Numerische Mathematik 1

Übung

October 31, 2018

Contents

1 Gleitkommaarithmetik

In diesem Kapitel beschäftigen wir uns mit der Zahlendarstellung im Rechner. Wenn wir verstehen, wie Zahlen im Rechner abgespeichert werden, können wir auch erkennen, wie groß die Rundungsfehler sind.

1.1 Binärdivision

Zur Wiederholung ein kleiner Vergleich zwischen Dezimal- und Binärdarstellung einiger Zahlen:

dezimal	binär
1	1
2	10
3	11

Zunächst wiederholen wir die Division im gewohnten Dezimalsystem. Wir wollen 1 durch 3 teilen: Wir überprüfen, wie oft 3 in 1 passt.

$$1 : 3 = 0,$$

Dann ziehen wir $0 \cdot 3$ von 1 ab und schreiben das Ergebnis in die nächste Zeile.

$$\begin{array}{ccccc} 1 & : & 3 & = & 0, \\ 1 & & & & \end{array}$$

Dann müssen wir eine Null hinzufügen (Das ist wie eine Erweiterung der Zähler auf beiden Seiten der Gleichung mit 10).

$$\begin{array}{ccccc} 1 & : & 3 & = & 0, \\ 1 & 0 & & & \end{array}$$

Und dann fängt das Ganze von vorn an. 3 passt dreimal in 10. Wir stellen fest, dass $10 - 3 \cdot 3 = 1$ und fügen wieder eine Null an.

$$\begin{array}{ccccc} 1 & : & 3 & = & 0,3 \\ 1 & 0 & & & \\ & 1 & 0 & & \end{array}$$

Wir erkennen eine Periode und ergänzen den Periodenstrich:

$$\begin{array}{ccccc} 1 & : & 3 & = & 0,\overline{3} \\ 1 & 0 & & & \\ & 1 & 0 & & \end{array}$$

Nun kommen wir zur Binärdivision. Wir wollen wieder eins (1 in binär) durch drei (11 in binär) teilen. Das Ergebnis sollte folgendermaßen aussehen.

$$\begin{array}{ccccccc} 1 & : & 1 & 1 & = & 0,\overline{01} \\ 1 & 0 & & & & \\ 1 & 0 & 0 & & & \\ & & 1 & & & \end{array}$$

Versuchen Sie, diese Rechnung nachzuvollziehen. Es geht am Einfachsten, wenn nirgends ins Dezimalsystem umgerechnet wird.

1.2 Null abziehen, Null hinzufügen

Es ist sehr hilfreich, sich zu vergewissern, was passiert, wenn Nullen an eine Zahl angehängt werden, oder wenn wir das Komma verschieben. Dazu folgende Beispiele:

Dezimal:

$1 \xrightarrow{0\text{ dazu}} 10$	mal zehn
$120 \xrightarrow{0\text{ weg}} 12$	durch zehn

Binär:

$1 \xrightarrow{0\text{ dazu}} 10$	mal zwei
$100 \xrightarrow{0\text{ weg}} 10$	durch zwei

Im letzten Kästchen sehen wir wie aus einer Eins durch Verdopplung eine Zwei wird und wie aus einer Vier durch Halbieren eine Zwei wird.

Aber was, wenn keine Nullen übrig sind, um sie hinten wegzunehmen? Dann können wir das Komma verschieben.

Binär:

$11 \xrightarrow{0\text{ weg}} 1,1$	durch zwei

Hier werden aus einer Drei durch Halbieren Drei Halbe.

1.3 Matlab/Octave Erste Schritte

Mit Alt+F2 bekommen wir in Linux in der Regel eine Möglichkeit ein Programm durch Eingabe des Programmnamens zu starten. Wir wollen ein Terminal starten (zum Beispiel eines der folgenden: xterm, konsole, terminal, ...). Im Terminal können wir dann

```
octave -h
```

eingeben, um eine Auflistung aller Parameter für octave zu sehen. Wir verwenden dann

```
octave --no-gui
```

um Octave im Terminal, also ohne GUI, zu starten. Nun erscheint die Eingabeaufforderung von Octave statt der des Terminals. Testen wir, ob wir richtig sind mit

```
5*3
```

und Enter. Nun können wir folgende Befehle ausprobieren und genau beobachten, was passiert, um Matlab zu erlernen.

```
A=[1 2 3];
A=[1 2 3]
A=[1;2;3]
A=[1 4;2 5;3 6]
A*[1 2]
A*[1 2 3]
```

Das Gleichungssystem

$$Ax = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

können wir dann mit dem Befehl

```
A\[1 2 3]
```

lösen. Der Backslash-Operator soll aber im Folgenden nicht verwendet werden, da wir die Algorithmen erlernen wollen, die hinter diesem Operator stehen. Mit

```
exit
```

kann man Octave wieder beenden. In beliebigem Texteditor wollen wir nun die Datei

```
factorial.m
```

editieren, die eine Matlab-Function mit der Bezeichnung

```
factorial
```

enthalten soll. Wichtig ist, dass die Datei genauso heißt, wie die Funktion, die in ihr beschrieben wird. Die Datei sieht nun folgendermaßen aus:

```
function y=factorial(x)
    y=1;
    for i=1:1:x
        y=y*i;
    end
end
```

Die Zeile

```
for i=1:1:x
```

bedeutet, dass i im ersten Schleifendurchlauf gleich 1 sein soll. Zwischen den Schleifendurchläufen soll es jeweils um 1 erhöht werden. Und der letzte Durchlauf soll mit $i = x$ geschehen.

1.4 Hexadezimalsystem

binär	hexadezimal	dezimal
1	1	1
10	2	2
11	3	3
$\vdots$	$\vdots$	$\vdots$
1001	9	9
1010	A	10
1011	B	11
$\vdots$	$\vdots$	$\vdots$
1111	F	15
10000	10	16

Die Umrechnung zwischen Binär und Hexadezimalsystem ist erstaunlich einfach. Wenn mann im Hexadezimalsystem das Komma um eine Stelle verschiebt, wird versechzehnfacht. Das enspricht vier Stellen Kommaverschiebung im Binärsystem. Ein Beispiel:

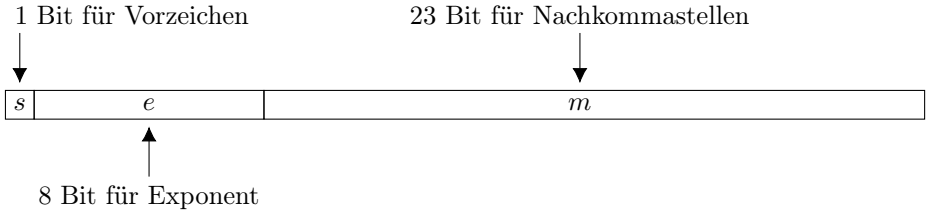
$$CB_{hex} = 1100\ 1011_{bin}$$

Wir müssen also nur die Viererblöcke in der Binärdarstellung mit obiger Tabelle übersetzen.

1.5 IEEE 754 (single precision)

Es gibt mehrere Formate, in denen Zahlen im Rechner gespeichert werden. Diese Formate werden im Standard IEEE 754 beschrieben. Eines davon ist das Gleitkommaformat (floating point). Davon gibt es die Varianten 32-bit (single precision) und 64-bit (double precision). Bei ersterem werden 32 Bit im Speicher belegt und bei letzterem 64 Bit.

Wir werden nun am sehen, was beim single-precision-Format in diesen 32 Bit gespeichert wird. Die grundlegende Aufteilung der 32 Bit im Speicher ist im folgenden Bild beschrieben.



Wie werden jetzt genau diese Bits bestimmt?

1.5.1 Umrechnung: Zahl → Speicher

Haben wir eine Zahl, so müssen wir sie erst in die Form

$$a \cdot 2^b, \text{ mit } a = 1, ...$$

umschreiben. Dann gilt die folgende Vorschrift.

$$\boxed{\begin{array}{c} \text{Zahl positiv} \Rightarrow s = 0 \\ \\ e = 01111111_{bin} + b \\ \\ m = a - 1 \end{array}}$$

Warum gerade so?

Da a immer eine 1 vor dem Komma hat, muss diese nicht gespeichert werden. Man nennt das auch implizite Eins. Deshalb speichern wir also nur die Nachkommastellen $m = a - 1$ von a. m wird auch als *Mantisse* bezeichnet.

Außerdem sehen wir, dass der Exponent kein eigenes Vorzeichenbit hat. Wir möchten aber trotzdem gerne positive und negative Exponenten darstellen. Der Trick ist, auf den Exponenten $127 = 01111111_{bin}$ draufzuaddieren. Somit wird aus allen Exponenten im Intervall $[-126, 127]$ eine positive Zahl generiert, die dann in den 8 Exponentenbits abgespeichert wird.

Ja, die $+128$ müssen wir ausschließen, weil der Standard vorsieht, dass die Zahl ∞ durch die Exponentenbits $e = 11111111$ kodiert wird. Die -127 müssen wir auch ausschließen. Wenn in den Exponentenbits $e = 00000000$ steht, so verändert sich die obige Vorschrift zu $b = -126$ und $m = a$ statt vorher $m = a - 1$.

Wir wollen ein Beispiel betrachten. Wie wird die Zahl 1 im Rechner gespeichert? Wir schreiben Sie in der Form

$$1 = 1 \cdot 2^0$$

Also ist das Vorzeichenbit

$$s = 0.$$

Die Exponentenbits sind

$$e = 01111111 + 0 = 01111111$$

und die Nachkommastellen/Mantisse sind

$$m = 00000000000000000000000$$

Zusammen sieht es also folgendermaßen aus.

$$\underbrace{\boxed{0}}_{} \underbrace{\boxed{011\ 1111}}_{3\quad F} \underbrace{\boxed{1}}_{} \underbrace{\boxed{00000000000000000000000}}_{8}$$

Mit geschweiften Klammern haben wir jeweils Viererblöcke zu einer Hexadezimalzahl zusammengefasst. Dadurch können wir leicht die Hexadezimaldarstellung der im Rechner gespeicherten Bits aufschreiben

```
3F800000
```

und das mit dem Matlab-Befehl

```
num2hex(single(1))
```

überprüfen. Die Funktion

```
single()
```

sorgt dafür, dass die Zahl mit 32 statt 64 Bit abgespeichert wird und

```
num2hex()
```

gibt die Hexadezimaldarstellung des von der übergebenen Zahl eingenommenen Speicherbereiches an.

2 Eliminationsverfahren, LR-Zerlegung

Die Untersuchung des Gauß'schen Eliminationsverfahrens dient in dieser Veranstaltung mehr dem Erlernen der Stabilitätsanalyse. Außerdem sollen einige Konzepte der Programmierung wiederholt werden.

2.1 Ein paar Hinweise zum Programmieren

Zum Beispiel um die Laufzeit der Implementierung eines Algorithmus' zu untersuchen, ist es nützlich einen Graphen (engl. plot) der Laufzeit zu erzeugen.

2.1.1 Plotting mit Octave

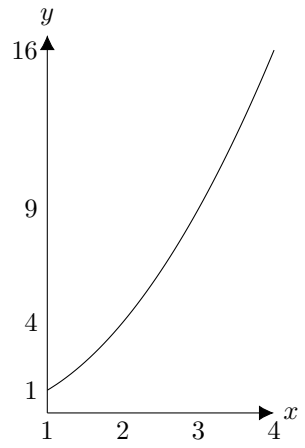
Der Befehl

```
plot(a,b)
```

macht aus zwei Vektoren einen Plot, wobei im ersten Vektor die x-Koordinaten und im zweiten die y-Koordinaten gespeichert sind. Also erzeugt zum Beispiel der Befehl

```
plot([1 2 3 4],[1 4 9 16]);
```

den Graphen einer quadratischen Funktion:

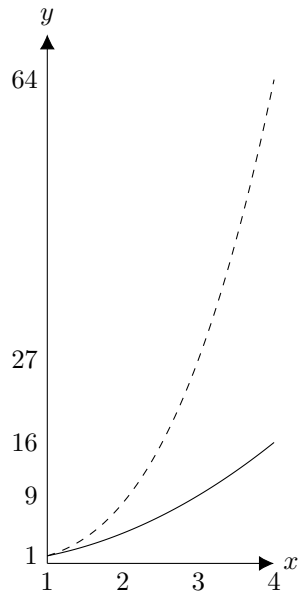


Natürlich stellt sich die Frage, woher Octave weiß, wie der Graph sich zwischen den in diesem Beispiel nur vier angegebenen Wertepaaren verhält. Natürlich kann es das nicht wissen. Um sicher zu gehen, dass der richtige Graph gezeichnet wird, sollte man so viele Wertepaare angeben, wie der Graph Pixel hat - also auf üblichen Displays müsste man jeweils etwa 1000 x und y-Werte angeben.

Wenn wir mehrere Funktionen in einem Graphen abbilden möchten, so kann man an `plot` auch eine Matrix übergeben, deren Spalten die y-Koordinaten der einzelnen Funktionen enthalten. Die x-Koordinaten werden dabei implizit als `[1 2 3 ...]` angenommen. Der Befehl

```
plot([1 1;4 8;9 27;16 64]);
```

zeichnet die Graphen einer quadratischen und einer kubischen Funktion wie folgt:



2.1.2 Laufzeit von Vektoroperationen

In Octave kann man

```
A(1,:)=B(1,:)+C(1,:)
```

schreiben, um die Summe aus erster Zeile von `B` und erster Zeile von `C` in die erste Zeile von `A` abzuspeichern. Obwohl die Zeile nur ein Pluszeichen enthält, werden für die Berechnung natürlich so viele Additionen benötigt, wie die Matrizen Spalten haben. Die Laufzeit muss also gleich der des äquivalenten Codes

```
for i=1:size(B,2)
    A(1,i)=B(1,i)+C(1,i);
end
```

angenommen werden. `size(B,2)` im obigen Code gibt übrigens die Anzahl der Spalten von `B` an. Die Spalten werden durch den zweiten Index gezählt. `size(B,1)` würde die Anzahl der Zeilen angeben.

2.1.3 Zufällige Matrizen

Der Befehl

```
rand(10)
```

erzeugt eine 10×10-Matrix voller zufälliger Zahlen. Diese Zahlen sind auf dem Intervall $[0, 1]$ *gleichverteilt*. Wenn eine nicht-quadratische Matrix gewünscht wird, kann der Befehl

```
rand(n,m)
```

genutzt werden.

```
randn(10,1)
```

wiederum gibt einen Vektor der Länge 10 aus, der *normalverteilte* Zufallsvariablen enthält. Dafür steht das `n` hinter `rand`.

2.1.4 Laufzeitmessung

Mit dem Befehl `tic` können wir eine Stoppuhr starten, deren Zeit wir mit `toc` ablesen können. Der Code

```
tic;
...some code here
y=toc;
```

speichert in `y` die Zeit, die der Code zwischen `tic` und `toc` benötigt hat.

2.1.5 Variable Taktrate (frequency scaling)

Fast alle modernen Prozessoren besitzen die Fähigkeit ihre Geschwindigkeit der momentanen Belastung anzupassen. Kriegt der Prozessor keine Rechenoperationen fährt er seine Geschwindigkeit auf ein Minimum herunter. Kommen dann aber plötzlich viele Anweisungen dauert es einige Sekunden, bis der Prozessor auf höchster Geschwindigkeit läuft. Aus diesem Grund muss der Prozessor vor der Benutzung von `tic` und `toc` aufgeweckt werden, damit die Ergebnisse vergleichbar sind. Dies kann man zum Beispiel tun, indem vor dem ersten Aufruf von `tic` eine etwa drei Sekunden dauernde Rechenoperation ausgeführt wird.

2.1.6 Erkennen der Asymptotischen Laufzeit

Wir haben also mit Hilfe von `tic` und `toc` die Laufzeit t für verschiedene Problemgrößen n ermittelt und erhalten somit zum Beispiel die folgende Tabelle.

n	t
2	3
4	24

Unter der Annahme, dass die Laufzeit sich als polynomielle Funktion

$$g : n \mapsto c \cdot n^k$$

darstellen lässt, können wir nun das Gleichungssystem

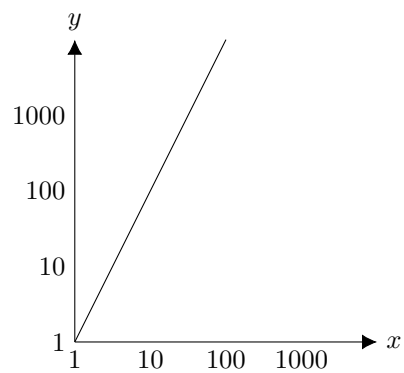
$$\begin{aligned} 3 &= c \cdot 2^k \\ 24 &= c \cdot 24^k \end{aligned}$$

aufstellen und lösen, um den Exponenten k zu bestimmen.

Eine andere Methode ist, einen logarithmischen Graphen mittels

```
loglog(a,b)
```

zu erzeugen. Dieser hat auf beiden Achsen eine logarithmische Skala. Das sieht wie folgt aus.



Im Graphen können wir die Steigung ablesen (im Beispiel ist die Steigung 2) und dadurch feststellen, welchen Grad das Polynom hat (im Beispiel eine quadratische Funktion).

2.2 Die LR- bzw. LU-Zerlegung

Ziel dieses Abschnittes ist es, das lineare Gleichungssystem

$$Ax = b$$

zu lösen, wobei die quadratische Matrix A und der Vektor b gegeben sind und x gesucht. Sie haben wahrscheinlich bereits den Gauß'schen Eliminationsalgorithmus zum Lösen linearer Gleichungssysteme kennengelernt. Dieser kann auch verwendet werden, um die Matrix A in ein Produkt

$$A = LR$$

zu zerlegen, wobei L eine linke untere Dreiecksmatrix und R eine rechte obere Dreiecksmatrix ist. Diese Zerlegung macht dann das Lösen des Gleichungssystems sehr einfach.

Example 1 (LR-Zerlegung ohne Pivotisierung)**.** Beginnen wir zur Veranschaulichung mit einem Beispiel. Dafür Benötigen wir nur die Matrix A. Die rechte Seite b des Gleichungssystems ist für die LR-Zerlegung erstmal nicht nötig.

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{pmatrix}$$

Jetzt wird der Gauß'sche Algorithmus gestartet. Wir wollen ein vielfaches der ersten Zeile auf die zweite und dritte Zeile addieren, um alle bis auf den ersten Eintrag in der ersten Spalte von A zu eliminieren (auf Null zu bringen). Nennen wir die erste Zeile I, die zweite Zeile II und die dritte Zeile III, dann sind die Operationen die folgenden:

$$I := I$$

$$II := II - 2 \cdot I$$

$$III := III + 1 \cdot I$$

Diese drei Operationen lassen sich als Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ +1 & 0 & 1 \end{pmatrix}$$

schreiben, die von links an A multipliziert wird. Diese Matrix wollen wir

$$(I - G_1)$$

nennen, so dass

$$G_1 = \begin{pmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

Wenn wir $A_0 := A$ definieren, haben wir nach einem Schritt des Gauß'schen Algorithmus

$$A_1 = (I - G_1)A_0$$

Nun schreiben wir im allgemeinen Fall

$$A_{n-1} = (I - G_{n-1})A_{n-1} = (I - G_{n-1}) \cdot ... \cdot (I - G_1)A_0,$$

wobei A_{n-1} die gesuchte Matrix R ist, also

$$R = (I - G_{n-1}) \cdot ... \cdot (I - G_1)A$$

gilt. Multiplikation von links mit allen $(I - G_i)$ liefert

$$LR = A$$

wobei

$$L = \cdot(I + G_1) \cdot ... \cdot (I + G_{n-1}),$$

wobei verwendet wurde, dass $(I - G_i)^{-1} = (I + G_i)$, wenn G_i nur in einer Spalte unterhalb der Diagonalen Nichtnulleinträge hat.

2.2.1 Implementierung des Gauß'schen Algorithmus'

Es fällt auf, dass $L = (I + G_1 + G_2 + ... + G_{n-1})$ ist. Außerdem benötigen wir die Einträge aus A, zu denen wir die Einträge von L bereits kennen nicht mehr. Als Beipiel können wir im ersten Gauß-Schritt die Einträge von G_1 direkt in A abspeichern und die vorhandenen Einträge überschreiben, da sie sowieso Null werden. So benötigen wir keinen zusätzlichen Speicherplatz.

Example 2. Wir beginnen mit der Matrix A:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{pmatrix}$$

Im nächsten Schritt berechnen wir die Einträge von G_1, als Quotienten $2 = \frac{4}{2}$ und $-1 = \frac{-2}{2}$ und speichern sie an die passende Stelle:

$$\begin{pmatrix} \underline{2} & 1 & 1 \\ 2 & |1 & 0 \\ -1 & |2 & 1 \end{pmatrix}$$

Jetzt stehen links von den Strichen schon die richtigen Einträge von L. Rechts davon muss jetzt auf die Zeilen ein Vielfaches der ersten Zeile addiert werden. Dann sieht das Resultat so aus:

$$\begin{pmatrix} \underline{2} & 1 & 1 \\ 2 & |-1 & -2 \\ -1 & | \ 3 & 2 \end{pmatrix}$$

Danach geht es mit der nächsten Spalte weiter, und so weiter.

2.2.2 Lösung des LGS mittels LR-Zerlegung

Wenn wir

$$y := Rx$$

definieren, dann zerlegt sich das Gleichungssystem

$$LRx = b$$

in zwei Teilsysteme

$$Ly = b$$

und

$$Rx = y$$

, die nacheinander gelöst werden können. Beim ersten wird mit dem Lösen der ersten Zeile begonnen. Beim zweiten System Lösen wir von hinten, also beginnend mit der letzten Zeile. Diese naheliegenden Verfahren heißen *Vorwärts- bzw. Rüchwärtseinsetzen.*

2.2.3 Komplexität

Der Algorithmus zur Berechnung der LR-Zerlegung benötigt

$$\frac{1}{3}n^3 - \frac{1}{3}n$$

Operationen. Das Lösen der zwei Dreieckssysteme benötigt insgesamt n^2 Operationen.

2.2.4 Ausführbarkeit, Pivotisierung

Bei der Matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

kann der oben angegebene Algorithmus nicht ausgeführt werden, da schon bei der Berechnung der Elemente von G_1 durch Null geteilt werden müsste!

Dieses Problem wird gelöst, indem man zusätzlich eine Permutationsmatrix P in der Zerlegung

$$PA = LR$$

zulässt. Der Algorithmus muss entsprechend angepasst werden. Zentrale Idee ist es, die Null auf der Diagonalen zu vermeiden, indem die Zeile, die die Null enthält mit einer anderen Zeile weiter unten getauscht wird. Es wird also erst getauscht

$$\tilde{A}_0 = T_1 A_0$$

wobei T_1 eine Vertauschungsmatrix (auch Transpositionsmatrix genannt) ist. Erst danach wird ein Gauß-Schritt angewendet

$$A_1 := (I - G_1) \cdot \tilde{A}_0 = (I - G_1) \cdot T_1 \cdot A_0.$$

Das wird so fortgeführt bis zu

$$A_{n-1} := (I - G_{n-1}) \cdot T_{n-1} \cdot A_{n-2}$$

woraus sich zusammengefasst

$$R := A_{n-1} = (I - G_{n-1}) \cdot T_{n-1} \cdot ... \cdot (I - G_1) \cdot T_1 \cdot A$$

ergibt. Wir wollen die Transpositionsmatrizen T_i alle aufmultiplizieren, um die Permutationsmatrix P zu erhalten. Leider stehen sie nicht alle nebeneinander. Dank der Eigenschaft

$$T_j \cdot (I - G_i) = (I - T_j \cdot G_i) \cdot T_j \text{ für } j > i$$

können wir die T_i alle nach rechts *durchziehen* und erhalten

$$R = (I - G_{n-1})(I - T_{n-1}G_{n-2}) \cdot ... \cdot (I - T_{n-1} \cdot ... \cdot T_2 G_1) \cdot (T_{n-1} \cdot ... \cdot T_1) \cdot A.$$

Somit ist

$$P = T_{n-1} \cdot ... \cdot T_1$$

und

$$L = (I + T_{n-1} \cdot ... \cdot T_2 G_1) \cdot ... \cdot (I + T_{n-1} G_{n-2})$$

Damit zeigt sich, dass wenn die Einträge von L und R in derselben Matrix gespeichert werden, die Vertauschungen auf beide gleichzeitig angewendet werden müssen/können!

2.2.5 A-priori-Fehler

A priori nennt man in der Numerik jede Aussage, die ohne eine berechnete Lösung getätigt werden kann. Das sind konkret Fehlerabschätzungen, die nicht von der berechneten Lösung abhängen. Im Gegensatz dazu steht der *A-posteriori*-Fehler, welcher nur anhand der schon berechneten Lösung bestimmt werden kann und von dieser abhängt. Der A-priori-Fehler ist abhängig von den Größen der Einträge von L und R. Siehe dazu auch die Mitschrift zur Vorlesung von Harry Yserentant, Satz 3.21, 3.22 und 3.23. Die Größe der Elemente von L und R lässt sich übrigens gut beschränken, wenn eine Pivotisierung verwendet wird, die immer das größte Element auf die Diagonale tauscht.

2.2.6 relative Kondition

Wir möchten die Lösungsfunktion f definieren. Sie gibt uns die Lösung zu einem vorgegebenen numerischen Problem.

Example 3. Soll das Gleichungssystem $Ax = b$ gelöst werden, so ist die Lösungsfunktion

$$f : \mathbb{R}^{n \times n} \times \mathbb{R}^n \to \mathbb{R}^n : (A, b) \mapsto x = A^{-1} b.$$

Example 4. Soll die Zahl x quadriert und dann auf 2 addiert werden, so ist die Lösungsfunktion

$$f : \mathbb{R} \to \mathbb{R} : x \mapsto 2 + x^2.$$

Da die Eingabewerte (im ersten Beispiel die Matrix A und die rechte Seite b bzw. im zweiten Beispiel die Zahl x) im Rechner im Gleitkommaformat abgespeichert werden sollen, lässt sich ein Rundungsfehler nicht verhindern. Auch sonst sind Eingabedaten in der Praxis in der Regel mit Fehlern behaftet.

Die Fragestellung ist nun, wie sich dieser Fehler auf das Ergebnis auswirken kann (selbst wenn man einen exakten Algorithmus annimmt).

Der *relative Fehler* von x ist der (größtmögliche) Fehler von x geteilt durch x, also

$$\frac{\Delta x}{x}$$

.

Die *relative Kondition* κ_{rel} ist nun das Verhältnis zwischen den relativen Fehler der Ausgabe und dem relativen Fehler der Eingabe, also

$$\kappa_{rel} = \frac{\text{relativer Fehler der Ausgabe}}{\text{relativer Fehler der Eingabe}}.$$

Anders ausgedrückt können wir den (möglichen) relativen Fehler der Ausgabe berechnen, indem wir den relativen Fehler der Eingabe mit der relativen Kondition multiplizieren.

2.2.7 A-posteriori-Fehler

siehe 2. Hausaufgabenblatt.

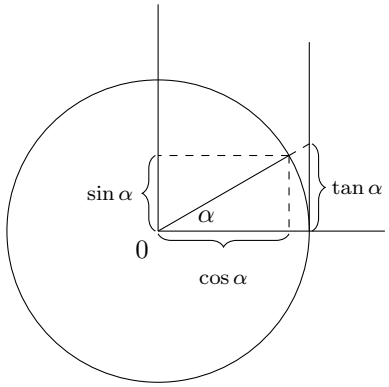
3 Die Orthogonale Projektion und die Bestapproximation

Die orthogonale Projektion ist vielleicht **das** Werkzeug schlechthin in der Numerischen Mathematik. Sie wird in fast jedem in dieser Vorlesung behandelten Themengebiet verwendet und vielleicht sogar überall in der Numerik. Die Diskrete Fouriertransformation (siehe Kapitel ...) **ist** tut nichts anderes, als die Längen der orthogonalen Projektionen auf besondere Basisvektoren zu berechnen. Ohne sie wäre die Übertragung und Speicherung von Bild, Ton und Film, so, wie wir sie heute kennen, undenkbar. Auch zur Lösung von partiellen Differentialgleichungen (Strömungen von Gasen und Flüssigkeiten, Elastizität von z. Bsp. Brücken, Moleküldynamik und Teilchenphysik) ist die orthogonale Projektion unabdingbar. Zum einen bildet sie die Grundlage des Verfahrens der konjugierten Gradienten (siehe Kapitel ...). Dieses kann große Gleichungssysteme, wie sie bei der Lösung von Differentialgleichungen vorkommen, erstaunlich schnell lösen. Außerdem basieren die Gauß'schen Quadraturformeln (siehe Kapitel ...) auf Orthogonalität. Auch diese werden zur Lösung partieller Differentialgleichungen fast überall benötigt.

Nicht zuletzt benötigen wir die orthogonale Projektion im folgenden Kapitel zur Beschreibung von Spiegelungen, welche ihrerseits wiederum die Grundlage des Householder-Algorithmus für die QR-Zerlegung bilden.

3.1 Orthogonale Projektion auf eindimensionalen Unterraum

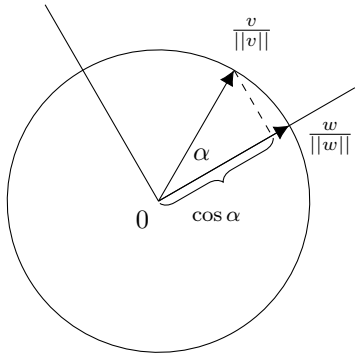
Wir wiederholen zunächst die Definition von Sinus, Kosinus und Tangens. Am Einheitskreis können wir die drei schön visualisieren:



3.1.1 Skalarprodukt zweier Vektoren der Länge 1

Insbesondere der Kosinus ist für das hier folgende interessant. Im nächsten Bild betrachten wir zwei Vektoren der Länge 1. Wir können den Einheitskreis in das Bild hineinzeichnen und sehen dann, dass die orthogonale Projektion von $\frac{v}{||v||}$ auf die von w aufgespannte Gerade $\cos\alpha$ ist. Außerdem kennen wir die Definition des Skalarproduktes als

$$\langle \frac{v}{||v||}, \frac{w}{||w||} \rangle = \cos\alpha.$$



Es folgt

$$\left|\left| P_{\mathrm{span}(w)} \left(\frac{v}{||v||} \right) \right|\right| = \langle \frac{v}{||v||}, \frac{w}{||w||} \rangle$$

wobei

$$P_W(v)$$

(in manchen Quellen statt P auch Π) für die orthogonale Projektion von v auf den Unterraum W steht.

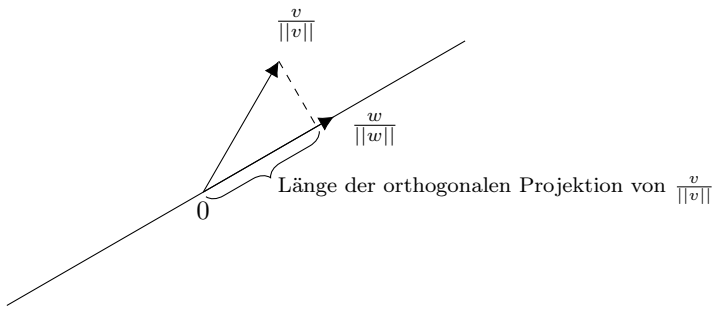
Der Kosinus nimmt nur Werte zwischen 1 und -1 an. Und genauso kann das Skalarprodukt zweier normierter Vektoren natürlich auch nur zwischen 1

und -1 liegen.

$$-1 \leq \langle \frac{v}{||v||}, \frac{w}{||w||} \rangle \leq 1$$

Dies ist natürlich nichts anderes als die Cauchy-Schwarz'sche Ungleichung.

3.1.2 Projektion auf 1-dimensionalen Teilraum (Gerade durch 0)



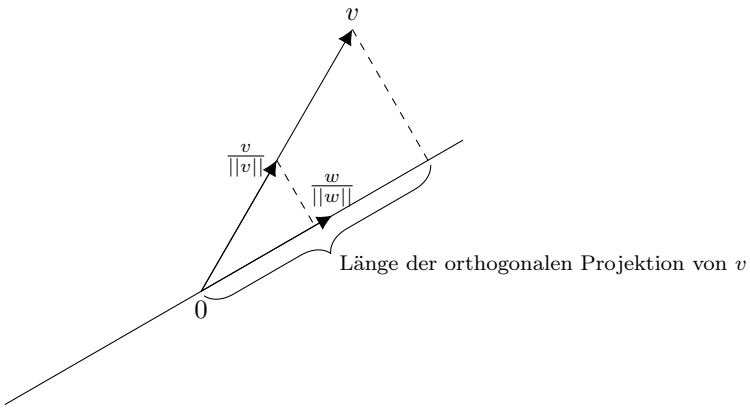
Wir sehen im obigen Bild nochmal ohne Kreis, die **Länge**

$$\langle \frac{v}{||v||}, \frac{w}{||w||} \rangle$$

der orthogonalen Projektion von

$$\frac{v}{||v||}$$

auf span(w) abgebildet. Wollen wir Vektoren beliebiger Länge projizieren, können wir den Strahlensatz, wie im folgendem Bild anwenden.



Wir sehen, dass die Länge der orthogonalen Projektion von v auf span(w) genau

$$||v|| \langle \frac{v}{||v||}, \frac{w}{||w||} \rangle = \langle v, \frac{w}{||w||} \rangle$$

ist.

Die Richtung der orthogonalen Projektion kann durch einen Vektor der Länge 1 angegeben werden und ist im eindimensionalen Fall natürlich

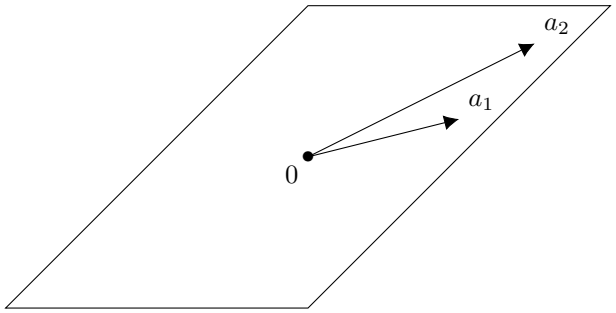
$$\frac{w}{||w||}.$$

Wenn wir Richtung mit Länge multiplizieren erhalten wir die Formel für die orthgonale projektion von v auf $\mathrm{span}(w)$ als

$$P_{\mathrm{span}(w)}(v) = \frac{w}{||w||}\langle v, \frac{w}{||w||}\rangle = w\frac{\langle v, w\rangle}{\langle w, w\rangle}.$$

3.2 Projektion auf 2-dimensionalen Untervektorraum

Jetzt wo wir alles über die Projektion auf einen 1-dimensionalen Untervektorraum wissen, können wir uns an mehrdimensionale Unterräume wagen. Dazu seien a_1 und a_2 zwei linear unabhängige Vektoren, die eine Ebene W aufspannen.



Wir fordern hier nichts weiter als die lineare Unabhängigkeit von a_1 und a_2. Sie können durchaus sehr dicht beieinander sein oder sehr unterschiedliche Längen haben. Die Aufgabe ist nun, die orthogonale Projektion auf die Ebene W zu berechnen. Ein intuitiver Ansatz wäre, auf die Gerade durch a_1 und auf die Gerade durch a_2 zu projizieren und die Ergebnisse zu addieren:

$$a_1\frac{\langle v, a_1\rangle}{\langle a_1, a_1\rangle} + a_2\frac{\langle v, a_2\rangle}{\langle a_2, a_2\rangle} \quad ?!$$

Dies liefert allerdings leider nicht die orthogonale Projektion. Es ist nicht einmal allein durch die Ebene W wohldefiniert. Man könnte Abhilfe schaffen, indem man die Vektoren a_i orthogonalisiert, zum Beispiel mittels Gram-Schmidt. In diesem Fall ist die obige Formel tatsächlich die orthogonale Projektion. Es gibt aber eine viel elegantere Formel, die vor allem in der Theorie von enormer Nützlichkeit ist. Diese wird nach dem folgenden Kapitel hergeleitet.

3.3 Allgemeine Definition der orthogonalen Projektion

Zunächst wollen wir rigoros definieren, was wir unter orthogonaler Projektion im Allgemeinen verstehen

Definition 5 (Orthogonale Projektion)**.** Sei V ein reeller Vektorraum mit Skalarprodukt und W ein Untervektorraum. y^* heißt orthogonale Projektion von x auf W falls

$$1. \quad y^* \in W \quad \textit{und}$$

$$2. \quad y^* - x \perp y \quad \forall y \in W.$$

Die zweite Bedingung kann auch als

$$\langle y^* - x, y \rangle = 0 \quad \forall y \in W$$

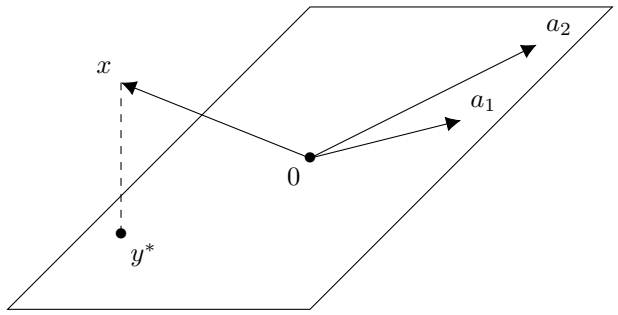
geschrieben werden.

Die erste Bedingung sagt, dass die Projektion natürlich in W sein soll. Die zweite Bedinung sagt, dass die Verbindungslinie zwischen zu projiziertem Punkt und seiner Projektion senkrecht auf dem Untervektorraum W steht.

3.4 Schöne Formel für die orthogonale Projektion

Wir starten also wieder mit der Situation, dass ein Untervektorraum W des $\mathbb{R}^n$ als Spann der linear unabhängigen Spalten $a_1, ..., a_m$ von A definiert sei. $(a_i)_{i=1,...,m}$ bildet also eine Basis von W.

Gesucht ist eine Formel für die orthogonale Projektion y^* von beliebigem Vektor x auf W wie in der folgenden Abbildung



Als Übungsaufgabe sei zu überprüfen, dass $A^T A$ invertierbar und die orthogonale Projektion eine lineare Abbildung ist. Hinweis: $A^T A$ ist symmetrisch und positiv definit. Der Fakt, dass ein Vektor v in W enthalten ist lässt sich ausdrücken durch die Existenz eines Vektors β so dass $v = A\beta$ (also v eine Linearkombination der Spalten von A) ist.

Weil die Projektion eine lineare Abbildung ist, gibt es eine Matrix so dass

$$y^* = Bx$$

und da y^* in W ist, gibt es einen Vektor β so dass

$$y^* = A\beta$$

gilt. Es folgt

$$A\beta = Bx$$

und nach Multiplikation mit A^T

$$A^T A\beta = A^T Bx.$$

Weil $A^T A$ invertierbar ist, können wir mit der Inversen multiplizieren und erhalten

$$\beta = (A^T A)^{-1} A^T Bx.$$

Es folgt also mit obigen Gleichungen

$$y^* = A(A^T A)^{-1} A^T Bx.$$

Dies setzen wir in Bedingung 2 der Definition der orthogonalen Projektion ein und rechnen

$$0 = \langle y^* - x, y\rangle = \langle y, A(A^T A)^{-1} A^T Bx - x\rangle.$$

Weil y beliebig aus W gewählt werden kann, können wir auch die Spalten von A wählen und erhalten

$$0 = A^T(A(A^T A)^{-1} A^T Bx - x) = A^T Bx - A^T x.$$

Wir sehen, dass diese Bedinung erfüllt ist, falls B die Identität ist. Also haben wir durch

$$y^* = A(A^T A)^{-1} A^T x$$

eine valide und elegante Formel für die orthogonale Projektion gefunden. Kleine Bemerkung am Rande, die in einem späteren Kapitel wiederkehren wird:

$$(A^T A)^{-1} A^T$$

ist gerade die Moore-Penrose-Pseudoinverse der Matrix A falls A vollen Rang und mehr Zeilen als Spalten hat. Wie ändert sich die Formel falls A mehr Spalten als Zeilen hat?

3.5 Element bester Approximation

Wir werden sehen, dass das *Element bester Approximation* identisch mit der orthogonalen Projektion ist. Allerdings benötigt seine Definition kein Skalarprodukt, sondern nur eine Norm. Es benötigt eigentlich auch keinen Untervektorraum sondern nur eine abgeschlossene Menge.

Definition 6 (Element bester Approximation)**.** Sei V ein reeller normierter Vektorraum und W ein Untervektorraum. y^* heißt Element bester Approximation von x auf W falls

$$1. \quad y^* \in W \quad \textit{und}$$

$$2. \quad ||x - y^*|| \leq ||x - y|| \quad \forall y \in W.$$

Die zweite Bedingung kann auch als

$$\langle y^* - x, y\rangle = 0 \quad \forall y \in W$$

geschrieben werden.

Theorem 7. *Orthogonale Projektion und Element bester Approximation sind identisch.*

Proof. Sei $y^* \in W$ das Element bester Approximation an x. Dann hat die Funktion

$$F : \alpha \mapsto ||x - (y^* - \alpha v)||^2$$

bei $\alpha = 0$ ein globales Minimum. F ist eine 1-dimensionale stetig differenzierbare Funktion. Aus der Schule wissen wir, dass bei einem Minimum die Ableitung verschwindet, also

$$0 = F'(0) = \frac{d}{d\alpha}|_{\alpha=0}||x - (y^* - \alpha v)||^2$$

$$= \frac{d}{d\alpha}|_{\alpha=0}\langle x - (y^* - \alpha v), x - (y^* - \alpha v)\rangle = -2\langle x - y^*, v\rangle.$$

Dies ist nichts anderes als die zweite Bedingung für die orthogonale Projektion.

Umgekehrt sei y^* die orthogonale Projektion von x auf W. Es gelte also

$$\langle x - y^*, v\rangle = 0 \quad \forall v \in W.$$

Dann gilt für alle nicht-verschwindenden $y \in W$

$$||x - y||^2 = ||x - y + y^* - y^*||^2 = \langle (x - y^*) + (y^* - y), (x - y^*) + (y^* - y)\rangle$$

$$= ||(x - y^*)||^2 + ||y^* - y||^2 + 2\langle (x - y^*), (y^* - y)\rangle \geq ||(x - y^*)||^2.$$

Hierbei wurde verwendet, dass $\langle (x - y^*), (y^* - y)\rangle$ verschwindet, da $(y^* - y)$ ein Element in W ist. $\square$

4 QR-Zerlegung (1)

5 Ausgleichsprobleme

6 Interpolation (1)

6.1 Interpolation allgemein

6.2 Polynominterpolation

6.2.1 Lagrangepolynome

6.2.2 Newton-Interpolation

6.2.3 Interpolationsfehler (1)

6.2.4 Hermite-Interpolation

7 Чебышёв(Tschebyscheff)-Polynome (1)

8 Diskrete Fouriertransformation (5)

Die Fouriertransformation

$$DFT : (f_0, ..., f_{n-1}) \mapsto (d_0, ..., d_{n-1})$$

ist definiert durch

$$d_j = \frac{1}{n} \sum_{k=0}^{n-1} f_k e^{-2\pi i \frac{k}{n} j}$$

Es fällt auf, dass bei geradem j und geradem n der Term

$$e^{-2\pi i \frac{k}{n} j}$$

sich eine gerade Anzahl an Umdrehungen in der komplexen Ebene um den Nullpunkt dreht. Definiere $n = 2m$ und $j = 2l$. Genauer gesagt gilt dann

$$e^{-2\pi i \frac{k+m}{n} j} = e^{-2\pi i \frac{k+m}{2m} 2l} = e^{-2\pi i \frac{k+m}{m} l} = e^{-2\pi i \frac{k}{m} l} e^{-2\pi i l} = e^{-2\pi i \frac{k}{m} l} = e^{-2\pi i \frac{k}{n} j}.$$

Daraus ergibt sich das folgende Lemma:

Lemma 8. *Sei $n = 2m$ und $DFT(f_0, ..., f_{n-1}) = (d_0, ..., d_{n-1})$. Dann gilt*

$$d_{2l} = \frac{1}{m} \sum_{k=0}^{m-1} \frac{1}{2}(f_k + f_{k+m}) e^{-2\pi i \frac{lk}{m}} = DFT(\frac{1}{2}(f_0 + f_m), ..., \frac{1}{2}(f_{m-1} + f_{2m-1}))$$

bzw.

$$d_{2l+1} = \frac{1}{m} \sum_{k=0}^{m-1} \frac{1}{2}(f_k - f_{k+m}) e^{-2\pi i \frac{k}{n}} e^{-2\pi i \frac{lk}{m}} = DFT(\frac{1}{2}(f_0 - f_m) e^{-2\pi i \frac{0}{n}}, ..., \frac{1}{2}(f_{m-1} - f_{2m-1}) e^{-2\pi i \frac{m-1}{n}})$$

Beachte, dass die DFTs auf der rechten Seite nur DFTs auf Vektoren von halber Länge sind. Wir können nun also eine DFT der Länge $2m$ berechnen, indem wir zwei DFTs der Länge m berechnen und dann geeignet zusammenfügen.

9 Numerische Integration (2)

9.1 Lineare Quadraturformeln

9.1.1 optimale Gewichte

9.1.2 Historische Formeln

9.1.3 Maximaler polynomieller Exaxtheitsgrad

9.1.4 Orthogonalpolynome

9.1.5 Gauß-Quadraturformeln

9.2 Romberg-Quadratur (R)

10 Wdh. Die Fixpunktiteration und Banach (1)

11 Newton-Verfahren

Das Ziel des Newtonverfahrens ist es, eine Nullstelle einer Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ zu finden. Dies ist insbesondere von Nutzen, um nichtlineare Gleichungssysteme zu lösen. Diese lassen sich nämlich immer in der Form

$$f(x) = 0$$

schreiben.

Das Newtonverfahren darf auf keinen Fall mit dem Gradientenverfahren verwechselt werden. Beim einen wird eine Nullstelle, beim anderen ein Minimum gesucht. Das eine konvergiert wahnsinnig schnell, das andere gemächlich. Das eine wird auf Funktionen $\mathbb{R}^n \to \mathbb{R}^n$, das andere auf Funktionen $\mathbb{R}^n \to \mathbb{R}$ angewendet.

Allerdings: Einen Zusammenhang gibt es zwischen beiden Verfahren. Das Gauß-Newton-Verfahren benutzt das Newton-Verfahren, um ein Minimum zu finden, indem die Nullstelle der Ableitung der zu minimierenden Funktion mittels Newton-Verfahren bestimmt wird.

11.1 Eindimensional

Wir möchten mit dem Newton-Verfahren eine Nullstelle einer eindimensionalen Funktion $f : \mathbb{R} \to \mathbb{R}$ finden.

Die Idee ist folgende (siehe dazu auch Abbildung 1): Wir beginnen wieder mit einem Startwert x_0, der möglichst schon in der Nähe der Nullstelle liegt.

An dieser Stelle linearisieren wir die Funktion. Die Linearisierung oder auch Tangente der Funktion ist nichts anderes als das erste Taylorpolynom.

Erinnerung: Wir hatten in Analysis die Ableitbarkeit einer Funktion definiert als die Existenz einer solchen affinen Funktion, die in einer Umgebung gegen die gegebene Funktion konvergiert, also tangential daran liegt.

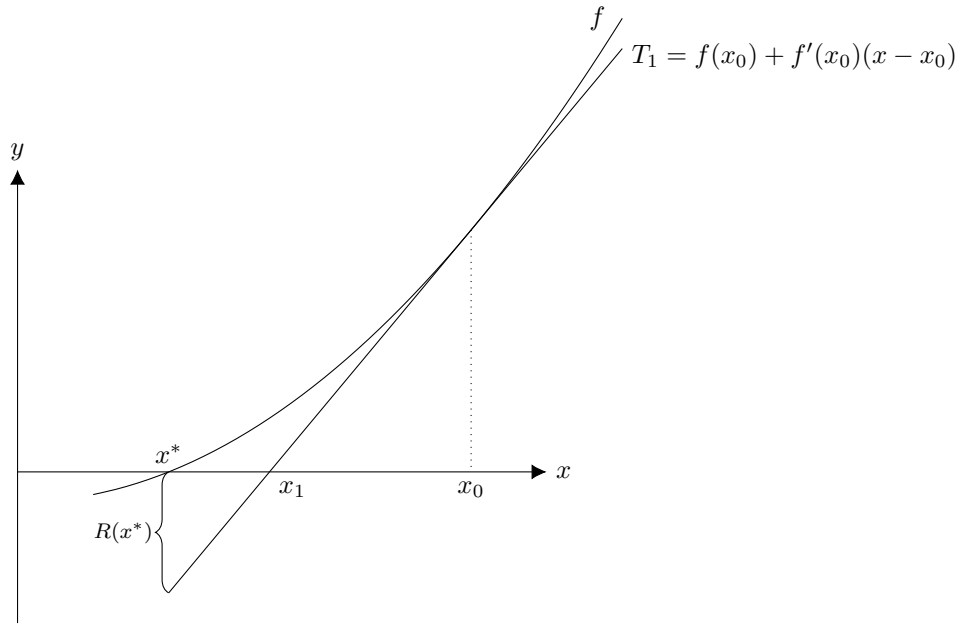


Figure 1: Newtonverfahren

Also diese Linearisierung am Punkt x_0 ist

$$f(x_0) + f'(x_0)(x - x_0)$$

Statt nun die Nullstelle von f zu bestimmen, bestimmen wir einfach die Nullstelle x_1 der Linearisierung. Wir lösen also

$$f(x_0) + f'(x_0)(x - x_0) = 0$$

nach x auf und erhalten somit eine (hoffentlich) bessere Näherung

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

11.1.1 Konvergenzgeschwindigkeit

Wir möchten nun den Fehler der aktuellen Iterierten

$$|x_1 - x^*|$$

durch den Fehler der vorangegangenen Iterierten

$$|x_0 - x^*|$$

abschätzen.

Wir sehen in Abbildung 1, dass $|x_1 - x^*|$ durch den Quotient aus Taylorschem Restglied an der Stelle x^* und der Steigung $f'(x_0)$ der Linearisierung an x_0 dargestellt werden kann, also

$$|x^* - x_1| = \frac{R(x^*)}{f'(x_0)} = \frac{|f''(x_0)|}{2|f'(x_0)|}|x^* - x_0|^2$$

Nun fällt folgendes auf:

1. Wir benötigen zweifache Differenzierbarkeit für dieses Argument.

2. Der Quotient
$$\frac{|f''(x_0)|}{2|f'(x_0)|}$$
lässt sich in einer Umgebung von x^* mittels Stetigkeit von f' und f'' gegen die Konstante
$$\frac{\max|f''(x_0)|}{2\min|f'(x_0)|}$$
abschätzen.

3. Wir sagen in diesem Fall, dass die Folge der Iterierten x_i "quadratisch" gegen die Nullstelle x^* konvergiert. Das heißt, dass die Anzahl der richtigen Stellen sich in jedem Schritt verdoppelt! Beachten Sie, dass diese Notation vielleicht zu Verwirrung führt, da "lineare" Konvergenz bedeutet, dass der Fehler exponentiell abfällt.

4. Die Ableitung bei der Iterierten darf nicht verschwinden. Um das zu gewährleisten, müssen wir verlangen, dass die erste Ableitung bei der Nullstelle nicht verschwindet und x_0 in einer genügend kleinen Umgebung der Nullstelle gewählt wird.

5. Falls $f'(x^*) = 0$, dann wird der Quotient
$$\frac{|f''(x_i)|}{2|f'(x_i)|}$$
beliebig groß für $x_i \to x^*$. Das können wir verhindern, indem wir das Newton-Verfahren statt auf f auf
$$k\frac{f}{f'}$$
anwenden. Hierbei soll k die Vielfachheit der Nullstelle sein.

6. Oft ist es schwierig die Ableitung von f zu bestimmen. Der Trick, um das zu umgehen, ist folgender: Starte mit zwei Startwerten x_0, x_1 und bestimme die Nullstelle der Geraden, die durch $f(x_0)$ und $f(x_1)$ geht. Die Verfahrensvorschrift schreibt sich dann als

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$

   Das Verfahren nennt sich "Regula Falsi", ist ein sogenanntes Sekantenverfahren und hat sogar eine noch höhere Konvergenzrate als das Newton-Verfahren.

7. Wenn die Ableitung $f'(x_0)$ für alle nachfolgenden Schritte wiederverwendet wird, statt sie an der aktuellen Iterierten auszuwerten, bekommen wir ein linear konvergentes Verfahren, d.h.

$$|x_{n+1} - x^*| < C|x_n - x^*|$$

   Versuchen Sie, sich davon zu überzeugen. Man spricht dann vom vereinfachten Newton-Verfahren.

11.2 Mehrdimensional

12 Iterative Verfahren für Lineare Gleichungssysteme

Wir wollen wieder das Gleichungssystem

$$Ax = b$$

mit einer invertierbaren Matrix A lösen. Wenn die Matrix A dünnbesetzt ist, heißt das nicht, dass in einer LR-Zerlegung L und R auch dünnbesetzt sein müssen. Betrachte zum Beispiel die Matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Wenn wir den Gaußschen Eliminationsalgorithmus darauf anwenden, benötigen wir genau so viele Schritte, wie bei einer dichtbesetzten Matrix, also $O(n^3)$ und L ist dichtbesetzt.

Die meisten Iterationsverfahren benötigen nur Matrixmultiplikationen mit der ursprünglichen (dünnbesetzten) Matrix. **Eine Matrix-Vektor-Multiplikation mit einer dünnbesetzten Matrix braucht nur $O(n)$ Operationen im Gegensatz zu $O(n^2)$ für die Multiplikation mit einer dichtbesetzten Matrix**.

Für den interessierten Leser folgt ein Beispiel für das Auftreten von dünnbesetzten Matrizen.

Example 9 (Finite Differenzen)**.** Wir wollen das Anfangswertproblem

$$f''(x) = g(x),\ f(a) = f(b) = 0$$

für eine vorgegebene Funktion g lösen. Dazu diskretisieren wie unseren Definitionsbereich in

$$a = x_0 < x_1 < ... < x_n = b$$

wobei der Abstand zwischen den Stützstellen h betragen soll. Wir nähern die zweite Ableitung mit Hilfe des Satzes von Taylor durch

$$f''(x_1) = \frac{2f(x_1) - f(x_0) - f(x_2)}{h^2} + R(h) \text{ mit } R \in O(h^2)$$

und wollen nur die Werte von f an den Stützstellen bestimmen. Diese bezeichnen wir mit dem Vektor

$$\begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}$$

Aus der Forderung $f''(x_i) = g(x_i)$ ergibt sich die lineare Gleichung $2f_i - f_{i-1} - f_{i+1} = h^2 g(x_i)$. In Matrixschreibweise können wir diese Bedingungen für alle Stützstellen zusammenfassen (Erkennen sie, warum jetzt die Randpunkte nicht mehr auftreten?) und erhalten ein lineares Gleichungssystem mit **dünnbesetzter** Koeffizientenmatrix:

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots & 0 \\ \vdots & & \ddots & & & & \vdots \\ \vdots & & & \ddots & & & \vdots \\ \vdots & & & & \ddots & & \vdots \\ 0 & \dots & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} f_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_{n-1} \end{pmatrix} = h^2 \begin{pmatrix} g(x_1) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ g(x_{n-1}) \end{pmatrix}$$

12.1 Jacobi und Gauß-Seidel

Problem: Es soll das Gleichungssystem $Ax = b$ gelöst werden, wobei $A \in \mathbb{R}^{n \times n}$. Für große n sind direkte Verfahren oft zu langsam.

Lösung: Die Matrizen in der Anwendung sind oft dünnbesetzt. Verwende deshalb ein Verfahren, das in jeder Iteration nur mit einer dünnbesetzten Matrix multipliziert.

12.1.1 Jacobi-Verfahren (aka. Gesamtschrittverfahren)

Es wird empfohlen, zunächst das Beispiel am Ende des Kapitels nachzurechnen.

$Ax = b$ ist ein System aus n Gleichungen mit n Variablen:

$$\begin{array}{ccccccc} a_{11}x_1 & + & \dots & + & a_{1n}x_n & = & b_1 \\ \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & \dots & + & a_{nn}x_n & = & b_n \end{array}$$

Wir lösen die i-te Gleichung nach der i-ten Variable auf und erhalten

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j \right)$$

Für das Jacobi-Verfahren setzen wir nun die aktuelle Iterierte

$$x^m = \begin{pmatrix} x_1^m \\ \vdots \\ x_n^m \end{pmatrix}$$

in die rechte Seite all dieser Gleichungen ein, um die nächste Iterierte x^{m+1} zu bekommen.

12.1.2 Konvergenz des Jacobi-Verfahrens

In Matrixschreibweise ergibt sich

$$x^{m+1} = D^{-1}(b - (A - D)x^m)$$

wenn D die Matrix ist, die auf der Diagonalen dieselben Einträge wie A hat und sonst mit Nullen aufgefüllt ist.

Das können wir als Fixpunktiteration $x^{m+1} = \varphi(x^m)$ mit

$$\varphi : x \mapsto D^{-1}b + (I - D^{-1}A)x$$

schreiben, wobei die exakte Lösung x^* zu $Ax = b$ ein Fixpunkt $\varphi(x^*) = x^*$ ist.

Aus

$$||\varphi(x - y)|| = ||(I - D^{-1}A)(x - y)|| \leq ||I - D^{-1}A|| \cdot ||x - y||$$

sieht man, dass $||I - D^{-1}A||$ eine Lipschitzkonstante der Fixpunktiteration ist und falls $||I - D^{-1}A|| < 1$ ist, haben wir es mit einer Kontraktion zu tun und das Verfahren konvergiert.

Lemma 10. *Bezüglich der Zeilensummennorm ist genau dann $||I - D^{-1}A||_\infty < 1$ also die Iterationsabbildung des Jacobi-Verfahrens eine Kontraktion, wenn A strikt diagonaldominant ist.*

Proof.

$$||I - D^{-1}A||_\infty = \max_i \left(\left(1 - \frac{|a_{ii}|}{|a_{ii}|} \right) + \sum_{k \neq i} \frac{|a_{ik}|}{|a_{ii}|} \right) = \max_i \sum_{k \neq i} \frac{|a_{ik}|}{|a_{ii}|}$$

Also ist $||I - D^{-1}A|| < 1$ genau dann, wenn $max_i \sum_{k \neq i} \frac{|a_{ik}|}{|a_{ii}|} < 1$, also wenn für alle i gilt, dass $\sum_{k \neq i} \frac{|a_{ik}|}{|a_{ii}|} < 1$. Multiplikation mit $|a_{ii}|$ ergibt die strikte Diagonaldominanz. $\square$

Example 11 (Jacobi-Verfahren)**.** Wir wollen das lineare Gleichungssystem

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

lösen. Dies ist äquivalent zu den drei Gleichungen

$$\begin{array}{ccccccc} 2x & + & y & & & = & 1 \\ x & + & 2y & + & z & = & 1 \\ & & y & + & 2z & = & 1 \end{array}$$

Für das Jacobiverfahren brauchen wir, wie für jedes iterative Verfahren einen Startvektor, zum Beispiel den Vektor

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

den wir dann in jeder Iteration verbessern bis schließlich eine zufriedenstellende Näherung der Lösung des Gleichungssystems erreicht wird.

Die Idee des Jacobiverfahrens ist es nun, die erste Zeile des Gleichungssystems nach der ersten Variable und die zweite nach der zweiten Variable usw. umzuformen und dann in die rechte Seite die aktuelle Näherung einzusetzen um auf der linken Seite die neue Näherung zu erhalten. Die Umformung ergibt

$$\begin{array}{ccccccc} x & = & \frac{1}{2} & - & \frac{1}{2}y & & \\ y & = & \frac{1}{2} & - & \frac{1}{2}x & - & \frac{1}{2}z \\ z & = & \frac{1}{2} & - & \frac{1}{2}y & & \end{array}$$

und einsetzen ergibt dann die Iterationsvorschrift

$$\begin{array}{ccccccc} x^1 & = & \frac{1}{2} & - & \frac{1}{2}y^0 & & \\ y^1 & = & \frac{1}{2} & - & \frac{1}{2}x^0 & - & \frac{1}{2}z^0 \\ z^1 & = & \frac{1}{2} & - & \frac{1}{2}y^0 & & \end{array}$$

Probieren Sie selbst ein paar Schritte des Jacobi-Verfahrens auszuführen. Es sollten sich die Iterierten

$$\begin{pmatrix} 0 \\ -\frac{1}{2} \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{3}{4} \\ \frac{1}{2} \\ \frac{3}{4} \end{pmatrix}$$

ergeben.

12.1.3 Gauß-Seidel-Verfahren (aka. Einzelschrittverfahren)

Beim Gauß-Seidel-Verfahren formen wir das Gleichungssystem auf die gleiche Weise um, wie beim Jacobi-Verfahren, so dass wir wieder

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j \right)$$

erhalten. Nur diesmal **berechnen wir die Komponenten der neuen Iterierten eine nach der anderen**. Also beginnend mit der aktuellen Iterierten

$$x^m = \begin{pmatrix} x_1^m \\ \vdots \\ x_n^m \end{pmatrix}$$

berechnen wir zunächst

$$x_1^{m+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=2}^{n} a_{ij} x_j^m \right)$$

wie gewohnt. Um die zweite Komponente zu berechnen setzen wir aber schon die erste Komponente der neuen Iterierten in die rechte Seite ein:

$$x_2^{m+1} = \frac{1}{a_{ii}} \left(b_i - a_{11} x_1^{m+1} \sum_{j=3}^{n} a_{ij} x_j^m \right)$$

Die Iterierten verändern sich mit diesem Verfahren folgendermaßen:

$$\begin{pmatrix} x_1^0 \\ \vdots \\ x_n^0 \end{pmatrix} \to \begin{pmatrix} x_1^1 \\ x_2^0 \\ \vdots \\ x_n^0 \end{pmatrix} \to \begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^0 \\ \vdots \\ x_n^0 \end{pmatrix} \cdots \to \begin{pmatrix} x_1^1 \\ \vdots \\ x_n^1 \end{pmatrix}$$

Dies lässt sich auch wieder gut in Matrixschreibweise darstellen als

$$x^{m+1} = (D+L)^{-1}(b - Ux^m) = (D+L)^{-1}b + (I - (D+L)^{-1}A)x^m$$

Für die Konvergenz benötigen wir wieder, dass

$$\varphi : x \mapsto (D+L)^{-1}b + (I - (D+L)^{-1}A)x$$

eine Kontraktion bezüglich irgendeiner Norm ist. Analog zum Jacobi-Verfahren benötigen wir, dass

$$||I - (D+L)^{-1}A|| < 1$$

gilt. Dies ist zum Beispiel dann der Fall, wenn A strikt diagonaldominant ist oder symmetrisch positiv definit.

Proof. ohne Beweis. □

12.2 Gradientenverfahren

12.2.1 Der Gradient

Für eine differenzierbare Funktion $f : \mathbb{R}^n \to \mathbb{R}$ ist der Gradient definiert als

$$\nabla f := \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} = (f')^T$$

Wir wollen zunächst ein Beispiel betrachten.

Example 12. Ein Gebirge, das genügend rundliche Kuppen hat und keine Überhänge kann man durch eine Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ beschreiben, die zu jeder Längengrad(x)/Breitengrad(y)-Koordinate die Höhe des Gebirges angibt.

Dass die Funktion f differenzierbar ist, heißt hierbei folgendes: Wenn man sich ein ganz kleines Stück des Gebirges herausschneidet, sieht es aus wie eine schiefe Ebene. Die Fortsetzung dieser Ebene in alle Richtungen ist die Linearisierung, also das erste Taylorpolynom der Funktion an der Stelle, die wir herausgeschnitten haben.

Der Gradient an einer Längen/Breitengrad-Koordinate ist die Richtung, in die es am steilsten bergauf geht. Der Gradient zeigt dabei nicht schräg nach oben, sondern liegt gänzlich in der (x, y)-Ebene. Seine Länge gibt an, wie steil es bergauf geht.

Question 13. *Überzeugen Sie sich davon, dass der Gradient immer senkrecht auf den Niveaulinien steht.*

Question 14. *Wo im Gebirge hat der Gradient Länge* 0*?*

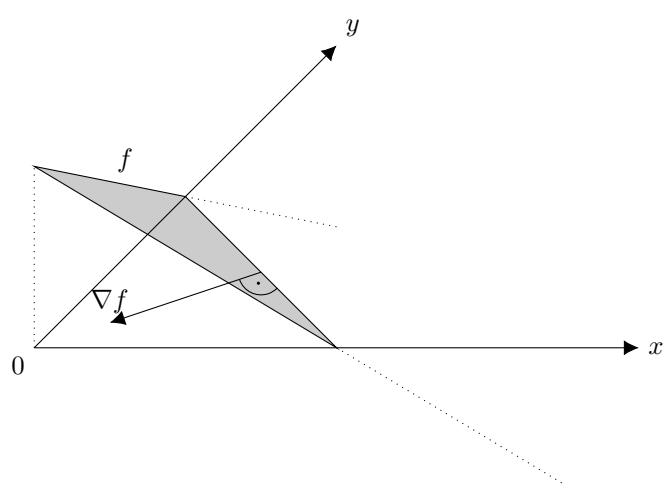


Figure 2: Gradient

Example 15. Wenn $f : x \mapsto a^T x + c$ mit $a \in \mathbb{R}^n,\ c \in \mathbb{R}$ tatsächlich affin ist und eine Ebene definiert, ist der Gradient $\nabla f = a$ leicht anzugeben.

Lemma 16. *Wir fassen noch einmal zusamen:*

1. *∇f zeigt in die Richtung des steilsten Anstiegs, aber nicht schräg nach oben. Für die Richtungsableitung von f in Richtung v gilt*

$$\frac{\partial f}{\partial v} = (f') \cdot v = (\nabla f)^T \cdot v = \langle \nabla f, v \rangle$$

   Diese Richtungsableitung ist offensichtlich maximal, wenn das Skalarprodukt am Ende maximal wird, also wenn $v = \frac{\nabla f}{||\nabla f||}$ die Richtung des Gradienten ist.

2. *Die Länge $||\nabla f||_2$ des Gradienten ist die Größe des Anstiegs.*

3. *$-\nabla f$, häufig auch Antigradient genannt, zeigt in Richtung des steilsten Abfalls.*

12.2.2 Das Gradientenverfahren

Wir möchten ein lokales Minimum einer differenzierbaren Funktion $f : \mathbb{R}^n \to \mathbb{R}$ finden.

Idee: Starte mit beliebigem Vektor $x_0 \in \mathbb{R}^n$. Berechne den Gradienten bei x_0, also $(\nabla f)(x_0)$. Gehe entgegen der Gradientenrichtung, um ein besseres x_1 zu finden. Besser heißt, dass f an bei x_1 kleiner sein soll, als bei x_0.

Das Verfahren lässt sich also folgendermaßen schreiben:

$$x_{k+1} = x_k - \lambda_k (\nabla f)(x_k)$$

λ_k ist hierbei die Schrittweite. Wir gehen also λ_k weit in Richtung des steilsten Abfalls.

Es stellen sich nun zwei Fragen:

1. Wie weit sollte ich gehen, damit x_{k+1} möglichst klein wird?

2. Ist es auch sinnvoll in andere Richtungen zu gehen?

Die erste Frage werden wir in folgenden Kapitel für eine spezielle Klasse von f beantworten, nämlich solche, mit denen man das Gleichungssystem $Ax = b$ lösen kann.

Die Suchrichtung sollte zumindest immer positives Skalarprodukt mit $-\nabla f$ haben, damit die Richtungsableitung von f negativ ist. Das CG-Verfahren benutzt Richtungen, mit denen man lineare Gleichungssysteme schnell lösen kann.

12.2.3 Liniensuchverfahren zur Lösung von Ax=b

Liniensuchverfahren sind eine Verallgemeinerung des Gradientenverfahren, bei denen die Suchrichtung nicht unbedingt $-\nabla f$ sein muss.

Wir behandeln hier den speziellen Fall, dass das Gleichungssystem $Ax = b$ gelöst werden soll, indem es in ein Minimierungsproblem umgeschrieben wird.

Dafür müssen wir zunächst annehmen, dass A symmetrisch positiv definit ist. Diese Annahme beschränkt die Allgemeinheit nicht, da bei invertierbarer Matrix A die Lösung des Gleichunssystems $Ax = b$ identisch ist mit der Lösung von $(A^T A)x = (A^T b)$. (Die Begründung ist einfacher als die der Gaußschen Normalengleichungen, obwohl sie gleich aussehen.)

Question 17. *Finden Sie den Unterschied, zwischen dieser Gleichung und den Gaußschen Normalengleichungen. Welche sind schwerer herzuleiten?*

Theorem 18. *Sei A symmetrisch positiv definit. Dann ist x genau dann die Läsung von $Ax = b$, wenn x eines der beiden "Energiefunktionale"*

$$\begin{aligned} E : x \mapsto & \quad \tfrac{1}{2}x^T Ax - x^T b \\ F : x \mapsto & \quad E(x) + \tfrac{1}{2}x^{*T} Ax^* \quad = \quad \tfrac{1}{2}(x - x^*)^T A(x - x^*) \end{aligned}$$

minimiert, wobei $x^* = A^{-1}b$ die exakte Lösung des Gleichungssystems sein soll.

Proof. E und F haben das Minimum an derselben Stelle, weil $F = E$+Konstante. Die Funktion $f(x) = \frac{1}{2}x^T Ax$ definiert ein Skalarprodukt (weil A symmetrisch positiv definit ist) und hat somit ihr einziges Minimum bei $x = 0$. Die ist auch das einzige lokale Minimum, weil f konvex ist. (Erinnerung: f ist das Quadrat der zum Skalarprodukt gehörenden Norm und ist somit ein elliptischer Paraboloid.) $F = f(x - x^*)$ hat folglich sein eindeutiges Minimum bei $x = x^* = A^{-1}b$. □

Wir werden nun also, statt $Ax = b$ direkt zu lösen, das Funktional

$$E : x \mapsto \frac{1}{2}x^T Ax - x^T b$$

minimieren.

Liniensuchverfahren für das Funktional E: Starte mit beliebigem $x_0 \in \mathbb{R}^n$. Iterierte mit

$$x_{n+1} = x_n + \alpha d_n$$

wobei d_n die Suchrichtung ist und α die Schrittweite.

Das heißt, von x_n ausgehend, optimieren wir nur auf dem 1-dimensionalen affinen Unterraum $x_n + \mathbb{R}d_n$, um x_{n+1} zu finden. Im Falle von $E : x \mapsto \frac{1}{2}x^T Ax - x^T b$ können wir dieses 1-dimensionale Problem folgendermaßen sogar exakt lösen.

Theorem 19. *Bezeichne den Gradienten bei x_n mit $r_n := (\nabla E)(x_n) = Ax_n - b$. r_n steht für Residuum, weil $r_n = A(x_n - x^*)$. Für A symmetrisch positiv definit und $E : x \mapsto \frac{1}{2}x^T Ax - x^T b$ ist die optimale Schrittweite*

$$\alpha = -\frac{d_n^T r_n}{d_n^T d_n}$$

***Optimal deshalb, weil sie die Funktion** E **zumindest auf der Geraden** $x_n + \alpha d_n$ **exakt minimiert wird**, das heißt*

$$min_{t \in \mathbb{R}} E(x_n + td_n) = E(x_n + \alpha d_n)$$

Proof. Der Beweis benutzt nur eindimensionale Kurvendiskussion, die sie bereits aus dem Abitur kennen:

Die Funktion $t :\mapsto E(x_n + td_n)$ bildet von $\mathbb{R}$ nach $\mathbb{R}$ ab und ist sogar ein quadratisches Polynom. Das Minimum finden wir, indem wir diese Funktion nach t ableiten und die Ableitung gleich null setzen. Versuchen Sie es selbst! Tipp: Die Funktion lässt sich einfacher nach t ableiten, wenn zunächst alle Potenzen von t ausgeklammert werden, also $E(x_n + td_n) = t^2(...) + t(...) + ...$ □

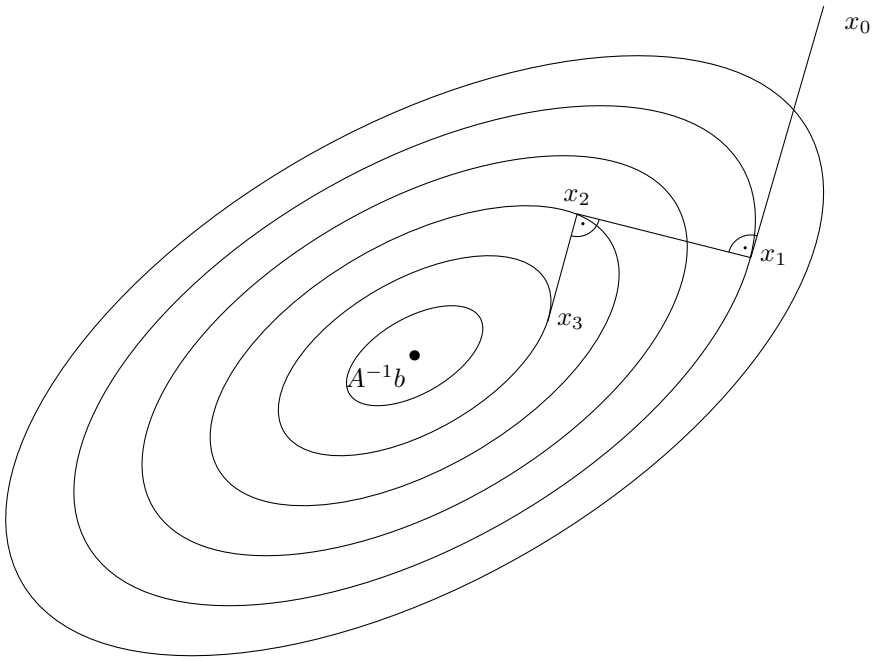


Figure 3: Gradientenverfahren

Example 20. In Abbildung 3 sind die Niveaulinien eines Funktionals $E : \mathbb{R}^2 \to \mathbb{R}$ eingezeichnet. Das sind die Kurven entlang derer die Funktion $E : x \mapsto$

$\frac{1}{2}x^T Ax - x^T b$ konstant ist, wie Höhenlinien in einer Wanderkarte. Wir hatten ja schon gesehen, dass der Gradient immer senkrecht auf den Niveaulinien stehen muss. Im Bild kann man außerdem erkennen: Wenn die Niveaulinien kreisrund wären, würde das Gradientenverfahren nur einen einzigen Schritt benötigen. Die Streckung der Ellipse entspricht der Kondition der Matrix A. Wenn A also schlecht konditioniert ist, können wir wie im Beispiel Startwerte finden, für die das Gradientenverfahren recht langsam konvergiert.

Theorem 21. *Die Konvergenz des Gradientenverfahrens zur Lösung von $Ax = b$ lässt sich durch*

$$||x_n - x^*||_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^n ||x_0 - x^*||_A$$

abschätzen.

12.2.4 Suchrichtungswahl, CG-Verfahren (3)

13 Eigenwertprobleme (R)

13.1 Гершгорин(Gerschgorin)-Kreise

13.2 Rayleigh-Quotient (1)

13.3 Vektoriteration (aka. Potenzmethode)

13.4 QR-Verfahren von Francis

14 Spline-Interpolation (R)