Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn, Oleksii Saukh 27. Mai 2013

Einführung in die Optimierung

Sommersemester 2013

Übungsblatt 2

Abgabe: Donnerstag, 13. Juni

Aufgabe 1: Line Search

In vielen iterativen Optimierungsverfahren muss als Teil jeder Iteration eine eindimensionale Optimierungsaufgabe (approximativ) gelöst werden („line search“):

$$\arg\min_\tau f(\boldsymbol{x_k} + \tau \cdot \boldsymbol{d_k}).$$

Hierbei ist $f : \mathbb{R}^n \to \mathbb{R}$ die Funktion, welche minimiert werden soll, $\boldsymbol{x_k} \in \mathbb{R}^n$ ist der Startpunkt für die lineare Suche und $\boldsymbol{d_k}$ ist die Suchrichtung.
In der Vorlesung wurde die folgende einfache Methode für das eindimensionale Suchproblem vorgestellt:

Algorithm 1 Line Search

```
τ = 1.0;
β = 0.9;
while Armijo Bedinung nich erfüllt ist do
    τ = τ · β
end while
return τ
```

Als erste Aufgabe, um mit der Matlab/Octave-Umgebung etwas vertraut zuwerden, soll der obige “Line Search”-Algorithmus als Funktion in Matlab/Octave implementiert werden. Legen Sie dazu eine Datei „linesearch.m“ mit folgender Struktur an:

```
function tau=linesearch(f, d, x)
    ...
    tau = ...;
```

Hierbei ist f der Name der Funktion, welche minimiert werden soll, d die Minimierungsrichtung ($-\nabla f(x)$ im Fall des Gradientenverfahrens) und x der Startpunkt.

Anmerkung: Wählen Sie den Parameter δ der Armijo Bedingung als $\delta = 10^{-4}$. Beachten Sie auch die Hinweise am Ende des Übungsblattes.

Aufgabe 2: Gradienten- und konjugiertes Gradientenverfahren

Wir haben in der Vorlesung mehrere Algorithmen zur Minimierung (ohne Nebenbedingungen) gesehen. Die folgende Aufgabe besteht daraus, das Gradientenverfahren, sowie das konjugierte Gradientenverfahren in Matlab/Octave zu programmieren. Erstellen Sie dazu zwei Funktionen *GD* und *CG*, welche als Argumente die Funktion f, den Gradienten *df*, den Startwert *x0*, sowie einen Parameter *epsilon* haben:

```
function [x, niter] = GD/CG(f, grad, x0, epsilon)
    ...
    x = ...;
    niter = ...;
```

Der Parameter *epsilon* beschreibt die gewünschte Genauigkeit. Der iterative Prozess soll abgebrochen werden, sobald der Wert des Gradienten kleiner als *epsilon* wird und man damit genug nahe bei einem stationären Punkt ist. Verwenden Sie bei *GD* und *CG* die Funktion *linesearch* aus Aufgabe 1, um die Schrittweite zu bestimmen.
Testen Sie *GD* und *CG* für folgende Funktionen. Verwenden Sie verschiedene *epsilon*-Werte und untersuchen Sie die Anzahl benötigter Iterationen der zwei Verfahren in Abhängigkeit von *epsilon*.

- $f_1(x) = x^2 + 2x - 7$, $x_0 = 10$
- $f_2(\boldsymbol{x}) = \boldsymbol{x_1}^2 + \boldsymbol{x_2}^2 - 1$, $\boldsymbol{x_0} = (0.2, 0.3)^\top$
- $f_3(\boldsymbol{x}) = 0.5\boldsymbol{x}^\top Q_1 \boldsymbol{x} + \boldsymbol{c_1}^\top \boldsymbol{x} + 10$, wobei $Q_1 = \begin{pmatrix} 1 & -5 \\ -5 & 100 \end{pmatrix}$, $\boldsymbol{c_1} = \begin{pmatrix} -15 \\ 150 \end{pmatrix}$, $\boldsymbol{x_0} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}$
- $f_4(\boldsymbol{x}) = 0.5\boldsymbol{x}^\top Q_2 \boldsymbol{x} + \boldsymbol{c_2}^\top \boldsymbol{x} + 10$, wobei $Q_2 = \begin{pmatrix} 8.5 & -1.5 & 3 \\ -1.5 & 8.5 & 3 \\ 3 & 3 & 4 \end{pmatrix}$, $\boldsymbol{c_2} = \begin{pmatrix} -10 \\ -10 \\ -10 \end{pmatrix}$, $\boldsymbol{x_0} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$

Hinweise

In Matlab/Octave gibt es sogenannte „function handles“ als Möglichkeit den Namen einer Funktion an eine andere Funktion zu übergeben (vgl. mit „function pointers“ in C). Das folgende Beispiel illustriert wie's funktioniert. Die erste Funktion berechnet x^2 (Datei „square.m“ anlegen).

```
function y = square(x)
    y = x*x;
```

Die zweite Funktion (in Datei „execfunc.m“) hat zwei Argumente, eine Variable x und ein Funktionsname *fun*, und berechnet $fun(x)$:

```
function y = execfunc(fun, x)
    y = fun(x);
```

Die Funktion kann folgendermassen in Matlab/Octave aufgerufen werden, um mit Hilfe der Funktion *square* eine Zahl zu quadrieren:

```
octave:> execfunc(@square, 2)
ans = 4
```

Abgabe

Zur Lösung der Aufgabe müssen folgende Skripte abgegeben werden:

- f1.m, df1.m, f2.m, df2.m, f3.m, df3.m, f4.m, df4.m : Funktionen und ihre Gradienten
- linesearch.m - Line Search Algorithmus
- GD.m - Gradientenverfahren (gradient descent)
- CG.m - Konjugiertes Gradientenverfahren (conjugate gradients)

Abgeben können sie die Lösungen per E-Mail an saukho@cs.uni-freiburg.de.