Einführung

Die Numerische Mathematik — oder kurz Numerik — beschäftigt sich mit der Entwicklung und der Analyse von Algorithmen, mit denen mathematische Probleme am Computer gelöst werden können. Im Gegensatz zu symbolischen oder analytischen Rechnungen will man hier keine geschlossenen Formeln oder algebraischen Ausdrücke als Ergebnisse erhalten, sondern quantitative Zahlenwerte[1], daher der Name „Numerische Mathematik“.

In dieser Grundvorlesung zur Numerik werden die folgenden mathematische Probleme behandelt:

- Lösung linearer Gleichungssysteme
- Berechnung von Eigenwerten
- Interpolation
- Integration
- Nichtlineare Gleichungen und Gleichungssysteme

Ein wichtiger Bereich, der in dieser Aufzählung fehlt, sind die Differentialgleichungen; diese werden jedoch aus Zeitgründen nicht in der Vorlesung vorkommen.

Die Fülle unterschiedlicher Probleme aus der Analysis und der linearen Algebra bringt es mit sich, dass ganz verschiedene mathematische Techniken aus diesen Gebieten zur numerischen Lösung verwendet werden. Aus diesem Grund wird gerade die erste Numerik-Vorlesung oft als „Gemischtwarenladen“ aufgefasst, in dem verschiedene Themen scheinbar zusammenhanglos abgehandelt werden. Tatsächlich gibt es aber — trotz der unterschiedlichen Mathematik — eine Reihe von Grundprinzipien, die in der Numerik wichtig sind. Bevor wir im nächsten Kapitel mit der „harten“ Mathematik beginnen, wollen wir diese Prinzipien in dieser Einführung kurz und informell erläutern.

- **Korrektheit**: Eine der wesentlichen Aufgaben der numerischen Mathematik ist es, die Korrektheit von Algorithmen zu überprüfen, d.h. sicherzustellen, dass und unter welchen Voraussetzungen an die Problemdaten tatsächlich das richtige Ergebnis berechnet wird. Diese Überprüfung soll natürlich mit mathematischen Methoden durchgeführt werden, d.h. am Ende steht ein formaler mathematischer Beweis, der die korrekte Funktion eines Algorithmus sicher stellt. In vielen Fällen wird ein

[1] die dann natürlich oft grafisch aufbereitet werden

Algorithmus kein exaktes Ergebnis in endlich vielen Schritten liefern, sondern eine Näherungslösung bzw. eine Folge von Näherungslösungen. In diesem Fall ist zusätzlich zu untersuchen, wie groß der Fehler der Näherungslösung in Abhängigkeit von den vorhandenen Parametern ist bzw. wie schnell die Folge von Näherungslösungen gegen den exakten Wert konvergiert.

- **Effizienz**: Hat man sich von der Korrektheit eines Algorithmus' überzeugt, so stellt sich im nächsten Schritt die Frage nach der Effizienz eines Algorithmus. Liefert der Algorithmus ein exaktes Ergebnis in endlich vielen Schritten, so ist im Wesentlichen die Anzahl der Operationen „abzuzählen", falls eine Folge von Näherungslösungen berechnet wird, so muss die Anzahl der Operationen pro Näherungslösung und die Konvergenzgeschwindigkeit gegen die exakte Lösung untersucht werden.

  Oft gibt es viele verschiedene Algorithmen zur Lösung eines Problems, die je nach den weiteren Eigenschaften des Problems unterschiedlich effizient sind.

- **Robustheit und Kondition**: Selbst wenn ein Algorithmus in der Theorie in endlich vielen Schritten ein exaktes Ergebnis liefert, wird dies in der numerischen Praxis nur selten der Fall sein. Der Grund hierfür liegt in den sogenannten *Rundungsfehlern*: Intern kann ein Computer nur endlich viele Zahlen darstellen, es ist also unmöglich, jede beliebige reelle (ja nicht einmal jede rationale) Zahl exakt darzustellen. Wir wollen diesen Punkt etwas formaler untersuchen. Für eine gegebene *Basis* $B \in \mathbb{N}$ kann jede reelle Zahl $x \in \mathbb{R}$ als

  $$x = m \cdot B^e$$

  dargestellt werden, wobei $m \in \mathbb{R}$ die *Mantisse* und $e \in \mathbb{Z}$ der *Exponent* genannt wird. Durch geeignete Wahl von e reicht es dabei, Zahlen der Form $m = \pm 0.m_1 m_2 m_3 \ldots$ mit den Ziffern $m_1, m_2, \ldots$ mit $m_i \in \{0, 1, \ldots, B-1\}$ zu benutzen. Computerintern wird üblicherweise die Basis $B = 2$ verwendet, da die Zahlen als *Binärzahlen* dargestellt werden. Im Rechner stehen nun nur endlich viele Stellen für m und e zur Verfügung, z.B. l Stellen für m und n Stellen für e. Wir schreiben $m = \pm 0.m_1 m_2 m_3 \ldots m_l$ und $e = \pm e_1 e_2 \ldots e_n$. Unter der zusätzlichen *Normierungs-Bedingung* $m_1 \neq 0$ ergibt sich eine eindeutige Darstellung der sogenannten *maschinendarstellbaren Zahlen*

  $$\mathcal{M} = \{x \in \mathbb{R} \,|\, \pm 0.m_1 m_2 m_3 \ldots m_l \cdot B^{\pm e_1 e_2 \ldots e_n}\} \ \cup \ \{0\}.$$

  Zahlen, die nicht in dieser Menge $\mathcal{M}$ liegen, müssen durch Rundung in eine maschinendarstellbare Zahl umgewandelt werden.

  Die dadurch entstehenden Ungenauigkeiten beeinflussen offensichtlich das Ergebnis numerischer Algorithmen. Die Robustheit eines Algorithmus ist nun dadurch bestimmt, wie stark diese Rundungsfehler sich im Ergebnis auswirken. Tatsächlich betrachtet man die Robustheit mathematisch für allgemeine Fehler, so dass egal ist, ob diese durch Rundung oder weitere Fehlerquellen (Eingabe- bzw. Übertragungsfehler, Ungenauigkeiten in vorausgegangenen Berechnungen etc.) hervorgerufen worden sind.

  Ein wichtiges Hilfsmittel zur Betrachtung dieser Problemstellung ist der Begriff der *Kondition* eines mathematischen Problems. Wenn wir das Problem abstrakt als Abbildung $\mathcal{A} : D \to L$ der Problemdaten D (z.B. Matrizen, Messwerte. . .) auf die

Lösung L des Problems betrachten, so gibt die Kondition an, wie stark sich kleine Änderungen in den Daten D in der Lösung L auswirken, was durch die Analyse der Ableitung von $\mathcal{A}$ quantitativ bestimmt wird. Wenn kleine Änderungen in D große Änderungen in L verursachen können spricht man von einem *schlecht konditionierten* Problem. Beachte, dass die Kondition eine Eigenschaft des gestellten Problems und damit unabhängig von dem verwendeten Algorithmus ist. Allerdings ist die Robustheit eines Algorithmus besonders bei schlecht konditionierten Problemen wichtig, da hier kleine Fehler im Algorithmus große Fehler im Ergebnis verursachen können. Am schönsten entwickelt ist diese Theorie bei linearen Gleichungssystemen; in diesem Rahmen werden wir sie auch ausführlich untersuchen.

Kapitel 1

Lineare Gleichungssysteme

Algorithmen zur Lösung linearer Gleichungssysteme bilden die Basis für viele Anwendungen der Numerik und stehen daher traditionell am Anfang vieler Numerik-Vorlesungen. Ausführlich aufgeschrieben besteht das Problem darin, Zahlen $x_1, \dots, x_n \in \mathbb{R}$ zu bestimmen, für die das Gleichungssystem

$$\begin{array}{ccccccccc} a_{1\,1}x_1 & + & a_{1\,2}x_2 & + & \dots & + & a_{1\,n}x_n & = & b_1 \\ & & & & & & & \vdots & \\ a_{n\,1}x_1 & + & a_{n\,2}x_2 & + & \dots & + & a_{n\,n}x_n & = & b_n \end{array} \tag{1.1}$$

erfüllt ist. Die ausführliche Schreibweise in (1.1) ist etwas unhandlich, weswegen wir lineare Gleichungsysteme in der üblichen Matrix-Form schreiben werden, nämlich als

$$Ax = b, \tag{1.2}$$

mit

$$A = \begin{pmatrix} a_{1\,1} & a_{1\,2} & \dots & a_{1\,n} \\ \vdots & \vdots & & \vdots \\ a_{n\,1} & a_{n\,2} & \dots & a_{n\,n} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}. \tag{1.3}$$

Diese Schreibweise hat nicht nur den Vorteil, dass man ein Gleichungssystem viel kürzer aufschreiben kann, es wird sich auch zeigen, dass gewisse Eigenschaften der Matrix A entscheiden, was für ein Verfahren zur Lösung von (1.2) sinnvollerweise eingesetzt werden kann.

Einige kurze Bemerkungen zur Notation: Wir werden Matrizen typischerweise mit großen Buchstaben bezeichnen (z.B. A) und Vektoren mit kleinen (z.B. b). Ihre Einträge werden wir mit indizierten Kleinbuchstaben bezeichnen, wie in (1.3). Mit einem hochgestellten „T“ bezeichnen wir transponierte Matrizen und Vektoren, für A und x aus (1.3) also z.B.

$$x = (x_1, \dots, x_n)^T, \quad A^T = \begin{pmatrix} a_{1\,1} & a_{2\,1} & \dots & a_{n\,1} \\ \vdots & \vdots & & \vdots \\ a_{1\,n} & a_{2\,n} & \dots & a_{n\,n} \end{pmatrix}.$$

Da die Anzahl n der Gleichungen in (1.1) gleich der Anzahl der Unbekannten $x_1, \dots, x_n$ ist, ist A in (1.2) eine quadratische Matrix der Dimension $n \times n$. Für quadratische Matrizen

ist aus der linearen Algebra bekannt, dass genau dann eine eindeutige Lösung von (1.2) existiert, wenn die Matrix invertierbar ist. Wir werden in diesem Kapitel immer annehmen, dass dies der Fall ist.

In den Übungen werden Beispiele von Problemen aus den Wirtschaftswissenschaften und der Physik behandelt, die auf die Lösung linearer Gleichungssysteme führen. Hier werden wir einige „innermathematische“ Probleme betrachten, deren numerische Behandlung ebenfalls auf die Lösung linearer Gleichungssysteme führt.

1.1 Eine Anwendung linearer Gleichungssysteme: Ausgleichsrechnung

Das erste unserer Anwendungsbeispiele ist für viele praktische Zwecke besonders wichtig, weswegen wir es etwas genauer untersuchen wollen.

Nehmen wir an, wir haben ein Experiment durchgeführt, bei dem wir für verschiedene Eingabewerte $t_1, t_2, \ldots, t_m$ Messwerte $z_1, z_2, \ldots, z_m$ erhalten. Auf Grund von theoretischen Überlegungen (z.B. auf Grund eines zugrundeliegenden physikalischen Gesetzes), kennt man eine Funktion $f(t)$, für die $f(t_i) = z_i$ gelten sollte. Diese Funktion wiederum hängt aber nun von unbekannten Parametern $x_1, \ldots, x_n$ ab; wir schreiben $f(t; x)$ für $x = (x_1, \ldots, x_n)^T$, um dies zu betonen. Z.B. könnte $f(t; x)$ durch

$$f(t; x) = x_1 + x_2 t \quad \text{oder} \quad f(t; x) = x_1 + x_2 t + x_3 t^2$$

gegeben sein. Im ersten Fall beschreibt die gesuchte Funktion eine Gerade, im zweiten eine (verallgemeinerte) Parabel. Wenn wir annehmen, dass die Funktion f das Experiment wirklich exakt beschreibt und keine Messfehler vorliegen, so könnten wir die Parameter x_i durch Lösen des (im Allgemeinen nichtlinearen) Gleichungssystems

$$\begin{aligned} f(t_1; x) &= z_1 \\ &\vdots \\ f(t_k; x) &= z_k \end{aligned} \tag{1.4}$$

nach x bestimmen. In vielen praktischen Fällen ist dieses Gleichungssystem linear, so z.B. in den zwei obigen Beispielen, in denen sich (1.4) zu $\widetilde{A}x = z$ mit

$$\widetilde{A} = \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_m \end{pmatrix} \quad \text{bzw.} \quad \widetilde{A} = \begin{pmatrix} 1 & t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 \end{pmatrix} \quad \text{und} \quad z = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix}$$

ergibt. Diese linearen Gleichungssysteme haben m Gleichungen (eine für jedes Wertepaar (t_i, z_i)) und n Unbekannte (nämlich gerade die unbekannten Parameter x_i), wobei m üblicherweise sehr viel größer als n ist. Man sagt, dass das Gleichungsystem *überbestimmt* ist. Da Messwerte eines Versuchs praktisch immer mit Fehlern behaftet sind, ist es sicherlich zu optimistisch, anzunehmen, dass das Gleichungssystem $\widetilde{A}x = z$ lösbar ist (überbestimmte Gleichungssysteme haben oft keine Lösung!). Statt also den (vermutlich vergeblichen) Versuch zu machen, eine exakte Lösung x dieses Systems zu finden, wollen wir probieren,

eine möglichst gute Näherungslösung zu finden, d.h. wenn $\widetilde{A}x = z$ nicht lösbar ist, wollen wir zumindest ein x finden, so dass $\widetilde{A}x$ möglichst nahe bei z liegt. Dazu müssen wir ein Kriterium für „möglichst nahe“ wählen, das sowohl sinnvoll ist als auch eine einfache Lösung zulässt. Hier bietet sich das sogenannte *Ausgleichsproblem* (auch *Methode der kleinsten Quadrate* genannt) an:

Finde $x = (x_1, \dots, x_n)^T$, so dass $\varphi(x) := \|z - \widetilde{A}x\|^2$ minimal wird.

Hierbei bezeichnet $\|y\|$ die euklidische Norm eines Vektors $y \in \mathbb{R}^n$, also

$$\|y\| = \sqrt{\sum_{i=1}^{n} y_i^2}.$$

Um die Funktion φ zu minimieren, setzten wir den Gradienten $\nabla\varphi(x)$ gleich Null. Beachte dazu, dass der Gradient der Funktion $g(x) := \|f(x)\|^2$ für beliebiges $f : \mathbb{R}^n \to \mathbb{R}^n$ durch $\nabla g(x) = 2Df(x)^T f(x)$ gegeben ist. Wir erhalten also

$$\nabla\varphi(x) = 2\widetilde{A}^T\widetilde{A}x - 2\widetilde{A}^T z.$$

Falls $\widetilde{A}$ vollen Spaltenrang besitzt, ist die zweite Ableitung $D^2\varphi(x) = 2\widetilde{A}^T\widetilde{A}$ positiv definit, womit jede Nullstelle des Gradienten $\nabla\varphi$ ein Minimum von φ ist. Folglich minimiert ein Vektor x die Funktion φ genau dann, wenn die sogenannten *Normalengleichungen* $\widetilde{A}^T\widetilde{A}x = \widetilde{A}^T z$ erfüllt sind. Das Ausgleichsproblem wird also wie folgt gelöst:

$$\text{löse} \quad Ax = b \quad \text{mit} \quad A = \widetilde{A}^T\widetilde{A} \text{ und } b = \widetilde{A}^T z.$$

Das zunächst recht kompliziert scheinende Minimierungsproblem für φ wird also auf die Lösung eines linearen Gleichungssystems zurückgeführt.

Neben der Ausgleichsrechnung gibt es viele weitere Anwendungen in der Numerik, die auf die Lösung eines linearen Gleichungssystems führen. Einige davon werden uns im weiteren Verlauf dieser Vorlesung noch begegnen, z.B. die Lösung *nichtlinearer* Gleichungssysteme mit dem *Newton-Verfahren*, bei dem eine Folge linearer Gleichungssysteme gelöst werden muss, oder die *Interpolation* von Punkten mittels *Splines*.

1.2 Das Gauß'sche Eliminationsverfahren

Wir werden nun ein erstes Verfahren zur Lösung linearer Gleichungssysteme kennen lernen. Das *Gauß'sche Eliminationsverfahren* ist ein sehr anschauliches Verfahren, das zudem recht leicht implementiert werden kann. Es beruht auf der einfachen Tatsache, dass ein lineares Gleichungssystem $Ax = b$ leicht lösbar ist, falls die Matrix A in *oberer Dreiecksform* vorliegt, d.h.

$$A = \begin{pmatrix} a_{1\,1} & a_{1\,2} & \dots & a_{1\,n} \\ 0 & a_{2\,2} & \dots & a_{2\,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n\,n} \end{pmatrix}.$$

In diesem Fall kann man $Ax = b$ leicht mittels der rekursiven Vorschrift

$$x_n = \frac{b_n}{a_{n\,n}}, \;\; x_{n-1} = \frac{b_{n-1} - a_{n-1\,n}x_n}{a_{n-1\,n-1}}, \ldots, \;\; x_1 = \frac{b_1 - a_{1\,2}x_2 - \ldots - a_{1\,n}x_n}{a_{1\,1}}$$

oder, kompakt geschrieben,

$$x_i = \frac{b_i - \sum\limits_{j=i+1}^{n} a_{i\,j}x_j}{a_{i\,i}}, \qquad i = n,\, n-1, \ldots, 1 \tag{1.5}$$

(mit der Konvention $\sum_{j=n+1}^{n} a_{i\,j}x_j = 0$) lösen. Dieses Verfahren wird als *Rückwärtseinsetzen* bezeichnet.

Die Idee des Gauß'schen Eliminationsverfahrens liegt nun darin, das Gleichungssystem $Ax = b$ in ein Gleichungssystem $\widetilde{A}x = \tilde{b}$ umzuformen, so dass die Matrix $\widetilde{A}$ in oberer Dreiecksform vorliegt. Wir wollen dieses Verfahren zunächst an einem Beispiel veranschaulichen.

Beispiel 1.1 Gegeben sei das lineare Gleichungssystem (1.2) mit

$$A = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 2 & 3 & 4 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 29 \\ 43 \\ 20 \end{pmatrix}.$$

Um die Matrix A auf obere Dreiecksgestalt zu bringen, müssen wir die drei Einträge 7, 2 und 3 unterhalb der Diagonalen auf 0 bringen („eliminieren"). Wir beginnen mit der 2. Hierzu subtrahieren wir 2-mal die erste Zeile von der letzten und erhalten so

$$A_1 = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 0 & -7 & -8 \end{pmatrix}$$

Das Gleiche machen wir mit dem Vektor b. Dies liefert

$$b_1 = \begin{pmatrix} 29 \\ 43 \\ -38 \end{pmatrix}.$$

Nun fahren wir fort mit der 7: Wir subtrahieren 7-mal die erste Zeile von der zweiten, sowohl in A_1 als auch in b_1, und erhalten

$$A_2 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & -7 & -8 \end{pmatrix} \quad \text{und} \quad b_2 = \begin{pmatrix} 29 \\ -160 \\ -38 \end{pmatrix}.$$

Im dritten Schritt „eliminieren" wir die -7, die jetzt an der Stelle der 3 steht, indem wir 7/26-mal die zweite Zeile von der dritten subtrahieren:

$$A_3 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & 0 & \frac{22}{13} \end{pmatrix} \quad \text{und} \quad b_3 = \begin{pmatrix} 29 \\ -160 \\ \frac{66}{13} \end{pmatrix}.$$

Hiermit sind wir fertig und setzen $\widetilde{A} = A_3$ und $\tilde{b} = b_3$. Rückwärtseinsetzen gemäß (1.5) liefert dann

$$x_3 = \frac{\frac{66}{13}}{\frac{22}{13}} = 3, \;\; x_2 = \frac{-160 - 3 \cdot (-36)}{-26} = 2 \;\text{ und }\; x_1 = \frac{29 - 2 \cdot 5 - 3 \cdot 6}{1} = 1.$$

□

Wir formulieren den Algorithmus nun für allgemeine quadratische Matrizen A.

Algorithmus 1.2 (Gauß-Elimination, Grundversion)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(1) Für j von 1 bis $n-1$ (j = Spaltenindex des zu eliminierenden Eintrags)

(2) Für i von n bis $j+1$ (rückwärts zählend)
(i = Zeilenindex des zu eliminierenden Eintrags)
Subtrahiere a_{ij}/a_{jj}-mal die j-te Zeile von der i-ten Zeile:

Setze $\alpha := a_{ij}/a_{jj}$ und berechne

$$a_{ik} := a_{ik} - \alpha\, a_{jk} \;\text{ für } k = j, \ldots, n, \quad b_i := b_i - \alpha\, b_j$$

Ende der i-Schleife

(3) Ende der j-Schleife

□

Es kann passieren, dass dieser Algorithmus zu keinem Ergebnis führt, obwohl das Gleichungssystem lösbar ist. Der Grund dafür liegt in Schritt (1), in dem durch das Diagonalelement a_{jj} geteilt wird. Hierbei wurde stillschweigend angenommen, dass dieses ungleich Null ist, was aber im Allgemeinen nicht der Fall sein muss. Glücklicherweise gibt es eine Möglichkeit, dieses zu beheben:

Nehmen wir an, dass wir Schritt (1) für gegebene Indizes i und j ausführen möchten und $a_{jj} = 0$ ist. Nun gibt es zwei Möglichkeiten: Im ersten Fall ist $a_{ij} = 0$. In diesem Fall brauchen wir nichts zu tun, da das Element a_{ij}, das auf 0 gebracht werden soll, bereits gleich 0 ist. Im zweiten Fall gilt $a_{ij} \neq 0$. In diesem Fall können wir die i-te und j-te Zeile der Matrix A sowie die entsprechenden Einträge im Vektor b vertauschen, wodurch wir die gewünschte Eigenschaft $a_{ij} = 0$ erreichen, nun allerdings nicht durch Elimination sondern durch Vertauschung. Dieses Verfahren nennt man *Pivotierung* und der folgende Algorithmus bringt nun tatsächlich jedes lineare Gleichungssystem in Dreiecksform.

Algorithmus 1.3 (Gauß-Elimination mit Pivotierung)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(1) Für j von 1 bis $n-1$ (j = Spaltenindex des zu eliminierenden Eintrags)

(1a) Falls $a_{j\,j} = 0$ wähle ein $p \in \{j+1, \dots, n\}$ mit $a_{p\,j} \neq 0$ und vertausche $a_{j\,k}$ und $a_{p\,k}$ für $k = j, \dots, n$ sowie b_p und b_j

(2) Für i von n bis $j+1$ (rückwärts zählend)
(i = Zeilenindex des zu eliminierenden Eintrags)
Subtrahiere $a_{i\,j}/a_{j\,j}$-mal die j-te Zeile von der i-ten Zeile:
Setze $\alpha := a_{i\,j}/a_{j\,j}$ und berechne

$$a_{i\,k} := a_{i\,k} - \alpha\, a_{j\,k} \text{ für } k = j, \dots, n, \quad b_i := b_i - \alpha\, b_j$$

Ende der i-Schleife

(3) Ende der j-Schleife

□

Das Element $a_{i\,j}$, das in Schritt (1a) mit $a_{j\,j}$ getauscht wird, nennt man *Pivotelement*. Wir werden später im Abschnitt 1.4 eine Strategie kennen lernen, bei der — auch wenn $a_{j\,j} \neq 0$ ist — gezielt ein Element $a_{k\,j} \neq 0$ als Pivotelement ausgewählt wird, mit dem man ein besseres Robustheitsverhalten des Algorithmus' erhalten kann.

1.3 LR-Faktorisierung

In der obigen Version des Gauß-Verfahrens haben wir die Matrix A auf obere Dreiecksform gebracht und zugleich alle dafür notwendigen Operationen auch auf den Vektor b angewendet. Es gibt alternative Möglichkeiten, lineare Gleichungssysteme zu lösen, bei denen der Vektor b unverändert bleibt. Wir werden nun ein Verfahren kennen lernen, bei dem die Matrix A in ein Produkt von zwei Matrizen L und R zerlegt wird, also $A = LR$, wobei R in oberer Dreiecksform und L in *unterer Dreiecksform*

$$L = \begin{pmatrix} l_{1\,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n-1\,1} & \dots & l_{n-1\,n-1} & 0 \\ l_{n\,1} & l_{n\,2} & \dots & l_{n\,n} \end{pmatrix}.$$

vorliegt. Die Zerlegung $A = LR$ wird als *LR-Faktorisierung* oder *LR-Zerlegung* bezeichnet.

Um $Ax = b$ zu lösen, kann man dann $LRx = b$ wie folgt in zwei Schritten lösen: Zunächst löst man das Gleichungssystem $Ly = b$. Dies kann, ganz analog zum Rückwärtseinsetzen (1.5) durch *Vorwärtseinsetzen* geschehen:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{i\,j} y_j}{l_{i\,i}}, \qquad i = 1, 2, \dots, n \tag{1.6}$$

Im zweiten Schritt löst man dann durch Rückwärtseinsetzen das Gleichungssystem $Rx = y$. Dann gilt

$$Ax = LRx = Ly = b,$$

womit das gewünschte System $Ax = b$ gelöst ist.

Das Gauß'sche Eliminationsverfahren lässt sich so erweitern, dass damit — bis auf die Zeilenvertauschungen in A — eine LR-Zerlegung einer invertierbaren Matrix A möglich ist:

Die Subtraktion „i-te Zeile – α-mal j-te Zeile" im m-ten Schritt des Algorithmus lässt sich durch Multiplikation von links mit der Matrix

$$F_m = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & -\alpha & & \ddots & \\ & & & & \ddots \\ & & & & & 1 \end{pmatrix} \quad \text{mit} \quad F_m^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & \alpha & & \ddots & \\ & & & & \ddots \\ & & & & & 1 \end{pmatrix}$$

erzielen, hierbei steht α in der i-ten Zeile und j-ten Spalte.

Wenn sich die Gauß-Elimination ohne Zeilenvertauschung durchführen lässt, so werden nur Operationen mit solchen Matrizen benötigt. Wenn man die dabei benötigten Matrizen $F_1, F_2, \ldots, F_k$ zu einer Matrix $F = F_k \cdot F_{k-1} \cdots F_1$ aufmultipliziert, so erhält man $A = LR$ mit $L = F^{-1} = F_1^{-1} \cdot F_2^{-1} \cdots F_k^{-1}$, also die LR-Faktorisierung (beachte, dass F^{-1}-Matrix als Produkt unterer Dreiecksmatrizen wieder eine untere Dreiecksmatrix ist).

Sind allerdings wegen der Pivotierung Zeilenvertauschungen nötig, so lässt sich i.A. keine LR-Zerlegung erhalten.[1] Die Vertauschung der i-ten und der j-ten Zeile in A im m-ten Schritt des Algorithmus lässt sich durch die Multiplikation von links mit Matrizen der Form

$$P_m = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & 0 & & & & 1 & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & 1 & & & & 0 & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}$$

erreichen, wobei die 0-Einträge gerade an der i-ten und j-ten Stelle stehen. Sind solche Zeilenvertauschungen notwendig, so kann man die benötigten Matrizen $P_1, \ldots, P_l$ ebenfalls zu einer Matrix $P = P_l \cdots P_1$ aufmultiplizieren (P nennt man *Permutationsmatrix*).

Diese Permutation muss dabei bei der Konstruktion der Matrix L berücksichtigt werden. Um dies zu erklären, verwenden wir die folgende Notation:

[1] tatsächlich existiert für allgemeine invertierbare Matrizen im Allgemeinen keine LR-Zerlegung

Es sei A_m die Matrix, die nach dem m-ten Schritt des Algorithmus erzeugt wurde (also $A_0 = A$ und $A_k = R$, wenn k die Gesamtanzahl der Schritte im Algorithmus ist). Es sei $P^{(m)} = P_m \cdots P_1$ die aufmultiplizierte Matrix der bis zum Schritt m angefallenen Permutationen, wobei wir zur einfacheren Zählung $P_l = \mathrm{Id}$ (Einheitsmatrix) setzen, falls keine Permutation durchgeführt wurde. Schließlich sei $L^{(m)}$ die untere Dreiecksmatrix mit

$$P^{(m)} A = L^{(m)} A_m. \tag{1.7}$$

Falls bis zum m-ten Schritt keine Permutation durchgeführt wurde, gilt wie oben $L^{(m)} = F_1^{-1} \cdot F_2^{-1} \cdots F_m^{-1}$.

Nehmen wir nun an, dass vor der Durchführung des $m+1$-ten Schritts eine Pivotierung durchgeführt wird, bei der die i-te mit der j-ten Zeile vertauscht wird. Bezeichnen wir die zugehörige Permutationsmatrix mit P_{m+1}, so wird die Matrix A_m in (1.7) also verändert zu $\widetilde{A}_m = P_{m+1} A_m$. Aus der Konstruktion von $P^{(m)}$ folgt zudem $P^{(m+1)} = P_{m+1} P^{(m)}$. Damit (1.7) nach der Pivotierung weiterhin gültig ist, muss die Matrix $L^{(m)}$ also durch eine Matrix $\widetilde{L}^{(m)}$ ersetzt werden, für die die Gleichung

$$P^{(m+1)} A = \widetilde{L}^{(m)} \widetilde{A}_m$$

gilt. Aus (1.7) folgt nun

$$P^{(m+1)} A = P_{m+1} P^{(m)} A = P_{m+1} L^{(m)} A_m = P_{m+1} L^{(m)} P_{m+1}^{-1} \widetilde{A}_m$$

und wir erhalten $\widetilde{L}^{(m)} = P_{m+1} L^{(m)} P_{m+1}^{-1}$.

Aus der Form von P_{m+1} folgt $P_{m+1}^{-1} = P_{m+1}$. Die Multiplikation mit P_{m+1} von rechts bewirkt daher gerade die Vertauschung der i-ten und j-ten Spalte in $L^{(m)}$. Die Matrix $\widetilde{L}^{(m)}$ entsteht also, indem wir erst die i-te und j-te Zeile und dann die i-te und j-te Spalte in $L^{(m)}$ vertauschen.

Beachte dabei, dass $L^{(m)}$ auf Grund der Form der Matrizen F_l^{-1} stets 1-Einträge auf der Diagonalen besitzt und darüberhinaus Einträge ungleich Null nur unterhalb der Diagonalen und nur in den Spalten $1, \ldots, j-1$ besitzt. Weil im Algorithmus zudem stets $i > j$ gilt, werden durch die Zeilen- und Spaltenvertauschung daher gerade die Elemente l_{ik} und l_{jk} für $k = 1, \ldots, j-1$ vertauscht. Insbesondere ist daher auch $\widetilde{L}^{(m)}$ wieder eine untere Dreiecksmatrix.

Details zum Implementierung dieses Algorithmus werden in den Übungen besprochen. Sie finden sich darüberhinaus z.B. in den Büchern von Deuflhard/Hohmann [2], Schwarz/Köckler [9] oder Stoer [10].

1.4 Fehlerabschätzungen und Kondition

Wie bereits im einführenden Kapitel erläutert, können Computer nicht alle reellen Zahlen darstellen, weswegen alle Zahlen intern gerundet werden, damit sie in die endliche Menge der *maschinendarstellbaren Zahlen* passen. Hierdurch entstehen *Rundungsfehler*. Selbst wenn sowohl die Eingabewerte als auch das Ergebnis eines Algorithmus maschinendarstellbare Zahlen sind, können solche Fehler auftreten, denn auch die (möglicherweise nicht

darstellbaren) Zwischenergebnisse eines Algorithmus werden gerundet. Auf Grund dieser Fehler aber auch wegen Eingabe- bzw. Messfehlern in den vorliegenden Daten oder Fehlern aus vorhergehenden numerischen Rechnungen wird durch einen Algorithmus üblicherweise nicht die exakte Lösung x des linearen Gleichungssystems

$$Ax = b$$

berechnet, sondern eine Näherungslösung $\tilde{x}$. Um dies formal zu fassen, führt man ein „benachbartes“ oder „gestörtes“ Gleichungssystems

$$A\tilde{x} = b + \Delta b$$

ein, für das $\tilde{x}$ gerade die exakte Lösung ist. Der Vektor Δb heißt hierbei das *Residuum* oder der *Defekt* der Näherungslösung $\tilde{x}$. Den Vektor $\Delta x = \tilde{x} - x$ nennen wir den *Fehler* der Näherungslösung $\tilde{x}$. Da Rundung und andere Fehlerquellen i.A. nur kleine Fehler bewirken, ist es gerechtfertigt anzunehmen, dass $\|\Delta b\|$ „klein“ ist. Das Ziel dieses Abschnittes ist es nun, aus der Größe des Residuums $\|\Delta b\|$ auf die Größe des Fehlers $\|\Delta x\|$ zu schließen. Insbesondere wollen wir untersuchen, wie *sensibel* die Größe $\|\Delta x\|$ von $\|\Delta b\|$ abhängt, d.h. ob kleine Residuen $\|\Delta b\|$ große Fehler $\|\Delta x\|$ hervorrufen können. Diese Analyse ist unabhängig von dem verwendeten Lösungsverfahren, da wir hier nur das Gleichungssystem selbst und kein explizites Verfahren betrachten.

Um diese Analyse durchzuführen, brauchen wir das Konzept der *Matrixnorm*. Man kann Matrixnormen ganz allgemein definieren; für unsere Zwecke ist aber der Begriff der *induzierten Matrixnorm* ausreichend. Wir erinnern zunächst an verschiedene Vektornormen für Vektoren $x \in \mathbb{R}^n$. In dieser Vorlesung verwenden wir üblicherweise die *euklidische Norm* oder 2-*Norm*

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2},$$

welche wir meistens einfach mit $\|x\|$ bezeichnen. Weitere Normen sind die 1-*Norm*

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

und die *Maximums-* oder ∞-*Norm*

$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|.$$

Wir werden gleich sehen, dass die zwei letzten Normen im Zusammenhang mit linearen Gleichungssystemen gewisse Vorteile haben.

Für alle Normen im $\mathbb{R}^n$ kann man eine zugehörige *induzierte Matrixnorm* definieren.

Definition 1.4 Sei $\mathbb{R}^{n\times n}$ die Menge der $n \times n$-Matrizen und sei $\|\cdot\|_p$ eine Vektornorm im $\mathbb{R}^n$. Dann definieren wir für $A \in \mathbb{R}^{n\times n}$ die zu $\|\cdot\|_p$ gehörige *induzierte Matrixnorm* $\|A\|_p$ als

$$\|A\|_p := \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_p=1}} \|Ax\|_p = \sup_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|Ax\|_p}{\|x\|_p}.$$

□

Die Gleichheit der beiden Ausdrücke auf der rechten Seite folgt dabei aus der Beziehung $\|\alpha x\|_p = |\alpha| \, \|x\|_p$ für $\alpha \in \mathbb{R}$. Dass es sich hierbei tatsächlich um Normen auf dem Vektorraum $\mathbb{R}^{n\times n}$ handelt, wird in den Übungen bewiesen.

Da im Allgemeinen keine Verwechslungsgefahr besteht, bezeichnen wir die Vektornormen und die von ihnen induzierten Matrixnormen mit dem gleichen Symbol.

Satz 1.5 Für die zu den obigen Vektornormen gehörigen induzierten Matrixnormen und $A \in \mathbb{R}^{n\times n}$ gelten die folgenden Gleichungen

$$\|A\|_1 = \max_{j=1,\dots,n} \textstyle\sum_{i=1}^n |a_{i\,j}| \qquad \text{(Spaltensummennorm)}$$

$$\|A\|_\infty = \max_{i=1,\dots,n} \textstyle\sum_{j=1}^n |a_{i\,j}| \qquad \text{(Zeilensummennorm)}$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} \qquad \text{(Spektralnorm)},$$

wobei $\rho(A^T A)$ den maximalen Eigenwert der symmetrischen und positiv definiten Matrix $A^T A$ bezeichnet.

Beweis: Wir beweisen die Gleichungen, indem wir die entsprechenden Ungleichungen beweisen.

„$\|A\|_1$“: Für einen beliebigen Vektor $x \in \mathbb{R}^n$ gilt

$$\|Ax\|_1 = \sum_{i=1}^n \left| \sum_{j=1}^n a_{i\,j} x_j \right| \le \sum_{i=1}^n \sum_{j=1}^n |a_{i\,j}| \, |x_j| = \sum_{j=1}^n |x_j| \sum_{i=1}^n |a_{i\,j}|.$$

Sei nun j^* der Index, für den die innere Summe maximal wird, also

$$\sum_{i=1}^n |a_{i\,j^*}| = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{i\,j}|.$$

Dann gilt für Vektoren mit $\|x\|_1 = 1$

$$\sum_{j=1}^n |x_j| \sum_{i=1}^n |a_{i\,j}| \le \underbrace{\sum_{j=1}^n |x_j|}_{=1} \sum_{i=1}^n |a_{i\,j^*}| = \sum_{i=1}^n |a_{i\,j^*}|,$$

woraus, da x beliebig war, die Ungleichung

$$\|A\|_1 \le \sum_{i=1}^n |a_{i\,j^*}| = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{i\,j}|$$

folgt. Andererseits gilt für den j^*-ten Einheitsvektor e_{j^*}

$$\|A\|_1 \ge \|Ae_{j^*}\|_1 = \sum_{i=1}^n \underbrace{\left| \sum_{j=1}^n a_{i\,j} [e_{j^*}]_j \right|}_{=|a_{i\,j^*}|} = \sum_{i=1}^n |a_{i\,j^*}| = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{i\,j}|$$

und damit die behauptete Gleichung.

„$\|A\|_\infty$“: Ähnlich wie für die 1-Norm mit $x^* = (\pm 1, \ldots, \pm 1)^T$ an Stelle von e_{j^*}.

„$\|A\|_2$“: Da A^TA symmetrisch ist, können wir eine Orthonormalbasis von Eigenvektoren $v_1, \ldots, v_n$ wählen, also $\|v_i\|_2 = 1$ und $\langle v_i, v_j\rangle = 0$ für $i \neq j$. Sei nun $x \in \mathbb{R}^n$ ein beliebiger Vektor mit Länge 1, dann lässt sich x als Linearkombination $x = \sum_{i=1}^n \mu_i v_i$ mit $\sum_{i=1}^n \mu_i^2 = 1$ schreiben. Seien λ_i die zu den v_i gehörigen Eigenwerte von A^TA und sei λ_{i^*} der maximale Eigenwert, also $\lambda_{i^*} = \rho(A^TA)$. Dann gilt

$$\begin{aligned} \|Ax\|_2^2 &= \langle Ax, Ax\rangle = \langle A^TAx, x\rangle = \sum_{i,j=1}^n \mu_i\mu_j \langle \underbrace{A^TAv_i}_{=\lambda_i v_i}, v_j\rangle \\ &= \sum_{\substack{i,j=1 \\ i\neq j}}^n \mu_i\mu_j\lambda_i \underbrace{\langle v_i, v_j\rangle}_{=0} + \sum_{i=1}^n \mu_i^2\lambda_i \underbrace{\langle v_i, v_i\rangle}_{=1} = \sum_{i=1}^n \mu_i^2\lambda_i. \end{aligned}$$

Damit folgt

$$\|Ax\|_2^2 = \sum_{i=1}^n \mu_i^2\lambda_i \leq \underbrace{\sum_{i=1}^n \mu_i^2}_{=1} \lambda_{i^*} = \lambda_{i^*},$$

also, da x beliebig war, auch

$$\|A\|_2^2 \leq \lambda_{i^*} = \rho(A^TA).$$

Andererseits gilt die Ungleichung

$$\|A\|_2^2 \geq \|Av_{i^*}\|_2^2 = \langle A^TAv_{i^*}, v_{i^*}\rangle = \lambda_{i^*} \underbrace{\langle v_{i^*}, v_{i^*}\rangle}_{=1} = \lambda_{i^*} = \rho(A^TA).$$

□

Für jede Matrixnorm können wir die zugehörige *Kondition* einer invertierbaren Matrix definieren.

Definition 1.6 Für eine gegebene Matrixnorm $\|\cdot\|_p$ ist die *Kondition* einer invertierbaren Matrix A (bzgl. $\|\cdot\|_p$) definiert durch

$$\mathrm{cond}_p(A) := \|A\|_p\|A^{-1}\|_p.$$

□

Wenn wir das Verhältnis zwischen dem Fehler Δx und dem Residuum Δb betrachten, können wir entweder die *absoluten Größen* dieser Werte, also $\|\Delta x\|_p$ und $\|\Delta b\|_p$ betrachten, oder, was oft sinnvoller ist, die *relativen Größen* $\|\Delta x\|_p/\|x\|_p$ und $\|\Delta b\|_p/\|b\|_p$. Der folgende Satz zeigt, wie man den Fehler durch das Residuum abschätzen kann.

Satz 1.7 Sei $\|\cdot\|_p$ eine Vektornorm mit zugehöriger (und gleich bezeichneter) induzierter Matrixnorm. Sei $A \in \mathbb{R}^{n\times n}$ eine gegebene invertierbare Matrix und b, $\Delta b \in \mathbb{R}^n$ gegebene Vektoren. Seien x, $\tilde{x} \in \mathbb{R}^n$ die Lösungen der linearen Gleichungssysteme

$$Ax = b \quad \text{und} \quad A\tilde{x} = b + \Delta b.$$

Dann gelten für den Fehler $\Delta x = \tilde{x} - x$ die Abschätzungen

$$\|\Delta x\|_p \leq \|A^{-1}\|_p\|\Delta b\|_p \tag{1.8}$$

und

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \mathrm{cond}_p(A)\frac{\|\Delta b\|_p}{\|b\|_p}. \tag{1.9}$$

Beweis: Seien $C \in \mathbb{R}^{n\times n}$ und $y \in \mathbb{R}^n$ eine beliebige Matrix und ein beliebiger Vektor. Dann gilt nach Übungsaufgabe 6(b) (Blatt 2)

$$\|Cy\|_p \leq \|C\|_p \|y\|_p. \tag{1.10}$$

Schreiben wir $\tilde{x} = x + \Delta x$, und ziehen die Gleichung

$$Ax = b$$

von der Gleichung

$$A(x + \Delta x) = b + \Delta b$$

ab, so erhalten wir

$$A\Delta x = \Delta b.$$

Weil A invertierbar ist, können wir die Gleichung auf beiden Seiten von links mit A^{-1} multiplizieren und erhalten so

$$\Delta x = A^{-1}\Delta b.$$

Daraus folgt

$$\|\Delta x\|_p = \|A^{-1}\Delta b\|_p \leq \|A^{-1}\|_p \|\Delta b\|_p,$$

wobei wir (1.10) mit $C = A^{-1}$ und $y = \Delta b$ benutzt haben. Dies zeigt (1.8). Aus (1.10) mit $C = A$ und $y = x$ folgt

$$\|b\|_p = \|Ax\|_p \leq \|A\|_p \|x\|_p,$$

und damit

$$\frac{1}{\|x\|_p} \leq \frac{\|A\|_p}{\|b\|_p}.$$

Wenn wir erst diese Ungleichung und dann (1.8) anwenden, erhalten wir

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \frac{\|A\|_p \|\Delta x\|_p}{\|b\|_p} \leq \frac{\|A\|_p \|A^{-1}\|_p \|\Delta b\|_p}{\|b\|_p} = \mathrm{cond}_p(A) \frac{\|\Delta b\|_p}{\|b\|_p},$$

also (1.9). □

Für Matrizen, deren Kondition $\mathrm{cond}_p(A)$ groß ist, können sich kleine Fehler im Vektor b (bzw. Rundungsfehler im Verfahren) zu großen Fehlern im Ergebnis x verstärken. Man spricht in diesem Fall von *schlecht konditionierten* Matrizen. Das folgende Beispiel illustriert diesen Sachverhalt.

Beispiel 1.8 Betrachte das Gleichungssystem $Ax = b$ mit

$$A = \begin{pmatrix} 1 & 0 \\ 1000 & 1 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 0.001 \\ 1 \end{pmatrix}.$$

Durch Nachrechnen sieht man leicht, dass die Lösung gegeben ist durch $x = (0.001, 0)^T$. Ebenso überprüft man leicht, dass

$$A^{-1} = \begin{pmatrix} 1 & 0 \\ -1000 & 1 \end{pmatrix}$$

gilt. In der Zeilensummennorm gilt also $\|A\|_\infty = 1001$ und $\|A^{-1}\|_\infty = 1001$ und damit

$$\mathrm{cond}_\infty(A) = 1001 \cdot 1001 \approx 1\,000\,000.$$

Für die gestörte rechte Seite $\tilde{b} = (0.002, 1)^T$, d.h. für $\Delta b = (0.001, 0)^T$ erhält man die Lösung $\tilde{x} = (0.002, -1)^T$. Es gilt also $\Delta x = (0.001, -1)^T$ und damit

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} = \frac{1}{0.001} = 1000 \quad \text{und} \quad \frac{\|\Delta b\|_\infty}{\|b\|_\infty} = \frac{0.001}{1} = 0.001.$$

Der relative Fehler 0.001 in $\tilde{b}$ wird also um den Faktor 1 000 000 zu dem relativen Fehler 1000 in $\tilde{x}$ vergrößert, was fast exakt der Kondition $\mathrm{cond}_\infty(A)$ entspricht. □

Bei schlecht konditionierten Matrizen können sich Rundungsfehler im Algorithmus ähnlich stark wie Fehler in der rechten Seite b auswirken. Ein wichtiges Kriterium beim Entwurf eines Lösungsverfahrens ist es, dass das Verfahren auch für schlecht konditionierte Matrizen noch zuverlässig funktioniert. Beim Gauß-Verfahren kann man dazu die Auswahl der Pivotelemente so gestalten, dass möglichst wenig Rundungsfehler auftreten.

Hierzu muss man sich überlegen, welche Operationen in der Gauß-Elimination besonders fehleranfällig sind; dies kann man sehr ausführlich und formal durchführen, wir werden uns hier aber auf ein eher heuristisches Kriterium beschränken: Die Rechenoperationen in der Gauß-Elimination sind gegeben durch die Subtraktionen

$$a_{i\,k} - \frac{a_{i\,j}}{a_{j\,j}} a_{j\,k}, \quad b_i - \frac{a_{i\,j}}{a_{j\,j}} b_j$$

Im Prinzip können wir im Schritt (1a) des Algorithmus eine beliebige andere Zeile mit der j-ten Zeile vertauschen, solange diese die gleiche Anzahl von führenden Nulleinträgen besitzt, was gerade für die Zeilen $j, \ldots, n$ gilt. Dies gibt uns die Freiheit, den *Zeilen*index „j" durch einen beliebigen Index $p \in \{j, \ldots, n\}$ zu ersetzen, was im Algorithmus durch Tauschen der j-ten mit der p-ten Zeile realisiert wird. Beachte, dass das „j" in „$a_{i\,j}$" der *Spalten*index des zu eliminierenden Elementes ist, der sich durch die Vertauschung nicht ändert; wir können also durch Zeilenvertauschung nur die Elemente $a_{j\,j}$, $a_{j\,k}$ und b_j oder anders gesagt gerade die Brüche $a_{j\,k}/a_{j\,j}$ und $b_j/a_{j\,j}$ beeinflussen.

Die wesentliche Quelle für Rundungsfehler in einer Subtraktion „$c - d$" im Computer entsteht, wenn die zu berechnende Differenz betragsmäßig klein ist und die einzelnen Terme c und d im Vergleich dazu betragsmäßig groß sind. Um dies zu veranschaulichen nehmen wir an, dass wir im Dezimalsystem auf 5 Stellen genau rechnen. Wenn wir nun die Zahlen $1,234$ von der Zahl $1,235$ subtrahieren, so erhalten wir das richtige Ergebnis $0,001$, wenn wir aber die Zahl $1000,234$ von der Zahl $1000,235$ subtrahieren, so wird nach interner Rundung auf 5 Stellen die Rechnung $1000,2 - 1000,2 = 0$ ausgeführt, was zu einem deutlichen Fehler führt (dieser spezielle Fehler wird auch „Auslöschung" gennant). Die Strategie, solche Fehler in der Gauß-Elimination so weit wie möglich zu vermeiden, besteht nun darin, den Zeilenindex p bei der Pivotierung so auszuwählen, dass die zu subtrahierenden Ausdrücke betragsmäßig klein sind, ein Verfahren, das man *Pivotsuche* nennt. Da wir nur die Brüche $a_{j\,k}/a_{j\,j}$ $(k = j, \ldots, n)$ und $b_j/a_{j\,j}$ beeinflussen können, sollte man p dazu also so wählen, dass eben diese Brüche im Betrag möglichst klein werden. Eine einfache Variante,

die sich oft in der Literatur findet, besteht darin, das *Pivotelement* $a_{p\,j}$ (also den Nenner der Brüche) so zu wählen, dass $|a_{p\,j}|$ maximal wird. Im folgenden Algorithmus 1.9 verwenden wir eine etwas aufwändigere Strategie, bei der auch die Zähler der auftauchenden Brüche berücksichtigt werden.

Algorithmus 1.9 (Gauß-Elimination mit Pivotsuche)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(1) Für j von 1 bis $n-1$ (j = Spaltenindex des zu eliminierenden Eintrags)

(1a) Wähle aus den Zeilenindizes $p \in \{j,\ldots,n \,|\, a_{p\,j} \neq 0\}$ denjenigen aus, für den der Ausdruck

$$K(p) = \max\left\{ \max_{k=j,\ldots,n} \frac{|a_{p\,k}|}{|a_{p\,j}|}, \frac{|b_p|}{|a_{p\,j}|} \right\}$$

minimal wird. Falls $p \neq j$ vertausche $a_{j\,k}$ und $a_{p\,k}$ für $k = j,\ldots,n$ sowie b_p und b_j

(2) Für i von n bis $j+1$ (rückwärts zählend)
(i = Zeilenindex des zu eliminierenden Eintrags)
Subtrahiere $a_{i\,j}/a_{j\,j}$-mal die j-te Zeile von der i-ten Zeile:

Setze $\alpha := a_{i\,j}/a_{j\,j}$ und berechne

$$a_{i\,k} := a_{i\,k} - \alpha\, a_{j\,k} \text{ für } k = j,\ldots,n, \quad b_i := b_i - \alpha\, b_j$$

Ende der i-Schleife

(3) Ende der j-Schleife

□

Die hier verwendete Form der Pivotsuche wird *Spaltenpivotsuche* genannt, da in jedem Schritt innerhalb der j-ten Spalte nach dem besten Pivotelement $a_{p\,j}$ gesucht wird. Eine erweiterte Form ist die *vollständige* oder *totale Pivotsuche* bei der auch in den Zeilen gesucht wird und dann gegebenenfalls auch Spaltenvertauschungen durchgeführt werden. Gute Implementierungen der Gauß-Elimination verwenden immer solche Pivotsuchmethoden. Diese bietet eine Verbesserung der Robustheit aber keinen vollständigen Schutz gegen große Fehler bei schlechter Kondition — aus prinzipiellen mathematischen Gründen, wie wir im nächsten Abschnitt näher erläutern werden.

Eine allgemeinere Strategie zur Vermeidung schlechter Kondition ist die sogenannte *Präkonditionierung*, bei der eine Matrix $P \in \mathbb{R}^{n\times n}$ gesucht wird, für die die Kondition von PA kleiner als die von A ist, so dass dann das besser konditionierte Problem $PAx = Pb$ gelöst werden kann. Wir kommen hierauf bei der Betrachtung iterativer Verfahren zurück.

Eine weitere Strategie zur Behandlung schlecht konditionierter Gleichungssysteme, die wir nun genauer untersuchen wollen, ist die QR-Faktorisierung (oder QR-Zerlegung) einer Matrix.

1.5 QR-Faktorisierung

Die LR-Zerlegung, die explizit oder implizit Grundlage der bisher betrachteten Lösungsverfahren war, hat unter Konditions-Gesichtspunkten einen wesentlichen Nachteil: Es kann nämlich passieren, das die einzelnen Matrizen L und R der Zerlegung deutlich größere Kondition haben als die zerlegte Matrix A.

Beispiel 1.10 Betrachte die Matrix

$$A = \begin{pmatrix} 0.001 & 0.001 \\ 1 & 2 \end{pmatrix} \quad \text{mit} \quad A^{-1} = \begin{pmatrix} 2000 & -1 \\ -1000 & 1 \end{pmatrix}$$

und LR-Faktorisierung

$$L = \begin{pmatrix} 1 & 0 \\ 1000 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 0.001 & 0.001 \\ 0 & 1 \end{pmatrix}.$$

Wegen

$$L^{-1} = \begin{pmatrix} 1 & 0 \\ -1000 & 1 \end{pmatrix} \quad \text{und} \quad R^{-1} = \begin{pmatrix} 1000 & -1 \\ 0 & 1 \end{pmatrix}$$

gilt

$$\text{cond}_\infty(A) \approx 6000, \ \text{cond}_\infty(L) \approx 1000000 \ \text{und} \ \text{cond}_\infty(R) \approx 1000.$$

Die ∞-Kondition von L ist also etwa 166-mal so groß wie die von A. □

Selbst wenn wir eventuelle Fehler in der Berechnung von R und L vernachlässigen oder z.B. durch geschickte Pivotsuche vermindern, kann die schlechte Konditionierung von R und L durch die Verstärkung der beim Rückwärts- und Vorwärtseinsetzen auftretenden Rundungsfehler zu großen Fehlern Δx führen, besonders wenn die Matrix A selbst bereits schlecht konditioniert ist. Bei der LR-Faktorisierung kann es also passieren, dass die Kondition der Teilprobleme, die im Algorithmus auftreten, deutlich schlechter ist als die des Ausgangsproblems. Beachte, dass die Kondition des Ausgangsproblems nur von der Problemstellung abhängt, die der Teilprobleme aber von dem verwendeten Algorithmus, weswegen diese auch als *numerische Kondition* bezeichnet werden.

Um die numerische Kondition zu verringern, wollen wir nun eine andere Form der Zerlegung betrachten, bei der die Kondition der einzelnen Teilprobleme (bzw. der zugehörigen Matrizen) nicht größer ist als die des Ausgangsproblems (also der Ausgangsmatrix A).

Hierzu verwenden wir die folgenden Matrizen.

Definition 1.11 Eine Matrix $Q \in \mathbb{R}^{n \times n}$ heißt *orthogonal*, falls $QQ^T = \text{Id}$ ist[2]. □

Unser Ziel ist nun ein Algorithmus, mit dem eine Matrix A in ein Produkt QR zerlegt wird, wobei Q eine orthogonale Matrix ist und R eine obere Dreiecksmatrix.

[2]Das komplexe Gegenstück hierzu sind die unitären Matrizen, mit denen sich all das, was wir hier im Reellen machen, auch im Komplexen durchführen lässt

Offenbar ist ein Gleichungssystem der Form $Qy = b$ leicht zu lösen, indem man die Matrixmultiplikation $y = Q^T b$ durchführt. Deswegen kann man das Gleichungssystem $Ax = b$ wie bei der LR-Faktorisierung leicht durch Lösen der Teilsysteme $Qy = b$ und $Rx = y$ lösen.

Bevor wir den entsprechenden Algorithmus herleiten, wollen wir beweisen, dass bei dieser Form der Zerlegung die Kondition tatsächlich erhalten bleibt — zumindest für die euklidische Norm.

Satz 1.12 Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Matrix mit QR-Zerlegung. Dann gilt

$$\mathrm{cond}_2(Q) = 1 \ \text{ und } \ \mathrm{cond}_2(R) = \mathrm{cond}_2(A).$$

Beweis: Da Q orthogonal ist, gilt $Q^{-1} = Q^T$. Daraus folgt für beliebige Vektoren $x \in \mathbb{R}^n$

$$\|Qx\|_2^2 = \langle Qx, Qx\rangle_2 = \langle Q^T Qx, x\rangle_2 = \langle x, x\rangle_2 = \|x\|_2^2,$$

also auch $\|Qx\|_2 = \|x\|_2$. Damit folgt für invertierbare Matrizen $B \in \mathbb{R}^{n\times n}$

$$\|QB\|_2 = \max_{\|x\|_2=1} \|QBx\|_2 = \max_{\|x\|_2=1} \|Q(Bx)\|_2 = \max_{\|x\|_2=1} \|Bx\|_2 = \|B\|_2$$

und mit $Qx = y$ auch

$$\|BQ\|_2 = \max_{\|x\|_2=1} \|BQx\|_2 = \max_{\|Q^T y\|_2=1} \|By\|_2 = \max_{\|y\|_2=1} \|By\|_2 = \|B\|_2,$$

da mit Q auch $Q^T = Q^{-1}$ orthogonal ist. Also folgt

$$\mathrm{cond}_2(Q) = \|Q\|_2\|Q^{-1}\|_2 = \|Q\mathrm{Id}\|_2\|Q^{-1}\mathrm{Id}\|_2 = \|\mathrm{Id}\|_2\|\mathrm{Id}\|_2 = 1$$

und

$$\begin{aligned}\mathrm{cond}_2(R) &= \mathrm{cond}_2(Q^T A) = \|Q^T A\|_2\|(Q^T A)^{-1}\|_2 = \|Q^T A\|_2\|A^{-1}Q\|_2 \\ &= \|A\|_2\|A^{-1}\|_2 = \mathrm{cond}_2(A).\end{aligned}$$

□

Zwar gilt dieser Satz für andere Matrixnormen nicht, da für je zwei Vektornormen aber Abschätzungen der Form $\|x\|_p \le C_{p,q}\|x\|_q$ gelten, ist zumindest eine extreme Verschlechterung der numerischen Kondition auch bezüglich anderer induzierter Matrixnormen ausgeschlossen.

Beispiel 1.13 Als Beispiel betrachten wir noch einmal das Gleichungssystem aus Beispiel 1.10. Eine QR-Zerlegung von A ist gegeben durch

$$Q = \begin{pmatrix} -0.001 & -1 \\ -1 & 0.001 \end{pmatrix} \quad \text{und} \quad R = \begin{pmatrix} -1 & -2 \\ 0 & 0.001 \end{pmatrix}$$

mit

$$Q^{-1} = \begin{pmatrix} -0.001 & -1 \\ -1 & 0.001 \end{pmatrix} \quad \text{und} \quad R^{-1} = \begin{pmatrix} -1 & -2000 \\ 0 & 1000 \end{pmatrix}.$$

Hier gilt also $\mathrm{cond}_\infty(Q) \approx 1$ und $\mathrm{cond}_\infty(R) \approx 6000$, d.h. es entsteht auch bzgl. der ∞-Kondition keine wesentliche Verschlechterung der Kondition. □

Die Idee der Algorithmen zur QR-Zerlegung liegt nun darin, die Spalten der Matrix A als Vektoren aufzufassen und durch orthogonale Transformationen auf die gewünschte Form zu bringen. Orthogonale Transformationen sind dabei gerade die linearen Transformationen, die sich durch orthogonale Matrizen darstellen lassen. Geometrisch sind dies die Transformationen, die die (euklidische) Länge des transformierten Vektors sowie den Winkel zwischen zwei Vektoren unverändert lassen — nichts anderes haben wir im Beweis von Satz 1.12 ausgenutzt.

Zur Realisierung eines solchen Algorithmus bieten sich zwei mögliche Transformationen an: Drehungen und Spiegelungen. Wir wollen hier den nach seinem Erfinder benannten *Householder-Algorithmus* herleiten, der auf Basis von Spiegelungen funktioniert[3]. Wir veranschaulichen die Idee zunächst geometrisch: Sei $a_{\cdot j} \in \mathbb{R}^n$ die j-te Spalte der Matrix A. Wir wollen eine Spiegelung $H^{(j)}$ finden, die $a_{\cdot j}$ auf einen Vektor der Form $a_{\cdot j}^{(j)} = (\underbrace{*, *, \ldots, *}_{j \text{ Stellen}}, 0)^T$ bringt. Der Vektor soll also in die Ebene $E_j = \operatorname{span}(e_1, \ldots, e_j)$ gespiegelt werden.

Um diese Spiegelung zu konstruieren, betrachten wir allgemeine Spiegelmatrizen der Form

$$H = H(v) = \mathrm{Id} - \frac{2vv^T}{v^T v}$$

wobei $v \in \mathbb{R}^n$ ein beliebiger Vektor ist (beachte, dass dann $vv^T \in \mathbb{R}^{n\times n}$ und $v^T v \in \mathbb{R}$ ist). Diese Matrizen heißen nach dem Erfinder des zugehörigen Algorithmus *Householder-Matrizen.* Offenbar ist H symmetrisch und es gilt

$$HH^T = H^2 = \mathrm{Id} - \frac{4vv^T}{v^T v} + \frac{2vv^T}{v^T v}\frac{2vv^T}{v^T v} = \mathrm{Id},$$

also Orthogonalität. Geometrisch entspricht die Multiplikation mit H der Spiegelung an der Ebene mit Normalenvektor $n = v/\|v\|$.

Um nun die gewünschte Spiegelung in die Ebene E_j zu realisieren, muss man v geeignet wählen. Hierbei hilft das folgende Lemma.

Lemma 1.14 Betrachte einen Vektor $w = (w_1, \ldots, w_n)^T \in \mathbb{R}^n$. Für einen gegebenen Index $j \in \{1, \ldots, n\}$ betrachte

$$\begin{aligned} c &= \operatorname{sgn}(w_j)\sqrt{w_j^2 + w_{j+1}^2 + \cdots + w_n^2} \in \mathbb{R} \\ v &= (0, \ldots, 0, c + w_j, w_{j+1}, \ldots, w_n)^T \\ H &= \mathrm{Id} - \frac{2vv^T}{v^T v} \end{aligned}$$

mit den Konventionen $\operatorname{sgn}(a) = 1$, falls $a \geq 0$, $\operatorname{sgn}(a) = -1$, falls $a < 0$ und $H = \mathrm{Id}$ falls $v = 0$. Dann gilt

$$Hw = (w_1, w_2, \ldots, w_{j-1}, -c, 0, \ldots, 0).$$

Darüberhinaus gilt für jeden Vektor $z \in \mathbb{R}^n$ der Form $z = (z_1, \ldots, z_{j-1}, 0, \ldots, 0)$ die Gleichung $Hz = z$.

[3] ein Algorithmus auf Basis von Drehungen ist der sogenannte Givens-Algorithmus

Beweis: Falls $v \neq 0$ ist gilt

$$\begin{aligned} \frac{2v^T w}{v^T v} &= \frac{2((c+w_j)w_j + w_{j+1}^2 + \cdots + w_n^2)}{c^2 + 2cw_j + w_j^2 + w_{j+1}^2 + \cdots + w_n^2} \\ &= \frac{2(cw_j + w_j^2 + w_{j+1}^2 + \cdots + w_n^2)}{2cw_j + 2w_j^2 + 2w_{j+1}^2 + \cdots + 2w_n^2} = 1. \end{aligned}$$

Hieraus folgt

$$Hw = w - v\frac{2v^T w}{v^T v} = w - v = \begin{pmatrix} w_1 \\ \vdots \\ w_{j-1} \\ w_j \\ w_{j+1} \\ \vdots \\ w_n \end{pmatrix} - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c + w_j \\ w_{j+1} \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} w_1 \\ \vdots \\ w_{j-1} \\ -c \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Falls $v = 0$ ist, sieht man sofort, dass $w_{j+1} = \ldots = w_n = 0$ gelten muss. Damit folgt $c = w_j$, also $w_j + c = 2w_j$, weswegen auch $w_j = 0$ sein muss. In diesem Fall gilt also bereits $w_j = w_{j+1} = \ldots = w_n = 0$, so dass die Behauptung mit $H = \mathrm{Id}$ gilt.

Für die zweite Behauptung verwenden wir, dass für Vektoren z der angegebenen Form die Gleichung $v^T z = 0$ gilt, woraus sofort

$$Hz = z - v\frac{2v^T z}{v^T v} = z,$$

also die Behauptung folgt. □

Die Idee des Algorithmus liegt nun nahe:

Wir setzen $A^{(1)} = A$ und konstruieren im ersten Schritt $H^{(1)}$ gemäß Lemma 1.14 mit $j = 1$ und $w = a^{(1)}_{\cdot 1}$, der ersten Spalte der Matrix $A^{(1)}$. Damit ist dann $A^{(2)} = H^{(1)} A^{(1)}$ von der Form

$$A^{(2)} = H^{(1)} A^{(1)} = \begin{pmatrix} a^{(2)}_{1\,1} & a^{(2)}_{1\,2} & \cdots & a^{(2)}_{1\,n} \\ 0 & a^{(2)}_{2\,2} & \cdots & a^{(2)}_{2\,n} \\ \vdots & \vdots & & \vdots \\ 0 & a^{(2)}_{n\,2} & \cdots & a^{(2)}_{n\,n} \end{pmatrix}.$$

Im zweiten Schritt konstruieren wir $H^{(2)}$ gemäß Lemma 1.14 mit $j = 2$ und $w = a^{(2)}_{\cdot 2}$, der zweiten Spalte der Matrix $A^{(2)}$. Da die erste Spalte der Matrix $A^{(2)}$ die Voraussetzungen an den Vektor z in Lemma 1.14 erfüllt, folgt die Form

$$A^{(3)} = H^{(2)} A^{(2)} = \begin{pmatrix} a^{(3)}_{1\,1} & a^{(3)}_{1\,2} & a^{(3)}_{1\,3} & \cdots & a^{(3)}_{1\,n} \\ 0 & a^{(3)}_{2\,2} & a^{(3)}_{2\,3} & \cdots & a^{(3)}_{2\,n} \\ 0 & 0 & a^{(3)}_{3\,3} & \cdots & a^{(3)}_{3\,n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a^{(3)}_{n\,3} & \cdots & a^{(3)}_{n\,n} \end{pmatrix}.$$

Wenn wir sukzessive fortfahren, erhalten wir nach $n-1$ Schritten die gewünschte QR-Zerlegung mit

$$Q^T = H^{(n-1)} \cdots H^{(1)} \quad \text{und} \quad R = A^{(n)},$$

denn es gilt

$$\begin{aligned} QR &= {H^{(1)}}^T \cdots {H^{(n-1)}}^T A^{(n)} \\ &= {H^{(1)}}^T \cdots {H^{(n-2)}}^T A^{(n-1)} \\ &\vdots \\ &= {H^{(1)}}^T A^{(2)} \\ &= A^{(1)} = A \end{aligned}$$

Insgesamt ergibt sich der folgende Ablaufplan für die QR-Zerlegung:[4]

Algorithmus 1.15 [QR-Faktorisierung]
Gegeben: Matrix $A \in \mathbb{R}^{n\times n}$

Setze $A^{(1)} = A$ und $Q^T = Id$.
For j von 1 bis $n-1$ (Spaltenindex)

(1) Konstruiere $H^{(j)}$ gemäß Lemma 1.14 mit $w = a^{(j)}_{\star j}$ (j-te Spalte von $A^{(j)}$)

(2) Setze $A^{(j+1)} := H^{(j)} A^{(j)}$

(3) Setze $Q^T := H^{(j)} Q^T$

Ende der j-Schleife

Ausgabe: Matrix $R := A^{(n+1)}$ in oberer Dreiecksform und Matrix $Q^T \in \mathbb{R}^{n\times n}$ □

Beachte, dass die QR-Faktorisierung *immer* funktioniert, auch wenn A nicht invertierbar ist, denn die obigen Überlegungen liefern einen konstruktiven Beweis für die Existenz. Die resultierende Matrix Q ist immer invertierbar, die Matrix R ist invertierbar genau dann, wenn A invertierbar ist.

Bei der Lösung eines linearen Gleichungssystems sollte die Invertierbarkeit von R vor dem Rückwärtseinsetzen getestet werden (eine obere Dreiecksmatrix ist genau dann invertierbar, wenn alle Diagonalelemente ungleich Null sind). Dies kann schon im obigen Algorithmus geschehen, indem überprüft wird, ob die c-Werte (die gerade die Diagonalelemente von R bilden) ungleich Null sind.

Die QR-Zerlegung kann tatsächlich mehr als nur lineare Gleichungssysteme lösen: Wir haben im Abschnitt 1.1 das lineare Ausgleichsproblem kennen gelernt, bei dem $x \in \mathbb{R}^n$ gesucht ist, so dass der Vektor

$$r = z - \widetilde{A}x$$

minimale 2-Norm $\|r\|_2$ hat, und haben gesehen, dass dieses Problem durch Lösen der Normalengleichungen $\widetilde{A}^T\widetilde{A}x = \widetilde{A}^T z$ gelöst werden kann. Gerade diese Matrix $\widetilde{A}^T\widetilde{A}$ kann

[4]Für eine **Implementierung** verweisen wir auf das [3, Algorithmus 2.15] verwiesen.

aber (bedingt durch ihre Struktur, vgl. Aufgabe 10, Übungsblatt 4) sehr große Kondition haben, so dass es hier erstrebenswert ist, (a) ein robustes Verfahren zu verwenden und (b) die explizite Lösung der Normalengleichungen zu vermeiden. Mit dem QR-Algorithmus kann man beides erreichen, man kann nämlich das Ausgleichsproblem *direkt* lösen.

Die QR-Zerlegung (und auch der obige Algorithmus) kann auf die nichtquadratische Matrix $\widetilde{A}$ mit n Spalten und $m > n$ Zeilen angewendet werden, indem man j in Schritt (1) bis n und alle anderen Indizes mit Ausnahme von k in der Berechnung von $H^{(j)}A^{(j)}$ bis m laufen lässt. Das Resultat ist dann eine Faktorisierung $\widetilde{A} = QR$ mit

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

wobei $R_1 \in \mathbb{R}^{n\times n}$ eine obere Dreiecksmatrix ist. Beachte, dass die Normalengleichungen genau dann lösbar sind, wenn $\widetilde{A}$ vollen Spaltenrang besitzt, was wir annehmen. In diesem Fall ist auch die Matrix R_1 invertierbar.

Wenn man dann x so wählt, dass der Vektor

$$s = Q^T r = Q^T z - Q^T \widetilde{A} x$$

minimale 2-Norm besitzt, dann hat auch r minimale 2-Norm, da aus der Orthogonalität von Q^T die Gleichung $\|r\|_2 = \|s\|_2$ folgt. Wir zerlegen s in $s^1 = (s_1, \ldots, s_n)^T$ und $s^2 = (s_{n+1}, \ldots, s_m)^T$. Wegen der Form von $R = Q^T \widetilde{A}$ ist der Vektor s^2 unabhängig von x und wegen

$$\|s\|_2^2 = \sum_{i=1}^{m} s_i^2 = \sum_{i=1}^{n} s_i^2 + \sum_{i=n+1}^{m} s_i^2 = \|s^1\|_2^2 + \|s^2\|_2^2$$

wird diese Norm genau dann miminal, wenn die Norm $\|s^1\|_2^2$ minimal wird. Wir suchen also ein $x \in \mathbb{R}^n$, so dass

$$\|s^1\|_2 = \|y^1 - R_1 x\|_2$$

minimal wird, wobei y^1 die ersten n Komponenten des Vektors $y = Q^T z$ bezeichnet. Da R_1 eine invertierbare obere Dreiecksmatrix ist, kann man durch Rückwärtseinsetzen eine Lösung x des Gleichungssystems $R_1 x = y^1$ finden, für die dann

$$\|s^1\|_2 = \|y^1 - R_1 x\|_2 = 0$$

gilt, womit offenbar das Minimum erreicht wird. Zusammenfassend kann man also das Ausgleichsproblem wie folgt mit der QR-Faktorisierung direkt lösen:

Algorithmus 1.16 (Lösung des Ausgleichsproblems mit QR-Faktorisierung)
Eingabe: Matrix $\widetilde{A} \in \mathbb{R}^{m\times n}$ mit $m > n$ und (maximalem) Spaltenrang n, Vektor $z \in \mathbb{R}^m$

(1) Berechne Zerlegung $\widetilde{A} = QR$ mit $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$ und oberer Dreiecksmatrix $R_1 \in \mathbb{R}^{n\times n}$

(2) Löse das Gleichungssystem $R_1 x = y^1$ durch Rückwärtseinsetzen, wobei y^1 die ersten n Komponenten des Vektors $y = Q^T z \in \mathbb{R}^m$ bezeichnet

Ausgabe: Vektor $x \in \mathbb{R}^n$, für den $\|\widetilde{A}x - z\|_2$ minimal wird. □

Geometrisch passiert hier das Folgende: Das Bild von $\widetilde{A}$ wird durch die orthogonale Transformation Q^T längentreu auf den Unterraum $\operatorname{span}(e_1, \dots, e_n)$ abgebildet, in dem wir dann das entstehende Gleichungssystem lösen können, vgl. Abbildung 1.1.

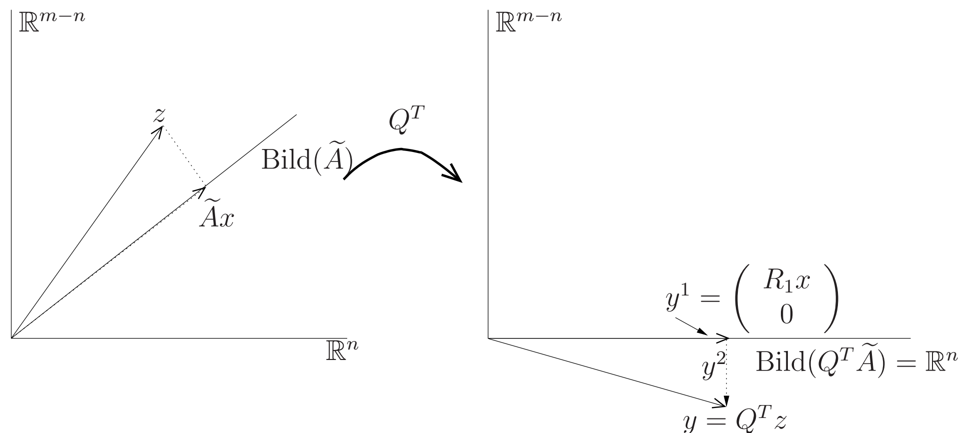

Abbildung 1.1: Veranschaulichung von Algorithmus 1.16

1.6 Aufwandsabschätzungen

Ein wichtiger Aspekt bei der Analyse numerischer Verfahren ist es zu untersuchen, wie lange diese Verfahren in der Regel benötigen, um zu dem gewünschten Ergebnis zu kommen. Da dies natürlich entscheidend von der Leistungsfähigkeit des verwendeten Computers abhängt, schätzt man nicht direkt die Zeit ab, sondern die Anzahl der Rechenoperationen, die ein Algorithmus benötigt. Da hierbei die *Gleitkommaoperationen*, also Addition, Multiplikation etc. von reellen Zahlen, die mit Abstand zeitintensivsten Operationen sind, beschränkt man sich in der Analyse üblicherweise auf diese[5].

Die Verfahren, die wir bisher betrachtet haben, liefern nach endlich vielen Schritten ein Ergebnis (man spricht von *direkten Verfahren*), wobei die Anzahl der Operationen von der Dimension n der Matrix abhängt. Zur *Aufwandsabschätzung* genügt es also, die Anzahl der Gleitkommaoperationen (in Abhängigkeit von n) „abzuzählen“. Wie man dies am geschicktesten macht, hängt dabei von der Struktur des Algorithmus ab. Zudem muss man einige Rechenregeln aus der elementaren Analysis ausnutzen, um die entstehenden Ausdrücke zu vereinfachen. Speziell benötigen wir hier die Gleichungen

$$\sum_{i=1}^{n} i = \frac{(n+1)n}{2} \quad \text{und} \quad \sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$

[5]Tatsächlich sind Multiplikation, Division und die Berechnung von Wurzeln etwas aufwändiger als Addition und Subtraktion, was wir hier aber vernachlässigen werden.

Wir beginnen mit dem **Rückwärtseinsetzen**, und betrachten zunächst die Multiplikationen und Divisionen: Für $i = n$ muss eine Division durchgeführt werden, für $i = n-1$ muss eine Multiplikation und eine Division durchgeführt werden, für $i = n-2$ müssen zwei Multiplikationen und eine Division durchgeführt werden, usw. So ergibt sich die Anzahl dieser Operationen als

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \frac{n^2}{2} + \frac{n}{2}.$$

Für die Anzahl der Additionen und Subtraktionen zählt man ab

$$0 + 1 + 2 + \cdots + n - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

Insgesamt kommt man also auf

$$\frac{n^2}{2} + \frac{n}{2} + \frac{n^2}{2} - \frac{n}{2} = n^2$$

Gleitkommaoperationen. Da das **Vorwärtseinsetzen** völlig analog funktioniert, gilt dafür die gleiche Abschätzung.

Bei der **Gauß-Elimination** betrachten wir hier die Version aus Algorithmus 1.3 ohne Pivotsuche und nehmen den schlechtesten Fall an, nämlich dass Schritt (2) jedes Mal durchgeführt wird. Wir gehen spaltenweise vor und betrachten die Elemente, die für jedes j eliminiert werden müssen. Für jedes zu eliminierende Element in der j-ten Spalte benötigt man je $n - (j-1) + 1$ Additionen und Multiplikationen (die „+1" ergibt sich aus der Operation für b) sowie eine Division zur Berechnung von α, d.h., $2(n+2-j)+1$ Operationen. In der j-ten Spalte müssen dabei $n-j$ Einträge eliminiert werden, nämlich für $i = n, \ldots, j+1$. Also ergeben sich für die j-te Spalte

$$(n-j)(2(n+2-j)+1) = 2n^2+4n-2nj-2jn-4j+2j^2+n-j = 2j^2-(4n+5)j+5n+2n^2$$

Operationen. Dies muss für die Spalten $j = 1, \ldots, n-1$ durchgeführt werden, womit wir auf

$$\begin{aligned}
&\sum_{j=1}^{n-1} \left(2j^2 - (4n+5)j + 5n + 2n^2\right) \\
&= 2\sum_{j=1}^{n-1} j^2 - (4n+5)\sum_{j=1}^{n-1} j + \sum_{j=1}^{n-1} \left(5n + 2n^2\right) \\
&= \frac{2}{3}(n-1)^3 + (n-1)^2 + \frac{1}{3}(n-1) - (4n+5)\frac{(n-1)n}{2} + (n-1)(5n+2n^2) \\
&= \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{13}{6}n
\end{aligned}$$

Operationen kommen.

Zur vollständigen **Lösung eines linearen Gleichungssystems** müssen wir nun einfach die Operationen der Teilalgorithmen aufaddieren.

Für den Gauß-Algorithmus kommt man so auf

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{13}{6}n + n^2 = \frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{13}{6}n$$

Operationen. Analoge Überlegungen führen für die QR-Zerlegung [3, Algorithmus 2.15] auf

$$\frac{4}{3}n^3 + 3n^2 + \frac{14}{3}n - 9 + n^2 = \frac{4}{3}n^3 + 4n^2 + \frac{14}{3}n - 9$$

Operationen. Berücksichtigt man, dass für große n die führenden "n^3-Terme" dominant werden, so ergibt sich, dass die Gauß-Elimination etwa 2-mal so schnell ist wie die QR-Faktorisierung.

Um einen Eindruck von den tatsächlichen Rechenzeiten zu bekommen, nehmen wir an, dass wir einen PC verwenden, der mit einem Core i7 Prozessor etwa eine Leistung von 33 GFLOPS (FLOPS = floating point operations per second; GFLOPS = GigaFLOPS = 10^9 Flops) schafft. Nehmen wir weiterhin (sehr optimistisch) an, dass wir Implementierungen der obigen Algorithmen haben, die diese Leistung optimal ausnutzen. Dann ergeben sich für $n \times n$ Gleichungssysteme die folgenden Rechenzeiten

n	Gauß	QR
1000	20.25 ms	40.49 ms
10000	20.21 s	40.41 s
100000	5.61 h	11.22 h
500000	29.23 d	58.46 d

Spätestens im letzten Fall $n = 500\,000$ sind die Zeiten kaum mehr akzeptabel: Wer will schon mehr als vier Wochen auf ein Ergebnis warten?

Zum Abschluss dieses Abschnitts wollen wir noch ein gröberes Konzept der Aufwandsabschätzung einführen, das für praktische Zwecke oft ausreicht. Oft ist man nämlich nicht an der exakten Zahl der Operationen für ein gegebenes n interessiert, sondern nur an einer Abschätzung für große Dimensionen. Genauer möchte man wissen, wie schnell der Aufwand in Abhängigkeit von n wächst, d.h. wie er sich *asymptotisch* für $n \to \infty$ verhält. Man spricht dann von der *Ordnung* eines Algorithmus.

Definition 1.17 Ein Algorithmus hat die *Ordnung* $O(n^q)$, wenn $q > 0$ die minimale Zahl ist, für die es eine Konstante $C > 0$ gibt, so dass der Algorithmus für alle $n \in \mathbb{N}$ weniger als Cn^q Operationen benötigt. □

Diese Zahl q ist aus den obigen Aufwandsberechnungen leicht abzulesen: Es ist gerade die höchste auftretende Potenz von n. Somit haben Vorwärts- und Rückwärtseinsetzen die Ordnung $O(n^2)$, während Gauß- und QR-Verfahren die Ordnung $O(n^3)$ besitzen.

Bemerkung 1.18 [Iterative Verfahren] Wir haben im letzten Abschnitt gesehen, dass die bisher betrachteten Verfahren — die sogenannten *direkten Verfahren* — die Ordnung $O(n^3)$ besitzen: Wenn sich also n verzehnfacht, so vertausendfacht sich die Anzahl der Operationen und damit die Rechenzeit. Für große Gleichungssysteme mit mehreren 100 000 Unbekannten, die in der Praxis durchaus auftreten, führt dies wie oben gesehen zu unakzeptabel hohen Rechenzeiten.

Eine Klasse von Verfahren, die eine niedrigere Ordnung hat, sind die *iterativen Verfahren*. Allerdings zahlt man für den geringeren Aufwand einen Preis: Man kann bei diesen Verfahren nicht mehr erwarten, eine (bis auf Rundungsfehler) exakte Lösung zu erhalten, sondern muss von vornherein eine gewisse Ungenauigkeit im Ergebnis in Kauf nehmen.

Das Grundprinzip iterativer Verfahren funktioniert dabei wie folgt:

Ausgehend von einem Startvektor $x^{(0)}$ berechnet man mittels einer Rechenvorschrift $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ iterativ eine Folge von Vektoren

$$x^{(i+1)} = \Phi(x^{(i)}), \qquad i = 0, 1, 2, \ldots,$$

die für $i \to \infty$ gegen die Lösung x^* des Gleichungssystems $Ax = b$ konvergieren, also $\lim_{i\to\infty} \|x^{(i)} - x^*\|_p = 0$. Wenn die gewünschte Genauigkeit erreicht ist, wird die Iteration abgebrochen und der letzte Wert $x^{(i)}$ als Näherung des Ergebnisses verwendet. □