Programmieraufgabe II (20 Punkte)

Abgabe in der Woche 29. April - 3. Mai

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Teilaufgabe a: Wiederholung zum Newton-Verfahren und Linesearch (10 Punkte)

In dieser Teilaufgabe wollen wir das aus der Vorlesung "Algorithmische Mathematik II" bekannte *Newton-Verfahren* als ein (lokal) schnelles Verfahren zur Optimierung zweimal stetig differenzierbarer Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ (ohne Nebenbedingungen) implementieren. Aus der "Einführung in die Grundlagen der Numerik" sind bereits allgemeine Abstiegsverfahren bekannt. Das Newton-Verfahren lässt sich in diesen Kontext integrieren, indem man im Punkt x als Suchrichtung/Abstiegsrichtung

$$d = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$$

verwendet, wobei $\nabla^2 f(x)$ die Hessematrix von f in x und $\nabla f(x)$ den Gradienten von f in x bezeichnet. Solange die Hessematrix positiv definit ist, ist offensichtlich mit $-\nabla f(x)$ auch d eine Abstiegsrichtung von f in x.

Aufgabe: Implementieren Sie das Newton-Verfahren mit Armijo-Linesearch, d.h. das Abstiegsverfahren mit Suchrichtungen d (wie oben angegeben) und Armijo-Linesearch. Funktionen, die f bzw. den Gradienten von f bzw. die Hessematrix von f berechnen, sollen als Input übergeben werden. Die Iteration soll abbrechen, wenn $\|\nabla f(x)\|_2 < tol$ mit einer als Input übergebenen Toleranz $tol > 0$ gilt. Weitere Inputs sollen x_0 (Startvektor), $c > 0$ (Parameter für die Armijo-Regel), $\beta > 0$ (Backtracking Parameters) sowie die maximale Anzahl der Iterationen des Abstiegsverfahrens $itmax$ und die maximale Anzahl der Backtracking-Iterationen $armijoitmax$ sein.

Passen Sie den Output an die unten beschriebenen Testfälle an.

In [2]:
```
def newton_armijo(f, gradf, hessf, x0, tol, c, beta, itmax, armijoitmax):

    return
```

Testen Sie Ihre Implementierung an folgender Funktion und den angegebenen Parametern:

- $f(x, y) := (1 - x)^2 + (y - x^2)^2$ mit Startwert $x_0 = (-0.5, 1.5)^T$ und $c = 5 \cdot 10^{-2}$, $\beta = \frac{1}{2}$, $tol = 10^{-10}$.

(Hierbei ist c die Konstante aus der Armijo-Regel und β der Parameter aus dem Backtracking-Algorithmus.)

Lassen Sie die Iterierten in einen Höhenlinienplot von f plotten. Erzeugen Sie einen Plot, aus dem die Konvergenzgeschwindigkeit des Algorithmus hervorgeht. Was können Sie in diesem Kontext hinsichtlich der vom Algorithmus bestimmten Schrittweiten beobachten?

Probieren Sie andere Startwerte x_0 aus. Was können Sie beobachten?

In [ ]:
```
```

Teilaufgabe b: Projiziertes Gradientenverfahren (10 Punkte)

Implementieren Sie das projizierte Gradientenverfahren wie es auf Übungsblatt 3 beschrieben wird.

In [4]:
```
def projected_gradient(f, gradf, Pc, x0, tol, c, beta, itmax, armijoitmax):

    return
```

Testen Sie Ihre Implementierung an der Funktion

$$f(x) := -\langle Ax, x\rangle$$

mit der Matrix $A = \mathrm{tridiag}(-1, 4, -1) \in \mathbb{R}^n$ und der konvexen Menge $C = \{x \in \mathbb{R}^n : \|x\|_2 \leq 1\}$. Die Projektion auf C ist bereits von Übungsblatt 2 bekannt.

Hinweis: Das globale Minimum von f über C ist durch $-\lambda_{\max}$ gegeben, wobei $\lambda_{\max}$ den größten Eigenwert der positiv definiten Matrix A bezeichnet. Zur Berechnung dieses Eigenwerts dürfen Sie Routinen aus numpy verwenden.

Plotten Sie für $n = 3$, $\beta = \frac{1}{2}$, $c = 10^{-2}$ und zufällig gewählte Startvektoren den Verlauf der Iterierten. Was können Sie beobachten?

In [ ]:
```

```

Was können Sie bei gleicher Parameterwahl für $n = 10$ und Startvektor $x_0 = (1, \ldots, 1)^T$ beobachten?

In [ ]:
```

```