Implementierungaufgabe:
Newton-Verfahren

für die Lehrveranstaltung: Optimierung 1, UE
gelesen von: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Bettina Klinz
am Institut für Diskrete Mathematik
Technische Universität Graz

Moritz Mossböck
11820925

24. August 2023

Inhaltsverzeichnis

1 Überblick über das Verfahren

Wir setzen im Folgenden eine Funktion $f \in \mathcal{C}^2(\mathbb{R}^n, \mathbb{R})$ voraus. Wie bei anderen Minimierungsverfahren suchen wir beim Newton-Verfahren (NV) einen Punkt $\boldsymbol{x}^* \in \mathbb{R}^n$, sodass $\nabla f(\boldsymbol{x}^*) = \mathbf{0}_n$ und $\nabla^2 f(\boldsymbol{x}^*) > 0$. Beim lokalen NV erzeugen wir eine Folge $\boldsymbol{x}^k$, die unter den gegebenen Umständen superlinear[1] gegen ein geeignetes $\boldsymbol{x}^*$ konvergiert. Wie in [Gei99, S. 83f] beschrieben wird, versuchen wir sukzessive die quadratische Näherung

$$q_k(\boldsymbol{x}) = f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^{\mathrm{t}}(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^{\mathrm{t}}\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k)$$

zu minimieren. Dabei bezeichnet $\boldsymbol{x}^k$ den aktuellen Iterationspunkt. Ableiten von q_k führt uns auf $\nabla q_k(\boldsymbol{x}) = \nabla f(\boldsymbol{x}^k) + \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k)$. Zum Auffinden eines stationären Punktes von q_k lösen wir das entsprechende inhomogene System und erhalten damit $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \nabla^2 f(\boldsymbol{x}^k)^{-1}\nabla f(\boldsymbol{x}^k)$. Die Folge der Richtungsvektoren $\boldsymbol{d}^k$ wird durch Lösen von $\nabla^2 f(\boldsymbol{x}^k)\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$ erzeugt. Den Umständen entsprechend ist es vorteilhaft, die explizite Berechnung der Inversen von $\nabla^2 f(\boldsymbol{x}^k)$ zu vermeiden und stattdessen direkt Gauss-Jordan[2] zu verwenden. Das liegt natürlich daran, dass wir nur eine Lösung, sofern existent, bestimmen wollen. Die Konvergenz des NV gegen einen stationären Punkt $\boldsymbol{x}^*$ von f wird durch den folgenden Satz aus [Gei99, S. 84] garantiert.

Satz 1.1 (Konvergenz des lokalen Newton-Verfahrens)**.** *Gegeben sei $f \in \mathcal{C}^2(\mathbb{R}^n, \mathbb{R})$, $\boldsymbol{x}^* \in \mathbb{R}^n$ ein stationärer Punkt von f und $\nabla^2 f(\boldsymbol{x}^*)$ regulär. Dann existiert ein $\varepsilon > 0$ sodass für jedes $\boldsymbol{x}^0 \in \mathcal{B}_\varepsilon(\boldsymbol{x}^*)$ folgt, dass das lokale NV eine gegen $\boldsymbol{x}^*$ konvergente Folge $(\boldsymbol{x}^k)_{k\in\mathbb{N}}$ erzeugt.*

Lokales Newtonverfahren

```
input:  ∇f: ℝ^n → ℝ^n ,  ∇²f: ℝ^n → ℝ_s^{n×n} ,  x^0 ∈ ℝ^n ,  ε ≥ 0
output:  x* ∈ ℝ^n

lnewton(∇f ,  ∇²f ,  x^0 ,  ε ):
    x^k ← x^0
    while  ‖∇f(x^k)‖ ≥ ε  do
        solve  d^k  as  ∇²f(x^k)d^k = −∇f(x^k)
        x^k ← x^k + d^k
    return  x^k
```

Das oben beschrieben Verfahren ist lediglich ein lokales Verfahren. Ein globalisiertes Verfahren wird in [Gei99, S. 85ff] vorgestellt, welches zusätzlich eine nicht-konstante Schrittweite t_k bestimmt und auf das Gradientenverfahren zurückgreift, falls $\nabla^2 f(\boldsymbol{x}^k)$ singulär ist.

[1] ist $\nabla^2 f$ lokal Lipschitz-stetig liegt sogar quadratische Konvergenz vor
[2] idealerweise die Cholesky-Zerlegung in Umgebung eines Minimums, siehe 1.1

Globalisiertes Newtonverfahren

```
input: ∇f: ℝ^n → ℝ^n, ∇²f: ℝ^n → ℝ_s^{n×n}, x^0 ∈ ℝ^n, ε ≥ 0, ϱ > 0, p > 2, β ∈ (0,1), σ ∈ (0, 1/2)
output: x* ∈ ℝ^n

gnewton(∇f, ∇²f, x^0, ε, ϱ, p, β, σ):
    x^k ← x^0
    while ‖∇f(x^k)‖ ≥ ε do
        t_k = 1
        solve d^k as ∇²f(x^k)d^k = −∇f(x^k) or d^k ← −∇f(x^k)
        if ∇f(x^k)^t d^k ≤ −ϱ‖d^k‖^p then d^k = −∇f(x^k)
        while f(x^k + t_k d^k) > f(x^k) + σ t_k ∇f(x^k)^t d^k do
            t_k ← β t_k
        x^k ← x^k + t_k d^k
    return x^k
```

1.1 Nicht-Monotone Armijo-Regel

Laut [Gei99, S. 86] weist das lokale Newtonverfahren oftmals besseres Konvergenzverhalten auf, als die globalisierte Variante. Daher zeigt es sich als Vorteilhaft möglichst oft die Schrittweite 1 zu wählen, was dem lokalen Newtonverfahren entspricht, sofern nicht der Gradientenschritt gewählt wird. Die Idee hinter dieser neuen Regel ist t_k durch die *nicht-monotone* Armijo-Regel zu bestimmen

$$f(\boldsymbol{x}^k + t_k \boldsymbol{d}^k) \leq \mathcal{R}_k + \sigma t_k \nabla f(\boldsymbol{x}^k)^{\mathrm{t}} \boldsymbol{d}^k.$$

Der Referenzwert $\mathcal{R}_k$ ist hierbei das Maximum der letzten $m_k + 1$ Funktionswerte, also

$$\mathcal{R}_k = \max_{k - m_k \leq j \leq k} f(\boldsymbol{x}^j).$$

Im Falle eines Gradientenschrittes wählt man $m_k = 0$, andernfalls setzen wir $\min\{m_{k-1}, m\}$, wobei m eine konstante natürliche Zahl ist. Da wir höchstens $m + 1$ mögliche Funktionswerte vergleichen speichern wir also zusätzlich die letzten m Funktionswerte ab. Damit erhalten wir also die folgende Variation des globalisierten Newtonverfahrens.

Globalisiertes Newtonverfahren mit nicht-monotoner Armijo-Regel

```
input: ∇f: ℝ^n → ℝ^n, ∇²f: ℝ^n → ℝ_s^{n×n}, x^0 ∈ ℝ^n, ε ≥ 0, ϱ > 0, p > 2, β ∈ (0,1), σ ∈ (0, 1/2)
       m ∈ ℕ
output: x* ∈ ℝ^n

gnewton(∇f, ∇²f, x^0, ε, ϱ, p, β, σ, m):
    x^k ← x^0
    R ← 0_m

    while ‖∇f(x^k)‖ ≥ ε do
        t_k = 1
        lshift(R, 1)
        R[m] = f(x^k)
        solve d^k as ∇²f(x^k)d^k = −∇f(x^k) and
            m_k ← min(m_{k−1} + 1, m)
        or
            d^k ← −∇f(x^k)
            m_k ← 0
        if ∇f(x^k)^t d^k ≤ −ϱ‖d^k‖^p then
            d^k = −∇f(x^k)
            m_k ← 0
        R_k = max_{j=m−m_k+1,...,m} R[j]
        while f(x^k + t_k d^k) > R_k + σ t_k ∇f(x^k)^t d^k do
            t_k ← β t_k
        x^k ← x^k + t_k d^k
    return x^k
```

2 Resultate

Beim Testen der Implementierungen wurden insgesamt drei verschiedene Funktionen getestet:

Verfahren	Verbesserungen aus 1.1
lokales Newtonverfahren	keine
globalisiertes Newtonverfahren	keine
globalisiertes Newtonverfahren	nicht monotone Armijo-Regel

Tabelle 1: Getestete Versionen des Newton-Verfahrens

Die getesteten Funktionen sind die Rosenbrock-Funktion $r(x_1, x_2)$, $f(\boldsymbol{x}) = -\exp(-\|\boldsymbol{x}\|^2)$, $h(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$, $d(x_1, x_2) = \sin(x_1 x_2)$ und $g(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1 x_2 + 2x_1^3 + x_1^4$. Im Anhang findet sich eine analytische Bestimmung der Minima. Getestet wurden nun:

1. die vergangene Zeit bis eine Abbruchbedingung erreicht wurde
2. welche Abbruchbedingung erreicht wurde
3. ob eine korrekte Minimalstelle ermittelt wurde, und der Fehler zum nächsten Minimum wenn nicht

In der Abgabe findet sich die Datei `testing.py` welche die unten aufgeführten Plots erzeugt. Ziel dabei war es das Verhalten der Algorithmen bei verschiedenen Startpunkten zu prüfen. Dazu wurde jeweils die Umgebung $[-5, 5]^2$ gewählt mit einer Auflösung von 10000 Punkten. Standardgemäß wurden dazu die folgenden Standard-Werte der Parameter verwendet:

Paramter					Abbruchbedingungen		
ϱ	p	β	σ	m	ε	M	N
1	$\frac{5}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	10	10^{-9}	100	50

Tabelle 2: Verwendete Parameter für alle Tests

Da insgesamt 60 Plots erzeugt wurden sind diese nicht in diesem Dokument eingefügt, sie sind aber in der Abgabe enthalten. Zu finden sind sie unter `<version>/`.

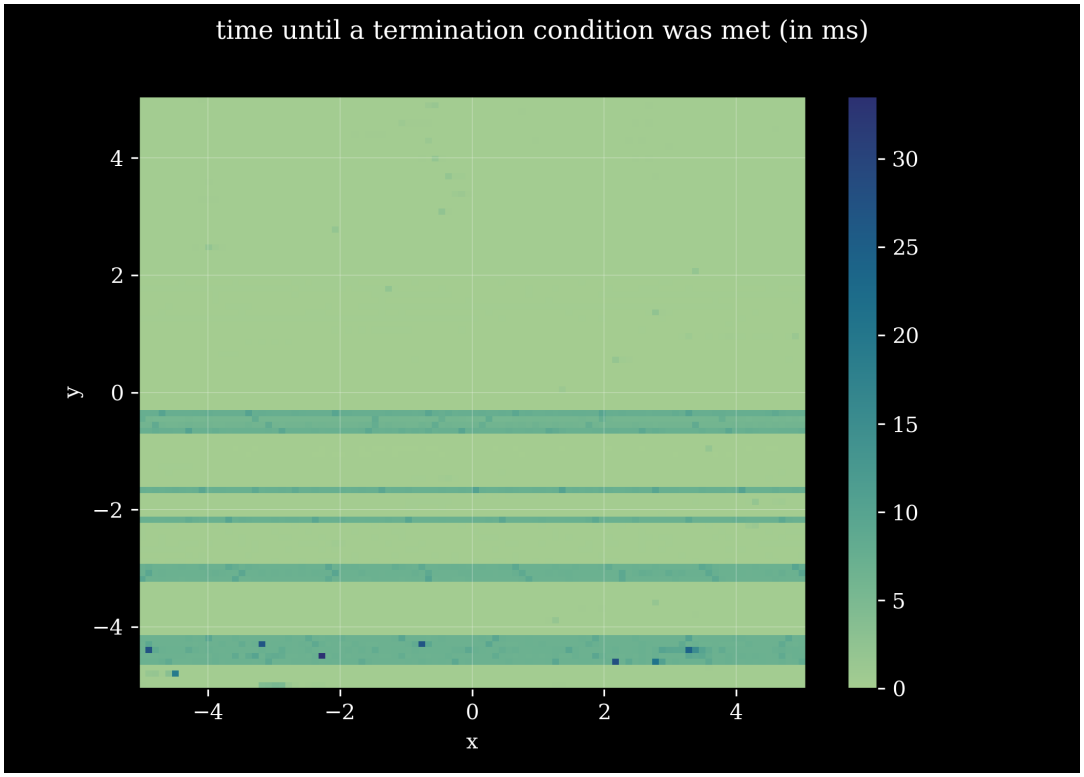

Abbildung 1: Vergangene Zeit bis eine Abbruchbedingung erreicht wurde ausgehend von verschiedenen Startpunkten, Globalisiertes Verfahren mit Funktion g

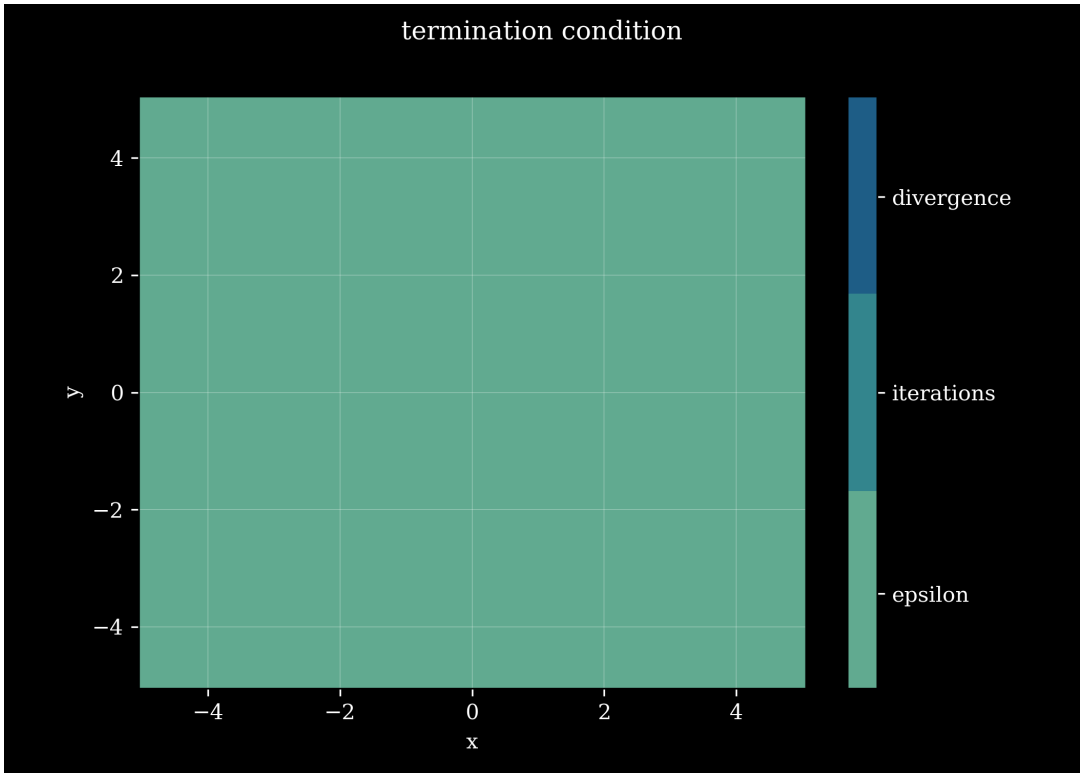

Abbildung 2: Welche Abbruchbedingung erreicht wurde, modifiziertes globalisiertes Verfahren mit d

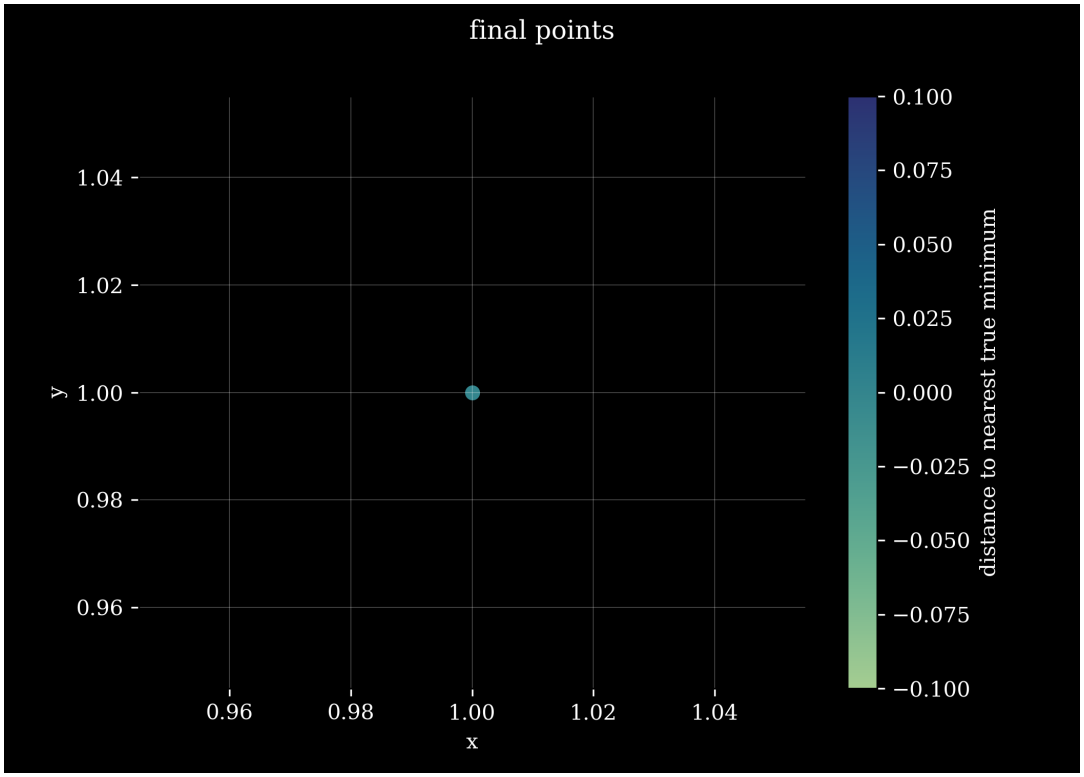

Abbildung 3: Welche Punkte durch das modifizierte lokale Newton-Verfahren als Minima von r bestimmt wurden

		Zeit in ms			**Abbruchbedingung**		
Verfahren	**Funktion**	min	avg	max	ε	N	M
lokal	r	0.022	0.074	2.304	9998	0	2
	f	0.005	0.192	10.031	10000	0	0
	g	0.057	0.159	18.257	10000	0	0
	h	0.696	0.792	14.276	0	10000	0
	d	0.028	0.091	2.631	9960	0	40
globalisiert	r	0.099	0.612	15.512	9977	23	0
	f	0.041	6.373	34.109	3248	6752	0
	g	0.126	1.248	33.548	8600	1400	0
	h	1.274	2.523	32.641	0	10000	0
	d	0.039	2.321	33.404	7148	2832	20
modifiziert globalisiert	r	0.181	0.448	22.359	9749	251	0
	f	0.067	1.929	30.238	5048	4952	0
	g	0.204	0.529	24.825	9998	2	0
	h	2.298	2.641	30.719	0	10000	0
	d	0.109	0.292	17.978	10000	0	0

Tabelle 3: Verschiedene Statistiken zu Laufzeiten und Fällen (die Spalten für die Fälle geben an wie viele Punkte jeweils welche Abbruchbedingung erwirkt haben)

3 Anhang

3.1 Struktur der Implementierung

Bei der abgegebenen Implementierung handelt es sich um einen objektorientierten Ansatz. Die primäre Motivation dahinter war es, die ständige Weitergabe von Parametern zu vermeiden. Werte wie N, ε oder σ werden als Attribute eines Objektes gespeichert. Das erlaubt es das Verfahren schnell mit mehreren Startpunkten anzuwenden und dabei die gewünschte Version auszuwählen.

Die Klasse findet sich in der Datei `newton.py` wieder und hat die Methoden `checkparams()`, `armijo()`, `modified_armijo()`, `local()`, `globalized()`, `modified_globalized()` und `run()` und den Konstruktor.

Die Methode `checkparams()` wird im Konstruktor aufgerufen um die Gültigkeit aller Parameter zu überprüfen. Das beinhaltet auch die übergebenen Funktionen, da diese potentiell `None` sein könnten (das Python-Analogon zu einer Null-Pointer Referenz).

Die Methode `armijo()` berechnet den Namen nach die Armijo-Schrittweite. Da der momentane Iterationspunkt als Attribut gespeichert wird, muss nur der aktuelle Richtungsvektor übergeben werden. Gleichfalls berechnet `modified_armijo()`, mit der aktuellen Richtung und dem Wert R_k, die nicht-monotone Armijo-Schrittweite im aktuellen Iterationspunkt.

Die Methoden `local()`, `globalized()` und `modified_globalized()` führen dann jeweils ein Verfahren aus, wobei `modified_globalized()` die nicht-monotone Armijo-Schrittweite verwendet. Alle drei Methoden nehmen als Parameter einen Startpunkt $\boldsymbol{x}^k$ an und geben der Angabe entsprechend den Punkt zurück, der eine Abbruchbedingung erreicht hat und welche Abbruchbedingung erreicht wurde.

Zuletzt bleibt `run()`. Diese Methode dient als Abstraktionsebene nach außen um mittels des Argumentes `vers` die Version des Newton-Verfahrens auszuwählen. Dabei bedeutet 0 das lokale, 1 das globalisierte und 2 das modifizierte globalisierte Verfahren.

Um unnötige if-Verzweigungen zu vermeiden wurden, sofern möglich, for-Schleifen verwendet. Bei diesen vermeiden wir es, eigens Zählvariablen anzulegen und zu inkrementieren, da diese streng genommen nicht von Bedarf sind. Lediglich für eine Statistik wie viele Iterationen gebraucht wurden würde die Speicherung der Zählvariable rechtfertigen.

3.2 Anmerkungen zum Pseudo-Code

Der verwendete Dialekt des oben verwendeten Pseudo-Codes ist meine persönliche Entwicklung. Großteils sollten die verwendeten Kontrollelemente und Funktionsaufrufe gängig genug sein um keiner weiteren Erklärung zu bedürfen. Die verwendete Funktion solve bedarf etwas Erklärung. Die grundlegende Idee ist die Try-Catch Struktur, die in vielen Programmiersprachen verfügbar ist. Im Try-Block soll versucht werden $\boldsymbol{d}^k$ als Lösung der Newton-Gleichung zu bestimmen. Im Anschließenden Block der durch or gekennzeichnet wird, ist der Catch-Fall, wenn keine Lösung existiert. Existiert eine Lösung und sollen zusätzlich noch weitere Instruktionen durchführt werden, so wird mit einem and der Try-Block erweitert.

Die Funktion lshift führt dem Namen nach einen Links-Shift durch:

$$\text{lshift}((x_1, \ldots, x_n)) = (x_2, \ldots, x_n, 0) \in \mathbb{R}^n$$

3.3 Anmerkungen zu Absicherungen gegen Endlosschleifen

Während der Entwicklung sind einige Probleme numerischer Natur aufgetreten. Diese könnten vermutlich durch geeignete Bibliotheken gelöst werden, allerdings handelte es sich bei den auftretenden Zahlen bereits um Größen im Bereich von 10^{-12} und darunter. Bei der Armijo-Regel findet sich ein zusätzlicher Check, ob $t_k \geq 10^{-12}$ gilt. Falls t_k diese Grenze unterschreitet wird abgebrochen. Dieselbe Problematik ist bei der nicht-monotonen Armijo-Regel aufgetreten.

3.4 Verschiedenes

$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^n$	kanonischer n-dimensionaler Vektorraum über $\mathbb{R}$
$\mathcal{C}(D, B)$	stetige Funktionen $f\colon D \to B$
$\mathcal{C}^k(D, B)$	k-fach stetig-differenzierbare Funktionen $f\colon D \to B$
$\mathrm{D}f$, auch $\mathrm{J}f$	Jacobi-Matrix von $f\colon \mathbb{R}^n \to \mathbb{R}^m$
∇f	Gradient einer Funktion ($\nabla f = \mathrm{D}f^{\mathrm{t}}$)
$\mathbb{R}^{m\times n}$	$m \times n$ Matrizen
$\mathbb{R}_s^{n\times n}$	symmetrische $n \times n$ Matrizen
$\nabla^2 f$, auch $\mathrm{H}f$	Hesse-Matrix von $f\colon \mathbb{R}^n \to \mathbb{R}$
A, M	Matrix
I	Einheitsmatrix
I_n	Einheitsmatrix in $\mathbb{R}^{n\times n}$
0	Nullmatrix
0_n	Nullmatrix in $\mathbb{R}^{n\times n}$
$\boldsymbol{x}, \boldsymbol{z}$	Vektor
$\boldsymbol{x}^*$	stationärer Punkt
$\mathbf{0}$	Nullvektor
$\mathbf{0}_n$	Nullvektor in $\mathbb{R}^n$
$\mathbf{1}$	$\begin{bmatrix}1 & \cdots & 1\end{bmatrix} \in \mathbb{R}^n$

Tabelle 4: Symbolindex

3.4.1 Analytische Untersuchung der getesteten Funktionen

mit den jeweiligen Ableitungen:

$$\nabla r(x_1, x_2) = \begin{bmatrix} -400(x_2 - x_1^2)x_1 - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

$$\nabla^2 r(x_1, x_2) = \begin{bmatrix} 800x_1^2 - 400(x_2 - x_1^2) + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

$$\nabla f(\boldsymbol{x}) = 2\exp(-\|\boldsymbol{x}\|^2)\boldsymbol{x}$$

$$\nabla^2 f(\boldsymbol{x}) = 2\exp(-\|\boldsymbol{x}\|^2) \begin{bmatrix} 1 - 2x_1^2 & -2x_1x_2 & \cdots & -2x_1x_{n-1} & -2x_1x_n \\ -2x_1x_2 & 1 - 2x_2^2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & -2x_{n-1}x_n \\ -2x_1x_n & & \ddots & -2x_{n-1}x_n & 1 - 2x_n^2 \end{bmatrix}$$

$$\nabla g(x_1, x_2) = 2\begin{bmatrix} 2x_1 - x_2 + 3x_1^2 + 2x_1^3 \\ x_2 - x_1 \end{bmatrix} \qquad \nabla^2 g(x_1, x_2) = 2\begin{bmatrix} 2 + 6x_1 + 6x_1^2 & -1 \\ -1 & 1 \end{bmatrix}$$

Am Gradienten von f sieht man schnell, dass nur $\mathbf{0}$ ein stationärer Punkt ist. Die Hesse-Matrix $\nabla^2 f(\mathbf{0})$ ist offensichtlich positiv definit, da $\nabla^2 f(\mathbf{0}) = 2\mathrm{I}_n$ gilt. Die Rosenbrock-Funktion r hat bekanntermaßen bei $\boldsymbol{x}^* = (1, 1)$ ihr Minimum. Für g bestimmen wir aus dem Gradienten $x_1 = x_2$ für stationäre Punkte und somit

$$2x_1 - x_1 + 3x_1^2 + 2x_1^3 \overset{!}{=} 0 \iff x_1(1 + 3x_1 + 2x_1^2) \overset{!}{=} 0$$

$$\iff x_1(x_1 + 1)\left(x_1 - \frac{1}{2}\right) = 0$$

$$\nabla^2 g(0, 0) = 2\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \rightsquigarrow \lambda_{1,2} = 3 \pm \sqrt{5} \implies \nabla^2 g(0, 0) > 0$$

$$\nabla^2 g(-1, -1) = 2\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \rightsquigarrow \lambda_{1,2} = 3 \pm \sqrt{5} \implies \nabla^2 g(0, 0) > 0$$

$$\nabla^2 g\left(\frac{1}{2}, \frac{1}{2}\right) = 2\begin{bmatrix} \frac{13}{2} & -1 \\ -1 & 1 \end{bmatrix} \rightsquigarrow \lambda_{1,2} = \frac{15 \pm \sqrt{137}}{2} \implies \nabla^2 g\left(\frac{1}{2}, \frac{1}{2}\right) > 0$$

Die Hesse-Matrix von g is also in allen drei stationären Punkten positiv definit und hat somit drei Minima.
h hat offensichtlich keine Minima:

$$\nabla h = \mathbf{1} \qquad \nabla^2 h = 0_n$$

Für d ergibt sich

$$\nabla d(x_1, x_2) = \cos(x_1 x_2) \begin{bmatrix} x_2 \\ x_2 \end{bmatrix} \qquad \nabla^2 d(x_1, x_2) = \begin{bmatrix} -x_2^2 \sin(x_1 x_2) & \cos(x_1 x_2) - x_1 x_2 \sin(x_1 x_2) \\ \cos(x_1 x_2) - x_1 x_2 \sin(x_1 x_2) & -x_1^2 \sin(x_1 x_2) \end{bmatrix}$$

Die Gleichung $\nabla d(\boldsymbol{x}) = \mathbf{0}$ hat einerseits die Lösung $\boldsymbol{x} = \mathbf{0}$ und für $x_1 \neq 0$ folgt

$$x_2 = \frac{\pi(2k-1)}{2x_1} \qquad k \in \mathbb{Z}$$

Offenbar ist $\mathbf{0}$ kein Minimum, da auf der Geraden $x_2 = 1$ bei $x_1 = 0$ kein Minimum vorliegt.

Literatur

[Gei99] Carl Geiger. *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben.* ger. Springer-Lehrbuch. 1999. ISBN: 9783540662204.