Programmierblatt 3.

Besprechungswoche: **13.12. – 17.12.2021**

Optimierungsprobleme treten in zahlreichen Anwendungen auf, seien sie wissenschaftlicher, technischer, oder wirtschaftlicher Natur. Viele Probleme aus der Physik lassen sich auf Energieminimierungsprobleme zurückführen, während Finanzdienstleister an der Maximierung eines Ertrages interessiert sind. Aus mathematischer Sicht spielt es dabei keine Rolle, ob eine Funktion minimiert oder maximiert wird, da das Minimieren einer Funktion f äquivalent zum Maximieren der Funktion $g = -f$ ist.
Wir werden auf diesem Programmierblatt verschiedene Algorithmen verwenden, um eine Funktion, beispielsweise die *Rosenbrock-Funktion*[1]

$$f(\mathbf{x}) = (a - x_1)^2 + b(x_2 - x_1^2)^2, \qquad a, b > 0, \tag{1}$$

zu minimieren. Deren globales Minimum $\mathbf{x} = [1, 1]^\mathsf{T}$ liegt in einem schmalen, parabolischen Bereich, was die Minimierung besonders anspruchsvoll macht.
Ausgehend von einem Startpunkt $\mathbf{x}_0$ versuchen wir, die Funktion iterativ zu minimieren, wobei wir eine Folge Iterierter $(\mathbf{x}_k)_k$ erhalten, welche der Rekursion

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k, \qquad k = 0, 1, \ldots$$

genügt. Dabei sind α_k Schrittweiten und $\mathbf{d}_k$ sinnvoll gewählte Suchrichtungen. Hierbei soll $\mathbf{d}_k$ stets eine Abstiegsrichtung sein, was heisst, dass wir für alle $k \geq 0$ die Bedingung $\mathbf{d}_k^\mathsf{T} \nabla f(\mathbf{x}_k) < 0$ fordern.

Armijo-Liniensuche

In vielen Optimierungsalgorithmen muss zu einem Punkt $\mathbf{x}$ und einer gegebenen Suchrichtung $\mathbf{d}$ die Schrittweite

$$\alpha \approx \arg\min_{t \in \mathbb{R}} f(\mathbf{x} + t\mathbf{d})$$

näherungsweise berechnet werden. Ausgehend von $\alpha = 1$ halbiert man α so lange, bis

$$f(\mathbf{x} + \alpha\mathbf{d}) \leq f(\mathbf{x}) - \mu\alpha\|\mathbf{d}\|_2^2$$

gilt.

Aufgabe 1. Schreiben Sie Funktionen

```
function xs = gradientDescent(f, Df, x0, mu, tol, maxIter)
function xs = nlcg(f, Df, x0, mu, tol, maxIter)
function xs = polakRibiere(f, Df, x0, mu, gammas, tol, maxIter)
function xs = quasiNewton(f, Df, x0, mu, gamma, nu, tol, maxIter)
```

welche die Minimierungsalgorithmen aus dem Skript implementieren. Die Funktion `gradientDescent` kann dabei analog zu Algorithmus 3.2 geschrieben werden, mit der Funktion `polakRibiere` soll das modifizierte Verfahren gemäss Algorithmus 4.16 implementiert werden.

[1] Benannt nach Howard H. Rosenbrock, 1920 – 2010

Nutzen Sie die Funktionen, um die Rosenbrock-Funktion (1) mit $a = 1$, $b = 100$ zu minimieren. Als Parameter benutzen Sie $\gamma = \nu = 1$, $\mu = 0.005$, `tol` $= 1e-8$, `maxIter` $= 10000$, sowie $\underline{\gamma} = 0.1$, $\overline{\gamma} = 10$ und $\mu = 0.7$. Als Startwerte verwenden Sie die Punkte

$$\mathbf{x}_0 = [-3, 1]^\mathsf{T}, \quad \mathbf{x}_1 = [-2, 4]^\mathsf{T}, \quad \mathbf{x}_2 = \left[-\frac{1}{2}, 5\right]^\mathsf{T}.$$

Stellen Sie in Subplots sowohl die Konturlinien der Rosenbrock-Funktion[2] als auch die Pfade der Optimierungsfunktionen gemäss Abbildung 1 graphisch dar. Geben Sie auch die Anzahl der benötigten Iterationen an.

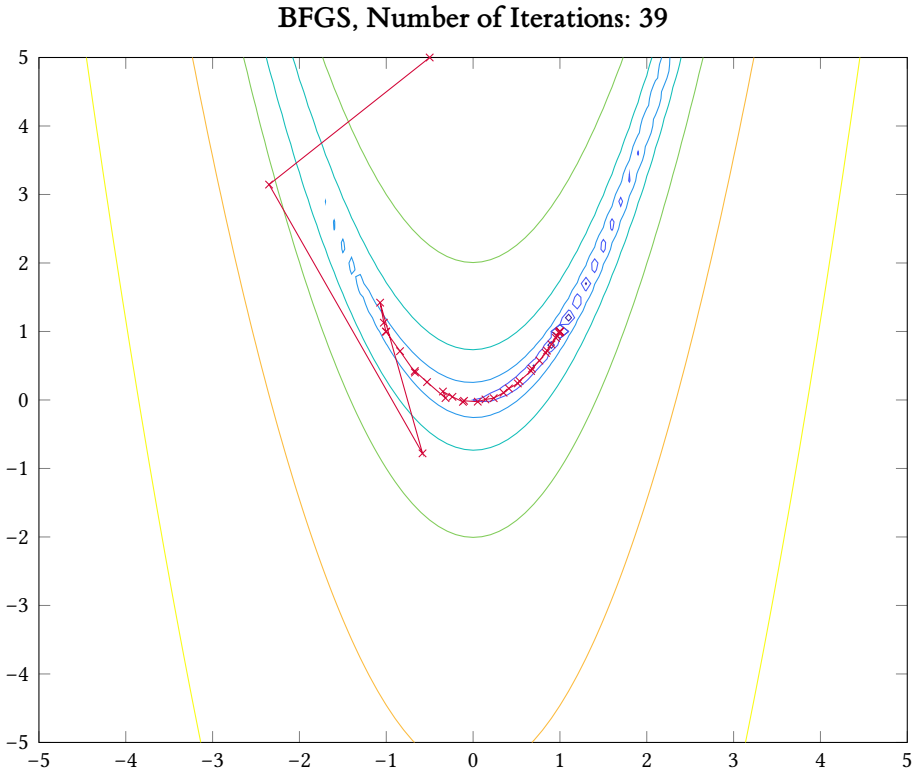

Abbildung 1: Konturlinien und Suchpfad des *BFGS-Verfahrens* für die Rosenbrock-Funktion (1) mit Startpunkt $\mathbf{x}_0 = [-\frac{1}{2}, 5]^\mathsf{T}$.

Optimierung mit Nebenbedingungen

In der Praxis treten vielerorts auch Minimierungsprobleme auf, welche unter Nebenbedingungen zu lösen sind. Ein Beispiel ist etwa die Entwicklung eines Fahrzeugs, welches möglichst energieeffizient gebaut werden sollte, jedoch gewisse Sicherheitsvorkehrungen erfüllen sollte. Auch im maschinellen Lernen treten Optimierungsprobleme dieser Art auf.
Wir beschränken uns im Folgenden auf die Optimierung unter Nebenbedingungen der Form

$$\text{minimiere } f(\mathbf{x}), \text{ so dass } \mathbf{g}(\mathbf{x}) = \mathbf{0}. \tag{2}$$

Mann kann zeigen, dass die Funktion (2) genau dann ein lokales Minimum in einem Punkt $\mathbf{x}^\star$, hat, wenn ein $\boldsymbol{\lambda}^\star$ existiert, so dass der Punkt $(\mathbf{x}^\star, \boldsymbol{\lambda}^\star)$ ein kritischer Punkt der zugehörigen *Lagrange-Funktion*

$$\mathcal{L}(\mathbf{x}; \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^\mathsf{T}\mathbf{g}(\mathbf{x})$$

[2] Der Matlab-Befehl `contour` ist hierbei hilfreich. Zur besseren Darstellung zeichnen Sie die Konturlinien von $\log f$.

ist. Da diese $(\mathbf{x}^\star, \boldsymbol{\lambda}^\star)$ im Allgemeinen aber keine Minima sind, definieren wir stattdessen die *Penalty-Funktion*

$$\mathcal{L}_\alpha(\mathbf{x}) := f(\mathbf{x}) + \alpha \|\mathbf{g}(\mathbf{x})\|_2^2, \tag{3}$$

welche wir für einen Parameter $\alpha \gg 1$ minimieren.

Aufgabe 2. Minimieren Sie die Funktion $f(\mathbf{x}) = x_1 x_2$ auf dem Einheitskreis. Benutzen Sie dabei sowohl das Verfahren des steilsten Abstiegs, als auch das Verfahren von Polak und Ribière. Wählen Sie als Parameter $\mu = 0.6$, $\underline{\gamma} = 0.1$, $\overline{\gamma} = 10$, `tol = 1e - 10`, `maxIter = 10000`. Als Parameter α gemäss (3) wählen Sie

$$\alpha = 2^k, \qquad k = 6, 7, \ldots, 14.$$

Stellen Sie, ausgehend vom Startwert $\mathbf{x}_0 = [1, 0.81]^\intercal$, den kleineren Fehler zu den analytischen Lösungen

$$\mathbf{x}^\star = \frac{1}{\sqrt{2}}[1, -1]^\intercal, \qquad \mathbf{x}^* = \frac{1}{\sqrt{2}}[-1, 1]^\intercal$$

gegen die Parameter α in einem `loglog`-Plot dar. Wie sieht Ihre Konvergenzrate aus?

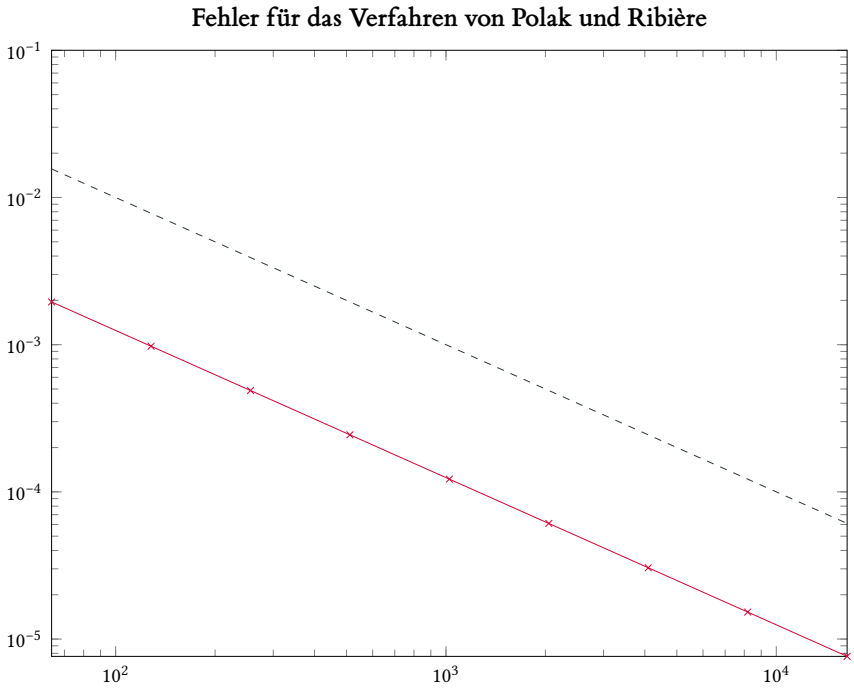

Abbildung 2: Fehlerverahlten des Verfahrens von Polak und Ribière abhängig von α.

Augmentierte Lagrange-Funktion

Der grosse Nachteil am obigen Verahren ist seine Instabilität. Weiter muss für eine beliebig genaue Annäherung α sehr gross werden. Wir benutzen daher eine weitere Möglichkeit zur Lösung von (2), nämlich die Methode der augmentierten Lagrange-Funktion. Dafür definieren wir für ein mässig grosses $\alpha > 1$ im k-ten Schritt die Funktion

$$\mathcal{L}_k(\mathbf{x}) := f(\mathbf{x}) + \frac{\alpha}{2}\|\mathbf{g}(\mathbf{x})\|_2^2 + \boldsymbol{\lambda}_k^\intercal \mathbf{g}(\mathbf{x}), \tag{4}$$

Das Optimierungsproblem (2) wird dann wie folgt iterativ gelöst:

Algorithmus 1 Augmentierte Lagrange-Methode

1: Wähle α, $\boldsymbol{\lambda}_0$, $\mathbf{x}_0$, $\texttt{tol} \ll 1$ und setze $k := 0$
2: Minimiere $\mathcal{L}_k$ gemäss (4)
3: Setze $\mathbf{x}_{k+1} := \arg\min_{\mathbf{y}} \mathcal{L}_k(\mathbf{y})$
4: Setze $\boldsymbol{\lambda}_{k+1} := \boldsymbol{\lambda}_k + \alpha \mathbf{g}(\mathbf{x}_{k+1})$
5: Falls $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_2 > \texttt{tol}$ erhöhe $k := k + 1$ und gehe nach 2

Aufgabe 3. Schreiben sie ein Skript, welches die Funktion $f(\mathbf{x}) = x_1 x_2$ mithilfe der augmentierten Lagrange-Methode auf dem Einheitskreis minimiert. Als Parameter wählen Sie $\alpha = 10$ und $\lambda_0 = 1$. Zur Minimierung von $\mathcal{L}_k$ benutzen Sie ein Verfahren Ihrer Wahl. Zeichnen Sie die Konturlinien der Funktion f, den Einheitskreis, sowie den Pfad ausgehend vom Startpunkt $\mathbf{x}_0 = [5, \frac{1}{5}]^\intercal$ in denselben Plot. Wie viele Iterationen sind notwendig, um eine analytische Lösung bis auf $\texttt{tol} = 1e-12$ anzunähern? Reicht das für die Methode der Penalty-Funktion auch aus?

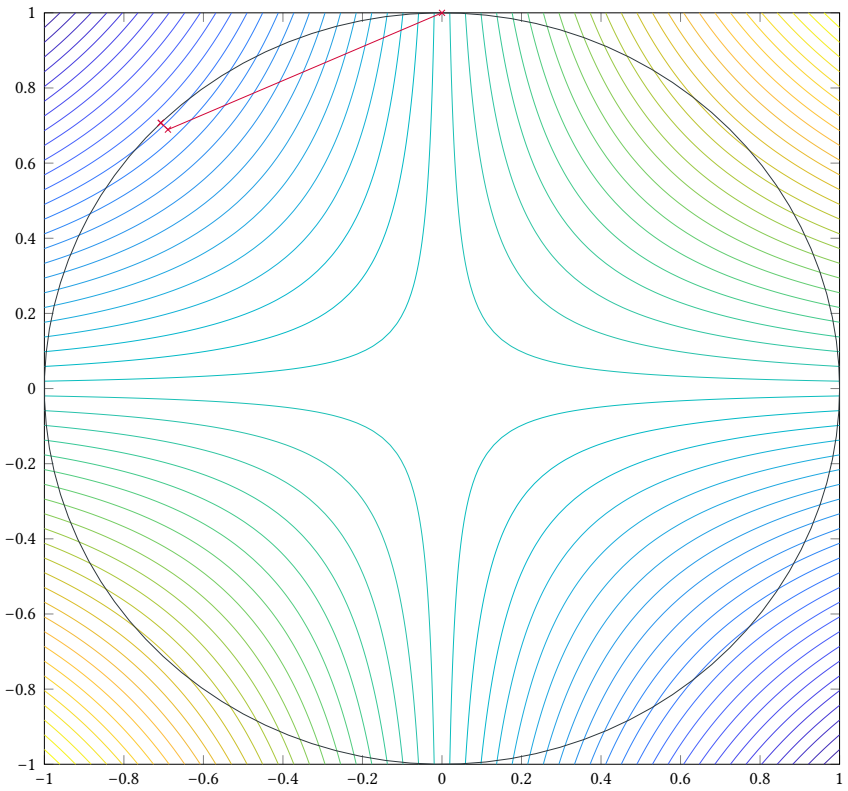

Abbildung 3: Kontur und Suchpfad für die augmentierte Lagrange-Methode.