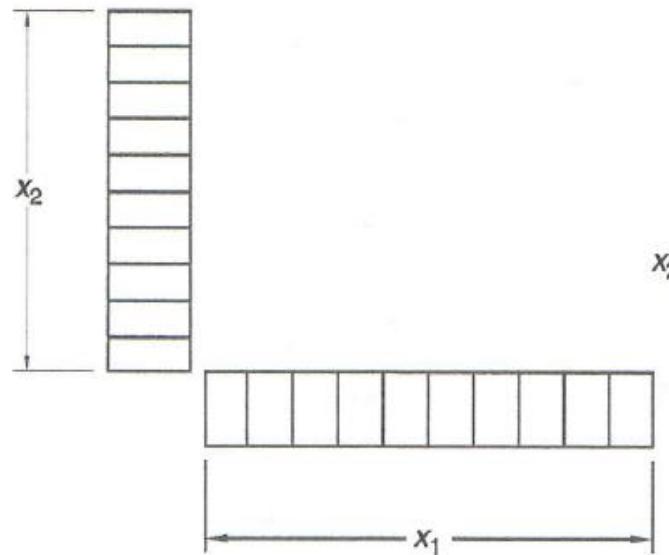
Surrogate Modellierung: Samplingstrategien (1)

- Erzeugung der Trainingsdaten durch „Messungen“:
 1. Festlegung der Positionen \mathbf{x}_i
 2. „Messung“: Ermitteln der zugehörigen \mathbf{y}_i
 - \mathbf{x}_i sollen Lösungsraum möglichst gut abdecken
- Sampling-Strategien:
- „uniform sampling“: äquidistantes Abtasten jeder Variable x_n + erschöpfende Kombination. Jedoch: exponentielles Wachstum mit steigendem N
 - „latin hypercube sampling“: äquidistantes Abtasten der x_n + Kombination so, dass jeder Abtastwert $x_{n,k}$ nur exakt einmal vorkommt!



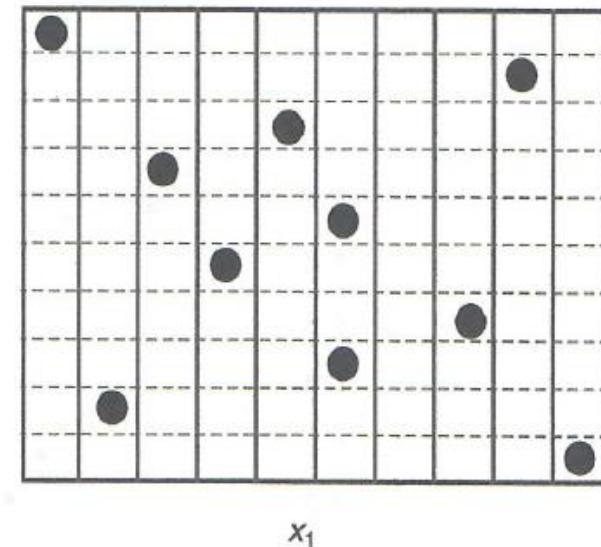
Surrogate Modellierung: Samplingstrategien (2)

- Beispiel für „latin hypercube sampling“ mit $N = 2$:



(a) Intervals Used for the LHS of Size
 $n = 10$

© Cambridge University Press / Messac



(b) 2-D Illustration of the LHS with
 $n = 10$