3 Lineare Gleichungssysteme, direkte Verfahren

Wir verwenden die Standard-Notation für ein System linearer Gleichungen:

$$A\mathbf{x} = \mathbf{b} \,,$$

worin A die Koeffizientenmatrix, $\mathbf{b}$ die rechte Seite und $\mathbf{x}$ den Lösungsvektor bezeichnet. Wir nehmen hier an, dass n die Anzahl der Gleichungen und Unbekannten bezeichnet und A eine reelle $n \times n$-Matrix ist.

Während für große, schwach besetzte Matrizen generell iterative Verfahren verwendet werden, löst man Systeme mit voll besetzten Matrizen üblicherweise durch *direkte* Verfahren; das sind solche, die nach einer fixen Zahl von Rechenschritten eine exakte Lösung (rundungsfehlerfreie Rechnung vorausgesetzt) liefern.

Referenz für Rechenprogramme in anerkannt hoher Qualität ist LAPACK (früher: LINPACK) `http://www.netlib.org/lapack/`. Diese Programme sind frei verfügbar. Sie werden auch in kommerziell angebotenen Paketen kaum bessere finden. Auch MATLAB enthält die LAPACK-Algorithmen. (Übrigens ist MATLAB ursprünglich als einfache Benutzerschnittstelle zu den LINPACK-Programmen entstanden).

3.1 Dreiecksmatrizen

Wenn A eine untere oder obere *Dreiecksmatrix* ist, findet man die Lösung direkt durch schrittweises Einsetzen (Vorwärts- bzw. Rückwärtssubstitution). Andernfalls *faktorisiert* man A zuerst in ein Produkt von Dreiecksmatrizen (und möglicherweise eine Diagonal- oder Permutationsmatrix).

Wir bezeichnen hier Dreiecksmatrizen mit L und R. In L sind nur Einträge im linken unteren Dreieck von Null verschieden und alle Einträge der Hauptdiagonale gleich eins. In R sind nur Einträge im rechten oberen Dreieck einschließlich der Hauptdiagonale ungleich Null. Beispiel für $n = 4$:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{21} & 1 & 0 & 0 \\ \ell_{31} & \ell_{32} & 1 & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{bmatrix}, \qquad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ 0 & r_{22} & r_{23} & r_{24} \\ 0 & 0 & r_{33} & r_{34} \\ 0 & 0 & 0 & r_{44} \end{bmatrix}$$

Wenn die Elemente einer Dreiecksmatrix L auf einem Feld `a[i][j]` und die rechte Seite b auf einem Vektor `b[i]` gespeichert sind, löst das folgende Java-Programmsegment das Gleichungssystem $L\mathbf{x} = \mathbf{b}$ durch schrittweises Einsetzen („Vorwärts-Substitution“. Achtung, Java zählt Feldindizes von 0 bis $n-1$; konventionelle Mathematik-Schreibweise zählt Zeilen und Spalten von 1 bis n.)

```
for (int i=0; i<n; i++) {
   x[i] = b[i];
   for (int j=0; j<i; j++) {
      x[i] -= a[i][j] * x[j];
   }
}
```

Ähnlich kompakt lässt sich die Lösung eines Gleichungssystems mit rechter oberer Dreiecksmatrix R formulieren. Unterschiede zu vorhin: Das Einsetzen beginnt in der letzten Zeile und

schreitet von unten nach oben fort („Rückwärts-Substitution“); durch das Hauptdiagonalelement wird dividiert; das Programmsegment nimmt auch an, dass anfangs die rechte Seite bereits auf `x[ ]` gespeichert ist.

```
        for (int i=n-1; i>-1; i--) {
           for (int j=i+1; j<n; j++) {
              x[i] -= a[i][j] * x[j];
           }
           x[i] /= a[i][i];
        }
    }
```

Der Rechenaufwand, gemessen an der Zahl der Punktoperationen (Multiplikationen und Divisionen) beträgt für die Vorwärts-Substitution $n^2/2-n/2$. Die Rückwärts-Substitution braucht $n^2/2+n/2$ Punktoperationen. Für große n ist praktisch nur der quadratische Term ausschlaggebend. Wir sagen daher

Rechenaufwand bei der Vorwärts- oder Rückwärts-Substitution
Der Rechenaufwand bei einem $n \times n$-System beträgt $n^2/2 + O(n)$
Oft reicht auch schon die Aussage: Der Aufwand wächst quadratisch mit der Anzahl der Gleichungen, ist also $O(n^2)$

Frage: Was tut man, wenn ein Gleichungssystem nicht in Dreiecksform vorliegt? Antwort: man formt es in ein solches um (natürlich so, dass Originalsystem und umgeformtes System äquivalent sind, also genau die gleichen Lösungen haben).

3.2 Gauß-Elimination

Die einfache Gauß-Elimination läßt sich so formulieren

Einfache Gauß-Elimination
Gegeben eine $n \times n$-Matrix A und rechte Seite $\mathbf{b}$. Sofern keines der a_{kk} zu einer Division durch Null führt, transformiert dieses Verfahren das System $A\mathbf{x} = \mathbf{b}$ auf ein äquivalentes System in oberer Dreiecksform $R\mathbf{x} = \mathbf{c}$.

Für alle Spalten $k = 1, \ldots n-1$
in Spalte k eliminiere alle Einträge unterhalb des Diagonalelements

Das Eliminieren in Spalte k läuft dabei folgendermaßen ab:

Für alle Zeilen $i = k+1, \ldots, n$ unterhalb der Diagonale
setze $p = a_{ik}/a_{kk}$
subtrahiere das p-fache der k-ten Zeile von Zeile i

Die Subtraktion wird durch folgende Schleife bewerkstelligt:

Für alle Spalten $j = k, \ldots n$
$a_{ij} = a_{ij} - pa_{kj}$
Für rechte Seite: $b_i = b_i - pb_k$

Diese Rechenvorschrift *überschreibt* Einträge in A und $\mathbf{b}$ mit den jeweiligen Zwischenresultaten und letztlich mit den Einträgen von R und $\mathbf{c}$. Sie verzichtet darauf, Einträge unterhalb der Diagonale (die eigentlich gleich Null sein sollen) zu löschen.

Der Rechenaufwand beträgt $n^3/3 - n/3$ Punktoperationen für die Transformation der Matrix und $n^2/2 - n/2$ Punktoperationen für die Transformation der rechten Seite.

Ein Computerprogramm implementiert dieses Verfahren verblüffend kurz als dreifache Schleife. Es nimmt an, dass anfangs die rechte Seite als Vektor `x[ ]` gespeichert ist.

```
for (int k=0; k<n; k++) {
   for (int i=k+1; i<n; i++) {
      double p = a[i][k] / a[k][k];
      for (int j=k+1; j<n; j++) {
         a[i][j] -= p * a[k][j];
      }
      x[i] -= p * x[k];
   }
}
```

Das Programm spart es sich, im k-ten Schritt die Elemente unterhalb der Hauptdiagonale in der k-ten Spalte zu berechnen, weil ohnehin 0 herauskommen muss. Es verzichtet auch darauf, diese Nullen explizit in die Matrix hineinzuschreiben, sondern lässt dort die Zwischenresultate einfach stehen.

Das schrittweise Rückwärts-Einsetzen läßt sich mit $n^2/2 + O(n)$ Punktoperationen genäß dem Programmsegment aus dem vorigen Abschnitt erledigen. (Diese Doppelschleife verwendet nur das obere Dreieck von A; die Zwischenresultate $\neq 0$ unterhalb der Diagonale von vorhin stören daher nicht.)

Kombiniert liefern diese beiden Codesegmente einen einfachen Gleichungslöser.

Rechenaufwand bei einfacher Gauss-Elimination
Gauß-Elimination errechnet die Lösung eines $n \times n$-Systems $A\mathbf{x} = \mathbf{b}$ mit $n^3/3 + O(n^2)$ Punktoperationen (exakt: $n^3/3 + n^2 - n/3$).
Oft reicht auch schon die Aussage: Der Aufwand wächst kubisch mit der Anzahl der Gleichungen, ist also $O(n^3)$

3.3 Pivotsuche

Die einfache Gauß-Elimination hat einen Haken: Der Rechenschritt $p = a_{ik}/a_{kk}$ kann zu einer Division durch Null führen. Für eine Matrix zufällig gewählter reeller Zahlen ist das extrem unwahrscheinlich, aber Murphy's Gesetz besagt: *If anything can go wrong, it will.* Und tatsächlich versagt das Verfahren bei so simplen Systemen wie

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

weil gleich als erste Rechenoperation durch Null dividiert wird. Das ist definitiv nicht Schuld des Gleichungssystems, es hat nämlich die eindeutige Lösung $x_1 = 1, x_2 = 1$. Vertauscht man andererseits erste und zweite Gleichung, dann läuft das Verfahren problemlos.

Auch aus Gründen der Rechengenauigkeit kann es günstig sein, Gleichungen oder Unbekannte systematisch zu vertauschen. Man nennt diese Vorgehensweise *Pivotierung*.

Gauß-Elimination mit vollständiger Pivotsuche

Gegeben eine $n \times n$-Matrix A und rechte Seite $\mathbf{b}$. Dieses Verfahren transformiert das System $A\mathbf{x} = \mathbf{b}$ auf obere Dreiecksform $R\mathbf{x} = \mathbf{c}$. Die Matrix A wird dabei durch R und der Vektor $\mathbf{b}$ durch $\mathbf{c}$ überschrieben.

Für alle Spalten $k = 1, \ldots, n-1$
    suche das betragsgrößte Element in der
    quadratischen Untermatrix aus den Zeilen
    und Spalten k bis n.

    Bringe dieses Element durch geeignetes Vertauschen
    von Gleichungen und Unbekannten an die Stelle a_{kk}.

    Wenn $a_{kk} = 0$
        Stopp.
    sonst
        in Spalte k eliminiere alle
        Einträge unterhalb des Diagonalelementes
        wie in der simplen Gauß-Elimination.

Pivot bedeutet „Drehachse, Angelpunkt“. Bei der Elimination dreht sich alles darum, welche Gleichung jeweils benützt werden soll, um die entsprechende Unbekannte in den verbleibenden Gleichungen zu eliminieren. Angelpunkt ist das Element a_{kk}, mit dessen Hilfe der Faktor $p = a_{ik}/a_{kk}$ berechnet wird. Es heißt deswegen das *Pivotelement*. Ein günstiges Pivotelement durch geeignete Vertauschungen zu finden heißt *Pivotsuche* oder *-wahl*.

Die notwendigen Zeilen- und Spaltenvertauschungen komplizieren ein Rechenprogramm in Vergleich zur Dreifachschleife der einfachen Gauß-Elimination erheblich. Der Gewinn an zusätzlicher Einsicht in das Verfahren steht dazu in keinem annehmbaren Verhältnis. Deswegen ist hier kein Programm zur vollständigen Pivotsuche abgedruckt.

Üblicherweise führen Gleichungslöser keine vollständige, sondern nur *Zeilen-Pivotsuche* durch. Das heißt, sie beschränken sich der Einfachheit halber auf Vertauschen von Zeilen (=Gleichungen). Die Empfindlichkeit gegenüber Rundungsfehlern ist bei Zeilen-Pivotsuche etwas höher als bei vollständiger Pivot-Suche. Ansonsten gelten die gleichen Aussagen.

3.4 Lösbarkeit linearer Gleichungssysteme

Die Faustregel „Bei genausoviel Gleichungen wie Unbekannten gibt es immer eine Lösung“ *ist falsch*. Ich wiederhole (weil ich es bei Prüfungen immer wieder so höre): *ist falsch*.

Bei den drei Gleichungssystemen

$$\begin{aligned} x + y &= 2 \\ 2x + 2y &= 4 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ 2x + 2y &= 3 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ x + 2y &= 3 \end{aligned}$$

sehen Sie hoffentlich mit freiem Auge: Beim ersten und beim dritten sind $x = 1, y = 1$ Lösungen. Das mittlere System ist unlösbar. Beim ersten System gibt es allerdings noch unendlich viele weitere Lösungen. Allgemein gilt: Ein System n linearer Gleichungen in n Unbekannten ist entweder eindeutig, nicht eindeutig oder überhaupt nicht lösbar. Sie haben das in der Mathematik-Grundvorlesung gelernt.

In diesem Abschnitt geht es darum, wie man *rechnerisch* bestimmen kann, welcher Fall vorliegt. Eine wichtige Eigenschaft des Gaußschen Eliminationsverfahrens ist, dass es diese Entscheidung treffen kann.

3.4.1 Eliminationsverfahren

Gauß-Elimination mit vollständiger oder Zeilen-Pivotsuche transformiert die Originalmatrix A und rechte Seite $\mathbf{b}$ auf ein System in *Stufenform*: In jeder Zeile verringert sich die Zahl der Unbekannten um mindestens eine, die dann auch in den darauffolgenden Zeilen nicht mehr vorkommt.

Mögliche Fälle nach Abschluss des Eliminationsverfahrens
(das System ist auf Stufenform transformiert)

- Es treten Nullzeilen in A auf und alle entsprechenden Einträge in b sind ebenfals Null: unendlich viele Lösungen
- Es treten Nullzeilen in A auf, aber zumindest ein entsprechender Eintrag in b ist nicht Null: keine Lösung
- Es treten keine Nullzeilen in A auf: eindeutige Lösung

Der MATLAB-Befehl `rref([A,b])` (rref steht für *reduced row echelon form*, reduzierte Stufenform) führt eine Variante des Eliminationsverfahrens durch (Gauß-Jordan Verfahren), allerdings nur mit Spalten-Pivotsuche. Für die Ergebnismatrix in der rref-Form gelten die gleichen Aussagen wie oben.

In der Praxis, bei Rechnungen am Computer, lässt sich nicht so leicht überprüfen, ob Einträge exakt gleich Null sind. Durch Rundungsfehler in den Eingabedaten und während der Rechnung werden Matrixelemente oft nicht exakt Null, sondern nur extrem klein. Man muss eine Schranke angeben, ab der Einträge als Null und die Matrix damit als numerisch singulär anzusehen sind.

3.4.2 Rang der Matrix und der erweiterten Matrix

Der MATLAB-Befehl `rank(A)` bestimmt den Rang (die Anzahl linear unabhängiger Zeilen oder Spalten) der $n \times n$-Matrix A, und `rank([A,b])` den Rang der *erweiterten Koeffizientenmatrix* (der Matrix des Gleichungssystems wird die rechte Seite als letzte Spalte angefügt).

Mögliche Fälle bei der Rang-Bestimmung

- `rank(A)` = `rank([A,b])` $< n$: unendlich viele Lösungen
- `rank(A)` $< n$ und `rank(A)` $\neq$ `rank([A,b])`: keine Lösung
- `rank(A)` $= n$: eindeutige Lösung

Es gibt verschiedene Methoden, den Rang einer Matrix zu berechnen, zum Beispiel Transformation auf Stufenform durch Gauß-Elimination: Der Rang ist die Anzahl der von Null verschiedenen Zeilen. Es gibt dabei aber kein einfaches Entscheidungskriterium, ob ein Wert

bloß infolge Rundungsfehlern oder „echt“ ungleich Null ist. MATLAB verwendet ein sehr rechenaufwendiges Verfahren (Singulärwertzerlegung), das aber die verläßlichsten Aussagen liefert.

Falls ein Gleichungssystem unendlich viele Lösungen hat, lässt sich die allgemeine Lösung entweder aus dem `rref([A,b])`-Ergebnis ablesen oder durch folgende Befehle finden:

`pinv(A)*b` liefert eine spezielle Lösung
`null(A)` liefert den *Nullraum* von A: eine Liste von linear unabhängigen Lösungen des *homogenen Systems* $A\mathbf{x} = 0$.

Die allgemeine Lösung ist die Summe aus der speziellen Lösung und einer beliebigen Linearkombination aus dem Nullraum.

`null(A,'r')` liefert ebenfalls eine Liste von linear unabhängigen Lösungen des homogenen Systems, aber bei einfachen Beispielmatrizen mit „schöneren“ Zahlen. Dafür sind bei „realistischen“ Matrizen die Ergebnisse eher durch Rundungsfehler verfälscht. (Lassen Sie sich nie von äußerer Schönheit blenden, wenn dahinter Falschheit lauert!)
Am Beispiel des Systems $A\mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 6 \\ -1 & -2 & -2 & -2 \\ 3 & 6 & 8 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 2 \end{bmatrix}, \quad \texttt{rref([A,b])} = \begin{bmatrix} 1 & 2 & 0 & -2 & -2 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Aus dem `rref([A,b])`-Ergebnis lässt sich ablesen: $x_2 = \lambda$ und $x_4 = \mu$ sind frei wählbare Parameter, $x_3 = 1 - 2\mu, x_1 = -2 - 2\lambda + \mu$. In Vektorform:

$$\mathbf{x} = \begin{bmatrix} -2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \lambda \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \mu \begin{bmatrix} 2 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

`pinv(A)*b` und `null(A)`liefern eine andere Darstellung,

$$\mathbf{x} = \begin{bmatrix} -0.2069 \\ -0.41379 \\ 0.034483 \\ 0.48276 \end{bmatrix} + \lambda \begin{bmatrix} -0.77069 \\ 0.10421 \\ 0.56227 \\ -0.28113 \end{bmatrix} + \mu \begin{bmatrix} -0.48335 \\ 0.54725 \\ -0.61115 \\ 0.30558 \end{bmatrix}$$

Obwohl MATLAB dieses Resultat mit mit aufwändigen und verlässlichen numerischen Verfahren berechnet, ist es gerade bei diesem einfachen Beispiel durch Rundungsfehler ungenauer als die erste Darstellung mit den schöneren Zahlen. (Schönheit hängt oft doch auch mit Wahrheit zusammen).

3.4.3 Determinante

Die Determinante determiniert, ob ein Gleichungssystem eindeutig lösbar ist.

Gleichungssysteme $A\mathbf{x} = \mathbf{b}$ mit $\det A \neq 0$ sind eindeutig lösbar.
Allerdings ist diese Regel für das numerische Rechnen unbrauchbar.

Die folgende symmetrische 8×8-Matrix lässt sich in MATLAB durch `A=rosser` erzeugen.

$$A = \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

Für sie berechnet MATLAB derzeit[4] $\det A = -9478.9$, also deutlich $\det A \neq 0$. Damit wären Gleichungssysteme mit A eindeutig lösbar. Als Rang berechnet MATLAB aber (korrekt) `rank(A)`=7, und weil $7 < 8$ ist, kann es keine eindeutigen Lösungen geben.

Für die 6×6-Matrix H, eine sogenannte Hilbert-Matrix,

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{bmatrix}$$

berechnet MATLAB $\det A = 5.367299886816843e-018$, und das könnte man durchaus schon als $\det A = 0$ durchgehen lassen. Als Rang berechnet MATLAB aber (korrekt) `rank(H)`=6. Gleichungssysteme mit H sind also eindeutig lösbar (allerdings extrem anfällig gegenüber Rundungsfehlern).

Diese Beispiele illustrieren:

> Der numerisch berechnete Wert einer Determinante sagt nichts über die Lösbarkeit eines Gleichungssystems aus.

3.5 LR-Zerlegung

Die einfache Gauß-Elimination liefert (wenn sie nicht abbricht) mehr als die Transformation auf Dreiecksgestalt. Sie kann gleichzeitig auch eine Zerlegung

$$A = LR$$

errechnen, wobei L eine untere Dreiecksmatrix mit Einsen in der Hauptdiagonale und R eine obere Dreiecksmatrix ist.

> **LR-Zerlegung**
> Das Gaußsche Eliminationsverfahren ohne Pivotwahl faktorisiert (wenn es nicht abbricht) eine Matrix A in ein Produkt $A = LR$ aus einer linken unteren Dreiecksmatrix L und einer rechten oberen Dreiecksmatrix R.

Nach Durchführen einer Gauß-Elimination mit Pivot-Suche enthalten oberes und unteres Dreieck der Ergebnismatrix die LR-Zerlegung einer Matrix mit - im Vergleich zur Ausgangsmatrix - entsprechend vertauschten Zeilen und Spalten.

Die Elemente von L sind 1 entlang der Hauptdiagonale, und darunter gleich den Multiplikatoren $p = a_{ik}/a_{kk}$ an den entsprechenden Stellen (i,k). Die Elemente von R sind genau jene, die das Eliminationsverfahren in das obere rechte Dreieck schreibt. Die einzige Änderung im obigen Programm zum Eliminationsverfahren ist, sich die Zwischenresultate `p` zu merken.

[4] Ältere Versionen vor 2013 liefern $\det A = -13017$, die 2015-Version bringt -9448.8 . Es sollte Sie beunruhigen, dass ein Rechenergebnis je nach Programmversion so unterschiedlich ausfällt!

Praktischerweise lässt sich jedes `p` auf dem entsprechenden Feldelement `a[i][k]` speichern; das Verfahren eliminiert nämlich gerade diesen Eintrag, erzeugt also an dieser Stelle eine Null. Die Null muss man sich nicht merken, aber dafür kann man an der dadurch freigewordenen Stelle das Zwischenresultat `p` speichern.

> Computerprogramme formulieren das Verfahren in aller Regel so, dass die Originalmatrix A durch R und L überschrieben wird. Das obere Dreieck von A enthält nach erfolgreichem Ablauf die Nichtnull-Einträge von R; die Einsen in der Hauptdiagonale von L verstehen sich von selbst, man braucht sie nicht zu speichern; die restlichen Nichtnull-Elemente von L finden unterhalb der Hauptdiagonale von A Platz. Das Elegante an dieser Speicherung ist, dass sie sich im Verlauf des Verfahrens quasi von selbst ergibt.

Für die LR-Zerlegung braucht man keine rechte Seite. Die kommt erst später ins Spiel. Zur Lösung des Systems $A\mathbf{x} = \mathbf{b}$ formt man nämlich um:

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} && \\ (LR)\mathbf{x} &= \mathbf{b} && \\ L(R\mathbf{x}) &= \mathbf{b} && \text{setze } \mathbf{y} = R\mathbf{x} \\ L\mathbf{y} &= \mathbf{b} && \text{löse durch Vorwärts-Substitution nach } \mathbf{y} \\ R\mathbf{x} &= \mathbf{y} && \text{löse durch Rückwärts-Substitution nach } \mathbf{x} \end{aligned}$$

Der Rechengang und der Rechenaufwand sind der einfachen Gauß-Elimination völlig äquivalent. Der Vorteil der LR-Zerlegung zeigt sich aber, wenn mehrere Systeme mit der selben Matrix A und unterschiedlichen rechten Seiten $\mathbf{b}_1$,$\mathbf{b}_2, \ldots$ gelöst werden sollen. Die LR-Zerlegung ist der arbeitsintensive Teil ($\sim n^3/3$ Punktoperationen) und muss nur einmal durchgeführt werden. Die einzelnen Lösungen kosten dann nur mehr $\sim n^2$ Punktoperationen pro rechter Seite.

> Für symmetrische Matrizen lassen sich spezielle Varianten der Gauß-Elimination durchführen. Ausnützen der Symmetrie spart Rechenzeit und Speicherplatz. Eine mögliche Zerlegung ist
>
> $$A = LDL^T \ ,$$
>
> mit einer Diagonalmatrix D. Für symmetrisch positiv definiter Matrizen ist die Zerlegung
>
> $$A = LL^T$$
>
> von Bedeutung.

3.6 Ein Rechenbeispiel zu Gauß-Elimination und LR-Zerlegung

Das Gaußsche Eliminationsverfahren ist eine Rechenvorschrift zur systematischen Elimination von Unbekannten. Bei Gleichungssystemen mit „einfachen“ Zahlen weicht man oft vom systematischen Weg ab und versucht, den Rechengang abzukürzen (z. B. schon vorhandene Nullen auszunützen). Dabei besteht immer die Gefahr, „im Kreis herum“ zu rechnen, Gleichungen nicht oder doppelt zu verwenden. Es ist daher wichtig, eine allgemein gültige Rechenvorschrift angeben zu können.
Gegeben sei das Gleichungssystem $A \cdot \mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 6 \\ 6 \\ 14 \end{bmatrix}$$

Man rechnet man mit der *erweiterten Koeffizientenmatrix*

$$[A\,\mathbf{b}] = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 10 & 20 & 23 & 6 \\ 15 & 50 & 67 & 14 \end{bmatrix}$$

Elimination in der ersten Spalte

(Vergleichen Sie mit dem Algorithmus auf Seite 25)

$n = 3$, in der Spalte $k = 1$ sollen alle Einträge unterhalb des Diagonalelementes eliminiert werden, das sind die Elemente in Zeilen $i = 2{,}3$

$k = 1, i = 2$: Elimination in erster Spalte, zweite Zeile

Elimination von $a_{ik} = a_{21} = 10$ mittels des Diagonalelements $a_{kk} = 5$ von Zeile $k = 1$. Multipliziere erste Zeile mit $a_{ik}/a_{kk} = 2$ und subtrahiere:

$$\begin{array}{cccc|} 10 & 20 & 23 & 6 & \\ 10 & 12 & 14 & 12 & |- \\ \hline 0 & 8 & 9 & -6 & \end{array}$$

$k = 1, i = 3$: Elimination in erster Spalte, dritte Zeile

Elimination von $a_{ik} = a_{31} = 15$ mittels des Diagonalelements $a_{kk} = 5$ von Zeile $k = 1$. Multipliziere erste Zeile mit $a_{ik}/a_{kk} = 3$ und subtrahiere:

$$\begin{array}{cccc|} 15 & 50 & 67 & 14 & \\ 15 & 18 & 21 & 18 & |- \\ \hline 0 & 32 & 46 & -4 & \end{array}$$

Transformierte erweiterte Koeffizientenmatrix

nach Elimination in der ersten Spalte

$$[A\,\mathbf{b}]^{(1)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 32 & 46 & -4 \end{bmatrix}$$

Elimination in der zweiten Spalte

in der Spalte $k = 2$ sollen alle Einträge unterhalb des Diagonalelementes eliminiert werden, das ist nur mehr das Element in Zeile $i = 3$

$k = 2, i = 3$: Elimination in zweiter Spalte, dritte Zeile

Elimination von $a_{ik} = a_{32} = 32$ mittels des Diagonalelements $a_{kk} = 8$ von Zeile $k = 2$. Multipliziere zweite Zeile mit $a_{ik}/a_{kk} = 4$ und subtrahiere:

$$\begin{array}{cccc|} 0 & 32 & 46 & -4 & \\ 0 & 32 & 36 & -24 & |- \\ \hline 0 & 0 & 10 & 20 & \end{array}$$

Transformierte erweiterte Koeffizientenmatrix

nach Elimination in der zweiten Spalte ist das Eliminationsverfahren beendet.

$$[A\,\mathbf{b}]^{(2)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 0 & 10 & 20 \end{bmatrix}$$

Rücksubstitution

Aus der dritten Zeile:

$$\begin{aligned} 10x_3 &= 20 \\ x_3 &= 2 \end{aligned}$$

Einsetzen in zweite Zeile

$$\begin{aligned} 8x_2 + 9x_3 &= -6 \\ 8x_2 + 18 &= -6 \\ 8x_2 &= -24 \\ x_2 &= -3 \end{aligned}$$

Einsetzen in erste Zeile

$$\begin{aligned} 5x_1 + 6x_2 + 7x_3 &= 6 \\ 5x_1 - 18 + 14 &= 6 \\ 5x_1 &= 10 \\ x_1 &= 2 \end{aligned}$$

Der Matlab-Befehl `x = A\b` arbeitet im Prinzip nach diesem Verfahren, allerdings in Form der LR-Zerlegung mit Spaltenpivotsuche.

Mehrere rechte Seiten

Sind zur selben Matrix A Lösungen für mehrere rechte Seiten zu berechnen, fügt man diese der erweiterten Koeffizientenmatrix als weitere Spalten an und rechnet in gleicher Weise.

Pivotwahl

Bei diesem Beispiel war es nicht notwendig, Gleichungen zu tauschen, um Divisionen durch Null zu vermeiden. Bei Spaltenpivotwahl hätte man vor dem ersten Schritt die dritte Gleichung zur ersten gemacht (weil sie den betragsgrößten Eintrag in der ersten Spalte hat)

LR-Zerlegung

Die auf Dreiecksform transformierte Matrix und die entsprechenden Pivot-Faktoren liefern auch die LR-Zerlegung. Eine rechte Seite ist für die LR-Zerlegung nicht notwendig.

$$\begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & 10 \end{bmatrix}$$

Der MATLAB-Befehl `[L U]=lu(A)` verwendet im Prinzip das Gaußsche Eliminationsverfahren, aber liefert zu diesem Zahlenbeispiel eine andere „quasi"-LR-Zerlegung. Die U- bzw. R-Matrix ist eine echte obere Dreiecksmatrix, L ist eine „durcheinandergeratene" untere Dreiecksmatrix. Begründung: Spaltenpivotsuche vertauscht Zeilen in der Matrix und ändert im Rechengang Reihenfolge und Zahlenwerte. Sie verringert dadurch die Rundungsfehler.

$$\begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} = \begin{bmatrix} 1/3 & 4/5 & 1 \\ 2/3 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 15 & 50 & 67 \\ 0 & -40/3 & -65/3 \\ 0 & 0 & 2 \end{bmatrix}$$

3.7 Weitere Anwendungen der LR-Zerlegung

3.7.1 Determinante

Determinante
Wenn eine Faktorisierung $A = LR$ gegeben ist, ist $\det A$ das Produkt der Hauptdiagonalelemente von R.

Begründung: Die Determinante einer Dreiecksmatrix ist das Produkt der Hauptdiagonalelemente. In L stehen dort lauter Einsen, somit ist $\det L = 1$. Nach den Rechenregeln für Determinanten ist dann

$$\det A = \det(LR) = (\det L)(\det R) = \det R \; .$$

Rechenaufwand: $n^3/3 + 2n/3 - 1$ Punktoperationen.

Vergleiche dazu das klassische Verfahren, Entwickeln nach Zeilen oder Spalten: Für eine $n \times n$-Matrix sei dafür $w(n)$ der Rechenaufwand an Punktoperationen. Im allgemeinen Fall sind n Unterdeterminanten von $(n-1) \times (n-1)$-Matrizen zu berechnen und noch mit den jeweiligen Elementen zu multiplizieren. Für den Rechenaufwand gilt also

$$w(n) = nw(n-1) + n = n(w(n-1) + 1) \; .$$

Die Funktion $w(n)$ wächst rasch, stärker als die Faktorielle $n!$. Für die Determinante einer 10×10-Matrix braucht ein schneller PC (10^7 Multiplikationen/sec) eine knappe Sekunde, schon bei einer 13×13-Matrix geht sich eine Viertelstunde Kaffeepause aus, auf die Determinante einer 15×15-Matrix warten Sie zweieinhalb Tage, dreizehn Jahrtausende bei einer 20×20-Matrix, und eine 25×25-Matrix wäre nach 80 Milliarden Jahren noch nicht fertig.
Das rasante Wachsen von $w(n)$ und den vergleichsweise geringen Aufwand der LR-Zerlegung illustriert die folgende Tabelle. Sie soll eindringlich auf die Bedeutung rechengünstiger Algorithmen und den Unterschied zwischen polynomialer und exponentieller Laufzeit hinweisen.

n	$w(n)$	$n^3/3 + 2n/3 - 1$
2	2	3
3	9	10
4	40	23
5	205	44
6	1 236	75
7	8 659	118
8	69 280	175
9	623 529	248
10	6 235 300	339
15	2 246 953 104 075	1 134
20	4 180 411 311 071 440 000	2 679
25	26 652 630 354 867 072 870 693 625	5 224

3.7.2 Inverse

Die zu einer (nichtsingulären) Matrix A inverse Matrix A^{-1} braucht kaum ein Rechenverfahren in expliziter Form. Soll etwa der Vektor $\mathbf{x} = A^{-1}\mathbf{y}$ berechnet werden, formt man um in $A\mathbf{x} = \mathbf{y}$ (beide Seiten mit A multiplizieren). Daraus gewinnt man $\mathbf{x}$ als Lösung des Gleichungssystems, und das ist von Rechenaufwand und -genauigkeit her günstiger.

Warnung 1: Bevor Sie eine Inverse berechnen, fragen Sie dreimal nach, ob Sie diese wirklich brauchen.

Warnung 2: Falls Sie sich noch an die aus der linearen Algebra bekannte Formel (die mit den Determinanten der Kofaktoren) erinnern: vergessen Sie diese. Sie ist von theoretischer Bedeutung, weil sie die Existenz von Inversen nichtsingulärer Matrizen zeigt. Berechnen sollten Sie die Inverse so nicht (überlegen Sie: Rechenaufwand $O(n^5)$, wenn Sie die einzelnen Unterdeterminanten mittels LR-Zerlegung rechnen; exponentieller Rechenaufwand, wenn Sie die Determinanten entwickeln).

Wenn es denn sein muss, gehen Sie so vor: Nennen Sie die erste Spalte der Inversen $\mathbf{x}_1$. Die erste Spalte der Einheitsmatrix I ist der Einheitsvektor $\mathbf{e}_1 = (1,0,\ldots,0)^T$. Laut Definition gilt

$$AA^{-1} = I \ .$$

Daher gilt insbesondere

$$A\mathbf{x}_1 = \mathbf{e}_1 \ .$$

Das heißt: die erste Spalte der Inversen erhalten Sie als Lösung eines Gleichungssystems mit dem ersten Einheitsvektor als rechter Seite.

In offenkundiger Verallgemeinerung:

Inverse
Die i-te Spalte von A^{-1} ist Lösung des Gleichungssystems

$$A\mathbf{x}_i = \mathbf{e}_i \ .$$

Hier haben Sie also Gleichungssysteme mit derselben Matrix A und mehreren rechten Seiten zu lösen. Vorgangsweise:

Zerlegung $A = LR$; (Aufwand $(n^3 - n)/3$).
Für $i = 1,\ldots,n$
    Lösung $LR\mathbf{x}_i = \mathbf{e}_i$; (Aufwand jeweils n^2).

Rechenaufwand gesamt $(4n^3 - n)/3$.

3.7.3 Symmetrisch positiv definite Matrizen

Einfache Argumente der linearen Algebra zeigen: Für symmetrisch positiv definite Matrizen bricht die einfache Gauß-Elimination nie mit $a_{kk} = 0$ ab. Pivot-Suche ist daher unnötig. Umgekehrt kann man symmetrisch positiv definite Matrizen dadurch charakterisieren, dass die im k-ten Schritt der LR-Zerlegung auftretenden a_{kk} alle > 0 sind.

Allerdings führt man bei symmetrisch positiv definiten Matrizen eher Zerlegungen der Form $A = LL^T$ (Cholesky-Zerlegung) oder $A = LDL^T$ durch. Vorteil: bei geschickter Programmierung weniger Rechenzeit und Speicherplatz erforderlich.

3.7.4 Unvollständige Zerlegung

Auch wenn in einer Matrix A die meisten Elemente null sind, müssen die Faktoren L und R nicht schwach besetzt sein. Durch Gauß-Elimination werden zusätzliche *Füllterme* erzeugt (also Elemente $\neq 0$ an Stellen, wo die Ausgangsmatrix Elemente $= 0$ hatte). Wenn man alle (oder kleine) Füllterme einfach vernachlässigt, reduzieren sich der Rechenaufwand und benötigte Speicherplatz einer solchen *unvollständigen Zerlegung* drastisch. Natürlich gilt dann

nicht mehr $LR = A$, sondern $LR = \tilde{A}$ für eine Näherung $\tilde{A}$ an A. Unvollständigen Zerlegungen sind leistungsfähige Präkonditionierer für iterative Gleichungslöser.

Das Prinzip lässt sich durch eine geringfügige Änderung im Beispielprogramm zur LR-Zerlegung illustrieren: Man ersetze dort in der innersten Schleife

```
For j = k + 1 To n
   a(i, j) = a(i, j) - p * a(k, j)
Next
```

durch

```
For j = k + 1 To n
   if a(i, j) <> 0 then
      a(i, j) = a(i, j) - p * a(k, j)
Next
```

Damit ist aus der LR-Zerlegung eine unvollständige Zerlegung ohne zusätzliche Füllterme geworden. In dieser Form spart das Programm allerdings keinen Speicherplatz und nicht wirklich Rechenzeit. Dazu wären Datenstrukturen notwendig, die gezielt nur die Nichtnull-Elemente speichern, auf diese zugreifen und damit manipulieren (Speicherformate für schwach besetzte Matrizen).

3.8 Fehlerempfindlichkeit

Rundungsfehler und Fehler in den Eingabedaten verfälschen eine Matrix A zu $A + \delta A$ und die rechte Seite $\mathbf{b}$ zu $\mathbf{b} + \delta\mathbf{b}$. Die Lösung dieses Systems wird um $\delta\mathbf{x}$ von der Lösung des unverfälschten Systems abweichen:

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \quad .$$

Wie hängt $\delta\mathbf{x}$ von δA und $\delta\mathbf{b}$ ab?

Konditionszahl
Die *Konditionszahl* $\kappa(A)$ einer Matrix A misst, wie empfindlich der relative Fehler von $\mathbf{x}$ von kleinen relativen Änderungen in A und $\mathbf{b}$ abhängt.

Eine genauere Analyse benötigt eine *Matrixnorm*, um die „Größe“ oder „Kleinheit“ einer Matrix A zu messen. Beispiel: Die Maximums- oder Zeilensummennorm, die mit $\|A\|_\infty$ bezeichnet wird.

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|$$

Damit lässt sich schreiben (ohne Beweis):

$$\frac{\|\delta\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \approx \kappa(A) \left(\frac{\|\delta A\|_\infty}{\|A\|_\infty} + \frac{\|\delta\mathbf{b}\|_\infty}{\|\mathbf{b}\|_\infty} \right) \quad \text{mit} \quad \kappa(A) = \|A\|_\infty \|A^{-1}\|_\infty$$

Der relative Fehler in $\mathbf{x}$ kann also $\kappa(A)$-mal größer als der relative Fehler in A und $\mathbf{b}$ sein. Ein Gleichungssystem, dessen Matrix eine große Konditionszahl hat, wird sehr empfindlich auf Fehler in den Eingabedaten reagieren. Ein solches System heißt *schlecht konditioniert*. Geometrische Veranschaulichung: schleifender Schnitt zweier Geraden in der Ebene.

Die Berechnung der Konditionszahl direkt gemäß der Definition würde die Berechnung der Inversen erfordern und wäre unsinnig aufwendig. Viele Gleichungslöser liefern Schätzungen von $\kappa(A)$ als Nebenprodukt. Es gilt zum Beispiel

$$\kappa(A) \geq \left| \frac{\lambda_{max}}{\lambda_{min}} \right|$$

(Verhältnis von betragsgrößtem zu -kleinsten Eigenwert; Eigenwerte werden im nächsten Kapitel behandelt).

4 Iterative Lösungsverfahren für lineare Gleichungssysteme

Gegeben sei ein lineares Gleichungssystem in n Gleichungen und Unbekannten.

$$\begin{array}{lcl} a_{11}x_1 + a_{12}x_2 + \quad \ldots \quad + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \quad \ldots \quad + a_{2n}x_n & = & b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \quad \ldots \quad + a_{nn}x_n & = & b_n \end{array} \tag{7}$$

In Matrixschreibweise

$$A\mathbf{x} = \mathbf{b}\,. \tag{8}$$

Das klassische Lösungsverfahren dafür, jedenfalls für Systeme von zwei bis einigen hundert Gleichungen, ist Gauß-Elimination. Sie wird im nächsten Kapitel systematisch behandelt. Aus vielen Anwendungen (Strömungssimulation, Seismik, Computertomographie, Festigkeitsrechnungen mit finiten Elementen...) resultieren Gleichungssysteme mit vielen tausend oder sogar Millionen Unbekannten. Solche Systeme werden in aller Regel nur iterativ gelöst. Sie lernen hier einige Basis-Verfahren kennen, auf denen die leistungsfähigeren Methoden aufbauen.

4.1 Einfache iterative Verfahren: Jacobi, Gauß-Seidel, SOR

Angenommen, die Diagonalelemente a_{ii} in A sind alle ungleich null. Dann wäre folgendes Rezept möglich (ein Fixpunkt-Verfahren):

Jacobi-Verfahren für $A\mathbf{x} = \mathbf{b}$, locker formuliert
Löse jede Gleichung nach ihrem Diagonal-Term auf, setze Startwerte ein, iteriere.

Manipulationen mit Gleichungssystemen lassen sich in Matrix-Schreibweise übersichtlich formulieren. Wir definieren eine Matrix $D = (d_{ij})$ mit den gleichen Diagonalelementen wie A und null in allen Nichtdiagonalelementen. Die restlichen Elemente von A schreiben wir in eine Matrix E:

$$A = D + E \text{ mit } D = (d_{ij}), \quad d_{ij} = \begin{cases} a_{ii} & \text{falls } i = j, \\ 0 & \text{sonst.} \end{cases} \qquad E = A - D\,.$$

Das Gleichungssystem (8) lässt sich dann äquivalent umformen zu

$$\begin{array}{rcl} A\mathbf{x} & = & \mathbf{b} \\ (D+E)\mathbf{x} & = & \mathbf{b} \\ D\mathbf{x} & = & \mathbf{b} - E\mathbf{x} \\ \mathbf{x} & = & D^{-1}(\mathbf{b} - E\mathbf{x}) \end{array}$$

Die letzte Form ist eine Fixpunktgleichung. Die entsprechende Fixpunkt-Iteration

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{9}$$

heißt *Jacobi-Verfahren*.

Wenn Ihnen die komponentenweise Schreibung (7) des Gleichungssystems lieber ist, können sie so vorgehen: Bringen Sie jeweils in der i-ten Zeile alle Terme bis auf den i-ten auf die rechte Seite und lösen Sie nach x_i auf. Ein entsprechend umgeformtes 3×3-System sieht dann so aus:

$$\begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \end{aligned}$$

Angenommen, $\mathbf{x}^{(k)}$ ist ein näherungsweiser Lösungsvektor. Das Jacobi-Verfahren erzeugt eine neue Näherung durch

$$\begin{aligned} x_1^{(k+1)} &= (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11} \\ x_2^{(k+1)} &= (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)})/a_{22} \\ x_3^{(k+1)} &= (b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})/a_{33} \end{aligned}$$

Für allgemeines n haben wir

Iterationsschritt des Jacobi-Verfahrens

Matrix-Schreibweise

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k)})$$

Komponenten-Schreibweise
für $i = 1, \ldots, n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right) / a_{ii}$$

Das Jacobi-Verfahren verwendet nicht die aktuellste Information zur Berechnung von $x_i^{(k+1)}$. Beispielsweise wird $x_1^{(k)}$ zur Berechnung von $x_2^{(k+1)}$ herangezogen, obwohl $x_1^{(k+1)}$ bereits bekannt wäre. Wenn wir das Verfahren so formulieren, dass es immer die aktuellsten Näherungswerte an die x_i verwendet, erhalten wir das *Gauß-Seidel-Verfahren*.

Iterationsschritt des Gauß-Seidel-Verfahrens

locker formuliert
Löse der Reihe nach jede Gleichung nach ihrem Diagonal-Term auf, setze Startwerte ein, iteriere, verwende jeweils neuerste Näherungswerte.

Komponenten-Schreibweise
für $i = 1, \ldots, n$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right) / a_{ii}$$

Das Gauß-Seidel-Verfahren lässt sich oft deutlich beschleunigen, wenn man den aus der Iterationsformel erhaltenen Wert nicht direkt verwendet, sondern die Änderung von $x_i^{(k)}$ zu $x_i^{(k+1)}$

noch um einen Faktor $\omega > 1$ vergrößert. Dieses iterative Verfahren heißt *SOR-Verfahren* (SOR steht für *successive overrelaxation*)

Iterationsschritt des SOR-Verfahrens

locker formuliert

Jeweils neuer Näherungswert zuerst als Zwischenresultat $y_i^{(k+1)}$ aus Gauß-Seidel-Schritt; endgültiger Näherungswert durch Extrapolation (Überrelaxation) aus alter Näherung und Zwischenresultat: $x_i^{(k+1)} = \omega y_i^{(k+1)} + (1-\omega)x_i^{(k)}$

Komponenten-Schreibweise

für $i = 1,\ldots,n$

$$y_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

$$x_i^{(k+1)} = \omega y_i^{(k+1)} + (1-\omega)x_i^{(k)}$$

Ein geeigneten Wert für ω lässt sich aber nicht so einfach angeben. Die Theorie sagt: $1 \leq \omega < 2$, mit Werten eher in der Nähe von 2. Für $\omega = 1$ reduziert sich SOR auf Gauß-Seidel.

Die wichtigste Anwendung iterativer Lösungsverfahren für lineare Gleichungssysteme ist die numerische Lösung partieller Differentialgleichungen. Dabei treten nämlich Gleichungssysteme mit sehr vielen Unbekannten auf (eine Million Unbekannte sind durchaus möglich). In diesen Gleichungssystemen sind aber pro Gleichung nur wenige Koeffizienten ungleich null. Man spricht in so einem Fall von einer *schwach besetzten* Matrix. Iterative Gleichungslöser für solche Matrizen sind ein sehr aktives Forschungsgebiet der numerischen linearen Algebra. Für Systeme mit bis zu einigen hundert Gleichungen und Unbekannten sind jedoch in der Regel Eliminationsverfahren (siehe nächstes Kapitel) günstiger.

4.2 Konvergenz des Jacobi- und des Gauß-Seidel-Verfahrens

Nicht für jede beliebige Matrix A konvergieren die drei oben vorgestellten Verfahren. Die Konvergenz des Jacobi-Verfahrens lässt sich zeigen, indem man nachweist, dass es sich bei der Fixpunkt-Iteration um eine kontrahierende Abbildung handelt. Dazu definieren wir:

Eine $n \times n$-Matrix $A = (a_{ij})$ ist *stark diagonaldominant*, wenn

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \text{für} \quad i = 1,2,\ldots,n$$

Es muss also in jeder Zeile die Summe der Beträge der Nichtdiagonalelemente kleiner sein als der Betrag des Diagonalelementes.

Konvergenz des Jacobi-Verfahrens

Das Jacobi-Verfahren konvergiert bei Gleichungssystemen mit stark diagonaldominanter Matrix für beliebige Startwerte zur exakten Lösung.

Beweis: Wir zeigen, dass die zur Iterationsvorschrift (9) gehörige Funktion $\mathbf{\Phi}(\mathbf{x}) = D^{-1}(\mathbf{b} - E\mathbf{x})$ für beliebige $\mathbf{x}$,$\mathbf{y} \in \mathbb{R}^n$ eine kontrahierende Abbildung in der Maximumsnorm ist. Wir vereinfachen erst einmal

$$\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y}) = D^{-1}(\mathbf{b} - E\mathbf{x}) - D^{-1}(\mathbf{b} - E\mathbf{y}) = D^{-1}E(\mathbf{y} - \mathbf{x})$$

Die i-te Zeile der Matrix $D^{-1}E$ lautet

$$\frac{a_{i1}}{a_{ii}} \quad \frac{a_{i2}}{a_{ii}} \quad \cdots \quad \frac{a_{i,i-1}}{a_{ii}} \quad 0 \quad \frac{a_{i,i+1}}{a_{ii}} \quad \cdots \quad \frac{a_{in}}{a_{ii}}$$

Die Summe der Elementbeträge in dieser Zeile ist < 1 (Begründung: Aus der Summe $1/|a_{ii}|$ herausheben, Diagonaldominanz ausnützen).

Für einen beliebigen Vektor $\mathbf{z} \in \mathbb{R}^n$ lautet die i-te Komponente von $D^{-1}E\mathbf{z}$

$$\frac{a_{i1}}{a_{ii}}z_1 + \frac{a_{i2}}{a_{ii}}z_2 + \cdots + \frac{a_{i,i-1}}{a_{ii}}z_{i-1} + 0 + \frac{a_{i,i+1}}{a_{ii}}z_{i+1} + \cdots + \frac{a_{in}}{a_{ii}}z_n$$

Der Betrag dieses Ausdrucks lässt sich von oben her abschätzen, indem man die Dreiecksungleichung anwendet und anschließend alle z_j durch die Komponente mit maximalem Betrag ersetzt; das ist auch genau der Wert von $\|\mathbf{z}\|_\infty$. Herausheben liefert

$$\left|\frac{a_{i1}}{a_{ii}}\right| + \left|\frac{a_{i2}}{a_{ii}}\right| + \cdots + \left|\frac{a_{i,i-1}}{a_{ii}}\right| + \left|\frac{a_{i,i+1}}{a_{ii}}\right| + \cdots + \left|\frac{a_{in}}{a_{ii}}\right| \|\mathbf{z}\|_\infty$$

Die Summe in den Klammern ist, wie gesagt, < 1. Damit ist der gesamte Ausdruck $< \|\mathbf{z}\|_\infty$.

Was für die i-te Komponente von $D^{-1}E\mathbf{z}$ gilt, $i = 1,\ldots,n$, muss insbesondere für die Komponente mit maximalem Betrag gelten; deren Wert ist aber genau $\|D^{-1}E\mathbf{z}\|_\infty$.

Zusammengefasst haben wir somit gezeigt:

$$\|D^{-1}E\mathbf{z}\|_\infty < C\|\mathbf{z}\|_\infty \quad \text{mit} \quad C < 1\ .$$

Wir brauchen nur noch $\mathbf{z} = \mathbf{x} - \mathbf{y}$ zu wählen; dann können wir schreiben

$$\|\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y})\|_\infty = \|D^{-1}E(\mathbf{y} - \mathbf{x})\|_\infty < C\|\mathbf{y} - \mathbf{x}\|_\infty\ ,$$

quod erat demonstrandum.

Mit beträchtlich mehr Aufwand lässt sich zeigen, dass auch für eine größere Klasse von Matrizen, nämlich *schwach diagonaldominante, irreduzible* Matrizen, das Jacobi-Verfahren konvergiert. Diese Aussage ist wichtig, weil viele Aufgaben aus der Praxis (numerische Lösung partieller Differentialgleichungen) genau solche Matrizen liefern. Der Vollständigkeit halber hier die Definitionen:

Eine $n \times n$-Matrix $A = (a_{ij})$ ist *schwach diagonaldominant*, wenn

$$|a_{ii}| \geq \sum_{j=1,j\neq i}^{n} |a_{ij}| \quad \text{für} \quad i = 1,2,\ldots,n$$

und zumindest für ein i echte Ungleichheit gilt. Um Irreduzibilität zu untersuchen, zeichnen Sie für jedes i einen Punkt. Für jedes Matrixelement $a_{ij} \neq 0$ in A verbinden Sie die Punkte i und j durch einen Pfeil in Richtung $i \to j$. Die Zeichnung stellt einen *gerichteten Graph* dar. Wenn man von jedem beliebigen Punkt zu jedem anderen gelangen kann, indem man den Pfeilen folgt, dann heißt dieser Graph *zusammenhängend* und die Matrix A ist *irreduzibel.*

In der Regel konvergiert das Gauß-Seidel-Verfahren rascher als das Jacobi-Verfahren. Es braucht für die gleiche Genauigkeit typischerweise nur halb so viele Iterationen. Das SOR-Verfahren mit optimal gewähltem Relaxationsparameter ω braucht größenordnungsmäßig $\sqrt{N}$ Iterationen, wo das Jacobi-Verfahren N Iterationen braucht.

Konvergenz des Gauß-Seidel-Verfahrens (Stein, Rosenberg, 1948)
Wenn A positive Elemente in der Hauptdiagonale hat und alle andern Elemente ≤ 0 sind, dann konvergiert das Gauß-Seidel-Verfahren genau dann, wenn das Jacobi-Verfahren konvergiert. Wenn beide Verfahren konvergieren, dann ist das Gauß-Seidel-Verfahren asymptotisch schneller.

Es gibt Matrizen (welche die Voraussetzung des obigen Satzes nicht erfüllen), für die Jacobi konvergiert, Gauß-Seidel nicht.

Konvergenz des Gauß-Seidel-Verfahrens für symmetrisch positiv definite Matrizen
Ist A symmetrisch und positiv definit, dann konvergiert das Gauß-Seidel-Verfahren.

4.3 Moderne iterative Gleichungslöser

Gleichungssysteme aus dem Bereich der Strömungssimulation, der Festigkeitsanalyse, der Finanzmathematik und vieler weiterer Anwendungsgebiete erreichen leicht eine Größe von mehreren Millionen Unbekannten. Dafür sind aber pro Matrixzeile nur wenige Elemente von Null verschieden (So eine Matrix heißt *schwach besetzt*). Für die Lösung solcher Systeme werden heute fast ausschließlich iterative Verfahren eingesetzt. Die klassischen Methoden (Jacobi, Gauß-Seidel) konvergieren aber zu langsam und erfordern daher zuviel Rechenaufwand.

4.3.1 Splittings, Präkonditionierung

Angenommen, Sie sollen das System

$$A\mathbf{x} = \mathbf{b}$$

lösen. Günstige Taktik: Sie ersetzen die Matrix A in dieser Aufgabe durch eine andere Matrix $\tilde{A}$, für die Sie Gleichungssysteme viel leichter lösen können. Sie können es sich dabei einfach machen und für $\tilde{A}$ die Einheitsmatrix I wählen, oder den diagonalen Anteil von A, oder gezielt nur bestimmte Matrixelemente aus A herausstreichen.

Schreiben Sie $A = \tilde{A} + E$. Eine solche Aufspaltung heißt ein *Splitting* von A in eine Näherung (auch: *Präkonditionierer*) und einen Restanteil E. Das ursprüngliche Gleichungssystem formulieren Sie dann als Fixpunkt-Aufgabe um.

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ (\tilde{A} + E)\mathbf{x} &= \mathbf{b} \\ \tilde{A}\mathbf{x} + E\mathbf{x} &= \mathbf{b} \\ \tilde{A}\mathbf{x} &= \mathbf{b} - E\mathbf{x} \\ \mathbf{x} &= \tilde{A}^{-1}(\mathbf{b} - E\mathbf{x}) \end{aligned}$$

Das Jacobi-Verfahren benützt diese Idee mit $\tilde{A} = D$, dem diagonalen Anteil. Das Gauß-Seidel-Verfahren lässt sich ebenfalls in dieser Form darstellen, indem man $\tilde{A}$ aus A durch Streichen sämtlicher Elemente oberhalb der Hauptdiagonale gewinnt. Je besser $\tilde{A}$ die Matrix A des

ursprünglichen Problems approximiert, um so rascher konvergiert ein solches iterative Verfahren. Besonders gute Splittings entstehen aus *unvollständiger LR-Zerlegung*. Diese Methoden werden später, im Kapitel über Eliminationsverfahren, behandelt.

In der oben angegebenen Fixpunkt-Form wird das Verfahren aber nicht implementiert, da man die Matrix $\tilde{A}^{-1}$ nur in einfachsten Fällen (wie etwa $\tilde{A} = D$) explizit bilden kann. Eine algebraisch gleichwertige, aber für Rechner geeignete Form lautet

Iteration, Grundschema für $A = \tilde{A} + E$

Für ein geeignetes Splitting $A = \tilde{A} + E$, einen beliebigen Startvektor $\mathbf{x}^{(0)}$ und eine vorgegebene Genauigkeitsschranke $\epsilon > 0$ findet dieser Algorithmus eine Näherungslösung von $A\mathbf{x} = \mathbf{b}$.

Beginne mit Startvektor $\mathbf{x}^{(0)}$
setze $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
iteriere für $k = 0,1,2,\ldots$
  löse $\tilde{A}\mathbf{d}^{(k+1)} = \mathbf{r}^{(k)}$
  setze $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k+1)}$
  setze $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - A\mathbf{d}^{(k+1)}$
bis $\|\mathbf{r}^{(k+1)}\| < \epsilon$
Ergebnis: Näherungslösung $\mathbf{x}^{(k+1)}$

Bei Iterationen dieser Form nennt man $\tilde{A}$ auch die *Präkonditionierungsmatrix*.

Für einen Vektor $\mathbf{x}$ und gegebenes A und $\mathbf{b}$ bezeichnet man den Ausdruck $\mathbf{b} - A\mathbf{x}$ als *Residuum* von $\mathbf{x}$. Die Aufgabe, ein Gleichungssystem $A\mathbf{x} = \mathbf{b}$ zu lösen, kann man also gleichwertig umformulieren in die Aufgabe, ein $\mathbf{x}$ mit verschwindendem Residuum zu finden. Man kann leicht nachprüfen, dass die Vektoren $\mathbf{r}^{(k)}$ im obigen Grundschema genau die jeweiligen Residuen sind: $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$. Das Abbruchkriterium des Verfahrens fordert also ein Residuum, das betragsmäßig kleiner als eine vorgegebene Schranke ist.

Vorsicht! Ein kleines Residuum bedeutet nicht automatisch, dass auch der Fehler $\mathbf{x} - \mathbf{x}^{(k)}$ zwischen exakter und genäherter Lösung klein ist. Es gilt beispielsweise, falls A symmetrisch ist,

$$\frac{\|\mathbf{r}^{(k)}\|_2}{|\lambda_{max}|} \leq \|\mathbf{x} - \mathbf{x}^{(k)}\|_2 \leq \frac{\|\mathbf{r}^{(k)}\|_2}{|\lambda_{min}|}$$

wobei λ_{max} und λ_{min} den betragsgrößten bzw. -kleinsten Eigenwert von A bezeichnen. Wenn also $|\lambda_{min}|$ nahe Null liegt, sagt ein kleines Residuum noch nicht viel über die Größe des Fehlers aus.

Ebensowenig kann man aus der Kleinheit der Korrekturen $\mathbf{d}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ unmittelbar auf die Kleinheit des Fehlers schließen. Moderne iterative Verfahren können allerdings praktischerweise Näherungen für die Eigenwerte mit geringem Zusatzaufwand mitrechnen und somit verlässliche Schranken für den Fehler liefern.

4.3.2 Konvergenzbeschleunigung durch Minimieren des Residuums

Häufig beobachtet man beim obigen Grundschema, dass sich die Vektoren $\mathbf{d}^{(k)}$, um die sich die Näherungsvektoren pro Iterationsschritt ändern, selber nur wenig ändern. Anstatt den Näherungsvektor pro Iterationsschritt nur um den Vektor $\mathbf{d}^{(k+1)}$ zu korrigieren, kann man daher versuchen, gleich ein Vielfaches ω dieser Korrektur anzubringen.

Wenn sich der Näherungsvektor beim Schritt von k nach $k+1$ um $\omega\mathbf{d}^{(k+1)}$ ändert, dann lässt sich leicht nachrechnen, dass sich der Restvektor um $-\omega A\mathbf{d}^{(k+1)}$ ändert. Man ändert also das Grundschema, setzt

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \omega\mathbf{d}^{(k+1)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}\end{aligned}$$

und wählt ω so, dass der Betrag $\|\mathbf{r}^{(k+1)}\|$ dadurch möglichst klein wird. Wie geht das? Mit der üblichen Vorgehensweise, wenn man einen Extremwert sucht: Differenzieren und Nullsetzen der Ableitung. (Es bedeutet hier$\|\cdot\|$ immer die 2-Norm, die euklidische Länge eines Vektors.)

Wir arbeiten der Einfachheit halber mit dem Betragsquadrat von $\mathbf{r}^{(k+1)}$. Es lässt sich als Funktion von ω schreiben:

$$\begin{aligned}\|\mathbf{r}^{(k+1)}\|^2 &= (\mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}) \\ &= \Big((\mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}) \cdot (\mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)})\Big) \\ &= \Big(\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)} - 2\omega(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + \omega^2(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)})\Big)\end{aligned}$$

Die einzelnen inneren Produkte sind hier konstante skalare Größen. Differenzieren nach ω und Nullsetzen der Ableitung liefert

$$\begin{aligned}0 &= \frac{d}{d\omega}\|\mathbf{r}^{(k+1)}\|^2 \\ &= \frac{d}{d\omega}\Big(\mathbf{r}^{(k)} \cdot \mathbf{r}^{(k)} - 2\omega(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + \omega^2(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)})\Big) \\ &= -2(\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}) + 2\omega(A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)}) \quad \text{, daraus} \\ \omega &= \frac{\mathbf{r}^{(k)} \cdot A\mathbf{d}^{(k+1)}}{A\mathbf{d}^{(k+1)} \cdot A\mathbf{d}^{(k+1)}}\end{aligned}$$

4.3.3 Konvergenzbeschleunigung durch orthogonale Suchrichtungen

Wir setzen nun also

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \omega A\mathbf{d}^{(k+1)}$$

wobei ω so optimal gewählt ist, dass der Betrag von $\mathbf{r}^{(k+1)}$ minimal wird. Das heißt, jede weitere Korrektur des Residuums in Richtung $A\mathbf{d}^{(k+1)}$ kann nur eine Verschlechterung bringen. Falls im nächsten Iterationsschritt

$$\mathbf{r}^{(k+2)} = \mathbf{r}^{(k+1)} - \omega A\mathbf{d}^{(k+2)}$$

die Korrektur $A\mathbf{d}^{(k+2)}$ eine Komponente in Richtung $A\mathbf{d}^{(k+1)}$ enthält, tritt aber genau das ein.

Daher: Ist das Residuum entlang einer Richtung bereits minimiert, dann darf es entlang dieser Richtung nicht mehr korrigiert werden. Wir brauchen also ein Verfahren, das aus der Residuums-Korrektur $A\mathbf{d}^{(k+2)}$ die unerwünschte Komponente in Richtung $A\mathbf{d}^{(k+1)}$ herausnimmt. Das läßt sich durch *Orthogonalisierung* erreichen.

Seien $\mathbf{p}$ und $\mathbf{q}$ zwei Vektoren $\neq 0$. Die Komponente von $\mathbf{p}$ in Richtung von $\mathbf{q}$ ist gegeben durch

$$\left(\frac{\mathbf{p} \cdot \mathbf{q}}{\mathbf{q} \cdot \mathbf{q}}\right) \mathbf{q}$$

Der Vektor

$$\mathbf{p} - \left(\frac{\mathbf{p}\cdot\mathbf{q}}{\mathbf{q}\cdot\mathbf{q}}\right)\mathbf{q}$$

enthält also keine Komponente mehr in Richtung $\mathbf{q}$, steht also orthogonal auf $\mathbf{q}$. (Sonderfall: wenn $\mathbf{p}$ ein skalares Vielfaches von $\mathbf{q}$ ist, liefert diese Rechnung den Nullvektor.)

In dieser Weise lassen sich aus einem Vektor $\mathbf{p}$ auch sukzessive alle Komponenten in Bezug auf ein System von Vektoren herausnehmen.

Orthogonalisieren

Gegeben m von 0 verschiedene Vektoren $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m \in \mathbb{R}^n$ und ein Vektor $\mathbf{p} \in \mathbb{R}^n$. Dieser Algorithmus entfernt aus $\mathbf{p}$ alle Komponenten in Richtung $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$.

Für $i = 1 \ldots m$
  rechne inneres Produkt $r_i = \mathbf{p}\cdot\mathbf{q}_i / \mathbf{q}_i\cdot\mathbf{q}_i$
  subtrahiere Komponente: ersetze $\mathbf{p} = \mathbf{p} - r_i\mathbf{q}_i$

P. K. W. Vinsome, damals bei einer Erdölgesellschaft angestellt, veröffentlichte 1976 ein Verfahren, ORTHOMIN, welches das Grundschema der Iteration um die beiden Ideen, Orthogonalisierung und Minimierung, ergänzt. Es gibt eine Fülle von Verfahren, die auf ähnlichen Prinzipien beruhen und auch ähnlich leistungsfähig sind. Beispielsweise ist in Simulationsprogrammen im Erdölwesen (SURE von Heinemann Oil Technology) oder der Verbrennungssimulation (FIRE von AVL List GmbH) ORTHOMIN im Einsatz.

Für symmetrisch positiv definite Matrizen wurde aus diesen Ideen schon früher ein besonders elegantes und effizientes Verfahren entwickelt, die Methode der konjugierten Gradienten (Hestenes und Stiefel, 1952). Sie ist das Standardverfahren zur iterativen Lösung schwach besetzter, symmetrisch positiv definiter Systeme.

5 Überbestimmte Systeme

Ein lineares Gleichungssystem $A\mathbf{x} = \mathbf{b}$ mit mehr Gleichungen als Unbekannten heißt *überbestimmt*. In so einem Fall ist A eine $n \times m$-Matrix mit $n > m$, also rechteckig, mit mehr Zeilen als Spalten. In der Regel hat ein solches System keine exakte Lösung.

(Regelfall: m linear unabhängige Spalten in A, $m+1$ linear unabhängige Spalten in der erweiterten Koeffizientenmatrix $[A,\mathbf{b}]$. Sonderfälle: genau m lin. unabh. Spalten in A und $[A,\mathbf{b}] \rightarrow$ eindeutige exakte Lösung; weniger als m lin. unabh. Spalten $\rightarrow$ Systeme ohne vollen Spaltenrang, haben ∞ viele Lösungen im Sinn der kleinsten Quadrate, das Verfahren der Normalengleichungen ist nicht anwendbar.)

Trotzdem kann man versuchen, eine „Kompromisslösung“ zu finden, die „am ehesten hinkommt“. Überbestimmte Systeme entstehen beispielsweise bei der Auswertung von Messergebnissen, wenn mehr Messungen vorliegen als Werte gesucht sind.

5.1 Normalengleichungen

Überbestimmte Systeme

Lösung nach der *Methode der kleinsten Quadrate*: suche jenes $\mathbf{x}$, für das die euklidische Norm des Residuenvektors

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}$$

minimal wird. Führt auf die *Normalengleichungen*

$$A^T A\mathbf{x} = A^T\mathbf{b}$$

Herleitung 1: Differenzieren von $\mathbf{r} \cdot \mathbf{r}$ nach den einzelnen Komponenten von $\mathbf{x}$; Nullsetzen der Ableitung.

Herleitung 2: Nehmen wir an, $\hat{\mathbf{x}}$ sei ein Vektor, der die Gleichung $A^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$ (gleichbedeutend mit der Normalengleichung) erfüllt. Wir zeigen nun: für jeden Vektor $\mathbf{x}$ gilt

$$\|\mathbf{b} - A\mathbf{x}\|_2 \geq \|\mathbf{b} - A\hat{\mathbf{x}}\|_2,$$

das heißt, $\hat{\mathbf{x}}$ minimiert das Residuum $\|\mathbf{b} - A\mathbf{x}\|$, ist in diesem Sinn also die beste Lösung des Systems $A\mathbf{x} = \mathbf{b}$.

Beweis: Man setze $\hat{\mathbf{r}} = \mathbf{b} - A\hat{\mathbf{x}}$ und $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. Dann können wir schreiben

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = (\mathbf{b} - A\hat{\mathbf{x}}) + A(\hat{\mathbf{x}} - \mathbf{x}) = \hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})$$

Die Rechenregeln fürs innere Produkt (Distributivgesetz, $\mathbf{u} \cdot (A\mathbf{v}) = (A^T\mathbf{u}) \cdot \mathbf{v}$, $A^T\hat{\mathbf{r}} = 0$ gemäß unserer Annahme am Beginn des Beweises) liefern als Zusammenhang zwischen $\mathbf{r} \cdot \mathbf{r}$ und $\hat{\mathbf{r}} \cdot \hat{\mathbf{r}}$

$$\mathbf{r} \cdot \mathbf{r} = (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})) \cdot (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})) = \hat{\mathbf{r}} \cdot \hat{\mathbf{r}} + (A(\hat{\mathbf{x}} - \mathbf{x})) \cdot (A(\hat{\mathbf{x}} - \mathbf{x}))\ .$$

Der letzte Term ist (als inneres Produkt eines Vektors mit sich selbst) immer ≥ 0, womit die Aussage bewiesen ist.

Herleitung 3: Geometrische Veranschaulichung im Fall $\mathbf{x} \in \mathbb{R}^2$ und A eine 3×2-Matrix. Die Vektoren aus der Menge$\{A\mathbf{x} : \mathbf{x} \in \mathbb{R}^2\}$ spannen eine Ebene im Raum auf. Es soll also $A\hat{\mathbf{x}}$ jener Punkt (Ortsvektor) auf der Ebene sein, der von $\mathbf{b}$ minimalen Abstand hat. Der kleinstmögliche Abstand ist der Normalabstand. Der Residuumsvektor $\mathbf{b} - A\hat{\mathbf{x}}$ steht somit normal auf die Ebene, also auf alle Vektoren der Form $A\mathbf{x}$. Aus

$$(A\mathbf{x})^T \cdot (\mathbf{b} - A\hat{\mathbf{x}}) = \mathbf{x}^T \cdot (A^T(\mathbf{b} - A\hat{\mathbf{x}})) = 0$$

folgt $A^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$, weil nur der Nullvektor auf alle Vektoren $\mathbf{x} \in \mathbb{R}^2$ orthogonal sein kann.

5.2 Weitere Verfahren

Die Lösung überbestimmeter Systeme über die Normalengleichungen ist ein klassisches Verfahren, aber deswegen nicht unbedingt der günstigste Algorithmus. Es ist bloß das einzige, das sich bei kleinen Beispielen mit vertrauten Methoden (Matrixmultiplikation, Elimination) händisch durchrechnen lässt. Modernere Programmpakete und Rechenumgebungen (wie MATLAB) verwenden als Standardverfahren die QR-Zerlegung. Bei Systemen *ohne vollen Spaltenrang* (seien sie überbestimmt oder nicht) ist die Singulärwertzerlegung (*singular value decomposition, SVD*) vorteilhaft. In diesem Fall gibt es nämlich unendlich viele Lösungen (exakt oder im Sinn der kleinsten Quadrate), und SVD liefert automatisch die betragskleinste. Aus der QR-Zerlegung lassen sich hingegen die Lösungen mit den meisten Null-Komponenten ablesen.

5.3 Beispiele zu überbestimmten Systemen

Gegeben sind drei Gleichungen in zwei Unbekannten,

$$\begin{array}{rcrcl} 2x & + & y & = & 19 \\ -4x & + & 4y & = & 13 \\ 4x & - & y & = & 17 \end{array}$$

Das Gleichungssystem in Matrixschreibweise lautet

$$A\mathbf{x} = \mathbf{b} \text{ mit } A = \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix}$$

Methode der Normalengleichungen

Bilde $A^T \cdot A$ und $A^T\mathbf{b}$:

$$A^T \cdot A = \begin{bmatrix} 2 & -4 & 4 \\ 1 & 4 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} 36 & -18 \\ -18 & 18 \end{bmatrix} = 18 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

$$A^T\mathbf{b} = \begin{bmatrix} 2 & -4 & 4 \\ 1 & 4 & -1 \end{bmatrix} \cdot \begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix} = \begin{bmatrix} 54 \\ 54 \end{bmatrix} = 18 \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Die Normalengleichungen lauten daher (schon durch 18 gekürzt):

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \text{ oder in ausgeschriebener Form } \begin{array}{rcrcl} 2x & - & y & = & 3 \\ -x & + & y & = & 3 \end{array}$$

Addition der beiden Gleichungen liefert sofort $x = 6$, und daraus durch Einsetzen $y = 9$.

Minimaler Fehler in verschiedenen Normen

Die Normalengleichungen liefern $\mathbf{x} = [6; 9]$. Einsetzen in die ursprünglichen Gleichungen zeigt aber: diese „Lösung“ erfüllt keine der drei Gleichungen exakt. Der Fehlervektor $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ lautet

$$\begin{bmatrix} 19 \\ 13 \\ 17 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix}$$

Der Fehlervektor hat Länge 3, und das ist die kleinstmögliche Länge. Gemeint ist hier die euklidische Länge, also die Zweinorm.

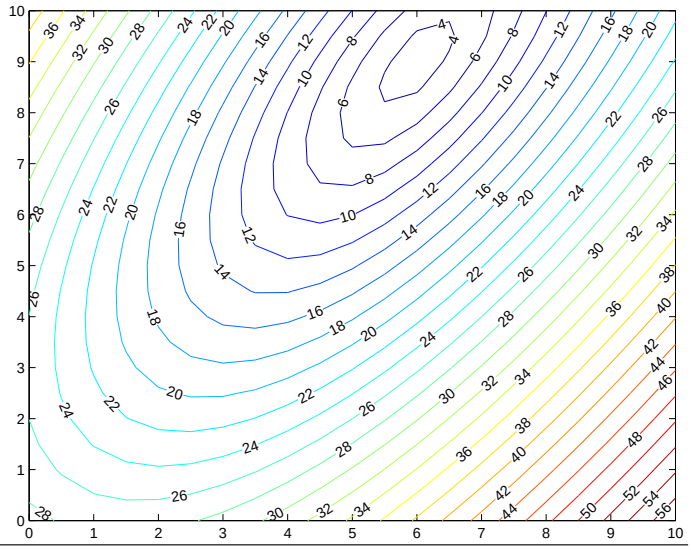
Ändert man beispielsweise den Vektor $\mathbf{x}$ geringfügig auf

$$\mathbf{x} = \begin{bmatrix} 6 \\ 8.8 \end{bmatrix} = \begin{bmatrix} 6 \\ 44/5 \end{bmatrix}$$

so lautet der Fehlervektor

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = \begin{bmatrix} -9/5 \\ 9/5 \\ 9/5 \end{bmatrix} = \begin{bmatrix} -1.8 \\ 1.8 \\ 1.8 \end{bmatrix}$$

mit euklidischer Länge $\|\mathbf{r}\|_2 = \frac{9}{5}\sqrt{3} = 3.1177$, also deutlich über dem Optimum.



Höhenschichtlinien der Länge des Fehlervektors in der Zweinorm. Das Fehlerminimum bei $[6; 9]$ ist deutlich zu erkennen.

In der Unendlichnorm (Maximum der Komponentenbeträge) ist der Fehler nun allerdings geringer: 1.8 statt 2.

Versuchen wir noch einen anderen Vektor $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} 6.15 \\ 9.4 \end{bmatrix}, \quad \text{zugehöriger Fehlervektor } \mathbf{r} = \mathbf{b} - A\mathbf{x} = \begin{bmatrix} -2.7 \\ 0 \\ 1.8 \end{bmatrix}$$

$$\|\mathbf{r}\|_2 = 3.2450, \quad \|\mathbf{r}\|_\infty = 2.7$$

Dieser Fehlervektor liegt also sowohl in der 2- als auch der ∞-Norm über den beiden vorherigen. Misst man aber die Summe der Absolutbeträge (die Einsnorm), so ergibt sich hier der Wert 4.5. Die beiden vorigen Fehlervektoren hatten Einsnormen von 5 bzw. 5.4. In der Einsnorm ist also die hier gewählte Lösung optimaler.

Lösung durch QR-Zerlegung von A

$$A = Q \cdot R\ , \quad \begin{bmatrix} 2 & 1 \\ -4 & 4 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} -1/3 & 2/3 & -2/3 \\ 2/3 & 2/3 & 1/3 \\ -2/3 & 1/3 & 2/3 \end{bmatrix} \cdot \begin{bmatrix} -6 & 3 \\ 0 & 3 \\ 0 & 0 \end{bmatrix}$$

Das transformierte Gleichungssystem $R\mathbf{x} = Q^T\mathbf{b}$ ist ebenfalls überbestimmt und lautet ausgeschrieben

$$\begin{array}{rcrcr} -6x & + & 3y & = & -9 \\ & & 3y & = & 27 \\ & & 0 & = & 3 \end{array}$$

Wenn man die ersten beiden Gleichungen exakt löst (wegen Dreiecksform einfach durch Rücksubstitution), liefern sie keinen Beitrag zum Residuum. Die letzte Gleichung hängt nicht von $\mathbf{x}$ ab. Keine Wahl von $\mathbf{x}$ kann den Beitrag dieser Gleichungen zum Residuum ändern.

Daher ist die aus den ersten beiden Gleichungen bestimmte Lösung optimal für das transformierte System. Weil die Transformation die Norm des Fehlervektors nicht beeinflusst (wegen Orthogonalität von Q), ist diese Lösung auch optimale Lösung von $A \cdot \mathbf{x} = \mathbf{b}$.

Weiteres Beispiel: Lösung mit Singulärwertzerlegung

Ein anderes überbestimmtes System lautet

$$A\mathbf{x} = \mathbf{b} \text{ mit } A = \begin{bmatrix} 14 & -2 \\ -4 & 22 \\ 16 & -13 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -80 \\ 40 \\ -145 \end{bmatrix}$$

Die Singulärwertzerlegung von A ist

$$A = U \cdot S \cdot V^T\ , \quad \begin{bmatrix} 14 & -2 \\ -4 & 22 \\ 16 & -13 \end{bmatrix} = \begin{bmatrix} -1/3 & -2/3 & -2/3 \\ 2/3 & -2/3 & 1/3 \\ -2/3 & -1/3 & 2/3 \end{bmatrix} \cdot \begin{bmatrix} 30 & 0 \\ 0 & 15 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -3/5 & -4/5 \\ 4/5 & -3/5 \end{bmatrix}^T$$

Das transformierte System lautet in diesem Fall

$$S \cdot \mathbf{y} = U^T \cdot \mathbf{b}\ , \quad \begin{bmatrix} 30 & 0 \\ 0 & 15 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 150 \\ 75 \\ -30 \end{bmatrix}$$

Wegen der Diagonalgestalt von S ist die Optimallösung direkt ablesbar: $\mathbf{y} = [5; 5]$. Die Lösung des Originalsystems erhält man über die Beziehung

$$\mathbf{x} = V \cdot \mathbf{y} = \begin{bmatrix} -3/5 & -4/5 \\ 4/5 & -3/5 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} -7 \\ 1 \end{bmatrix}$$

Wenn QR-Zerlegung oder SVD bereits gegeben sind, ist die Lösung von (überbestimmten) Gleichungssystemen relativ einfach. Der eigentliche Arbeitsaufwand steckt im Berechnen der Zerlegungen. Auf die dabei verwendeten Verfahren kann im Rahmen der Vorlesung nicht eingegangen werden.

MATLAB-Befehle

QR-Zerlegung einer Matrix A:	`[Q R]=qr(A)`
SVD, Singulärwertzerlegung $A = U \cdot S \cdot V^T$	`[U S V]=svd(A)`
Lösung eines überbestimmten Gleichungssystems	
... durch QR-Zerlegung	`A\b`
... durch SVD	`pinv(A)*b`

5.4 Anpassen eines linearen Modells (einer Ausgleichsebene)

Dieser Abschnitt erläutert anhand eines weiteren Beispiels die Lösung überbestimmter Systeme. Vom Thema her überschneidet er sich mit Kapitel 6; dort werden weitere Methoden zur Approximation von Daten behandelt. Was das Beispiel hier illustrieren soll: die unterschiedlichen Größenordnungen bei den Zwischenergebnissen und der daraus resultierene Einfluss der Rundungsfehler im Vergleich zwischen der Methode der Normalengleichung und der QR-Zerlegung.

Für die Blies (einen Nebenfluss der Saar) sollen die Hochwasserstände am Pegel Neunkirchen aus den Wasserständen des Pegels Ottweiler und des Pegels Hangard vorhergesagt werden. Es liegen die Daten der Scheitelwasserstände von 12 Winterhochwässern aus den Jahren 1963–1971 vor:

Wasserstand in cm												
Neunkirchen y	172	309	302	283	443	298	319	419	361	267	337	230
Ottweiler x_1	93	193	187	174	291	184	205	260	212	169	216	144
Hangard x_2	120	258	255	238	317	246	265	304	292	242	272	191

Daten-Quelle: U. Maniak, Hydrologie und Wasserwirtschaft, Springer, 1988

Wir unterstellen den Daten das lineare Modell $a_0 + a_1 x_1 + a_2 x_2 = y$. In diesem Ansatz sind a_0, a_1 und a_2 unbekannte Koeffizienten, die aus den zwölf gegebenen Werte-Tripeln möglichst gut bestimmt werden sollen. Geometrisch lassen sich die Tripel $(x_1|x_2|y)$ als Punkte im Raum interpretieren. Das lineare Modell entspricht dann einer Ebene, die möglichst gut an die Datenpunkte angepasst werden soll.

Einsetzen der Daten in den Ansatz liefert das Gleichungssystem

$$\begin{array}{rcrcrcr} a_0 & + & 93a_1 & + & 120a_2 & = & 172 \\ a_0 & + & 193a_1 & + & 258a_2 & = & 309 \\ a_0 & + & 144a_1 & + & 191a_2 & = & 230 \\ \vdots & & & & & & \vdots \\ a_0 & + & 187a_1 & + & 255a_2 & = & 302 \end{array}$$

Es liegt somit ein überbestimmtes Gleichungssystem $A\mathbf{x} = \mathbf{b}$ vor, mit 12×3-Matrix A und rechter Seite $\mathbf{b}$,

$$A = \begin{bmatrix} 1 & 93 & 120 \\ 1 & 193 & 258 \\ 1 & 187 & 255 \\ \vdots & \vdots & \vdots \\ 1 & 144 & 191 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 172 \\ 309 \\ 302 \\ \vdots \\ 230 \end{bmatrix}$$

Der klassische Weg zur Ausgleichslösung nach der Methode der kleinsten Quadrate führt über die Normalengleichungen $A^T \cdot A\mathbf{x} = A^T\mathbf{b}$, in diesem Beispiel

$$\begin{bmatrix} 12 & 2328 & 3000 \\ 2328 & 480202 & 609985 \\ 3000 & 609985 & 780572 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3740 \\ 766996 \\ 975996 \end{bmatrix}$$

Es treten recht unterschiedlich große Zahlen in diesem System auf (Grössenordnung 10^1 bis 10^6). Das ist typisch für Normalengleichungen und ein Indiz dafür, dass dieses System empfindlich gegenüber Rundungsfehlern ist. Numerisch günstiger, aber praktisch nur rechnergestützt durchführbar ist die QR-Zerlegung. Das transformierte System $R\mathbf{x} = Q^T\mathbf{b}$ lautet hier

$$\begin{bmatrix} -3.4641 & -672.0357 & -866.0254 \\ 0 & -169.0266 & -165.5656 \\ 0 & 0 & -56.2141 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1079.6 \\ -245.14 \\ -7.2659 \\ -2.4029 \\ \vdots \\ -11.003 \end{bmatrix}$$

Die ersten drei Gleichungen liegen als System in oberer Dreiecksform vor und sind durch Rücksubstitution exakt auflösbar. Die restlichen neun Gleichungen von der Form $0 = -2.4029, \ldots$ sind klarerweise unlösbar. Sie liefern den Beitrag zum Restvektor.

---

locker gesagt: QR-Zerlegung transformiert ein überbestimmtes System in ein exakt lösbares System in Dreiecksform und einen unlösbaren Rest. Ignorieren der unlösbaren Gleichungen liefert die bestmögliche Lösung im Sinn der kleinsten Quadrate

---

Anmerkung: Es ist zwar unmittelbar einsichtig, dass obiges Rezept die Lösung mit betragsmäßig minimalem Restvektor für das *transformierte* System liefert. Allerdings hat das *Originalsystem* einen anderen Restvektor. Die eigentliche Pointe des Verfahrens ist, dass die Transformation von dem einen zum anderen Restvektor durch Multiplikation mit einer *orthogonalen* Matrix passiert. Multiplikation mit einer orthogonalen Matrix lässt den Betrag eines Vektors unverändert. Daher hat auch der Restvektor des Originalsystems minimalen Betrag.

In MATLAB verwendet der Standard-Gleichungslöser-Befehl `A\b` im Falle eines überbestimmten Systems automatisch das QR-Verfahren und liefert hier

```
>> A\b
ans =
   22.5505
    1.3237
    0.1293
```

(Die aufwändigere Lösung mit Singulärwertzerlegung (MATLAB: `pinv(A)*b`) liefert hier das gleiche Resultat.)

Das Modell zur Vorhersage der Hochwasser-Scheitelwerte lautet somit

$$y = 22.5505 + 1.3237x_1 + 0.1293x_2$$

Man erkennt, dass die x_2-Daten gut zehnfach weniger Einfluss auf das Vorhersagemodell haben (weil der entsprechende Koeffizient im linearen Modell nur 0.1293 im Vergleich zu 1.3237 beträgt). Eine mögliche Interpretation wäre, dass der Wasserstand in Neunkirchen hauptsächlich vom Pegel Ottweiler und nicht wirklich vom Pegel Hangard abhängt.

Wie vertrauenswürdig ist dieses Modell? Einsetzen der Datenpunkte liefert den Restvektor.

Messwert y	172	309	302	283	443	298	319	419	361	267	337	230
Modellvorhersage	161	311	303	284	449	298	328	406	341	278	344	238
Residuum	11	-2	-1	-1	-6	0	-9	13	20	-11	-7	-8

Der mittlere absolute Fehler liegt bei 7,3 cm, der maximale Fehler allerdings bei 20 cm.

Eine genauere Beurteilung des Modells im Hinblick auf

- die Richtigkeit des angenommenen linearen Zusammenhangs,
- Einfluss der einzelnen Modellgrößen
- Wahrscheinlichkeit, dass der Prognosewert mit gewisser Genauigkeit zutrifft
- mögliche Ausreißer

erfordert Methoden der multivariaten Statistik und der Regressionsanalyse (und wohl auch eine größere Datenmenge).