Thema: Nichtlineare Optimierung – Gradientenmethode

Konsulentin: Selina Zschocke, VMB/203

1 Nichtlineare Optimierung

- Zielfunktion und/oder Nebenbedingungen nichtlinear
- Menge möglicher Lösungen der Optimierungsaufgabe nicht auf Eckpunkte des zulässigen Bereichs beschränkt

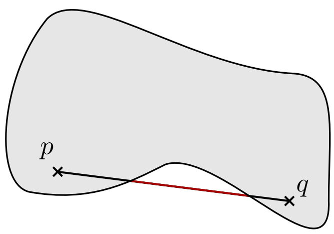
1.1 Konvergenz

- Annäherung an einen Grenzwert
- hier: Annäherung an Maximum/Minimum der Zielfunktion im Laufe des Iterationsprozesses
- Konvergenzkriterium k als Abbruchkriterium: definierte Bedingung zum Feststellen der Konvergenz, z.B.

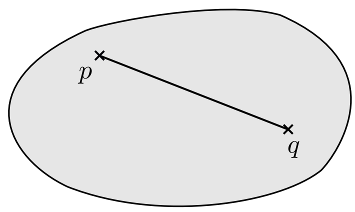
$$|z\left(\underline{x}^{[n+1]}\right) - z\left(\underline{x}^{[n]}\right)| < k \tag{1}$$

1.2 Konvexität

- Konvexes Optimierungsproblem
  - Zielfunktion und zulässiger Bereich sind konvex, siehe Abb. 1
  - Existenz eines globalen Optimums ohne zusätzliche lokale Optima gesichert
- Ingenieurpraxis: meist nicht bekannt ob das zu lösende Problem ein konvexes Optimierungsproblem darstellt



(a) nichtkonvexe Menge



(b) konvexe Menge

Abbildung 1: Konvexität von Mengen

2 Gradientenmethode

2.1 Einführung

- gerichtetes Suchverfahren, d.h. Verbesserung des Wertes der Zielfunktion $z(\underline{x})$ in jedem Schritt n

$$z\left(\underline{x}^{[n+1]}\right) < z\left(\underline{x}^{[n]}\right) \text{ (bei Minimumsuche)}$$
$$\underline{x}^{[n+1]} = \underline{x}^{[n]} + \lambda \cdot \underline{y}$$

mit

$$\lambda = \text{ Schrittweite}$$
$$\underline{y} = \text{ Suchrichtung in Abhängigkeit des Gradienten von } z \text{ in } \underline{x}^{[n]}$$

- **Gradient** – Vektor der partiellen Ableitungen von z zeigt in die Richtung des maximalen Anstiegs von z

$$\text{grad } z\left(\underline{x}^{[n]}\right) = \left.\begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{bmatrix}\right|_{x^{[n]}}$$

- Suchrichtung $\underline{y}$
  - Maximumsuche $\underline{y} = +\text{grad } z\left(\underline{x}^{[n]}\right)$
  - Minimumsuche $\underline{y} = -\text{grad } z\left(\underline{x}^{[n]}\right)$
- Voraussetzung: Differenzierbarkeit von $z(\underline{x})$
  - im Ingenieurkontext nicht immer gegeben
  - Alternative: Differentialquotient durch Differenzenquotient (numerische Näherung) ersetzten, Berechnung der part. Ableitungen entfällt
- Vorteil
  - effizienter als systematische und einfache (Zufalls-) Suche
- Nachteile
  - globales Optimum wird nicht immer gefunden
  - Startpunktabhängigkeit (mögliche Lösung: wiederholte Ausführung mit unterschiedlichen Startpunkten, siehe Abb. 2)

2.2 Einfache Gradientenmethode

- Schrittweite λ konstant

$$\underline{x}^{[n+1]} = \underline{x}^{[n]} \pm \lambda \cdot \frac{\text{grad } z\left(\underline{x}^{[n]}\right)}{\left|\text{grad } z\left(\underline{x}^{[n]}\right)\right|}$$

- Problem: Festlegung von λ hat einen starken Einfluss auf Effizienz und Effektivität
  - λ zu klein $\rightarrow$ langsame Konvergenz
  - λ zu groß $\rightarrow$ Überspringen des Optimums (siehe Abb. 3)
- Verbesserung ohne a priori verfügbares Wissen: λ mittels Erfolgsprüfung steuern

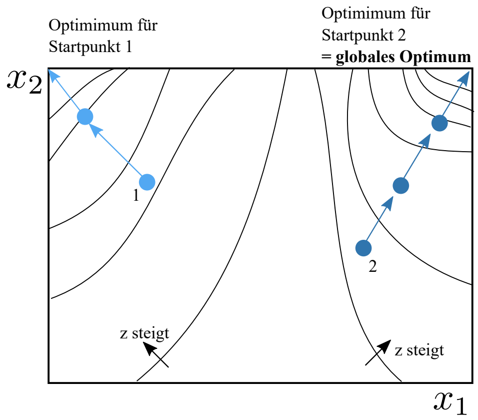


Abbildung 2: Startpunktabhängigkeit

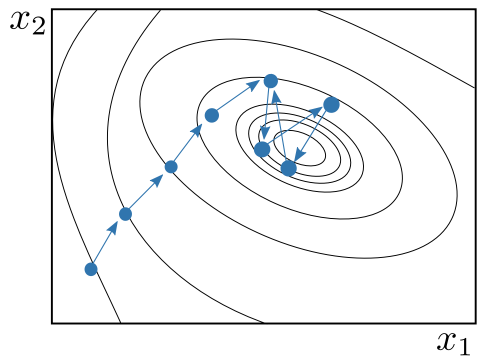


Abbildung 3: Überspringen des Optimums

2.3 Optimale Gradientenmethode

- Bestimmung der „optimalen Schrittweite“ λ_{opt} in Abhängigkeit von ZF–Werten

$$z\left(\underline{x}^{[n]} \pm \lambda \cdot \text{grad } z(\underline{x}^{[n]})\right) \rightarrow \text{opt. (nur abhäng. von } \lambda) \tag{2}$$

2.4 Diagonalschrittmethode

- Erweiterung des optimalen Gradienten durch Einfügen zusätzlicher Schritte mit

$$\underline{y}^{[n+1]} = \underline{x}^{[n]} - \underline{x}^{[n-2]} \tag{3}$$

- entspricht der 3. Seite des Dreiecks

2.5 Vorgehen am Rand des zulässigen Bereiches

- Rand ist erreicht, Gradient zeigt in den nichtzulässigen Bereich
- gesucht: Schrittrichtung im zul. Bereich mit Verbesserung des ZF-Wertes

- Methoden
  - systematische Richtungssuche
    - aufwändig (vor allem bei hochdimensionalen $\underline{x}$, da sehr viele Winkel innerhalb Hypersphäre abgesucht werden müssen)
    - sinnvoll bei nicht differenzierbaren NB
  - Zufallsauswahl (bei komplizierten Bereichsgrenzen)
  - Straffunktionen
  - Umrandungsgradient
    - Korrektur von grad $z\left(\underline{x}^{[n]}\right)$ durch Gradienten der verletzen NB mit Wichtungsfaktoren
    - Erläuterung an Prinzipskizze (Skript Abb. 3.14)
      - $\underline{b} = -\operatorname{grad} z(\underline{x}_n) \cdot \lambda$ – Schritt in den unzulässigen Bereich
      - $\underline{c} = -w_i \cdot \operatorname{grad} g_i(\underline{x}_n)$ – Korrektur des Vektors $\underline{b}$ in den zulässigen Bereich
      - $\underline{y} = \underline{b} + \underline{c}$ – Neuausrichtung der Suche

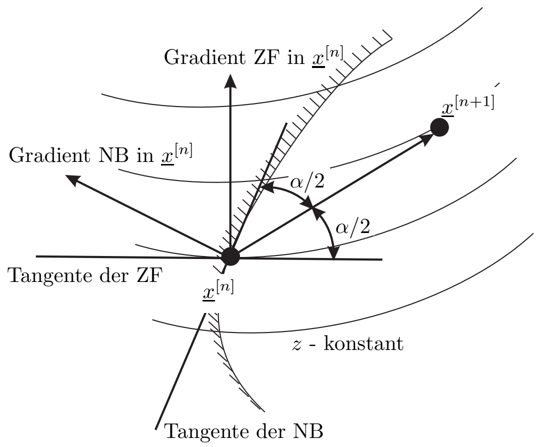


Abbildung 4: Umrandungsgradient

- Anwendung des Umrandungsgradienten für mehrere NB:

$$\underline{y} = -\operatorname{grad} z\left(\underline{x}^{[n]}\right) - \sum_{i=1}^{n_g} w_i \cdot I_i\left(\underline{x}^{[n]}\right) \cdot \operatorname{grad} g_i\left(\underline{x}^{[n]}\right)$$

$$I_i\left(\underline{x}^{[n]}\right) = \begin{cases} 0 & \text{NB nicht aktiv} \\ 1 & \text{NB aktiv} \end{cases}$$

$$w_i = \text{ Gewichtsfaktoren}$$

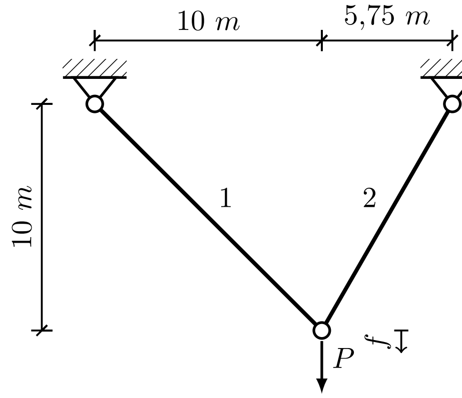
$$n_g = \text{ Anzahl der Ungleichungsnebenbedingungen}$$

- bei nur einer verletzten NB, Wahl der Wichtungsfaktoren z.B. als Winkelhalbierende zwischen der Tangente der NB und der Tangente von z im Punkt $\underline{x}^{[n]}$

$$w = \frac{\left|\operatorname{grad} z\left(\underline{x}^{[n]}\right)\right|}{\left|\operatorname{grad} g\left(\underline{x}^{[n]}\right)\right|}$$

3 Beispiel – Fachwerk

3.1 System



- $P = 100\,\mathrm{kN}$
- $E = 10^6\,\mathrm{kN/cm^2}$
- $\sigma_{zul} = 100\,\mathrm{kN/cm^2}$ (Maximalspannung)
- $f_{zul} = 1\,\mathrm{cm}$ (Maximalverschiebung)
- Stablängen $l_1 = 14{,}14\,\mathrm{m}$ und $l_2 = 11{,}53\,\mathrm{m}$

3.2 Optimierungsaufgabe

- Entwurfsvariablen: Querschnittsflächen x_1, x_2
- Zielfunktion — Minimum des Volumens (bei konstanter Dichte äquivalent zu masseoptimalem Problem)

$$z(x_1, x_2) = x_1 \cdot l_1 + x_2 \cdot l_2 \rightarrow \min$$

- Nebenbedingungen
  - Spannungsnebenbedingungen

    Stabkräfte (Kräftegleichgewicht am Knoten)

$$S_1 = 51{,}61\ \mathrm{kN}$$
$$S_2 = 73{,}24\ \mathrm{kN}$$

    Stabspannungen

$$\sigma_k = \frac{S_k}{x_k}$$

    Bedingung

$$\text{mit } \sigma_k \leq \sigma_{zul} \text{ und } x_k = \frac{S_k}{\sigma_k}$$
$$g_1(x_1) = 0{,}5161\ \mathrm{cm}^2 - x_1 \leq 0$$
$$g_2(x_2) = 0{,}7324\ \mathrm{cm}^2 - x_2 \leq 0$$

  - Verschiebungsnebenbedingungen

    Knotenverschiebung (Prinzip der virtuellen Verrückungen, Arbeitsgleichung)

$$f = \frac{\overline{S}_1 \cdot S_1 \cdot l_1}{x_1 \cdot E} + \frac{\overline{S}_2 \cdot S_2 \cdot l_2}{x_2 \cdot E}$$

Bedingung

$$f \leq f_{zul}$$
$$g_3(x_1, x_2) = \frac{1\ \text{cm}^3}{x_1} + \frac{1{,}6414\ \text{cm}^3}{x_2} - 2{,}6527\ \text{cm} \leq 0$$

3.3 Grafische Interpretation

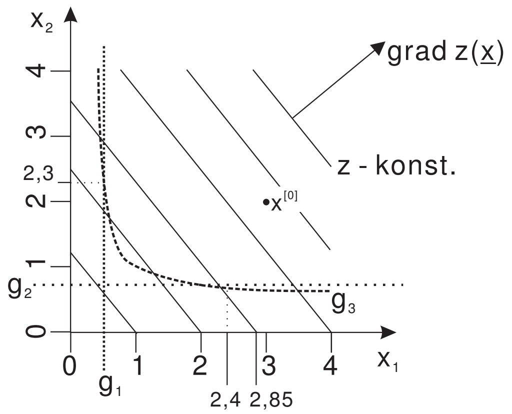


Abbildung 5: Ausgangsproblem

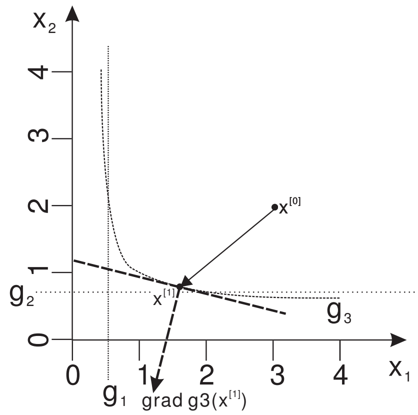


Abbildung 6: 1. Iterationsschritt

3.4 Anwendung optimales Gradientenverfahren

- Startpunkt innerhalb des zulässigen Bereichs frei gewählt — Abb. 5

$$\underline{x}^{[0]} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$
$$\text{grad } z\,(\underline{x}) = \begin{pmatrix} \frac{\partial z}{\partial x_1^{[0]}} \\ \frac{\partial z}{\partial x_2^{[0]}} \end{pmatrix} = \begin{pmatrix} 1414{,}2 \\ 1153{,}5 \end{pmatrix} \quad \text{hier: lineare ZF} \rightarrow \text{konstanter Gradient}$$

- 1. Iterationsschritt: Bestimmung von λ_{opt} (Abb. 6)

$$\underline{x}^{[1]} = \underline{x}^{[0]} - \lambda_{opt}^{[1]} \cdot \text{grad } z\left(\underline{x}^{[0]}\right)$$
$$\underline{x}^{[1]} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} - \lambda_{opt}^{[1]} \cdot \begin{pmatrix} 1414{,}2 \\ 1153{,}5 \end{pmatrix}$$

λ_{opt} — bis zum Rand, da Zielfunktion linear

Welches ist die aktive NB? — Berechnen des Abstands vom aktuellen Punkt bis zu allen Nebenbedinungen ($= \lambda_{opt}$) - kürzester ist Maßgebend — hier: g_3

$\underline{x}^{[1]}$ in g_3

$$0 = \frac{1}{3 - 1414{,}2 \cdot \lambda_{opt}} + \frac{1{,}6414}{2 - 1153{,}2 \cdot \lambda_{opt}} - 2{,}6527$$

$$\rightarrow \lambda_{opt}^{[1]} = 0{,}001025 \text{ (Nullstellensuche)}$$

$$\underline{x}^{[1]} = \begin{pmatrix} 1{,}5503 \\ 0{,}8176 \end{pmatrix}$$

$$z\left(\underline{x}^{[1]}\right) = 3135{,}6$$

- 2. Iteration — Nutzung **Umrandungsgradient** da der zulässige Bereich bei Fortschreiten in Richtung Gradienten verlassen wird

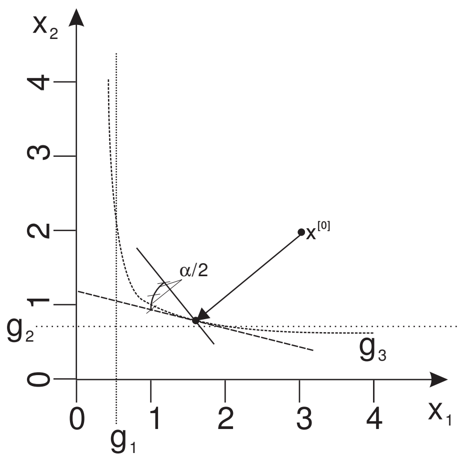


Abbildung 7: 2. Iterationsschritt - Winkelhalbierende

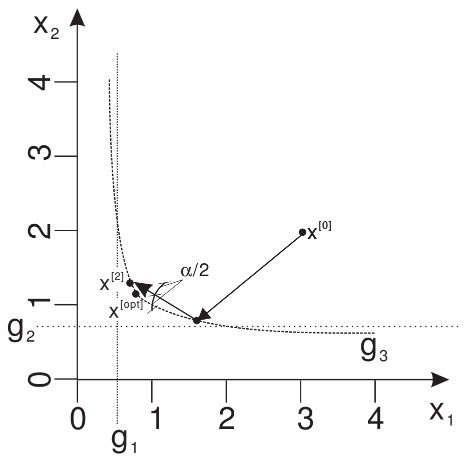


Abbildung 8: weitere Iterationen

$$\text{grad } g_3\left(\underline{x}\right) = \begin{pmatrix} \frac{-1}{x_1^2} \\ \frac{-1{,}6414}{x_2^2} \end{pmatrix} \qquad \rightarrow \text{grad } g_3\left(\underline{x}^{[1]}\right) = \begin{pmatrix} -0{,}4161 \\ -2{,}4556 \end{pmatrix}$$

Wichtungsfaktor — Abb. 7

$$w = \frac{\left|\text{grad } z\left(\underline{x}^{[1]}\right)\right|}{\left|\text{grad } g_3\left(\underline{x}^{[1]}\right)\right|} = \frac{1825}{2{,}406} = 732{,}74$$

neue Suchrichtung → Winkelhalbierende zwischen den Tangenten

$$\underline{y} = -\text{grad } z\left(\underline{x}^{[1]}\right) - w \cdot \text{grad } g_3\left(\underline{x}^{[1]}\right) = \begin{pmatrix} -1109{,}35 \\ 645{,}83 \end{pmatrix}$$

$$\underline{x}^{[2]} = \underline{x}^{[1]} + \lambda_{opt}^{[2]} \cdot \underline{y} \text{ (Hinweis: + , da Minimumsuche schon in } \underline{y}\text{)}$$

Aktive NB? g_3 immer noch aktiv (Abb. 8)

$$\lambda_{opt}^{[2]} = 7{,}498 \cdot 10^{-4}$$

$$\underline{x}^{[2]} = \begin{pmatrix} 0{,}7185 \\ 1{,}3019 \end{pmatrix}$$

$$z\left(\underline{x}^{[2]}\right) = 2517{,}8$$

- nach weiteren Iterationen — Abb. 8

$$\underline{x}^{[\mathrm{opt}]} = \begin{pmatrix} 0{,}8129 \\ 1{,}1532 \end{pmatrix} \qquad z\left(\underline{x}^{[\mathrm{opt}]}\right) = 2479{,}8$$

3.5 Vergleich mit analytischer Lösung

- $\underline{x}^{[\mathrm{opt}]}$ liegt auf g_3

$$\begin{aligned}
\mathrm{grad}\; g_3\left(\underline{x}\right) &= \lambda \cdot \mathrm{grad}\; z\left(\underline{x}\right) \\
\frac{1}{x_1^2} &= \lambda \cdot 1414{,}2 \\
\frac{1{,}6414}{x_2^2} &= \lambda \cdot 1153{,}5 \\
0 &= \frac{1}{x_1} + \frac{1{,}6414}{x_2} - 2{,}65270 \qquad \rightarrow \lambda = 1{,}0969 \cdot 10^{-3} \\
\underline{x}^{[opt]} &= \begin{pmatrix} 0{,}8129 \\ 1{,}1532 \end{pmatrix} \\
z\left(\underline{x}^{[opt]}\right) &= 2479{,}8\ \mathrm{cm}^3
\end{aligned}$$

3.6 Algorithmische Umsetzung

```
import numpy as np                          # numerische Operationen
import matplotlib.pyplot as plt             # grafisches Darstellen
from matplotlib import rc
from gooey import Gooey, GooeyParser        # GUI

############## Definition der Optimierungsaufgabe ##############
def zf(xVec, l1=1414.2, l2=1153.5):
    """
    Definiton Zielfunktion

    :param xVec: Vektor der Eingangsparameter x
    :param l1: Laenge Stab 1
    :param l2: Laenge Stab 2
    :return: Zielfunktion z = f(x)
    """

    z = xVec[0]*l1 + xVec[1]*l2
    return z


def gradZF(xVec, l1=1414.2, l2=1153.5):
    """
    Definition Gradient der Zielfunktion

    :param xVec: Vektor der Eingangsparameter x
    :param l1: Laenge Stab 1
    :param l2: Laenge Stab 2
    :return: Gradient der Zielfunktion
    """

    return np.asanyarray([l1, l2])
```

```
def nebenbedinungen(xVec):
  """
  Definition der Nebenbedingungen

  :param xVec: Vektor der Eingangsparameter x
  :return: Werte der Nebenbedingungen
  """

  g1 = 0.5163 - xVec[0]
  g2 = 0.7324 - xVec[1]
  g3 = (1 / xVec[0]) + (1.6414 / xVec[1]) - 2.6527

  return np.asanyarray([g1, g2, g3])


def gradientDerNB(xVec):
  """
  Gradient der Nebenbedingungen

  :param xVec: Vektor der Eingangsparameter x
  :return: Gradienten aller Nebenbedingungen
  """

  gradG1 = np.asanyarray([-1, 0])
  gradG2 = np.asanyarray([0, -1])
  gradG3 = np.asanyarray([-(1/xVec[0]**2), -(1.6414/xVec[1]**2)])

  return [gradG1, gradG2, gradG3]


def ausgabe(i, x):
  """
  Definition der Standardausgaabe des Iterationsfortschritts

  :param i: Iterationsschritt
  :param x: Eingabevektor
  :return: None
  """

  print('------ITERATION %i ------' %i)
  print('Entwurfsvektor: %2.4f %2.4f' %(x[0], x[1])+
  ' -- Zielfunktionswert: %f' %zf(x))
  print('Nebenbedingungen: %f %f %f' %(nebenbedinungen(x)[0],
  nebenbedinungen(x)[1], nebenbedinungen(x)[2]))
###############################################################

def run_optimierung(x, lambdaVal, iterationen, modus, args):
  xSpeicher = []  # Speicher-Vektor der Eingangsgr en
  zSpeicher = []  # Speicher-Vektor der Zielfunktion

  # Optimierung
  for i in range(iterationen):
    if modus == 'mitNorm':
      # Suchrichtung mit Gradientennormalisierung
      y = -(gradZF(x)/np.linalg.norm(gradZF(x)))

    elif modus == 'ohneNorm':
      # Suchrichtung ohne Gradientennormalisierung
      y = -gradZF(x)
```

```
        elif modus == 'umRdgsGrad':
          # Suchrichtung mit Nebenbedingungen
          y = -gradZF(x)
          nb = nebenbedinungen(x)
          for idxNB, gradNB in enumerate(gradientDerNB(x)):
            if nb[idxNB] > 0:
              y += -gradNB*(np.linalg.norm(gradZF(x))/np.linalg.norm(gradNB))
        else:
          raise ValueError("Falscher Auswertungsmodus")

        x = x+lambdaVal*y   # neuer Entwurfspunkt
        ausgabe(i, x)
        xSpeicher.append(x)
        zSpeicher.append(zf(x))

      # Plot Entwurfs- und Ergebnisraum
      if args.plot is True:
        # Erstellung der Grafik
        fig, ax = plt.subplots(1, 2)
        # plot Eingangsvektoren
        xArray = np.asanyarray(xSpeicher)
        ax[0].plot(xArray[:, 0], xArray[:, 1], 'r.-')
        # Beschriftungen
        ax[0].set_title('Entwurfsraum')
        ax[0].set_xlabel(r'$x_1$')
        ax[0].set_ylabel(r'$x_2$')
        # plot Zielfunktion
        ax[1].plot(zSpeicher, 'b.-')
        # Beschriftungen
        ax[1].set_title('Zielfunktion')
        ax[1].set_xlabel('Iterationen')
        ax[1].set_ylabel('Zielfunktionswert')

        for axi in ax:
          axi.grid() # erstelle Raster
        fig.tight_layout() # positioniere subplots
        plt.show()

#################### GUI und Eingabeparameter ####################
@Gooey()  # decorator
def GUI():
  parser = GooeyParser(description='Parametereingabe gradientenbasierte
                                    Optimierung')

  parser.add_argument('-Startpunkt-x1', '-my-arg', widget='IntegerField',
  gooey_options={'min': 2, 'max': 10, 'increment': 1})

  parser.add_argument('-Startpunkt-x2', '-my-arg2', widget='IntegerField',
  gooey_options={'min': 2, 'max': 10, 'increment': 1})

  parser.add_argument('-e', '--modus', nargs='+',
  choices=['mitNorm', 'ohneNorm', 'umRdgsGrad'])

  parser.add_argument('-Iterationen', widget='Slider',
  gooey_options={'min': 10, 'max': 100, 'increment': 10})

  parser.add_argument('-f', '--plot',
  metavar='Auswertungsabbildung anzeigen', action='store_true')

  args = parser.parse_args()
```

```
    return args


# Parameter der Optimierung (teilweise abgerufen aus GUI)
args = GUI()
x = [int(args.Startpunkt_x1), int(args.Startpunkt_x2)] # Startpunkt
lambdaVal = 1*10**-4 # Schrittweite
iterationen = int(args.Iterationen) # vorgegebene Anzahl der Iterationen
modus = str(args.modus[0])
# 'mitNorm' - mit Gradientennormalisierung
# 'ohneNorm' - ohne Gradientennormalisierung
# 'umRdgsGrad' - mit Umrandungsgradient

################################################################


run_optimierung(x, lambdaVal, iterationen, modus, args)
```

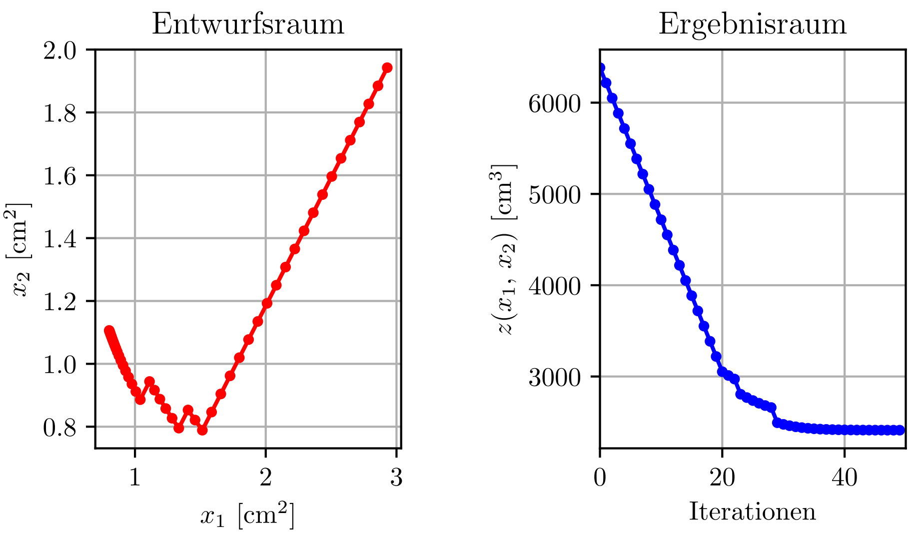


Abbildung 9: Ergebnisdarstellung