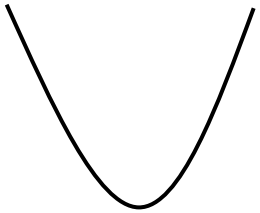
Optimierung

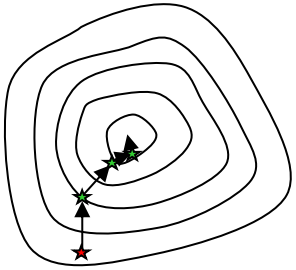
Vorlesung 3

Newton- und Quasi-Newton-Verfahren

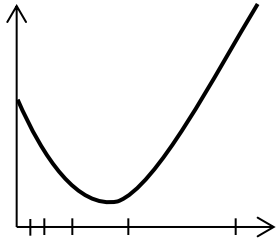
In den letzten Vorlesungen



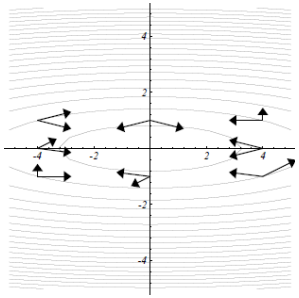
Konvexität (Garantie globaler Minima)



Gradientenabstieg
(allgemeines Verfahren für kontinuierliche Probleme)



Line Search (zur Bestimmung der optimalen Schrittweite)



Conjugate Gradients (CG)
(effizientes Verfahren für quadratische Funktionen)

Gradientenverfahren

- Iteratives Verfahren

  $$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tau^k \boldsymbol{d}^k$$

  mit einem Startpunkt $\boldsymbol{x}^0$, einer Änderungsrichtung $\boldsymbol{d}^k$ und einer Schrittweite τ^k

- Beim Gradientenverfahren entspricht die Änderungsrichtung dem negativen Gradienten der Zielfunktion f an der aktuellen Stelle x^k

  $\boldsymbol{d}^k := -\nabla f(\boldsymbol{x}^k)$ also $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \tau^k \nabla f(\boldsymbol{x}^k)$

- Gradientenverfahren verwenden nur die 1. Ableitung der Funktion. Die 2. Ableitung (Krümmung) wird ignoriert.

- Verfahren 1. Ordnung vs. Verfahren 2. Ordnung

- Annahme: die 2. Ableitung existiert, d.h. $f \in \mathcal{C}^2$ (2x stetig differenzierbar)

Newton-Verfahren (Motivation I)

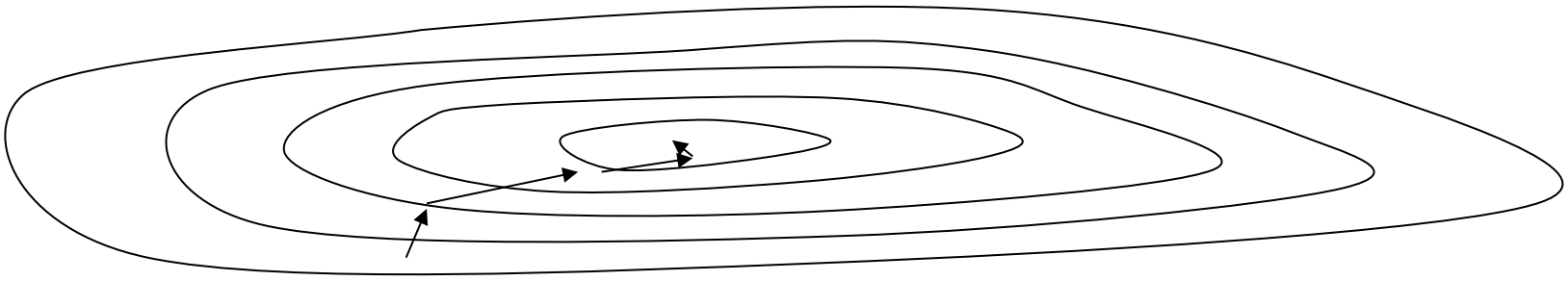
- Das **Newton-Verfahren** verwendet auch die 2. Ableitung (Krümmung):

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \tau^k H_f^{-1}(\boldsymbol{x}^k) \nabla f(\boldsymbol{x}^k)$$

- Im mehrdimensionalen Fall ist diese durch die **Hesse Matrix** H gegeben:

$$H_f(x_1, ..., x_n) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

- Motivation: größere Schritte in Richtungen mit schwacher Krümmung



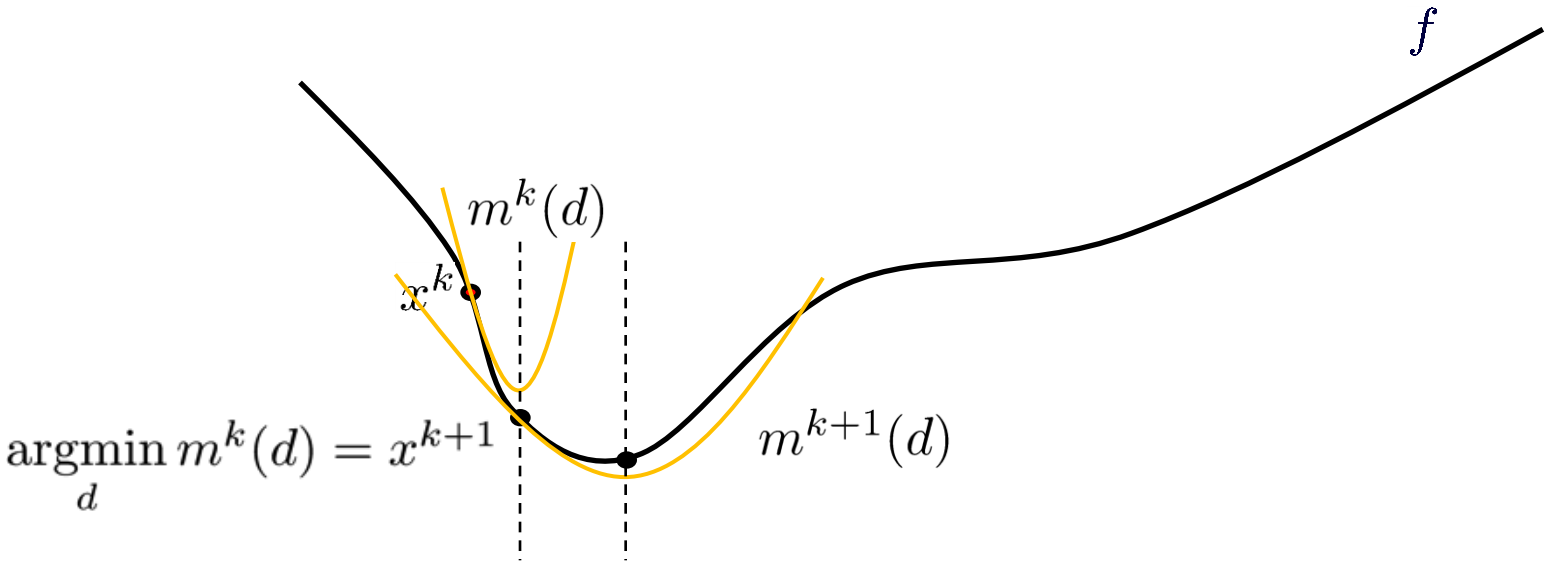
Weitere Motivation: quadratische Approximation der Zielfunktion

- **Zielfunktion lässt sich an der Stelle x^k durch Taylor-Reihe darstellen:**

$$f(\boldsymbol{x}^k+\boldsymbol{d}) = f(\boldsymbol{x}^k) + \boldsymbol{d}^\top \nabla f(\boldsymbol{x}^k) + \frac{1}{2}\boldsymbol{d}^\top H(\boldsymbol{x}^k)\boldsymbol{d} + O(\boldsymbol{d}^3)$$

→ **d.h. die Funktion wird lokal durch eine quadratische Funktion approximiert**

$$f(\boldsymbol{x}^k+\boldsymbol{d}) = f(\boldsymbol{x}^k) + \boldsymbol{d}^\top \nabla f(\boldsymbol{x}^k) + \frac{1}{2}\boldsymbol{d}^\top H(\boldsymbol{x}^k)\boldsymbol{d} = m^k(\boldsymbol{d})$$



Weitere Motivation: quadratische Approximation der Zielfunktion

- Für $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{d}^k$ müssen wir also $\operatorname{argmin}_{\boldsymbol{d}} m^k(\boldsymbol{d})$ bestimmen, wobei

$$m^k(\boldsymbol{d}) = f(\boldsymbol{x}^k) \; + \; \boldsymbol{d}^\top \nabla f(\boldsymbol{x}^k) \; + \; \frac{1}{2} \boldsymbol{d}^\top H(\boldsymbol{x}^k) \boldsymbol{d}$$

- → quadratisches Optimierungsproblem
  - 1. Ableitung von $m^k(\boldsymbol{d})$ nach $\boldsymbol{d}$:

$$\nabla m^k(\boldsymbol{d}) = \nabla f(\boldsymbol{x}^k) + H(\boldsymbol{x}^k)\boldsymbol{d} \qquad (*)$$

  - Setzten der Ableitung gleich 0 ergibt

$$\boldsymbol{d}^k = -H_f^{-1}(\boldsymbol{x}^k) \nabla f(\boldsymbol{x}^k)$$

- Bemerkung: (*) gilt nur da $H(\boldsymbol{x}^k)$ symmetrisch ist.

Vergleich zum Gradientenverfahren

- In beiden Verfahren haben wir eine Abstiegsrichtung $\boldsymbol{d}^k$

  $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tau^k \boldsymbol{d}^k$

  und müssen eine gute/optimale Schrittweite τ^k finden.

- Gradientenabstieg (Verfahren 1. Ordnung)

  $\boldsymbol{d}^k := -\nabla f(\boldsymbol{x}^k)$

- Newton-Verfahren (Verfahren 2. Ordnung)

  $\boldsymbol{d}^k := -H^{-1}(f(\boldsymbol{x}^k))\nabla f(\boldsymbol{x}^k)$

- Bestimmung der Schrittweite in beiden Fällen mittels Line Search

- Hinzunahme der Krümmung hat verschiedene Vor- und Nachteile

Konvergenzordnungen

- Konvergenz von Folgen $s_1, s_2, ..., \lim_{k \to \infty} s_k = \bar{s}$

- In unserem Fall: Die Konvergenz der Punkte x_k zur Lösung des Optimierungsproblems (bzw. zur Nullstelle der Ableitung)

- Lineare Konvergenz

$$\lim_{k \to \infty} \frac{|s_{k+1} - \bar{s}|}{|s_k - \bar{s}|} = C < 1$$

Beispiel: $s_k = 0.9^k$

- Superlineare Konvergenz

$$\lim_{k \to \infty} \frac{|s_{k+1} - \bar{s}|}{|s_k - \bar{s}|} = 0$$

Beispiel: $s_k = \frac{1}{k!}$

- Quadratische Konvergenz

$$\lim_{k \to \infty} \frac{|s_{k+1} - \bar{s}|}{|s_k - \bar{s}|^2} = C < 1$$

Beispiel: $s_k = 0.9^{(2^k)}$

Konvergenz Gradientenabstieg vs. Newton-Verfahren

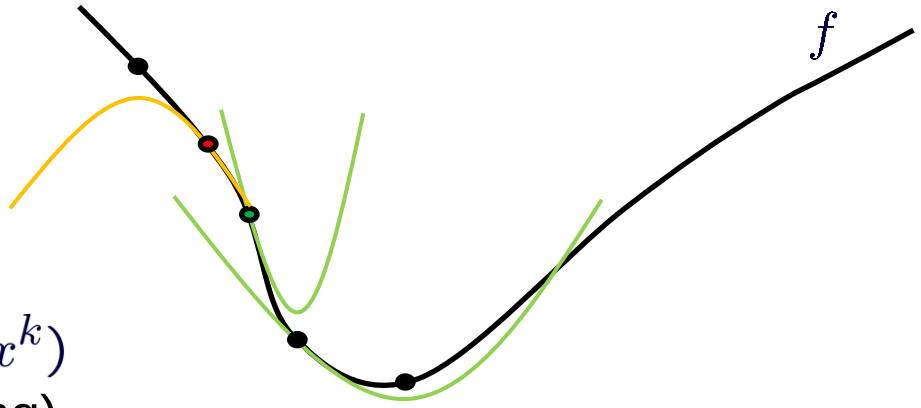
- Gradientenabstieg konvergiert linear
- Das Newton-Verfahren konvergiert für glatte, konvexe Funktionen nahe der Lösung quadratisch
- Insbesondere konvergiert das Newton-Verfahren für quadratische Funktionen in einem Schritt
- Das Newton-Verfahren ist also nominal wesentlich effizienter als Gradientenabstieg
- Zu berücksichtigen: Berechnung und Invertierung der Hesse-Matrix

$$d^k := -\nabla f(x^k) \quad \text{vs.} \quad d^k := -H^{-1}(f(x^k))\nabla f(x^k)$$

→ jeder einzelne Iterationsschritt ist wesentlich aufwendiger

Konvergenzradius: Wann konvergiert das Newton-Verfahren?

- Ist die Krümmung negativ, läuft das Newton-Verfahren in die falsche Richtung!



- Damit $d^k = -H_f^{-1}(x^k)\nabla f(x^k)$ eine Abstiegsrichtung ist, muss $H_f(x^k)$ positiv definit sein (positive Krümmung)

- Konvergenzgarantie für konvexe Funktionen (diese haben überall positive Krümmung)

- Bei nicht-konvexen Funktionen: Manipulation der Hesse Matrix, so dass alle ihre Eigenwerte positiv sind

- Beim Gradientenverfahren ist die Richtung per Definition eine Abstiegsrichtung

Übersicht

Gradientenabstieg

- Lineare Konvergenz
- Konvergenz für alle glatte Funktionen
- Jede Iteration:
  - Gradientenberechnung
  - Line Search

Newton-Verfahren

- Quadratische Konvergenz
- Konvergenz für glatte, streng konvexe Funktionen (sonst zusätzliche Maßnahmen nötig)
- Jede Iteration:
  - Gradientenberechnung
  - Berechnung Hesse Matrix
  - Invertierung der Hesse Matrix
  - Line Search (nicht nötig bei quadratischen Funktionen)

Gibt es Verfahren mit superlinearer Konvergenz aber schnelleren Iterationen?

Quasi-Newton-Verfahren

- Zwei Grundideen:
  - Approximation der Hesse Matrix mithilfe des Gradienten (1. Ordnung)
  - Berechnung der Approximation aus dem vorherigen Iterationsschritt

- Newton-Verfahren:

$$x^{k+1} = x^k - \tau^k H_f^{-1}(x^k) \nabla f(x^k)$$

- Quasi-Newton Verfahren:

$$x^{k+1} = x^k - \tau^k (B^k)^{-1} \nabla f(x^k)$$

- Wie erhält man eine gute Approximation der Hesse-Matrix?

Approximation der Hesse Matrix

- Taylor Approximation des Gradienten (ges.: Nullstelle der Ableitung!)

$$\nabla f(x^k + d) \approx \nabla f(x^k) + H_f(x^k)d$$

- Mit $d = x^{k+1} - x^k$ erhält man

$$\nabla f(x^{k+1}) - \nabla f(x^k) \approx H_f(x^k)(x^{k+1} - x^k)$$

- Die Approximation B^k sollte dies auch erfüllen.

→ **Quasi-Newton Bedingung (Sekantengleichung)**:

$$B^k s^k = y^k$$

mit $s^k := x^k - x^{k-1}$ $\qquad y^k := \nabla f(x^k) - \nabla f(x^{k-1})$

- Um diese Gleichung erfüllen zu können muss gelten (siehe nächste Folie)

$$(s^k)^\top y^k > 0$$

→ Bedingung für die Schrittweitensuche

Beweis Sekantengleichtung.

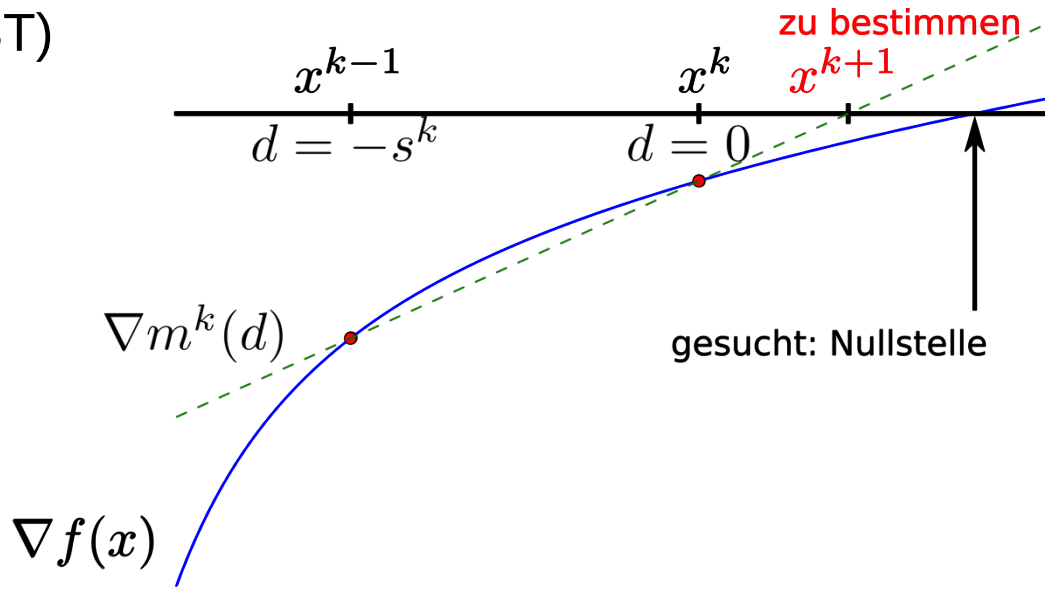
Beweis von $B^k s^k = y^k$ impliziert $(s^k)^\top y^k > 0$

- B Positive definite
  - Das heisst: $\forall z \neq \vec{0}: \quad z^\top B^k z > 0$
  - Aus $B^k s^k = y^k$ folgt $(s^k)^\top B^k s^k = (s^k)^\top y^k$
  - Es gilt $(s^k)^\top B^k s^k > 0$ und damit $(s^k)^\top y^k > 0$

Erklärung für Sekantengleichen

- Wir suchen eine Nullstelle (NST) der Ableitung $\nabla f(x)$
- Lineare Approximation mit Matrix B^k:

$$\nabla m^k(d) = \nabla f(x^k) + B^k d$$



- Sekantenmethode zur NST-Suche -> Bedingungen:

$$\nabla m^k(\vec{0}) = \nabla f(x^k) \text{ und } \nabla m^k(-s^k) = \nabla f(x^{k-1}) \quad \text{mit } s^k = x^k - x^{k-1}$$

- 1. Bedingung automatisch erfüllt, 2. Bedingung kann umformuliert werden da $\nabla m^k(-s^k) = \nabla f(x^k) - B^k s^k$ gilt:

$$\begin{aligned} \nabla f(x^k) - B^k s^k &= \nabla f(x^{k-1}) \\ B^k s^k &= \nabla f(x^k) - \nabla f(x^{k-1}) =: y^k \end{aligned}$$

Approximation der Hesse-Matrix

- Bisher folgende Bedingungen an B^k:
  - Symmetrische Matrix $B^k = (B^k)^\top$
  - Positiv definit (alle Eigenwerte > 0)
  - Sekantengleichung $B^k s^k = y^k$

- Dies führt immer noch zu vielen möglichen Lösungen
  → weitere Bedingung

$$B^k = \min_B \|B - B^{k-1}\|$$

(Approximation ist möglichst ähnlich zu der im letzten Iterationsschritt)

- Verschiedene Matrixnormen führen zu verschiedenen Unterverfahren.

- Mit einer gewichteten Frobeniusnorm, bei der die Gewichtsmatrix der mittleren Hesse-Matrix entspricht, führt dies zum Verfahren von Davidon, Fletcher und Powell (ohne Beweis).

Davidon-Fletcher-Powell Verfahren

- Von Davidon 1959 aus der Not entwickelt da sein Computer grundsätzlich abstürzte bevor die Optimierung abgeschlossen war.

- Neue Approximation der Hesse-Matrix wird aus der vorherigen berechnet:

$$B^{k+1} = \left(I - \rho^k y^k (s^k)^\top\right) \; B^k \; \left(I - \rho^k s^k (y^k)^\top\right) + \rho^k y^k (y^k)^\top$$

$$\rho^k := \frac{1}{(y^k)^\top y^k}$$

- Da man ohnehin an der inversen Hesse-Matrix interessiert ist, ist es vorteilhaft direkt die inverse Approximation $Q := B^{-1}$ zu iterieren:

$$Q^{k+1} = Q^k - \frac{Q^k y^k (y^k)^\top Q^k}{(y^k)^\top Q^k y^k} + \frac{s^k (s^k)^\top}{(y^k)^\top s^k}$$

BFGS-Verfahren

- Die Bedingung

$$B^k = \min_B \|B - B^{k-1}\|$$

kann auch auf die inverse Approximation angewendet werden:

$$Q^k = \min_Q \|Q - Q^{k-1}\|$$

- Dies führt zum noch etwas effizienteren Verfahren von Broyden-Fletcher-Goldfarb-Shanno (BFGS):

$$Q^{k+1} = \left(I - \rho^k \boldsymbol{s}^k (\boldsymbol{y}^k)^\top\right) Q^k \left(I - \rho^k \boldsymbol{y}^k (\boldsymbol{s}^k)^\top\right) + \rho^k \boldsymbol{s}^k (\boldsymbol{s}^k)^\top$$

- Aus Effizienzgründen ist die Reihenfolge der Operationen so zu wählen, dass keine Matrix-Matrix Multiplikationen entstehen.

(Multipliziere erst Q^k mit $\boldsymbol{y}^k$ und dann das Ergebnis mit $\boldsymbol{s}^k$)

BFGS-Verfahren

- Eingabe:
  - Startpunkt x^0
  - Initiale Approximation Q^0 (z.B. $Q^0 = I$)

- Iterationen:
  - Berechne Abstiegsrichtung:

$$\boldsymbol{d}^k = -Q^k \nabla f(\boldsymbol{x}^k)$$

  - Bestimme Schrittweite τ^k durch Line Search mit Wolfe Bedingungen (wichtig)
  - Berechne neue Lösung:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tau^k \boldsymbol{d}^k$$

  - Berechne neue Approximation der inversen Hesse Matrix:

$$\boldsymbol{s}^k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k, \quad \boldsymbol{y}^k = \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k), \quad \rho^k = \frac{1}{(\boldsymbol{y}^k)^\top \boldsymbol{y}^k}$$

$$Q^{k+1} = \left(I - \rho^k \boldsymbol{s}^k (\boldsymbol{y}^k)^\top\right) Q^k \left(I - \rho^k \boldsymbol{y}^k (\boldsymbol{s}^k)^\top\right) + \rho^k \boldsymbol{s}^k (\boldsymbol{s}^k)^\top$$

BFGS: Eigenschaften und Hinweise

- Superlineare Konvergenz. Was heißt das in der Praxis?

  Gradientenabstieg: 5264 Iterationen
  BFGS: 34 Iterationen
  Newton: 21 Iterationen

  Beispielproblem: $f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
  Abbruchkriterium: $\|\nabla f(\boldsymbol{x}^k)\| \leq 10^{-5}$

- Q^k ist immer positiv definit wenn Q^{k-1} positiv definit war.

- Line search sollte immer $\tau^k = 1$ testen, da diese Schrittweite in den meisten Iterationen akzeptiert wird und zu schneller Konvergenz führt.

- Die Wolfe Bedingungen sind wichtig, damit BFGS zeitweise schlechte Approximationen Q wieder korrigiert.

Least-Squares-Aufgaben

- Eine sehr häufige Klasse von Funktionen:

$$f(x) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(x)$$

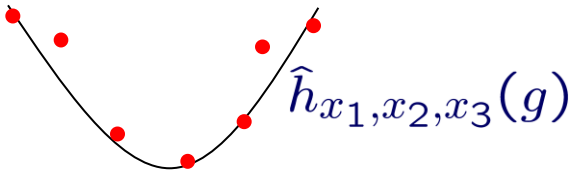
- Modell mit m Fehlertermen (**Residuen**) r_j

- Beispiel 1: Viele Probleme aus dem maschinellen Lernen
- Beispiel 2: Kurve an Datenpunkte anpassen

  Quadratische Kurve beschrieben durch drei Parameter $x = (x_1, x_2, x_3)^\top$

  Estimate $\hat{h}_j = x_1 g_j^2 + x_2 g_j + x_3$

  Gegeben m Datenpunkte $(g_j, h_j)^\top$



- Optimierungsproblem: $f(\boldsymbol{x}) = \frac{1}{2} \sum_{j=1}^{m} \underbrace{(\overbrace{x_1 g_j^2 + x_2 g_j + x_3}^{\hat{h}_j} - h_j)}_{r_j(\boldsymbol{x})}{}^2$

Least-Squares-Aufgaben

- Problemklasse hat eine nützliche Struktur

- Wir können die Residuen als einen **Residuenvektor** schreiben:

$$r(x) = (r_1(x), ..., r_m(x))^\top$$

- Die Zielfunktion lässt sich dann in Kurzform schreiben:

$$f(x) = \frac{1}{2}\|r\|^2$$

- Die Ableitungen der Residuen bilden die $m \times n$ **Jacobimatrix**

$$J(x) = \begin{pmatrix} \nabla r_1(x)^\top \\ \vdots \\ \nabla r_m(x)^\top \end{pmatrix}$$

- Damit lässt sich z.B. der Gradient von $f(x)$ kompakt darstellen:

$$\nabla f(x) = \frac{1}{2} \sum_{j=1}^{m} 2 r_j(x) \nabla r_j(x) = J(x)^\top r(x)$$

Least-Squares-Aufgaben

- Auch die Hesse-Matrix bekommt eine besondere Form:

$$H(x) = \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^\top + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

$$= J(x)^\top J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

- Teil der Hesse-Matrix lässt sich aus Ableitungen 1. Ordnung berechnen!

- Spezialfall linearer Residuen (**linear least squares**): $r_j = A_j x - b_j$
  (entspricht einem quadratischen Optimierungsproblem)

- In diesem Fall $J(x) = A$ und der zweite Term der Hesse-Matrix fällt weg

- Optimierungsproblem durch lineares Gleichungssystem beschrieben:

$$J^\top J x = J^\top b$$

Gauss-Newton-Verfahren

- Für nichtlineare Residuen bietet sich wieder eine Approximation der Hesse-Matrix an:

$$H(x) = J(x)^\top J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$
$$\approx J(x)^\top J(x)$$

→ Approximation des Newton-Verfahrens nur mit 1. Ableitungen

- Anders als bei BFGS durch Ausnutzen der speziellen Problemstruktur (Summe quadrierter Residuen)

- Durch Vernachlässigen des zweiten Terms, werden die Residuen lokal an der aktuellen Lösung x^k linearisiert:

$$r_j(x^k + d^k) = r_j(x^k) + J(x^k) d^k + \frac{1}{2} (d^k)^\top \nabla^2 r_j(x^k) d^k + \ldots$$

Gauss-Newton-Verfahren

- Linearisierung der Residuen

  $$r_j(x^k + d^k) \approx r_j(x^k) + J(x^k)d^k$$

  führt daher zu einem linearen Least-Squares-Problem mit Parametern d^k

- Lineares Gleichungssystem

  $$J^\top(x^k)J(x^k)d^k = -J^\top(x^k)r(x^k)$$

- Die Lösung d^k ist eine Abstiegsrichtung, mit der x^k verbessert wird:

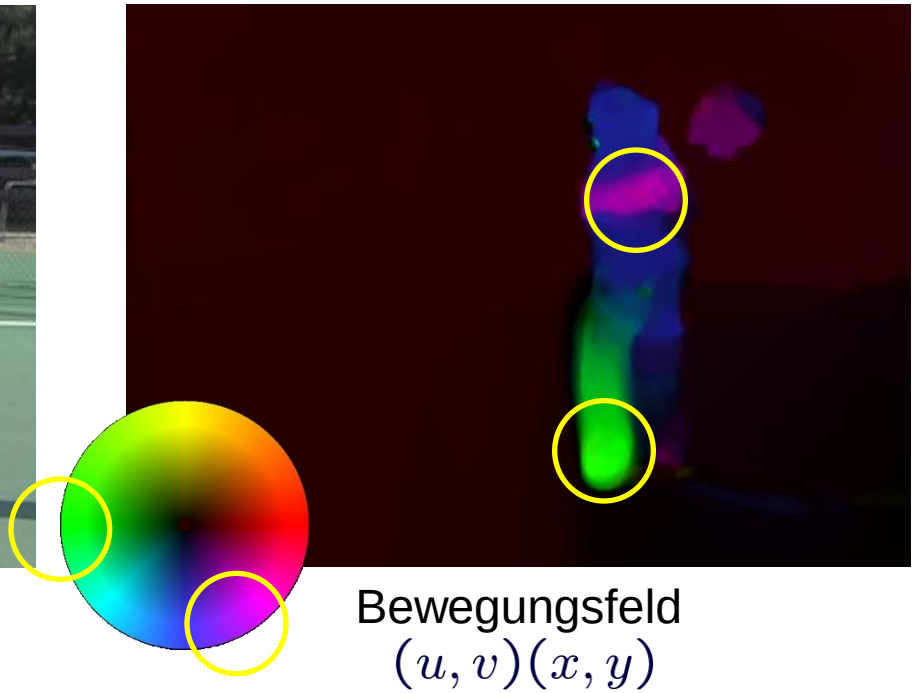
  $$x^{k+1} = x^k + \tau^k d^k$$

- Gauss-Newton löst also eine Sequenz von linearen Gleichungssystemen

- Gauss-Newton funktioniert, wenn alle Eigenwerte von $J^\top J$ deutlich positiv sind. Bei Eigenwerten nahe null → Levenberg-Marquardt

Beispiel für Gauss-Newton-Verfahren: Bewegungsschätzung

Zwei Bilder einer Bildsequenz
$I_1(x,y), I_2(x,y)$



Bewegungsfeld
$(u,v)(x,y)$

- Optimierungsproblem:

$$f(u,v) = \int (I_2(x+u(x,y), y+v(x,y)) - I_1(x,y))^2 + \alpha \left(|\nabla u(x,y)|^2 + |\nabla v(x,y)|^2 \right) \, dxdy$$

Zusammenfassung

- Das **Newton-Verfahren** berücksichtigt neben dem Gradienten zusätzlich die Krümmung zur Bestimmung der Abstiegsrichtung

- **Quasi-Newton-Verfahren** approximieren die Krümmung und erreichen so schnelle Konvergenz bei niedrigeren Kosten pro Iteration

- Auch das **Gauss-Newton-Verfahren** approximiert die Krümmung. Hierbei wird die spezielle Struktur von Least-Squares-Problemen ausgenutzt.

Übungsaufgabe

1. Laden Sie das Bild `puppy.png` mit `imread` aus der Library `scipy.misc` und lassen Sie es sich mit `pyplot's imshow` funktion anzeigen. Das Bild ist offenbar etwas verrauscht. Wir würden gerne ein schöneres Bild rekonstruieren, was auf das folgende Optimierungsproblem hinausläuft:

$$f(x) = \sum_{i,j} \left(\sqrt{(x_{ij} - y_{ij})^2 + 1} + \frac{1}{2} \sqrt{(x_{ij} - x_{i+1j})^2 + (x_{ij} - x_{ij+1})^2 + 1} \right)$$

   wobei y_{ij} die gemessenen Bildpixel darstellen und x_{ij} die Bildpixel des verbesserten Bildes.
   - Berechnen Sie den Gradienten der Funktion.
   - Optimieren Sie die Funktion mit Gradientenabstieg. Starten Sie mit dem Eingangsbild als Startpunkt, also $x = y$ (wandeln Sie am besten das Eingangsbild in `float64` um). Bestimmen Sie die Schrittweite wieder mit Backtracking Line Search. Speichern Sie die gewählte Schrittweite bei jeder Iteration und lassen Sie sich die Schrittweiten über die Iterationen später anzeigen. Verfolgen Sie auch die Konvergenz (also den Funktionswert über die Iterationen) und schauen Sie, wie sich die Lösung x in Form des Bildes über die Iterationen verändert.

2. Versuchen sie nun die Funktion von Scipy's `optimize` optimieren zu lassen. Vergleichen sie das BFGS und das Truncated Newton Verfahren. Welches benötigt weniger schritte? Wie lange dauert ein Schritt?