# Nichtlineare Programmierung

## Gradientenabstieg und Newton-Methode

Deutsch

**Wolfgang Böhme**

wolfgang.boehme@economic-modeling.de

www.economic-modeling.de

25. Oktober 2026

---

Dieses Dokument führt in die iterativen Verfahren der nichtlinearen Optimierung ein. Voraussetzung: Grundkenntnisse der Differentialrechnung und linearen Algebra.

Dokumentationen dieser Webseite wurden mit Unterstützung der KI Claude (Anthropic) entwickelt.

# Inhaltsverzeichnis


**1 Einführung und Motivation** **2**

**2 Mathematische Grundlagen** **2**
2.1 Das allgemeine nichtlineare Problem . . . 2
2.2 Optimalitätsbedingungen . . . 2
2.3 Gradient und Hesse-Matrix . . . 3
2.4 Numerische Differentiation (im Tool) . . . 3

**3 Gradientenabstieg** **3**
3.1 Grundidee . . . 3
3.2 Armijo-Liniensuche (Backtracking) . . . 4
3.3 Konvergenzrate . . . 4

**4 Newton-Methode** **4**
4.1 Grundidee . . . 4
4.2 Newton-Richtung . . . 5
4.3 Quadratische Konvergenz . . . 5
4.4 Einschränkungen der Newton-Methode . . . 5

**5 Vollständiges Beispiel: Beide Methoden** **6**
5.1 Problemstellung . . . 6
5.2 Methode 1: Gradientenabstieg . . . 6
5.2.1 Iteration $0 \to 1$ . . . 6
5.2.2 Iteration $1 \to 2$ . . . 6
5.2.3 Konvergenzverlauf . . . 7
5.3 Methode 2: Newton-Methode . . . 7
5.3.1 Iteration $0 \to 1$ . . . 7
5.3.2 Iterationstabelle Newton . . . 7
5.4 Grafische Darstellung: Optimierungspfade . . . 8
5.5 Rosenbrock-Funktion: Ein schwieriger Fall . . . 8

**6 Toolbeschreibung** **9**
6.1 Funktionseingabe . . . 9
6.2 Vordefinierte Beispielfunktionen . . . 9
6.3 Bedienungsanleitung . . . 9
6.4 Ausgabe-Interpretation . . . 10

**7 Zusammenfassung und Methodenvergleich** **10**

**8 Konturplot – Visualisierung der Zielfunktion** **11**
8.1 Was ist ein Konturplot? . . . 11
8.2 Farbcodierung im Tool . . . 11
8.3 Niveaukurven (weiße Linien) . . . 11
8.4 Gradient und Niveaukurven . . . 12
8.5 Optimierungspfade im Konturplot . . . 12
8.6 Interaktion: Startpunkt per Klick setzen . . . 12
8.7 Vollständige Legende: Konturplot-Beispiel . . . 13

# 1 Einführung und Motivation

Viele reale Optimierungsprobleme lassen sich **nicht** als lineare Programme formulieren. Die Zielfunktion oder die Nebenbedingungen sind nichtlinear – etwa quadratische Kostenfunktionen, Produktionsfunktionen mit abnehmenden Grenzrenditen oder physikalische Modelle.

**Typische nichtlineare Optimierungsprobleme**

- **Portfolio:** min $\mathbf{x}^\top \Sigma \mathbf{x} - \mu^\top \mathbf{x}$ (quadratische Optimierung)
- **Neuronale Netze:** min $\sum_i \ell(f_\theta(\mathbf{x}_i), y_i)$ (nichtlinearer Verlust)
- **Ingenieurwesen:** min Gewicht($\mathbf{x}$) unter nichtlinearen Festigkeitsbedingungen
- **Statistik:** max $\log L(\theta|\text{Daten})$ (Maximum-Likelihood-Schätzung)

Im Gegensatz zur linearen Programmierung gibt es bei nichtlinearen Problemen **keine allgemeine Garantie**, dass ein gefundenes lokales Optimum auch global optimal ist. Iterative Abstiegsverfahren nähern sich schrittweise einem Minimum an.

# 2 Mathematische Grundlagen

## 2.1 Das allgemeine nichtlineare Problem

**Definition: Unbeschränktes nichtlineares Problem**

Gesucht ist ein lokales Minimum der Funktion $f : \mathbb{R}^n \to \mathbb{R}$:

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

Dabei sei $f$ zweimal stetig differenzierbar ($f \in \mathcal{C}^2$).

## 2.2 Optimalitätsbedingungen

**Satz 1: Notwendige Bedingung erster Ordnung (FONC)**

Ist $\mathbf{x}^*$ ein lokales Minimum von $f$, so gilt:

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

Ein Punkt mit $\nabla f(\mathbf{x}^*) = \mathbf{0}$ heißt **stationärer Punkt**.

**Satz 2: Hinreichende Bedingung zweiter Ordnung (SOSC)**

Sei $\nabla f(\mathbf{x}^*) = \mathbf{0}$ und die Hesse-Matrix $\mathbf{H}(\mathbf{x}^*)$ **positiv definit**:

$$\mathbf{H}(\mathbf{x}^*) = \nabla^2 f(\mathbf{x}^*) \succ 0$$

Dann ist $\mathbf{x}^*$ ein **striktes lokales Minimum**.

## 2.3 Gradient und Hesse-Matrix

**Definition: Gradient und Hesse-Matrix**

Für $f : \mathbb{R}^n \to \mathbb{R}$ sind:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n, \qquad \mathbf{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

## 2.4 Numerische Differentiation (im Tool)

Da nicht immer analytische Ableitungen vorliegen, verwendet das Tool **numerische Differentiation**:

**Gradient (zentrale Differenzen, Ordnung $O(h^2)$):**

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \approx \frac{f(\mathbf{x} + h\,\mathbf{e}_i) - f(\mathbf{x} - h\,\mathbf{e}_i)}{2h}, \quad h = 10^{-5}$$

**Hesse-Matrix (zentrale Differenzen):**

$$\frac{\partial^2 f}{\partial x_i^2} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - 2f(\mathbf{x}) + f(\mathbf{x} - h\mathbf{e}_i)}{h^2}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + h\mathbf{e}_i - h\mathbf{e}_j) - f(\mathbf{x} - h\mathbf{e}_i + h\mathbf{e}_j) + f(\mathbf{x} - h\mathbf{e}_i - h\mathbf{e}_j)}{4h^2}$$

# 3 Gradientenabstieg

## 3.1 Grundidee

Der **negative Gradient** $-\nabla f(\mathbf{x})$ zeigt in die Richtung des steilsten Abstiegs. Der Gradientenabstieg bewegt sich iterativ in diese Richtung:

**Definition: Gradientenabstieg**

**Initialisierung:** Startpunkt $\mathbf{x}^{(0)}$, Toleranz $\varepsilon > 0$.
**Iteration** $k = 0, 1, 2, \ldots$:

1. Berechne Suchrichtung: $\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$
2. Bestimme Schrittweite $\alpha_k > 0$ (Liniensuche)
3. Aktualisiere: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$
4. Stopp falls $\|\nabla f(\mathbf{x}^{(k+1)})\| < \varepsilon$

## 3.2 Armijo-Liniensuche (Backtracking)

**Definition: Armijo-Bedingung**

Eine Schrittweite $\alpha$ erfüllt die **Armijo-Bedingung** (hinreichende Abstiegsbedingung), falls:

$$f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) \leq f(\mathbf{x}^{(k)}) + c \cdot \alpha \cdot \nabla f(\mathbf{x}^{(k)})^\top \mathbf{d}^{(k)}$$

mit $c \in (0,1)$ (typisch $c = 0{,}01$). Der **Backtracking-Algorithmus** startet mit $\alpha_0$ und halbiert solange ($\alpha \leftarrow \rho\alpha$, $\rho = 0{,}5$), bis die Bedingung erfüllt ist.

**Satz 3: Abstiegseigenschaft**

Mit der Armijo-Liniensuche gilt für jeden Schritt:

$$f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)})$$

solange $\|\nabla f(\mathbf{x}^{(k)})\| > 0$. Der Gradientenabstieg erzeugt eine streng monoton fallende Folge von Funktionswerten.

## 3.3 Konvergenzrate

**Satz 4: Lineare Konvergenz des Gradientenabstiegs**

Für eine stark konvexe Funktion $f$ mit $m \leq \lambda_{\min}(\mathbf{H}(\mathbf{x})) \leq \lambda_{\max}(\mathbf{H}(\mathbf{x})) \leq L$ gilt mit optimaler Schrittweite:

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2 \leq \left(\frac{L-m}{L+m}\right)^{2k} \|\mathbf{x}^{(0)} - \mathbf{x}^*\|^2$$

Die Konvergenzrate ist **linear** (geometrisches Fallen des Fehlers). Die Konvergenz ist langsam, wenn die Konditionszahl $\kappa = L/m$ groß ist (lang gezogene Ellipsen).

# 4 Newton-Methode

## 4.1 Grundidee

Die Newton-Methode approximiert $f$ lokal durch eine quadratische Funktion (Taylor-Entwicklung zweiter Ordnung) und minimiert diese:

$$f(\mathbf{x} + \mathbf{d}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{d} + \frac{1}{2} \mathbf{d}^\top \mathbf{H}(\mathbf{x}) \mathbf{d}$$

Minimum der Approximation: $\nabla_{\mathbf{d}}[\ldots] = \mathbf{H}(\mathbf{x})\mathbf{d} + \nabla f(\mathbf{x}) = \mathbf{0}$

**Definition: Newton-Methode**

**Initialisierung:** Startpunkt $\mathbf{x}^{(0)}$, Toleranz $\varepsilon > 0$.
**Iteration** $k = 0, 1, 2, \ldots$:

1. Berechne Gradient $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$ und Hesse-Matrix $\mathbf{H}^{(k)} = \mathbf{H}(\mathbf{x}^{(k)})$
2. Löse das lineare Gleichungssystem: $\mathbf{H}^{(k)}\mathbf{d}^{(k)} = -\mathbf{g}^{(k)}$
3. Bestimme Schrittweite $\alpha_k$ (Liniensuche oder $\alpha_k = 1$)
4. Aktualisiere: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$
5. Stopp falls $\|\mathbf{g}^{(k+1)}\| < \varepsilon$

## 4.2 Newton-Richtung

**Definition: Newton-Richtung**

Die **Newton-Richtung** $\mathbf{d}_N^{(k)} = -[\mathbf{H}^{(k)}]^{-1}\mathbf{g}^{(k)}$ ist eine Abstiegsrichtung, falls $\mathbf{H}^{(k)} \succ 0$:

$$(\mathbf{g}^{(k)})^\top \mathbf{d}_N^{(k)} = -(\mathbf{g}^{(k)})^\top [\mathbf{H}^{(k)}]^{-1}\mathbf{g}^{(k)} < 0$$

## 4.3 Quadratische Konvergenz

**Satz 5: Lokale quadratische Konvergenz**

Sei $\mathbf{x}^*$ ein striktes lokales Minimum mit $\mathbf{H}(\mathbf{x}^*) \succ 0$. Falls $\mathbf{x}^{(0)}$ hinreichend nah an $\mathbf{x}^*$ liegt, konvergiert die Newton-Methode mit $\alpha_k = 1$ **quadratisch**:

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C \cdot \|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2$$

für eine Konstante $C > 0$. Dies bedeutet: Verdopplung der gültigen Stellen pro Iteration.

**Vergleich:** Gradientenabstieg braucht typischerweise hunderte Iterationen; Newton konvergiert in 5–20 Iterationen. Dafür ist jede Newton-Iteration teurer: Sie erfordert die Berechnung und Invertierung der Hesse-Matrix ($O(n^3)$ für $n$ Variablen).

## 4.4 Einschränkungen der Newton-Methode

- Die Hesse-Matrix muss **positiv definit** sein (sonst keine Abstiegsrichtung).
- Nahe Sattelpunkten oder bei nicht-konvexen Funktionen kann die Methode **divergieren**.
- Bei großen $n$ ist die Invertierung der Hesse-Matrix teuer $\Rightarrow$ **Quasi-Newton-Methoden** (BFGS) approximieren $\mathbf{H}^{-1}$.

# 5 Vollständiges Beispiel: Beide Methoden

## 5.1 Problemstellung

**Beispielfunktion: Quadratisches Minimum**

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

**Startpunkt:** $\mathbf{x}^{(0)} = (0, 0)^\top$
Diese Funktion hat ein eindeutiges globales Minimum bei $\mathbf{x}^* = (2, 1)^\top$ mit $f(\mathbf{x}^*) = 0$.

**Analytische Ableitungen:**

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2(x_1 - 2) \\ 4(x_2 - 1) \end{pmatrix}, \qquad \mathbf{H}(\mathbf{x}) = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$$

Am Startpunkt $\mathbf{x}^{(0)} = (0, 0)^\top$:

$$\nabla f(\mathbf{x}^{(0)}) = \begin{pmatrix} -4 \\ -4 \end{pmatrix}, \qquad \mathbf{H}(\mathbf{x}^{(0)}) = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, \qquad f(\mathbf{x}^{(0)}) = 4 + 2 = 6$$

## 5.2 Methode 1: Gradientenabstieg

### 5.2.1 Iteration 0 → 1

**Suchrichtung:** $\mathbf{d}^{(0)} = -\nabla f(\mathbf{x}^{(0)}) = (4, 4)^\top$
**Liniensuche (Armijo, $c = 0{,}01$, $\rho = 0{,}5$, $\alpha_0 = 1$):**
Teste $\alpha = 1$: $\mathbf{x} = (4, 4)^\top$. $f(4, 4) = (4 - 2)^2 + 2(4 - 1)^2 = 4 + 18 = 22$.
Armijo-Bedingung: $22 \leq 6 + 0{,}01 \cdot 1 \cdot ((-4)(4) + (-4)(4)) = 6 - 0{,}32 = 5{,}68$? Nein.
Teste $\alpha = 0{,}5$: $\mathbf{x} = (2, 2)^\top$. $f(2, 2) = 0 + 2 = 2 \leq 6 - 0{,}16 = 5{,}84$? Ja.
$\mathbf{x}^{(1)} = (2, 2)^\top$, $f(\mathbf{x}^{(1)}) = 2$, $\nabla f = (0, 4)^\top$, $\|\nabla f\| = 4$.

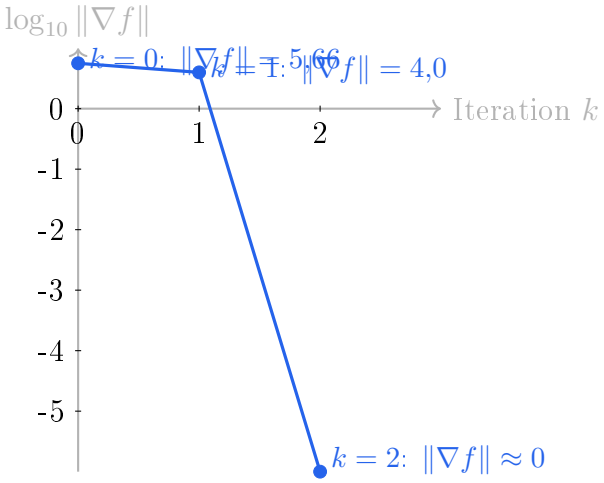
### 5.2.2 Iteration 1 → 2

$\mathbf{d}^{(1)} = (0, -4)^\top$. Liniensuche liefert $\alpha = 0{,}5$:
$\mathbf{x}^{(2)} = (2, 2 - 2)^\top = (2, 0)^\top$. $f(2, 0) = 2$. $\nabla f = (0, -4)^\top$.
Nochmals: $\alpha = 0{,}5$: $\mathbf{x}^{(3)} = (2, 2)^\top$... Dieses Zick-Zack ist typisch bei unterschiedlichen Eigenwerten.
**Tatsächliche Iterationen** (numerisch mit Backtracking):

| $k$ | $x_1^{(k)}$ | $x_2^{(k)}$ | $f(\mathbf{x}^{(k)})$ | $\|\nabla f\|$ | $\alpha_k$ |
|---|---|---|---|---|---|
| 0 | 0,0000 | 0,0000 | 6,0000 | 5,6569 | 0,5000 |
| 1 | 2,0000 | 2,0000 | 2,0000 | 4,0000 | 0,2500 |
| 2 | 2,0000 | 1,0000 | 0,0000 | 0,0000 | – |

**Kovergenz in 2 Iterationen** (da Hesse-Matrix diagonal und Suchrichtung sofort auf Minimum zeigt).

### 5.2.3 Konvergenzverlauf




## 5.3 Methode 2: Newton-Methode

### 5.3.1 Iteration $0 \to 1$

**Newton-Gleichungssystem:** $\mathbf{H}^{(0)}\mathbf{d}^{(0)} = -\nabla f(\mathbf{x}^{(0)})$

$$\begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix} \mathbf{d}^{(0)} = -\begin{pmatrix} -4 \\ -4 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

$$\mathbf{d}^{(0)} = \begin{pmatrix} 4/2 \\ 4/4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

**Schrittweite:** Liniensuche mit $\alpha_0 = 1$:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + 1 \cdot \mathbf{d}^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$f(\mathbf{x}^{(1)}) = (2-2)^2 + 2(1-1)^2 = 0 = f^*$$

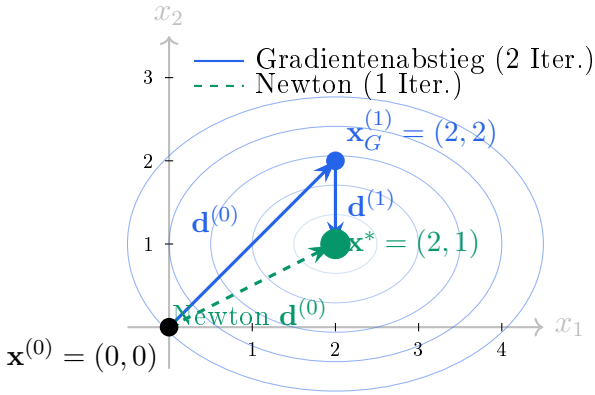
Die Newton-Methode erreicht das Optimum in **einer einzigen Iteration**!

**Warum?** Die Funktion ist quadratisch, daher ist die Taylor-Approximation zweiter Ordnung **exakt**. Newton findet das Minimum quadratischer Funktionen immer in einem Schritt.

### 5.3.2 Iterationstabelle Newton

| $k$ | $x_1^{(k)}$ | $x_2^{(k)}$ | $f(\mathbf{x}^{(k)})$ | $\|\nabla f\|$ | $\det(\mathbf{H})$ |
|---|---|---|---|---|---|
| 0 | 0,0000 | 0,0000 | 6,0000 | 5,6569 | 8 |
| 1 | 2,0000 | 1,0000 | 0,0000 | 0,0000 | 8 |

## 5.4 Grafische Darstellung: Optimierungspfade




## 5.5 Rosenbrock-Funktion: Ein schwieriger Fall

**Rosenbrock-Funktion (Bananentalfunktion)**

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

**Minimum:** $\mathbf{x}^* = (1,1)^\top$, $f(\mathbf{x}^*) = 0$. **Startpunkt:** $(-1, 0)$.
Die Rosenbrock-Funktion hat ein lang gezogenes, stark gekrümmtes Tal. Der Gradientenabstieg ist hier extrem langsam (Konditionszahl $\kappa \gg 1$), während Newton deutlich schneller konvergiert.

**Analytische Ableitungen:**

$$\frac{\partial f}{\partial x_1} = -400x_1(x_2 - x_1^2) - 2(1 - x_1), \qquad \frac{\partial f}{\partial x_2} = 200(x_2 - x_1^2)$$

$$\mathbf{H} = \begin{pmatrix} -400(x_2 - x_1^2) + 800x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

**Am Startpunkt** $(-1, 0)$**:**

$$\nabla f(-1,0) = \begin{pmatrix} -400(-1)(-1) - 2(2) \\ 200(-1) \end{pmatrix} = \begin{pmatrix} -404 \\ -200 \end{pmatrix}$$

$$\mathbf{H}(-1,0) = \begin{pmatrix} -400(-1) + 800 + 2 & 400 \\ 400 & 200 \end{pmatrix} = \begin{pmatrix} 1202 & 400 \\ 400 & 200 \end{pmatrix}$$

**Newton-Schritt:** $\mathbf{d} = -\mathbf{H}^{-1}\nabla f$. $\det(\mathbf{H}) = 1202 \cdot 200 - 400^2 = 240400 - 160000 = 80400$.

$$\mathbf{H}^{-1} = \frac{1}{80400}\begin{pmatrix} 200 & -400 \\ -400 & 1202 \end{pmatrix}$$

$$\mathbf{d} = -\frac{1}{80400}\begin{pmatrix} 200(-404) + (-400)(-200) \\ -400(-404) + 1202(-200) \end{pmatrix} = -\frac{1}{80400}\begin{pmatrix} -80800 + 80000 \\ 161600 - 240400 \end{pmatrix} = -\frac{1}{80400}\begin{pmatrix} -800 \\ -78800 \end{pmatrix} \approx$$

Nach dem ersten Newton-Schritt: $\mathbf{x}^{(1)} \approx (-0{,}990, 0{,}980)^\top$ – deutlicher Fortschritt auf das Minimum zu!

| Methode | Typische Iterationen (Rosenbrock) | Konvergenzrate |
|---|---|---|
| Gradientenabstieg | $> 1000$ | Linear |
| Newton-Methode | 15–30 | Quadratisch |

# 6 Toolbeschreibung

**NLP-Tool: Gradientenabstieg & Newton-Methode**

Das Tool ermöglicht die interaktive Visualisierung und den Vergleich beider Optimierungsverfahren für beliebige zweidimensionale Funktionen.

## 6.1 Funktionseingabe

Die Zielfunktion wird als **JavaScript-Ausdruck** eingegeben. Verfügbare Operatoren und Funktionen:

| Syntax | Bedeutung |
|---|---|
| `x1`, `x2` | Entscheidungsvariablen |
| `**` | Potenz (z. B. `x1**2`) |
| `Math.sqrt(x)`, `Math.abs(x)` | Wurzel, Betrag |
| `Math.exp(x)`, `Math.log(x)` | Exponential, natürl. Logarithmus |
| `Math.sin(x)`, `Math.cos(x)` | Trigonometrische Funktionen |
| `Math.PI` | Kreiszahl $\pi$ |

## 6.2 Vordefinierte Beispielfunktionen

| Name | Ausdruck | Minimum |
|---|---|---|
| Quadratisch | `(x1-2)**2 + 2*(x2-1)**2` | $(2, 1)$, $f^* = 0$ |
| Rosenbrock | `100*(x2-x1**2)**2 + (1-x1)**2` | $(1, 1)$, $f^* = 0$ |
| Himmelblau | `(x1**2+x2-11)**2 + (x1+x2**2-7)**2` | 4 Minima |
| Booth | `(x1+2*x2-7)**2 + (2*x1+x2-5)**2` | $(1, 3)$, $f^* = 0$ |
| Matyas | `0.26*(x1**2+x2**2) - 0.48*x1*x2` | $(0, 0)$, $f^* = 0$ |
| Rastrigin | `20 + x1**2 - 10*cos(2πx1) + ...` | $(0, 0)$, multimodal |

## 6.3 Bedienungsanleitung

1. **Funktion wählen:** Preset-Button klicken oder eigenen Ausdruck eingeben.
2. **Startpunkt setzen:** Numerisch eingeben oder direkt in den Konturplot klicken.
3. **Methode wählen:** Gradientenabstieg, Newton oder beide vergleichen.
4. **Parameter anpassen:** Max. Iterationen, Toleranz $\varepsilon$, Startschrittweite $\alpha_0$.

5. **Berechnen:** Klick auf "▶ Optimieren".

6. **Ergebnisse:** Optimierungspfade im Konturplot, Konvergenzplot und Iterationstabelle.

### 6.4 Ausgabe-Interpretation

- **Konturplot:** Blau = kleiner Funktionswert, Rot = großer Funktionswert. Weiße Linien = Niveaukurven. Blauer Pfad = Gradientenabstieg, Grüner Pfad = Newton.
- **Konvergenzplot:** $\log_{10} \|\nabla f\|$ über Iterationen. Lineare Konvergenz: gerade Linie. Quadratische Konvergenz: stark fallende Kurve nahe dem Optimum.
- $\|\nabla f^*\|$**:** Sollte $< \varepsilon$ sein (Stoppkriterium). Sehr kleine Werte ($< 10^{-6}$) zeigen numerische Konvergenz an.
- $\det(\mathbf{H})$**:** Positiv = lokales Minimum, negativ = Sattelpunkt, null = entarteter Fall.

## 7 Zusammenfassung und Methodenvergleich

**Kernpunkte: Nichtlineare Optimierung**

1. **Stationäre Punkte** ($\nabla f = 0$) sind notwendig für Minima; $\mathbf{H} \succ 0$ ist hinreichend.
2. Der **Gradientenabstieg** bewegt sich stets in Richtung des steilsten Abstiegs – einfach, robust, aber langsam (lineare Konvergenz).
3. Die **Armijo-Liniensuche** garantiert bei jedem Schritt einen Funktionswertabfall (Abstiegseigenschaft).
4. Die **Newton-Methode** löst $\mathbf{Hd} = -\nabla f$ und konvergiert *quadratisch* nahe dem Optimum – sehr schnell, aber teurer pro Iteration.
5. Für **quadratische Funktionen** ist Newton exakt und konvergiert in einem Schritt.
6. Die **Konditionszahl** $\kappa(\mathbf{H})$ bestimmt die Konvergenzgeschwindigkeit des Gradientenabstiegs; bei großem $\kappa$ ist Newton klar überlegen.
7. **Multimodale Funktionen** (mehrere lokale Minima) erfordern Globalstrategien (Mehrstarts, evolutionäre Algorithmen).
8. Das **NLP-Tool** visualisiert beide Methoden interaktiv im 2D-Konturplot und eignet sich zum Verstehen des Konvergenzverhaltens.

| Eigenschaft | Gradientenabstieg | Newton |
|---|---|---|
| Konvergenzrate | Linear | Quadratisch |
| Iterationen (typisch) | 100–10000 | 5–30 |
| Kosten pro Iteration | $O(n)$ | $O(n^3)$ |
| Benötigt Hesse-Matrix | Nein | Ja |
| Robustheit | Hoch | Mittel |
| Geeignet für | Große $n$, rauschige $f$ | Kleine $n$, glatte $f$ |

# 8 Konturplot – Visualisierung der Zielfunktion

## 8.1 Was ist ein Konturplot?

Ein **Konturplot** (Höhenliniendarstellung) stellt eine Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ zweier Variablen in der Ebene dar. Da wir drei Dimensionen bräuchten ($x_1$, $x_2$, $f$), wird der Funktionswert stattdessen durch **Farbe** und **Konturlinien** (Niveaukurven) codiert.

**Definition: Niveaukurve (Konturlinie)**

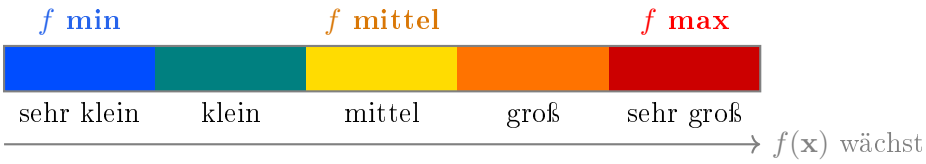
Die **Niveaukurve** zum Wert $c$ ist die Menge aller Punkte, an denen die Funktion genau den Wert $c$ annimmt:

$$\mathcal{L}_c = \{\mathbf{x} \in \mathbb{R}^2 \mid f(\mathbf{x}) = c\}$$

Niveaukurven sind die zweidimensionalen Analoga zu Höhenlinien einer topographischen Karte.

## 8.2 Farbcodierung im Tool

Das Tool verwendet ein **Spektrum von Blau über Gelb nach Rot**:

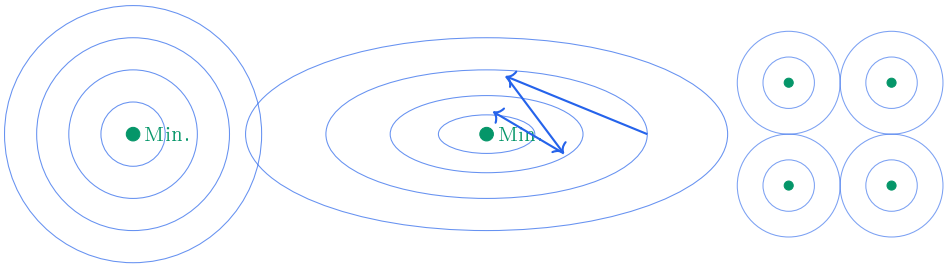



**Wichtig:** Die Farbe verwendet eine **logarithmische Skala**, d. h. $\log(f(\mathbf{x}) - f_{\min} + \epsilon)$ wird farbcodiert. Dies ermöglicht auch bei sehr großen Funktionswertbereichen (z. B. Rosenbrock) eine aussagekräftige Darstellung.

## 8.3 Niveaukurven (weiße Linien)

Die weißen Linien im Konturplot sind **Niveaukurven** mit gleichem Funktionswert. Sie werden mit dem Marching-Squares-Algorithmus berechnet: Das $x_1$-$x_2$-Gebiet wird in ein $200 \times 200$-Raster unterteilt, und für jeden Zelle wird geprüft, ob die Zielniveaukurve die Zelle schneidet.

**Was die Form der Niveaukurven aussagt:**




Kreise: $\kappa \approx 1$ Gradient schnell | Ellipsen: $\kappa \gg 1$ Gradient langsam | Mehrere Minima Lokale Optima

## 8.4 Gradient und Niveaukurven

**Satz 6: Orthogonalität von Gradient und Niveaukurve**

Der Gradient $\nabla f(\mathbf{x})$ steht **senkrecht** (orthogonal) auf der Niveaukurve durch $\mathbf{x}$ und zeigt in Richtung des steilsten Anstiegs.

*Proof.* Sei $\gamma(t)$ eine Kurve auf der Niveaukurve $\mathcal{L}_c$, d. h. $f(\gamma(t)) = c$ für alle $t$. Ableiten: $\nabla f(\gamma(t))^\top \dot{\gamma}(t) = 0$. Da $\dot{\gamma}$ tangential zur Niveaukurve ist, steht $\nabla f$ senkrecht darauf. □

**Konsequenz für den Gradientenabstieg:** Die Suchrichtung $-\nabla f$ steht stets senkrecht auf den Niveaukurven. Bei stark elliptischen Kurven (großes $\kappa$) zeigt $-\nabla f$ fast quer zur "Talrichtung" – daher das Zick-Zack.

## 8.5 Optimierungspfade im Konturplot

Das Tool zeichnet die Trajektorien beider Methoden in den Konturplot:

| Element | Farbe | Bedeutung |
|---|---|---|
| Pfad (Gradientenabstieg) | **Hellblau** | Suchrichtung $-\nabla f$ |
| Pfad (Newton) | **Hellgrün** | Newton-Richtung $-\mathbf{H}^{-1}\nabla f$ |
| Ausgefüllter Punkt | jeweilige Farbe | Startpunkt $\mathbf{x}^{(0)}$ |
| Weißer Punkt (Rand farbig) | jeweilige Farbe | Endpunkt / Optimum |
| Breite Niveaukurven | Weiß (halbtransp.) | Gleichwert-Linien $f = c$ |
| Blau → Rot | Farbverlauf | Kleiner → großer Funktionswert |

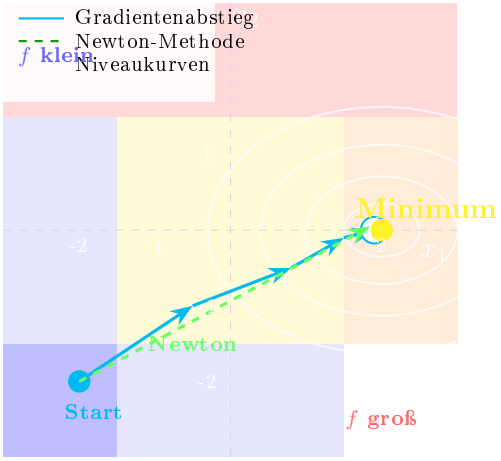
## 8.6 Interaktion: Startpunkt per Klick setzen

Ein **Klick** in den Konturplot setzt den Startpunkt $\mathbf{x}^{(0)}$ auf die angeklickte Position. Das Tool berechnet automatisch die Koordinaten aus der Pixelposition:

$$x_1 = \frac{\text{Pixelposition}_x}{\text{Breite}} \cdot 2R - R, \qquad x_2 = R - \frac{\text{Pixelposition}_y}{\text{Höhe}} \cdot 2R$$

wobei $R$ der eingestellte Plot-Bereich $\pm R$ ist. Dies ermöglicht das interaktive Erkunden verschiedener Startpunkte und deren Einfluss auf Konvergenz und Trajektorie.

## 8.7 Vollständige Legende: Konturplot-Beispiel




**Interpretation:**

- Der **Gradientenabstieg** (hellblauer Pfad) macht mehrere Schritte mit Richtungsänderungen entlang der Niveaukurven.
- **Newton** (gestrichelt) fährt direkt zum Minimum – bei quadratischen Funktionen in einem Schritt.
- Die **Dichte der Niveaukurven** zeigt die Steilheit: Eng beieinander liegende Kurven bedeuten starken Gradienten.
- Das **Minimum** liegt stets im blauen Bereich (niedrigster Funktionswert) – dem "Zentrum" des Konturplots.