Numerisches Programmieren (IN0019)

Frank R. Schmidt

Winter Semester 2016/2017

10. Minimieren von Funktionen 2 / 32

Gradienten-Abstieg 3 / 32

Minimierungsproblem

Viele Probleme lassen sich als Minimierungsproblem darstellen.

Hierzu definiert man eine Funktion, die zu jeder gültigen Lösung x eines Problems einen Wert $f(x)$ zuordnet. Das Problem ist gelöst, wenn man das x^* findet, das die **Auswertungsfunktion** f minimiert.

Im Folgenden wollen wir also das Minimum einer Funktion $f\colon \mathbb{R} \to \mathbb{R}$ finden, ohne alle Stellen $x \in \mathbb{R}$ auszuwerten.

Daher fordern wir eine gewisse **Regularität** von f. Wir benötigen mindestens **Stetigkeit**, da wir andernfalls das Minimum der Funktion

$$\begin{aligned} f\colon \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases} \end{aligned}$$

nur Finden können, wenn f an allen Stellen ausgewertet wird.

Differenzierbare Funktionen

Im Weiteren gehen wir sogar davon aus, dass $f \in C^1(\mathbb{R})$ ist, d.h. dass f mindestens einmal **stetig differenzierbar** ist.

Wir wissen dann, dass das Minimum x^* von E die Bedingung $f'(x^*) = 0$ erfüllt, d.h. x^* ist ein **kritischer Punkt** von f. Im Folgenden sind wir daran interessiert einen kritischen Punkte zu finden. Es handelt sich hierbei nicht notwendigerweise um ein globales Minimum.

Ist nun $f'(x_0) < 0$, so gibt es ein $\varepsilon > 0$, so dass

$$\begin{aligned} 0 &> \frac{f(x_0 + h) - f(x_0)}{h} && \forall h, |h| < \varepsilon \\ f(x_0) &> f(x_0 + h) && \forall h, 0 < h < \varepsilon \end{aligned}$$

Analog gilt für $f'(x_0) > 0$, dass $f(x_0) > f(x_0 - h)$, d.h. wenn man f in **Richtung der negativen Ableitung variiert**, sinkt der Wert von f.

Iterationsfolge

Benutzen wir die Taylorapproximation von f, so erhalten wir

$$\begin{aligned} f(x_0 + h) &\approx f(x_0) + f'(x_0) \cdot h \\ f(x_0 - t f'(x_0)) &\approx f(x_0) - \underbrace{t f'(x_0)^2}_{\text{erwartete Reduktion}} , \end{aligned}$$

d.h. wir können eine größere Reduktion von f erwarten, wenn t größer ist.

Allerdings gilt die obige Approximation nur für $t\,|f'(x_0)| < \varepsilon$.
Somit muss t groß gewählt sein, aber nicht zu groß.

Insgesamt erhalten wir also eine Iterationsfolge zum Finden eines lokalen Minimums von f:

$$x_{k+1} = x_k - t_k \cdot f'(x_k)$$

Aber: Wie ist t_k zu wählen?

Schrittweite

Bei der Iterationsfolge

$$x_{k+1} = x_k - t_k \cdot f'(x_k)$$

nennen wir t_k die **Schrittweite**. Eine optimale Schrittweite zu finden, d.h. eine Schrittweite, so dass f möglichst stark sinkt ist nicht einfach.

Ein bewährtes Verfahren besteht darin, mit einer festen Schrittweite, z.B. $t = 1$ zu starten und diese iterativ zu verbessern:

```
1    // f: function  -  x: position  -  d: derivative
2    while (f(x-t*d)>=f(x)) t /= 2.0;
3    x = x - t*d;
```

Hierbei wird benutzt, dass das komplette Intervall $(0; \varepsilon)$ Schrittweiten liefert, die alle die Funktion f reduzieren.

Konvergenz

Die Iterationsfolge x_k ist derart, dass $f(x_k)$ eine **monoton fallende Folge** ist, die durch das Minimum von f **nach unten beschränkt** ist.

Daher konvergiert $f(x_k)$. Allerdings konvergiert $f(x_k)$ nicht notwendigerweise zu einem lokalen Minimum.

Um ein lokale Minimum zu gewährleisten, muss man zusätzliche Eigenschaften von der Schrittweite fordern.

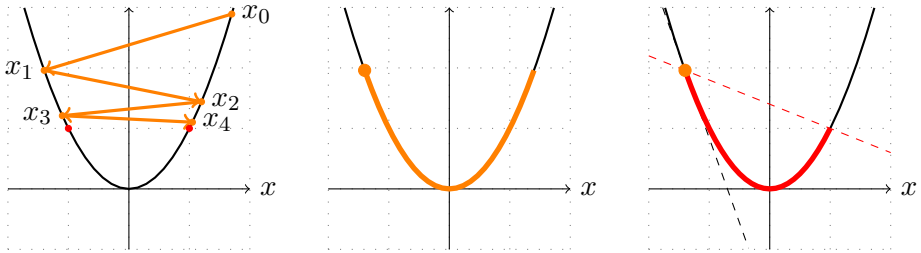
Ein Beispiel hierfür ist die **Armijo-Bedingung** $(0 < \alpha < 1)$

$$f(x_{k+1}) \leqslant f(x_k) - \alpha \cdot t_k \cdot f'(x_k)^2$$

Diese Bedingung garantiert ein lokales Minimum.

Oft wählt man ein kleines α (z.B. $\alpha = 10^{-4}$).

Armijo-Bedingung (Bsp.)



$f(x_k)$ konvergiert nicht notwendigerweise zum Minimum.

Dies liegt daran, dass die Menge gültiger Schrittweiten zu groß ist.

Die Armijo-Bedingung schränkt die Schrittweiten so ein,
dass Konvergenz zu einem lokalen Minimum garantiert wird.

Dies wird erreicht indem man nur solche Schrittweiten zulässt, die zumindest einen Anteil $\alpha < 1$ der erwarteten Reduktion garantiert.

Gradienten-Abstieg

In den meisten Anwendungen ist man an der Minimierung einer Funktion $f\colon \mathbb{R}^n \to \mathbb{R}$ interessiert.

Anstatt die Stelle x_n in die Richtung der negativen Ableitung zu verschieben, verschiebt man sie in die Richtung des negativen Gradienten. Man nennt dieses Verfahren das **Gradienten-Abstiegs-Verfahren (eng. Gradient Descent)**.

Die Iterationsfolge ist

$$x_{n+1} = x_n - t_n \cdot \nabla f(x_n)$$

Die Armijo-Bedingung wird dann zu

$$f(x_{n+1}) \leqslant f(x_n) - \alpha \cdot t_n \cdot \|\nabla f(x_n)\|^2$$

Newton-Verfahren im $\mathbb{R}^n$

Zur **Nullstellen-Berechnung** einer Funktion $g\colon \mathbb{R}^n \to \mathbb{R}^n$ können wir das Newton-Verfahren benutzen.

Für $n = 1$ erhalten wir die klassische Newton-Iteration

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

Für allgemeines n gilt dann

$$x_{k+1} = x_k - J[g](x)^{-1} g(x) \qquad J[g](x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

wobei $J[g](x)$ die **Jacobi-Matrix** von g an der Stelle x ist.

Minimieren mittels Newton-Verfahren

Wenn wir an einem **lokalen Minimum** von $f\colon \mathbb{R}^n \to \mathbb{R}$ interessiert sind, reicht es die Nullstellen von $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$ zu finden.

Hierzu können wir das Newton-Verfahren benutzen:

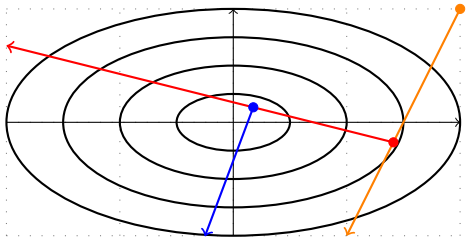
$$x_{k+1} = x_k - H(x)^{-1}\nabla f(x),$$

wobei $H(x)$ die (symmetrische) **Hesse-Matrix** von f ist:

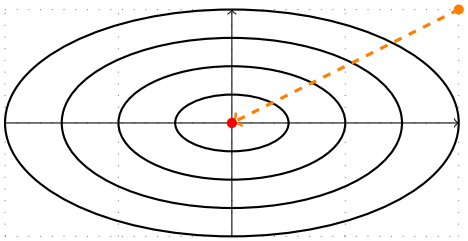
$$H(x) = J[\nabla f](x) \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix} \in \mathbb{R}^{n\times n}$$

In jedem Schritt muss ein lineares Gleichungssystem gelöst werden.

Newton-Verfahren (Bsp.)



Gradienten-Abstieg



Newton-Schritt

In jedem Schritt des **Gradienten-Abstiegs-Verfahrens** reduziert man die Funktion entlang einer Richtung (Gradienten-Richtung).

Das kann das zu vielen Iterationen führen.

Das Newton-Verfahren erreicht das Minimum in wesentlich weniger Iterationen.

Allerdings muss in jedem Newton-Schritt ein neues lineares Gleichungssystem (LGS) gelöst werden.

Iterative Lösung von LGS

Im Folgenden wollen wir uns mit der Lösung linearer Gleichungen $Ax = b$ beschäftigen, die beim Newton-Verfahren auftreten.

Weil Hesse-Matrizen symmetrisch sind, fordern wir, dass A **symmetrisch** ist.

Oft treten in der Praxis Matrizen auf, die nur $\mathcal{O}(n)$ Einträge besitzen, die $a_{ij} \neq 0$ erfüllen. Solche Matrizen heissen **dünn besetzt (eng. sparse)**.

Ist eine Matrix A dünn besetzt, so lässt sich die Matrix-Vektor-Multiplikation in $\mathcal{O}(n)$ Schritten durchführen.

Ziel: Finde ein iteratives Verfahren, dass schnell konvergiert.
Dann werden statt $\mathcal{O}(n^3)$ nur $\mathcal{O}(n) \cdot \#$Iterationen Schritte benötigt.

Vorteil: Der Speicherbedarf ist weiterhin $\mathcal{O}(n)$ und nicht $\mathcal{O}(n^2)$ wie bei der LU- oder der QR-Zerlegung. (Oft ist $n > 10^6$)

Richardson-Iteration

Ziel bei iterativen Verfahren ist es, ein Fixpunktproblem zu formulieren:

$$b = Ax = (A - I)x + x \quad \Leftrightarrow \quad x = x + (b - Ax)$$

Daraus ergibt sich die **Richardson-Iteration**

$$x_{k+1} = x_k + (b - Ax_k) =: \Phi(x_k)$$

Damit $x^* = A^{-1}b$ ein anziehender Fixpunkt ist, muss Folgendes gelten:

$$1 > \limsup_{h \to 0} \frac{\|\Phi(x^* + h) - \Phi(x^*)\|}{\|h\|} = \limsup_{h \to 0} \frac{\|(I - A)h\|}{\|h\|} = \|A - I\|_2$$

x^* ist ein anziehender Fixpunkt genau dann, wenn für alle **Eigenwerte** λ_i von A gerade $0 < \lambda_i < 2$ gilt.

Das Richardson-Verfahren ist also nur sinnvoll, wenn $A \approx I$ gilt.

Richardson-Iteration (Bsp.)

Betrachten wir beispielhaft folgende LGS

$$\underbrace{\begin{pmatrix} 0.5 & 0 \\ 0 & 1 \end{pmatrix}}_{A_0} \cdot x = \underbrace{\begin{pmatrix} 0.5 \\ 1 \end{pmatrix}}_{b_0} \quad \text{und} \quad \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}}_{A_1} \cdot x = \underbrace{\begin{pmatrix} 1 \\ 3 \end{pmatrix}}_{b_1},$$

die beide die eindeutige Lösung $x = \binom{1}{1}$ haben.

Initialisieren wir die Richardson-Iteration $(x + (b - Ax))$ mit dem Nullvektor, erhalten wir folgende Iterationsfolgen:

A_0 :	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.5 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0.75 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0.875 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0.9375 \\ 1 \end{pmatrix}$	$\cdots$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
A_1 :	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -15 \end{pmatrix}$	$\cdots$	

Diagonaldominanz

Wir wollen das Richardson-Verfahren auch auf eine größere Klasse von Matrizen anwenden. Dies sind die Matrizen, die im Wesentlichen durch ihre Diagonaleinträge bestimmt sind, d.h. es gilt:

$$|d_i| = |a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|$$

Diese Matrizen heissen **diagonaldominant**.

Im Weiteren bezeichnen wir mit D die **Diagonalmatrix** von A, d.h. die Matrix, die die Diagonaleinträge von A besitzt und sonst aus Nullen besteht.

Bezeichnet man mit $B := D^{-1}A$, so gilt für die Eigenwerte λ von B gerade

$$|\lambda - 1| \leqslant \sum_{\substack{j=1 \\ i \neq j}} \frac{|a_{ij}|}{|d_i|} < 1 \qquad \text{(Gerschgorin)}$$

Jacobi-Verfahren

Wir haben gesehen, dass wir bei einer diagonaldominanten Matrix A, die Richardson-Iteration auf $D^{-1}A$ anwenden können.

Die Gleichung $Ax = b$ wird zu $D^{-1}Ax = D^{-1}b$ und wir erhalten:

$$x_{k+1} = x_k + (D^{-1}b - D^{-1}Ax_k) = D^{-1}\left[b - (A - D)x_k\right]$$

Dieses Verfahren heisst **Jacobi-Verfahren**.

Zerlegt man A in seine drei Komponenten:

$$A = L + D + U,$$

wobei L (bzw. U) die Einträge unterhalb (bzw. oberhalb) der Diagonalen von A besitzt, so lässt sich das Jacobi-Verfahren wie folgt umschreiben:

$$x_{k+1} = D^{-1}\left[b - (L + U)x_k\right]$$

Implementierung (Jacobi)

Eine Jacobi-Iteration

$$x_{k+1} = D^{-1}\left[b - (L+U)x_k\right]$$

kann man wie folgt implementieren:

```
1     // A: matrix - x_old: current vector - x_new: new vector
2     for (unsigned i=0; i<n; i++) {
3       x_new[i] = b[i];
4       for (unsigned j=0; j<i; j++) {
5         x_new[i] -= A[i][j]*x_old[j];
6       }
7       for (unsigned j=i+1; j<n; j++) {
8         x_new[i] -= A[i][j]*x_old[j];
9       }
10      x_new[i] /= A[i][i];
11    }
```

Es wird also in jedem Schritt die i-te Zeile nach x_i aufgelöst.

Gauß-Seidel-Verfahren

Ein anderes Verfahren, dass auf der Richardson-Iteration beruht ist das **Gauß-Seidel-Verfahren**. Es benutzt die Matrix $B = (D + L)^{-1}A$.

Die Gleichung $Ax = b$ wird zu $(D + L)^{-1}Ax = (D + L)^{-1}b$ und wir erhalten:

$$\begin{aligned} x_{k+1} &= x_k + (D + L)^{-1}(b - Ax_k) \\ &= (D + L)^{-1}\left[b - (A - D - L)x_k\right] \\ &= (D + L)^{-1}\left[b - Ux_k\right] \end{aligned}$$

Im Unterschied zum Jacobi-Verfahren hängt die i-te Komponente von x_{k+1} nur von der j-ten Komponente von x_k ab, wobei $j > i$ ist.

Das heißt, wir können die Gauß-Seidel-Iteration **in place**, d.h. ohne zusätzlichen Speicherplatz durchführen.

Gauß-Seidel-Iteration

In jedem Schritt wird folgende Gleichung (nach y) gelöst:

$$(D + L) \cdot y = b - U \cdot x$$

Das heißt:

$$\begin{aligned} \sum_{j=1}^{i} a_{ij}y_j &= b_i - \sum_{j=i+1}^{n} a_{ij}x_j \\ a_{ii}y_i &= b_i - \sum_{j=i+1}^{n} a_{ij}x_j - \sum_{j=1}^{i-1} a_{ij}y_j \end{aligned}$$

y_1 hängt also nur von A und x ab und kann unabhängig von den übrigen y_i berechnet werden. Analog hängt y_2 nur von A, x und y_1 ab und kann unabhängig von den übrigen y_i als Nächstes berechnet werden etc.

Man kann also in jedem Schritt die i-te Zeile nach y_i auflösen, wenn man zuvor die x_j $(j < i)$ durch y_j ersetzt.

Implementierung (Gauß-Seidel)

Eine Gauß-Seidel-Iteration

$$x_{k+1} = (D + L)^{-1} [b - U \cdot x_k]$$

kann man also wie folgt implementieren:

```
1     // A: matrix - x: current vector
2     for (unsigned i=0; i<n; i++) {
3       x[i] = b[i];
4       for (unsigned j=0; j<i; j++) {
5         x[i] -= A[i][j]*x[j];
6       }
7       for (unsigned j=i+1; j<n; j++) {
8         x[i] -= A[i][j]*x[j];
9       }
10      x[i] /= A[i][i];
11    }
```

Die Implementierung benutzt also immer die aktuellsten Werte von x.

Zusammenfassung

Wir haben zwei verschiedene iterative Verfahren zum Lösen von Gleichungssystemen gefunden, die auf der **Richardson-Iteration** beruhen.

Das **Jacobi-Verfahren** und das **Gauß-Seidel-Verfahren** lösen die i-te Zeile von A nach der i-ten Variablen auf.

Das Jacobi-Verfahren benutzt die Werte x_k der vorherigen Iteration und benötigt daher einen **zusätzlichen Speicherplatz** von $\mathcal{O}(n)$. Da die Reihenfolge der Einträge von x_{k+1} nicht vorgegeben ist, ist es einfacher, das Jacobi-Verfahren zu **parallelisieren**.

Das Gauß-Seidel-Verfahren arbeitet **in-place** und benötigt keinen zusätzlichen Speicherplatz. Da die Reihenfolge der Einträge von x_{k+1} vorgegeben ist, ist eine Parallelisierung üblicherweise komplizierter.

Beide Verfahren konvergieren für **diagonaldominate Matrizen**.

Darüber hinaus konvergiert das Gauß-Seidel-Verfahren ebenfalls für **symmetrische positiv-definite** Matrizen. (ohne Beweis)

Gauß-Seidel und Jacobi (Bsp.)

Um zu sehen, dass das Gauß-Seidel-Verfahren eine größere Problemklasse als das Jacobi-Verfahren lösen betrachten wir folgendes LGS

$$\begin{pmatrix} 4 & 3 & 3 \\ 3 & 4 & 3 \\ 3 & 3 & 4 \end{pmatrix} \cdot x = \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix},$$

welches die eindeutige Lösung $x = (1\ 1\ 1)^\top$ besitzt.

Initialisieren wir Jacobi (J) und Gauß-Seidel (GS) mit dem Nullvektor, erhalten wir folgende Iterationsfolgen (gerundet auf 2 Nachkommastellen):

(GS) :	$(2.5\ 0.625\ 0.156)^\top$	$(1.91\ 0.947\ 0.354)^\top$	$(1.52\ 1.09\ 0.538)^\top$
(J) :	$(2.5\ 2.5\ 2.5)^\top$	$(-1.25\ -1.25\ -1.25)^\top$	$(4.38\ 4.38\ 4.38)^\top$

Entrauschen von Bildern

Original

Verrauscht

Sensoren sind oft ungenau und liefern meist **verrauschte Daten**.

Die Entfernung der Verrauschung nennt man Entrauschung (eng. Denoising).

Traditionelle Verfahren benutzen hierfür Bildglättung (siehe FFT-Vorlesung).

Moderne Verfahren beschreiben dies als Minimierung einer Funktion.

Entrauschung durch Minimierung

Sei $u \in \mathbb{R}^n$ eine Diskretisierung des Eingabebildes.
Wir sind auf der Suche des entrauschten Bilde $v \in \mathbb{R}^n$, so dass

- v sich nicht stark von u unterscheidet und
- v kleinere Ableitungen als u besitzt.

Bezeichnen wir mit $D_1, D_2 \in \mathbb{R}^{n \times n}$ die diskretisierten Ableitungen $\frac{\partial}{\partial x_1}$ bzw. $\frac{\partial}{\partial x_2}$, so kann man folgende Funktion beschreiben:

$$f(v) = \|v - u\|^2 + \lambda \cdot \left[\|D_1 v\|^2 + \|D_2 v\|^2 \right]$$

Der **Parameter** λ beschreibt wie wichtig die Glattheit von u ist. Für $\lambda = 0$ ist $v = u$ das Minimum von f. Für wachsendes λ wird das minimierende v immer glatter. Für $\lambda \to \infty$ wird v schließlich konstant.

Verbesserte Entrauschung

Da die quadratische Bestrafung der Ableitungen zu Überglättungen führen kann, benutzt man üblicherweise die folgende Funktion

$$f(v) = \|v - u\|^2 + \lambda \cdot \left[\|D_1 v\|^2 + \|D_2 v\|^2 \right]^{\frac{p}{2}}$$

Für $p = 2$ erhält man die gleiche Funktion wie zuvor.
Für $p < 2$ werden die Ableitungen schwächer bestraft und man erhält üblicherweise bessere Ergebnisse.

Die so erhaltene Funktion ist für $p > 1$ zweimal differenzierbar und kann mit Hilfe von Gradienten-Abstieg oder mit Hilfe des Newton-Verfahrens minimiert werden. Außerdem kann gezeigt werden, dass jedes lokale Minimum das globale Minimum ist.

Für $p = 2$ muss nur ein einziger Newton-Schritt berechntet werden, da f dann quadratisch in v ist.

Ergebnisse

Bild-Glättung (Faltung)

Entrauschung ($p = 2$)

Ergebnisse für verschiedene p

Entrauschung ($p = 2$)

Entrauschung ($p = 1$)