1 Nichtlineare Gleichungen in einer Unbekannten

1.1 Ein kurzer Rundgang im Garten der Gleichungen

Als Einstieg in die Numerische Mathematik behandeln wir numerische Lösungsverfahren für Gleichungen in einer Unbekannten. *Linear* sind solche Gleichungen, wenn sie sich in der Form

$$kx = d, \quad k, d \text{ gegeben}, \quad x \text{ gesucht}$$

schreiben lassen. Offensichtlich gibt es, falls $k \neq 0$, eine eindeutige Lösung. Dieses Thema ist also vorläufig abgehakt, wir kümmern uns nun um *nichtlineare* Gleichungen. (Die linearen Gleichungen werden uns erst dann intensiver beschäftigen, wenn sie in Massen, als Systemen mit *mehreren* Unbekannten auftreten.)

Analytische oder numerische Lösung Wenn sich durch algebraische Umformungen die Lösung einer Gleichung explizit, also in der Form $x = \ldots$, anschreiben lässt (im obigen Beispiel: $x = d/k$, allgemein ein Term, in dem nur die üblichen Standard-Rechenoperationen und -Funktionen auftreten), spricht man von einer *analytischen* Lösung

Analytisch lösbar sind beispielsweise *quadratische* Gleichungen, also solche, die sich als

$$x^2 + px + q = 0 \quad p, q \text{ gegeben}, \quad x \text{ gesucht}$$

schreiben lassen. Sie kennen sicherlich die Lösungsformel

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

> Es reicht aber nicht, eine Lösungsformel hinschreiben zu können, sie muss auch verlässlich genaue Ergebnisse liefern. Die scheinbar triviale Lösung einer quadratischen Gleichung nach obiger Formel kann recht ungenau werden. Lassen Sie Ihren Taschenrechner damit die kleinere Lösung der quadratischen Gleichung
>
> $$x^2 - 12345678x + 9 = 0$$
>
> berechnen. Der (sechzehnstellig) genaue Wert ist $x_1 = 7{,}290\,000\,597\,780\,479 \times 10^{-7}$. Obwohl übliche Rechner zehn- bis vierzehnstellig genau rechnen, liefern sie nur die ersten paar Stellen richtig. Die numerisch genauere Methode berechnet zuerst die *betragsmäßig größere* Lösung x_1 mit der klassischen Formel und findet dann die *betragsmäßig kleinere* Lösung x_2 mit der alternativen Lösungsformel
>
> $$x_2 = \frac{q}{x_1}.$$

Algebraische und transzendente Gleichungen Lineare, quadratische und kubische Gleichungen sind die einfachsten Beispiele *polynomialer* Gleichungen. Ein Polynom in einer Variablen x ist eine Summe von x-Potenzen, multipliziert mit Koeffizienten, also ein Ausdruck der Form

$$a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0.$$

Die höchste auftretende Potenz heißt die *Ordnung* oder der *Grad* des Polynoms oder der Gleichung.

Kubische Gleichungen und Gleichungen vierter Ordnung sind im Prinzip analytisch lösbar, aber die Formeln (Cardanische Formeln, N. Tartaglia[1], G. Cardano[2], L. Ferrari[3], um 1540) sind so unhandlich, dass sie praktisch kaum verwendet werden. Numerische Verfahren sind in diesen Fällen meist sinnvoller. Sie liefern Näherungen, die schrittweise, mit immer besserer Genauigkeit, die Lösungen anstreben. Ab Polynomgrad fünf gibt es ohnehin keine allgemeinen Lösungsformeln mehr.

> Der junge norwegische Mathematiker Niels Henrik Abel führt 1826 den „Beweis der Unmöglichkeit, algebraische Gleichungen von höheren Graden als dem vierten allgemein aufzulösen“. Ab dem fünften Grad lassen sich Gleichungen also (im Allgemeinen) nicht durch eine *endliche Zahl elementarer Rechenoperationen* (Addition, Subtraktion, Multiplikation, Division, Wurzelziehen) lösen.

Um die Vorstellung der verschiedenen Gleichungstypen zum Abschluss zu bringen: Gleichungen, in denen auch noch Bruchterme, Wurzeln oder rationale Exponenten vorkommen, lassen sich (möglicher Weise nur mit hohem Aufwand und komplizierten Umformungen) auf Systeme polynomialer (man sagt auch: algebraischer) Gleichungen zurückführen. Terme oder Funktionen, die sich nicht mittels endlich vieler elementarer Rechenoperationen formulieren lassen, sind etwas, das die Kräfte der Algebra übersteigt (*„quod vires algebrae transcendit“*, sagte Leibnitz) und heißen deswegen *transzendent*.

Beispielsweise sind die trigonometrischen Funktionen, die Exponentialfunktion und die entsprechenden Umkehrfunktionen transzendente Funktionen. Treten solche Funktionen in Gleichungen auf, ist normaler Weise nur numerische Lösung möglich.

> Explizite Lösungsformeln gibt es nur für polynomiale Gleichungen niedrigen Grades und die allereinfachsten transzendenten Glaichungen. In allen anderen Fällen können nur numerische Methoden eine Lösung finden.

1.2 Begriffe, Probleme, Lösungen

Hier behandelte Aufgabentypen:

$$\begin{aligned} g(x) &= h(x), && \text{Finden einer } \textit{Lösung} \text{ einer Gleichung} \\ f(x) &= 0, && \text{Finden einer } \textit{Nullstelle} \text{ der Funktion } f \\ x &= \phi(x), && \text{Finden eines } \textit{Fixpunktes} \text{ der Funktion } \phi \end{aligned}$$

> Eine Lösung der Gleichung $f(x) = 0$ heißt *Nullstelle* der Funktion f.
> Eine Lösung der Gleichung $x = \phi(x)$ heißt *Fixpunkt* der Funktion ϕ.

> Eine Fixpunkt-Gleichung $x = \phi(x)$ lässt sich natürlich sofort umformen auf $\phi(x) - x = 0$. Jeder Fixpunkt von ϕ ist also zugleich Nullstelle von $f(x) = \phi(x) - x$.
> Selbstverständlich muss der Funktionsterm in einer Nullstellen-Aufgabe nicht automatisch f heißen, ebensowenig wie in Fixpunkt-Gleichungen die Funktion mit ϕ bezeichnet sein muss. Die Vorlesungsunterlagen schreiben aber in der Regel $x = \phi(x)$, wenn diese Gleichung durch Umformen aus $f(x) = 0$ entstanden ist.

[1] Niccolò Fontana Tartaglia verriet Cardano die Lösung unter dem Siegel der Verschwiegenheit; war stinksauer, als der sie trotzdem veröffentlichte.
[2] auch bekannt durch Kardanwelle und kardanische Aufhängung, die er ebenfalls nicht erfunden hat
[3] Das Rennen um die Lösung für Gleichungen vierten Grades, sozusagen die Formel Vier, wurde damals von Ferrari gewonnen.

Nullstellen von Polynomen nennt man auch *Wurzeln*.[4]

Eine *analytische Lösung* ist ein expliziter Ausdruck, in dem nur bekannte Größen und Funktionen vorkommen.

> Welche Funktionen dabei als „bekannt“ vorausgesetzt werden, ist nicht exakt festgelegt. Letztlich lassen sich auch von so geläufigen Funktionen wie Sinus oder Cosinus Werte nur durch numerische Verfahren berechnen – auch wenn Ihnen der Taschenrechner diese Arbeit abnimmt.

Demgegenüber steht die *numerische Lösung*, eine Rechenvorschrift, die eine schon irgendwie bekannte Näherung schrittweise verbessert.

Mehrfache Nullstellen: Eine Funktion f hat an der Stelle x eine genau n-fache Nullstelle, wenn zugleich $f(x) = 0, f'(x) = 0, f''(x) = 0, \ldots, f^{(n-1)}(x) = 0$ und $f^{(n)}(x) \neq 0$. (Dabei setzen wir die Existenz stetiger Ableitungen mindestens bis zur n-ten Ordnung voraus.)

Die auftretenden Funktionen $f, g, \ldots$ und Variablen $x, y, \ldots$ bezeichnen in dieser Vorlesung in der Regel *reelle* Größen. Die *komplexen* Zahlen sind an sich der natürliche Lebensraumfür Polynome und Funktionen (unter anderem deswegen, weil Polynome n-ten Grades dort immer genau n Nullstellen haben–wobei manche mehrfach gezählt werden, damit das so stimmt; Fundamentalsatz der Algebra). Die meisten Definitionen und Verfahren lassen sich leicht für komplexe Variable und komplexwertige Funktionen verallgemeinern. Trotzdem beschränken wir uns (abgesehen von gelegentlichen Hinweisen) auf Rechenverfahren in den reellen Zahlen.

Checkliste zum Lösen nichtlinearer Gleichungen

Gleichzeitig Inhaltsangabe und Stoffübersicht der folgenden Abschnitte.

- Vorarbeiten
  - Überblicken Sie den Verlauf der Funktionen (Wertetabelle, graphische Darstellung).
  - Definitionsbereich? Wo können die Lösung liegen? Wie viele Lösungen gibt es?
  - Lassen sich günstige Umformungen finden?
- Trivialmethoden für Computer oder Taschenrechner
  - Systematisches Einsetzen in Wertetabelle
  - Hineinzoomen im Funktionsgraph
- Klassische Lösungsverfahren
  - Intervallhalbierung
  - Sekantenmethode und Regula Falsi
  - Newton-Verfahren (heißt auch Newton-Raphson-Verfahren)
  - Fixpunkt-Iteration

1.3 Beispiele zum Aufwärmen

In den Übungen und in der Vorlesung diskutieren wir Beispiele der folgenden Art. Auch die folgenden Abschnitte 1.5 und 1.6 bringen weitere Erklärungen.

[4] Allerdings klingt „Wurzel“ statt „Lösung“ oder „Nullstelle“ im heutigen Fachdeutsch eher veraltet; im Englischen ist *root of a polynomial* der gängige Fachausdruck, und auch *root of a function or an equation* ist neben *zero of a function or solution of an equation* durchaus üblich.

Aus der Finanzmathematik

Ein Kredit von 100.000 € soll in 180 Monatsraten zu je 900 € zurückgezahlt werden. Was ist der Zinssatz bei diesen Konditionen?

Die Rentenformel für nachschüssige Zahlung liefert für den (monatlichen) Aufzinsungsfaktor q die Gleichung

$$900 = 100\,000\frac{q-1}{1-q^{-180}}. \tag{1}$$

Zustandsgleichung eines realen Gases

Wie groß ist das Molvolumen von Stickstoff bei 20 C und 1 bar $= 10^5$ Pa nach der Van der Waals-Gleichung?

Die Zustandsgleichung

$$\left(p + \frac{a}{V_{mol}^2}\right)(V_{mol} - b) = RT$$

beschreibt den Zusammenhang zwischen Druck p, Molvolumen V_{mol} und Temperatur T. Die Konstanten a und b haben für Stickstoff die Werte

$$a = 0{,}129\,\mathrm{Pa\,m^6/mol^2}, \quad b = 38{,}6 \times 10^{-6}\,\mathrm{m^3/mol}.$$

Die molare Gaskonstante ist $R = 8{,}3145\,\mathrm{J/molK}$. Nach Einsetzen der Zahlenwerte verbleibt als Gleichung für V_{mol}:

$$\left(100\,000 + \frac{0{,}129}{V_{mol}^2}\right)(V_{mol} - 0{,}000\,038\,6) = 2437{,}4 \tag{2}$$

Widerstände in Rohrleitungen

Die Rohrreibungszahl λ hängt von der Reynoldszahl Re ab. Bei laminarer Strömung gilt einfach $\lambda = 64/\mathrm{Re}$. Im turbulenten Bereich, ab etwa $\mathrm{Re} > 2000$, listen technische Handbücher verschiedene, teilweise empirische Formeln für λ. Auf theoretischem Weg hat Prandtl für ein glattes Rohr die Beziehung

$$\lambda = \frac{1}{(2\log_{10}(\mathrm{Re}\sqrt{\lambda}) - 0{,}8)^2} \tag{3}$$

abgeleitet, die bis $\mathrm{Re} = 3{,}4 \times 10^6$ mit Versuchen übereinstimmt. Wie groß ist λ bei $\mathrm{Re} = 1 \times 10^6$?

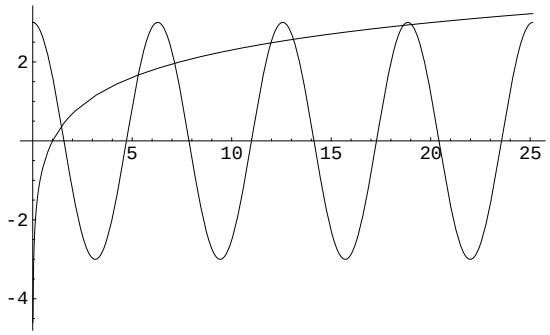


Abbildung 1: Schaubild zur Gleichung $3\cos x = \log x$. Den x-Werten der Schnittpunkte der Funktionsgraphen entsprechen die Lösungen der Gleichung.

Ohne tiefere Bedeutung

Es ist gut, wenn die bisherigen Beispiele den Eindruck einer gewissen Praxisnähe vermittelt haben. Der technischen Hintergrund und damit verbundene Verständnisschwierigkeiten verstellen aber den Blick auf die mathematischen Inhalte. Sie lernen hier nicht Physik, sondern numerische Verfahren, und die lassen sich leichter an einfachen Musterbeispielen illustrieren. Deswegen:

Finden Sie die Lösungen der Gleichung

$$3\cos x = \log x \tag{4}$$

Wichtiger Hinweis: hier meint log natürlich den natürlichen Logarithmus[5]. Argumente in Winkelfunktionen sind immer im Bogenmaß einzusetzen!

1.4 Graphische Lösung: Ein Bild sagt mehr als tausend Formeln

Entsprechend der Checkliste aus Kapitel 1.2 verschaffen wir uns am Beispiel von Gleichung 4 einen ersten Überblick. Diese Gleichung läßt nicht unmittelbar erkennen, ob, wo und wieviele Lösungen sie hat. Da sowohl Cosinus als auch Logarithmus geläufige Funktionen sind, bietet sich eine graphische Darstellung an. (Abbildung 1). Aus dem Schaubild lässt sich die Anzahl und ungefähre Lage der Lösungen erkennen. Rechenprogramme, die Wertetabellen berechnen oder in einen Funktionsgraphen hineinzoomen können, liefern rasch brauchbare Werte (die Checkliste nennt diese Vorgangsweisen „Trivialmethoden“).

1.5 Passende Umformungen: Nullstellen und Fixpunkte

Die Lösungen der Gleichung $3\cos x = \log x$ sind genau die Nullstellen der Funktion $f(x) = 3\cos x - \log x$. Ein Vergleich von Abbildung 1 mit Abbildung 2 stellt diesen Sachverhalt klar

[5] Für den dekadischen Logarithmus sprechen außer der evolutionsbedingten Zufälligkeit, dass Menschen zehn Finger haben, keine Argumente. Für Leute, die nicht bis drei zählen können, ist die Basis $e = 2{,}718\,281\,8\ldots$ ohnedies natürlicher.

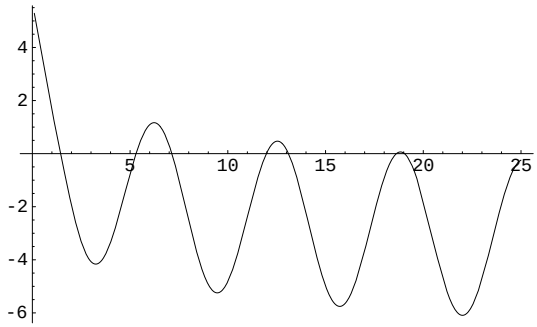


Abbildung 2: Schaubild zur Funktion $f(x) = 3\cos x - \log x$. Die Nullstellen von f entsprechen den x-Werten der Schnittpunkte in der Abbildung 1

und zeigt zum Beispiel: In der Nähe von $x = 5$, jedenfalls im Bereich $4 < x < 6$, muss eine der Nullstellen von f liegen.

Welche Form der graphischen Darstellung man günstigerweise wählt, hängt von der gegebenen Gleichung ab. In diesem Beispiel lassen sich cos und log als bekannte Funktionen leicht skizzieren, deswegen ist die Darstellung der Lösung durch die (x-Werte der) Schnittpunkte zweier Kurven übersichtlich. Andererseits lässt die Darstellung von $f(x) = 3\cos x - \log x$ die Nullstellen unmittelbar erkennen. Die klassischen Methoden zum Finden von Nullstellen ab Kapitel 1.7 erfordern ohnedies eine solche Umformung der Gleichung.

Die Gleichung $3\cos x = \log x$ lässt sich aber auch beispielsweise umformen zu

$$x = \arccos \frac{\log x}{3} \quad . \tag{5}$$

In dieser Form liegt eine Fixpunkt-Aufgabe $x = \phi(x)$ vor, mit $\phi(x) = \arccos((\log x)/3)$.

Fixpunkt-Iteration

Was passiert, wenn man auf der rechten Seite von Gleichung 5 einen Wert für x einsetzt, den Ausdruck ausrechnet und das Ergebnis wieder in der rechten Seite einsetzt? Beginnend etwa mit $x = 1$ liefert dieses Verfahren die Folge

$$1; \quad 1{,}5708; \quad 1{,}419\,69; \quad 1{,}453\,72; \quad 1{,}445\,76; \quad 1{,}447\,61; \quad 1{,}447\,18\ldots$$

Die Folge konvergiert gegen $\xi = 1{,}447\,258\,6$, das ist die kleinste Lösung der gegebenen Gleichung und gleichzeitig der einzige Fixpunkt der Funktion

$$\phi(x) = \arccos \frac{\log x}{3}.$$

Sie sehen hier ein Beispiel einer *Fixpunkt-Iteration*.

Fixpunkt-Iteration
Gegeben eine Gleichung $x = \phi(x)$.

> Beginne mit einem Startwert
> Setze Wert auf rechter Seite der Formel ein und werte aus
> Setze das Ergebnis wieder und wieder rechts in die Formel ein, bis sich die Resultate nicht mehr ändern

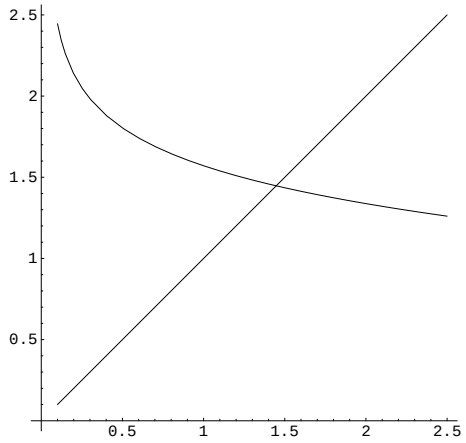


Abbildung 3: Schaubild zur Fixpunktaufgabe mit der Funktion $\phi(x) = \arccos((\log x)/3)$. Der Fixpunkt von ϕ entspricht der Nullstelle von f in der Nähe von 1,4. Weitere Fixpunkte von ϕ gibt es nicht. Durch die Umformulierung sind alle anderen Lösungen der ursprünglichen Gleichung verlorengegangen!

Weitere Beispiele von Fixpunkt-Iterationen:

- Geben Sie eine Zahl in den Taschenrechner ein und drücken Sie wiederholt auf die Wurzeltaste. Die Ergebnisse konvergieren gegen 1 (Fixpunkt von $f(x) = \sqrt{x}$).
- Geben Sie eine Zahl < 20 in den Taschenrechner ein und drücken Sie abwechselnd wiederholt auf die Tasten exp und $1/x$. Die Ergebnisse (nach dem $1/x$-Schritt) konvergieren gegen 0,567 14 (Fixpunkt von $f(x) = 1/\exp x$).
- Berechnen von Quadratwurzeln war schon in der griechischen Antike ein wichtiges Problem und (für rationale Zahlen) gelöst. Die Wurzel aus a ist definiert als Lösung von $x^2 = a$; eine für $x \neq 0$ äquivalente Umformung dieser Gleichung ist

  $$x = \frac{1}{2}\left(x + \frac{a}{x}\right) \ .$$

  Schon den Babyloniern soll die oft als Heron-Verfahren bezeichnete Iteration

  $$x^{(0)} = a; \quad x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \text{ für } k = 0, 1, 2, \ldots$$

  bekannt gewesen sein.
- Gleichung 3 ist eine Fixpunkt-Gleichung. Mit dem Startwert 0,05 liefern wenige Fixpunkt-Iterationen eine genaue Lösung.

Aber es funktioniert nicht immer: Eine andere mögliche Fixpunkt-Form von Gleichung 4 lautet

$$x = \exp(3\cos x) \ .$$

Wenn Sie hier $x = 1$ rechts einsetzen und das für die Ergebnisse jeweils wiederholen, erhalten Sie die Folge

$$1; \quad 5{,}057\,68; \quad 2{,}760\,46; \quad 0{,}061\,745\,5; \quad 19{,}971; \quad 3{,}6805\ldots$$

Ihre Werte wechseln unregelmäßig und konvergieren nicht.

Zusammenfassung

Nicht jede Fixpunkt-Iteration konvergiert. Passende Umformungen sind nicht immer leicht zu finden. Anderseits sind viele numerische Verfahren vom Typ einer Fixpunkt-Iteration. Das rechtfertigt eine ausführliche theoretische Untersuchung solcher Verfahren im Kapitel 1.12.

1.6 Diskussion der Beispiele: Wichtige und unwichtige Terme

Hier werden die in Kapitel 1.1 vorgestellten Beispiele ausführlich besprochen.

1.6.1 Eine fast lineare Gleichung

Die am Anfang von Kapitel 1.1 erwähnte Gleichung

$$x^2 - 12345678x + 9 = 0$$

ist, wenn es um die betragskleinere der beiden Lösungen geht, eigentlich keine quadratische Gleichung! Begründung: Die gesuchte Lösung ist von der Größenordnung 10^{-6} bis 10^{-7}; der Term x^2 in der Gleichung ist also gegenüber dem linearen Term $12345678x$ um mehr als zehn Größenordnungen kleiner. Für alle praktischen Zwecke ist eine solche Gleichung linear mit einem kleinen quadratischen Korrekturterm. Lösen Sie daher nach dem linearen Term auf:

$$x = \frac{1}{12345678}(x^2 + 9) \; .$$

Der Startwert $x^{(0)} = 0$ liefert selbst auf den billigsten Taschenrechnern ohne Wurzeltaste bereits ein bessere Näherung $x^{(1)} = 7{,}290\,000\,597\,78 \times 10^{-7}$ als die meisten Rechner durch Anwendung der Standard-Lösungsformel erreichen können.

> Locker formuliert: Viele Gleichungen enthalten Terme, in denen die Unbekannte zwar auftritt, aber im Vergleich zu anderen Termen wenig Einfluss hat. Wenn eine Gleichung dadurch leichter lösbar wird, lassen sich solche Terme in erster Näherung vernachlässigen. In weiteren Schritten korrigiert man das Ergebnis, indem man Näherungswerte in den anfangs vernachlässigten Termen einsetzt.

1.6.2 Van der Waals-Gleichung

Die Gleichung (2) lässt sich zu einer kubischen Gleichung umformen,

$$-4.9794 \cdot 10^{-6} + 0.129 V_{mol} - 2441.3 V_{mol}^2 + 100000 V_{mol}^3 = 0 \quad , \tag{6}$$

und wäre damit im Prinzip analytisch lösbar. Tun Sie's nicht! Ein wenig Einsicht in den physikalischen Hintergrund dieser Gleichung legt eine andere Vorgangsweise nahe: Bei Zimmertemperatur ist Stickstoff nahezu ein ideales Gas, das der Gleichung

$$pV_{mol} = RT$$

gehorcht. In der Van der Waals-Gleichung

$$\left(p + \frac{a}{V_{mol}^2}\right)(V_{mol} - b) = RT \tag{7}$$

ist der Term a/V_{mol}^2 eine Korrektur der idealen Gasgleichung und für die im Beispiel gegebenen Parameter gegenüber p vernachlässigbar klein. Dem umgeformten Polynom (6) sieht man es

nicht an, aber die ursprüngliche Gleichung (7) entspricht – im Bereich der gegebenen Daten – nicht einer „richtigen“ kubischen Gleichung, sondern vielmehr einer lineare Gleichung in V_{mol} plus einem kleinen Korrekturterm a/V_{mol}^2.

Daher lässt sich diese Gleichung auflösen, wenn man „unwichtige“ Terme der Unbekannten auf der rechten Seite stehen lässt. Hier formen wir um zu

$$V_{mol} = \frac{RT}{p + a/V_{mol}^2} + b = \frac{2437{,}4}{100000 + 0{,}129/V_{mol}^2} + 0{,}000\,038\,6$$

und ignorieren wir erst einmal den Korrekturterm a/V_{mol}^2. Das liefert eine nullte Näherung für das Molvolumen,

$$V_0 = \frac{2437{,}4}{100000} + 0{,}000\,038\,6 = 0{,}024\,413\ .$$

Der Trick ist nun, diese Näherung für V_{mol} in der rechten Seite der Gleichung einzusetzen und daraus eine verbesserte Näherung

$$V_1 = \frac{2437{,}4}{100000 + 0{,}129/0{,}024\,413^2} + 0{,}000\,038\,6 = 0{,}024\,360$$

zu berechnen. Wiederholtes Einsetzen liefert keine weitere Verbesserung:

$$V_2 = \frac{2437{,}4}{100000 + 0{,}129/0{,}024\,360^2} + 0{,}000\,038\,6 = 0{,}024\,360\ .$$

Damit haben wir (jedenfalls auf fünf Dezimalstellen genau) den Wert $V_{mol} = 0{,}024\,360\,\mathrm{m}^3$ bestimmt.

> Bußübung für die Fastenzeit: Schlagen Sie in Wikipedia die Cardanischen Formeln nach und lösen Sie die Aufgabe damit. Vergleichen Sie den Zeitaufwand mit der obigen Methode.

1.6.3 Finanzmathematik

In Gleichung 1 erwarten wir für den Aufzinsungsfaktor q einen Wert knapp über 1. Den Term q^{-180} im Nenner wird vermutlich $\ll 1$ und nicht so wichtig sein. Das motiviert, die Gleichung nach dem q im Zähler aufzulösen.

$$q = 1 + \frac{900}{100000}(1 - q^{-180})$$

Ignoriert man q^{-180} auf der rechten Seite, dann folgt als nullte Näherung

$$q_0 = 1 + \frac{900}{100000} = 1{,}009$$

Auch hier funktioniert der Trick, q_0 in der rechten Seite einzusetzen und daraus eine verbesserte Näherung

$$q_1 = 1 + \frac{900}{100000}(1 - 1{,}009^{-180}) = 1{,}007\,206$$

zu berechnen. Wiederholtes Einsetzen liefert

$$q_2 = 1{,}006\,529 \quad q_3 = 1{,}006\,210 \quad q_4 = 1{,}006\,047\ldots$$

Es braucht aber hier insgesamt 14 Iterationen, bis sich die Werte bei $q = 1{,}005\,851$ stabilisieren.

Bemerkungen zum Abschluss

Ist eine Gleichung in der Form $f(x) = g(x)$ gegeben (Beispiel: Gleichung 4), lässt sich nicht unmittelbar erkennen, welche Terme „wichtig“ oder „unwichtig“ sind. Regel: man löse nach jener Seite der Gleichung auf, welcher den *steileren* Funktionsgraph im Schnittpunkt hat.

Passende Umformungen für Fixpunkt-Iterationen erfordern oft ein tieferes Verständnis der einzelnen Terme in einer Gleichung. Es gibt zum Glück Lösungsverfahren, die mehr nach „Schema F“ ablaufen. Eines davon stellt das nächste Kapitel vor.

1.7 Intervallhalbierung

Kennen Sie die Geschichte von den zwei Möglichkeiten? Sie beginnt mit dem Zwischenwertsatz.

Zwischenwertsatz

Eine Funktion f, die auf einem abgeschlossenen Intervall $[a, b]$ stetig ist, nimmt in diesem Intervall auch jeden Wert zwischen $f(a)$ und $f(b)$ an.

Ist f insbesondere für $x = a$ negativ und für $x = b$ positiv (oder umgekehrt), dann garantiert der Zwischenwertsatz: f hat mindestens eine Nullstelle in diesem Intervall.

Es gibt immer zwei Möglichkeiten...

Angenommen, wir suchen eine Nullstelle einer im Bereich $a \leq x \leq b$ stetigen Funktion. Es lässt sich rechnerisch sofort prüfen, ob $f(a)$ und $f(b)$ unterschiedliches Vorzeichen haben. Wenn ja, dann garantiert der Zwischenwertsatz die Existenz eine Nullstelle im Bereich $a \leq x \leq b$, aber wir wissen nicht, wo sie liegt. Nun gibt es zwei Möglichkeiten: Entweder ist $b - a$ klein, dann ist es gut: Wir können sowohl a als auch b als Näherung für eine Nullstelle von f auffassen. Andernfalls berechnen wir den Mittelpunkt c des Intervalls, $c = (a + b)/2$. Nun gibt es wieder zwei Möglichkeiten. Ist $f(c) = 0$, so ist es gut: es liegt dort eine Nullstelle vor. Anderenfalls hat f an den Enden eines der Teilintervalle $a \leq x \leq c$ oder $c \leq x \leq b$ verschiedene Vorzeichen (klar? Das ist der springende Punkt!). In einem der beiden Intervalle muss also eine Nullstelle liegen. Betrachten wir dieses Intervall und nennen wir der Einfachheit die neuen Intervallgrenzen wieder a und b.

Nun gibt es zwei Möglichkeiten: Entweder ist $b-a$ klein, dann ist es gut: Wir können sowohl a als auch b als Näherung für eine Nullstelle von f auffassen. Andernfalls bilden wir $c = (a+b)/2$. Nun gibt es wieder zwei Möglichkeiten...

Sie können nun die Geschichte selber fortsetzen. Beachten Sie aber, dass die Intervalllänge in jedem Erzählschritt halbiert wird. Für jede beliebig klein vorgegebene Genauigkeitsschranke $\epsilon > 0$ erreichen Sie nach einer endlichen Anzahl von Schritten ein Intervall mit Länge $b-a < \epsilon$. Damit endet die Geschichte wie im wirklichen Leben: Es gibt immer zwei Möglichkeiten, aber jede Entscheidung schränkt den Freiraum für weitere Aktionen ein. Irgendwann sind die Alternativen dann doch ausgeschöpft.

Formalisiert angeschrieben, lautet dieses Verfahren

Intervallhalbierung (Bisektionsverfahren)
Gegeben eine Funktion f, zwei Werte a und b mit $f(a) \cdot f(b) < 0$, eine Fehlerschranke $\epsilon > 0$. Ist f im Intervall $a \leq x \leq b$ stetig, dann findet dieser Algorithmus die Näherung c an eine Nullstelle ξ von f mit Fehler $|c - \xi| < \epsilon$.

Wiederhole
  setze $c \leftarrow (a + b)/2$
  falls $f(a) \cdot f(c) < 0$
    setze $b \leftarrow c$
  sonst
    setze $a \leftarrow c$
bis $|b - a| < \epsilon$ oder $f(c) = 0$

Lineare Konvergenz

Die beste Schätzung für den Wert der Nullstelle ist der Mittelpunkt des Intervalls. Der maximale Fehlerbetrag ist dann durch $\epsilon_0 \leq |b-a|/2$ beschränkt; größer als die halbe Intervallbreite kann er nicht sein. Intervallhalbierung reduziert diese Fehlerschranke pro Schritt um den Faktor $1/2$ oder, da

$$\left(\frac{1}{2}\right)^{3{,}3} \approx \frac{1}{10} \quad ,$$

um einen Faktor $1/10$ pro (durchschnittlich) 3,3 Schritten. Man kann sagen: Intervallhalbierung produziert eine korrekte Dezimalstelle pro 3,3 Iterationen. Der maximale Fehler nach dem i-ten Schritt, ϵ_i, ist höchstens halb so groß wie der vorherige maximale Fehler ϵ_{i-1}. Es gilt also

$$\epsilon_i \leq C\epsilon_{i-1} \quad \text{mit } C = \frac{1}{2} \; .$$

Allgemein: Wenn bei einem Verfahren für die Fehlerschranken aufeinanderfolgender Iterationsschritte gilt

$$\epsilon_i \leq C\epsilon_{i-1} \quad \text{mit } C < 1 \; .$$

spricht man von *linearer* Konvergenz.

Vor- und Nachteile

Vorteile der Intervallhalbierung: einfach zu verstehen, leicht zu programmieren. Wenn die Voraussetzungen erfüllt sind, konvergiert es mit Sicherheit. Es ist ein *Einschlussverfahren*, das heißt, es liefert nicht nur einen Näherungswert, sondern grenzt die Lösung von beiden Seiten her ein.

Nachteile: Man braucht Startwerte–aber das ist ein Problem jedes numerischen Verfahrens. Intervallhalbierung ist langsam; nur lineare Konvergenz – die dafür aber sicher.

1.8 Regula Falsi (lineares Eingabeln)

Funktionen, die in der Umgebung der Nullstelle glatt verlaufen, lassen sich dort durch eine Gerade annähern. Statt, wie bei der Intervallhalbierung, den Wert c genau in der Mitte zwischen a und b anzunehmen, wählen wir c als Nullstelle der Gerade durch $(a, f(a))$ und $(b, f(b))$, siehe Abbildung 4.

$$c = a - f(a)\frac{a-b}{f(a)-f(b)} = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

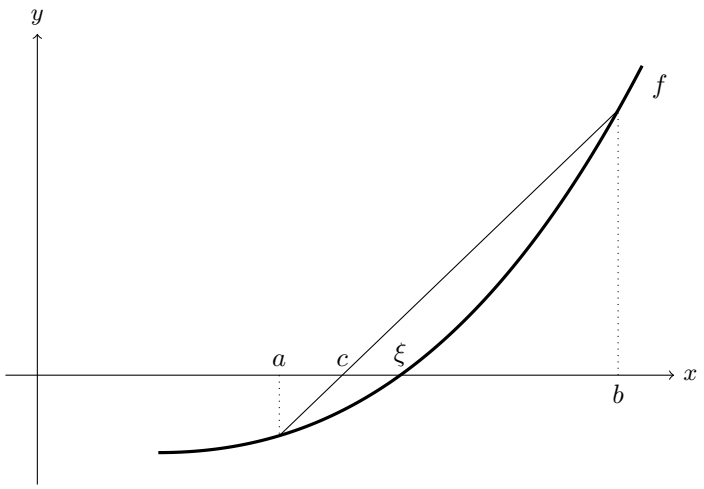


Abbildung 4: Die Regula Falsi berechnet c, die Nullstelle der Verbindungsgeraden, als Näherungswert an die Nullstelle ξ der Funktion f.

Regula Falsi (lineares Eingabeln)
Gegeben eine Funktion f, zwei Werte a und b mit $f(a) \cdot f(b) < 0$ und eine Genauigkeitsschranke $\epsilon > 0$. Ist $f(x)$ im Intervall $a \leq x \leq b$ stetig, dann findet dieser Algorithmus[a] eine Näherung c an eine Nullstelle ξ von f mit Genauigkeit $|c - \xi| < \epsilon$.

Wiederhole
  setze $c \leftarrow a - f(a)\dfrac{a-b}{f(a)-f(b)}$
  falls $f(b) \cdot f(c) < 0$
    setze $a \leftarrow b$
  sonst
    (klassische Version) nix
    (Illinois-Variante) reduziere $f(a)$ auf $\frac{1}{2}f(a)$
    (Pegasus-Variante) reduziere $f(a)$ auf $\dfrac{f(a)f(b)}{f(b)+f(c)}$
  setze $b \leftarrow c$
bis $|b-a| < \epsilon$ oder $f(c) = 0$

[a] mit dem hier gegebenen Abbruchkriterium allerdings nur die beiden Varianten

Allerdings bringt die Regula Falsi in der Standard-Version im Vergleich zur Intervallhalbierung kein wesentlich besseres Konvergenzverhalten. Typischer Weise bleibt nach einigen Iterationen die Intervallgrenze a fix, die andere Grenze b konvergiert zwar zur Nullstelle, aber die Abbruchbedingung $|b - a| < \epsilon$ wird nicht erreicht. Sorgfältige Programmierer würden im obigen Algorithmus jedenfalls noch eine Notbremse einbauen: zähle die Anzahl der Iterationen mit und brich ab, wenn eine Maximalzahl überschritten wird.

Die Illinois- oder die Pegasus-Variante verbessern das Konvergenzverhalten im Vergleich zur Intervallhalbierung deutlich; mutige Programmierer würden in diesem Fall auf die Abfrage

nach einer maximalen Iterationszahl verzichten.

Intervallhalbierung und die verschiedenen Regula-Falsi-Versionen haben gemeinsam, daß sie die Nullstelle von beiden Seiten her „eingabeln“ — sie sind Einschlussverfahren, das ist gut. Nachteilig ist, dass man zu Beginn des Verfahrens zwei Näherungswerte braucht, und zwar je einen auf jeder Seite der Nullstelle. Das kann sehr schwer zu erreichen sein, wenn man zwei nahe beisammen liegende Nullstellen hat, da dann eine der ursprünglichen Näherungen dazwischen liegen muß. Mehrfache Nullstellen gerader Ordnung können diese Verfahren überhaupt nicht finden.

> Was ist „falsch“ an der Regula Falsi? Natürlich nicht die Regel selbst, sondern die angenommenen Startwerte a und b. Aus diesen beiden „falschen Lösungen“ berechnet die Regel eine bessere Näherungslösung.
> Die Methode ist uralt, die Grundidee war schon Jahrhunderte vor Chr. weltweit bekannt: Babyloniern, Ägypter, Inder und Chinesen lösten damit lineare Gleichungen. Aus arabischen Quellen nach Europa bringt sie um 1200 Leonardo von Pisa, genannt Fibonacci. Er beschreibt mehrere Varianten, darunter die *regula duarum falsarum positionum*, die „Methode vom doppelten falschen Ansatz“. So sollte sie auch richtiger Weise heißen, aber es hat sich schlampig verkürzt „Regula Falsi“ durchgesetzt.
> Fibonacci löste damit nur lineare Probleme; da berechnet die Regel aus zwei falschen Startwerten sofort die richtige Lösung. Die Anwendung als iteratives Verfahren für Nullstellen nichtlinearer Funktionen ist dann doch nicht so alt. Mitte des vorigen Jahrhunderts fand man kleine, aber nicht unwesentliche Verbesserungen der Rechenregel (Pegasus-, Illinois-Variante). Sogar noch kürzlich, 2020, veröffentlichten Oliveira und Takahashi eine weitere Verbesserung (`https://en.wikipedia.org/wiki/ITP_method`).

1.9 Sekantenmethode

Die Sekantenmethode berechnet gleich wie die Regula Falsi eine neue Näherung durch lineare Interpolation, verlangt aber nicht, dass die Werte a und b die Nullstelle einschließen, siehe Abbildung 5.

Die formale Beschreibung des Verfahrens bezeichnet hier die Startwerte a und b mit $x^{(0)}$ und $x^{(1)}$ und die weiteren iterativ berechneten Näherungswerte mit $x^{(k)}, x^{(k+1)}, \ldots$

Sekantenmethode
Gegeben eine Funktion f, zwei Werte $x^{(0)}$ und $x^{(1)}$, eine Genauigkeitsschranke $\epsilon > 0$ und eine maximale Iterationsanzahl k_{max}. Für hinreichend gute Startwerte $x^{(0)}$ und $x^{(1)}$ findet dieser Algorithmus die Näherung $x^{(k)}$ an eine Nullstelle ξ von f mit Genauigkeit $|x^{(k)} - \xi| \approx \epsilon$ oder bricht nach einer Maximalzahl von k_{max} Schritten ab.

setze $k = 1$
Wiederhole
  setze $x^{(k+1)} = x^{(k)} - f(x^{(k)}) \dfrac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}$
  erhöhe $k = k + 1$
bis $|x^{(k+1)} - x^{(k)}| < \epsilon$ oder $k \geq k_{max}$

Superlineare Konvergenz

Die Sekantenmethode zeigt *superlineare* Konvergenz. (Notwendige technische Details: f zweimal stetig differenzierbar, keine mehrfache Nullstelle.) Das heißt, für die Fehlerschranken $|x^{(k+1)} - \xi|$ und $|x^{(k)} - \xi|$ aufeinanderfolgender Schritte gilt, sofern $|x^{(k)} - \xi|$ schon hinreichend klein ist:

$$|x^{(k+1)} - \xi| \leq C|x^{(k)} - \xi|^p \quad \text{mit } p > 1 \ .$$

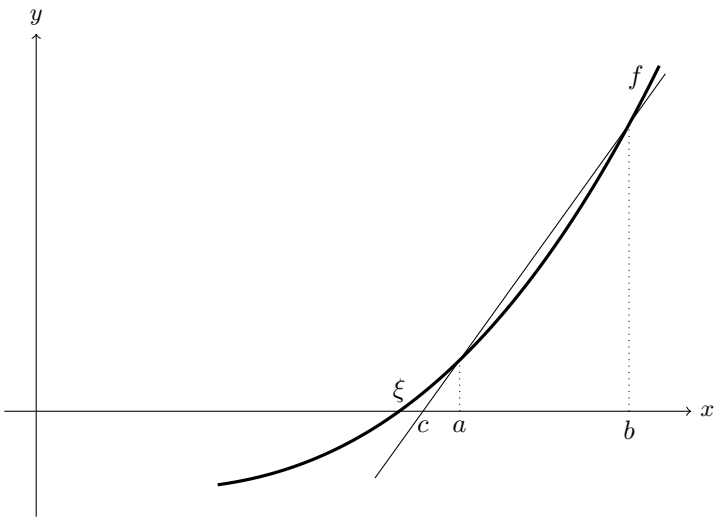


Abbildung 5: Die Sekantenmethode berechnet den nächsten Näherungswert c mittels einer Schnittgeraden (Sekante) durch zwei Punkte des Funktionsgraphen. Die beiden Werte a und b schließen die Nullstelle ξ jedoch nicht unbedingt ein.

Der Fehler reduziert sich also nicht bloß um einen Faktor C, sondern zusätzlich noch mit der Potenz p. Für die Sekantenmethode lässt sich zeigen

$$p = \frac{1+\sqrt{5}}{2} \approx 1{,}618 \ .$$

Angenommen, es ist $|x^{(k)} - \xi| = 0{,}01$. Überlegen Sie sich, was den Fehler stärker verringert: Multiplikation mit einem Faktor $C = 1/2$, oder Potenzieren mit $p = 1{,}6$!

1.10 Newton-Verfahren

Heißt auch Newton-Raphson-Verfahren, aber erst einige Jahrzehnte nach Isaac Newton und Joseph Raphson formuliert Thomas Simpson das Verfahren so, wie wir es heute kennen.

Gesucht sei eine Nullstelle der Funktion f. Gegeben sei ein Startwert $x^{(0)}$ in der Nähe der Nullstelle. Das Newton-Verfahren versucht, ähnlich der Sekantenmethode, die Funktion f durch eine lineare Funktion anzunähern und verwendet dazu die Tangente an f im Punkt $(x^{(0)}, f(x^{(0)}))$. Der Schnittpunkt der Tangente mit der x-Achse ist der nächste Näherungswert, siehe Abbildung 6.

Herleitung aus der Taylorentwicklung von f um den Punkt $x^{(0)}$. Ist f genügend oft differenzierbar, dann gilt:

$$f(x) = f(x^{(0)}) + (x - x^{(0)})f'(x^{(0)}) + \frac{(x-x^{(0)})^2}{2!}f''(x^{(0)}) + \dots$$

Es soll gelten $f(x) = 0$. Vernachlässigen von Gliedern höherer Ordnung liefert die Gleichung

$$0 = f(x^{(0)}) + (x - x^{(0)})f'(x^{(0)}) \ ,$$

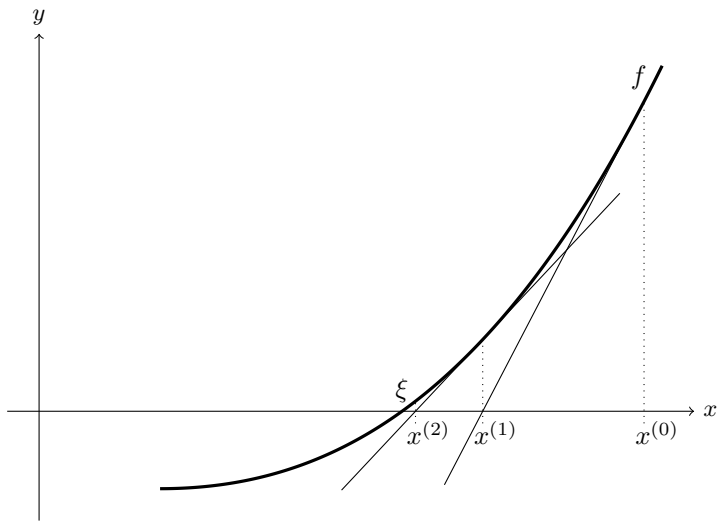


Abbildung 6: Graphische Deutung des Newton-Verfahrens: Die Tangente an f im Punkt $(x^{(0)}, f(x^{(0)}))$ schneidet die x-Achse in $x^{(1)}$. Der Wert $x^{(2)}$ im nächsten Schritt liegt schon nahe an der Nullstelle ξ.

aus der sich x ausdrücken lässt:

$$x = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} \ .$$

Newton-Verfahren
Gegeben eine differenzierbare Funktion f und ein Startwert $x^{(0)}$.
Gesucht eine Nullstelle von f.

Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad \text{für } k = 0, 1, 2 \ldots$$

Quadratische Konvergenz

Das Newton-Verfahren zeigt *quadratische* Konvergenz. Das heißt, für die Fehlerschranken $\epsilon_{k+1} = |x^{(k+1)} - x|$ und $\epsilon_k = |x^{(k)} - x|$ aufeinanderfolgender Schritte gilt, sofern ϵ_k schon hinreichend klein ist:

$$\epsilon_{k+1} \leq C\epsilon_k^2$$

Der neue Fehler ist also um einen Faktor C kleiner als das *Quadrat* des alten Fehlers. Der genaue Wert von C ist dabei nicht so wichtig.

Angenommen, es ist $\epsilon_k = 10^{-4}$. Das heißt, der Fehler beträgt eine Einheit in der vierten Nachkommastelle. Dann gilt bei quadratischer Konvergenz $\epsilon_{k+1} = C \cdot 10^{-8}$. Der Fehler beträgt also C Einheiten in der achten Nachkommastelle. Wenn C größenordnungsmäßig im Bereich 1 ist, hat sich die Anzahl der korrekten Stellen ungefähr verdoppelt.

Quadratische Konvergenz: Neuer Fehler $\sim$ Quadrat des alten Fehlers.

Faustregel: Sofern schon einige signifikante Stellen exakt sind, sind im nächsten Näherungswert etwa doppelt so viele signifikante Stellen korrekt.

1.11 Abbruchbedingungen

Rechner haben nur eine fixe Zahl von Binärstellen zur Verfügung, um Gleitkommazahlen zu speichern. Möglicherweise erreicht $f(x)$ für kein Gleitkomma-Argument x exakt den Wert Null. Wenn die Nullstelle ξ in der Gegend von 1 liegt, können Sie leicht eine Näherung x mit absolutem Fehler $|x - \xi| < 10^{-6}$ finden. Liegt die Nullstelle um $\xi \approx 10^{22}$, werden Sie einen absoluten Fehler dieser Güte nicht erreichen können. Eine übliche Wahl der Abbruchschranke ϵ ist $\varepsilon_m(|a| + |b|)/2$, wenn ε_m die Maschinengenauigkeit und a, b die ursprünglichen Intervallgrenzen sind. Wenn a, b und die Nullstelle selber nahe bei Null liegen, ist Vorsicht bei dieser Formel geboten. Die Abbruchschranke darf jedenfalls nicht kleiner als die kleinste positive Maschinenzahl sein (typischerweise um 10^{-38} für 4-Byte-Datentypen, 10^{-308} für 8-Byte-Datentypen).

Maschinengenauigkeit

Die Maschinengenauigkeit ε_m ist die kleinste positive Gleitkommazahl, die, zur Gleitkommazahl 1,0 addiert, eine von 1,0 verschiedene Summe ergibt (typischerweise um 10^{-7} für 4-Byte-Datentypen, 10^{-16} für 8-Byte-Datentypen).

1.12 Fixpunkt-Iteration

Im Abschnitt 1.5 haben wir bereits Fixpunkte von Funktionen durch wiederholtes Einsetzen bestimmt. Viele numerische Verfahren lassen sich als Spezialfälle einer Fixpunkt-Iteration betrachten. Aussagen über die Konvergenz von Fixpunkt-Iterationen sind deswegen von allgemeiner Bedeutung.

Fixpunkt-Iteration
Gegeben eine Funktion ϕ und ein Startwert $x^{(0)}$.
Gesucht ein Fixpunkt ξ von ϕ.

Iterationsvorschrift
$x^{(k+1)} = \phi(x^{(k)})$ für $k = 0, 1, 2 \ldots$

Fixpunkt-Iteration konvergiert für kontrahierende Abbildungen
Die Funktion ϕ besitze einen Fixpunkt ξ Sei ferner I ein offenes Intervall der Form $(\xi - r, \xi + r)$ um den Fixpunkt ξ, in dem ϕ als *kontrahierende Abbildung* wirkt, d. h.

$$|\phi(x) - \phi(y)| \leq C|x - y| \text{ gilt mit } C < 1 \text{ für alle } x, y \in I \ .$$

Dann konvergiert für alle $x^{(0)} \in I$ die Fixpunkt-Iteration $x^{(k+1)} = \phi(x^{(k)})$ mindestens linear gegen ξ.

Beweis: Zuerst zeigt man durch Induktion: $x^{(k)} \in I$ für alle $k = 0, 1, 2 \ldots$ Die Aussage ist laut Voraussetzung richtig für $k = 0$. Angenommen, es liegt bereits $x^{(k)} \in I$, also weniger als r von ξ entfernt: $|x^{(k)} - \xi| < r$. Dann können wir die Kontraktionsbedingung und Fixpunkt-Eigenschaft für $x^{(k)}$ und ξ anwenden und erhalten

$$|x^{(k+1)} - \xi| = |\phi(x^{(k)}) - \phi(\xi)| \leq C|x^{(k)} - \xi| < Cr \; .$$

Da $C < 1$, ist also auch

$$|x^{(k+1)} - \xi| < r \quad \text{und somit} \quad x^{(k+1)} \in I$$

Aus diesen Überlegungen folgt auch unmittelbar für die Fehler $\epsilon^{(k)} = |x^{(k)} - \xi|$ und $\epsilon^{(k+1)} = |x^{(k+1)} - \xi|$:

$$\epsilon^{(k+1)} \leq C\epsilon^{(k)} \leq C^k \epsilon_0, \text{ somit } \epsilon^{(k+1)} \to 0 \text{ für } k \to \infty \; .$$

So wie der Satz hier formuliert ist, setzt er die Existenz eines Fixpunktes voraus. Dadurch wird der Konvergenz-Beweis kurz und schmerzlos. Eine etwas allgemeinere Formulierung und ein technisch aufwändigerer Beweis zeigen, dass aus der Kontraktions-Eigenschaft auch schon die Existenz und Eindeutigkeit eines Fixpunktes folgen. Das ist der berühmte Fixpunktsatz von Banach.

Zusammenhang kontrahierende Abbildung–Steigung der Funktion

Die Eigenschaft $|\phi(x) - \phi(y)| \leq C|x - y|$ bedeutet für $C < 1$ anschaulich: Funktionswerte unterscheiden sich weniger als die Eingabewerte. Wie stark sich Funktionswerte im Verhältnis zu Eingabewerten ändern, ist (im Grenzwert für kleine Änderungen) durch die Steigung der Funktion bestimmt.

Ist ϕ in einer Umgebung von ξ stetig differenzierbar und $|\phi'(\xi)| < 1$, so ist in einer Umgebung von ξ die Kontraktionseigenschaft erfüllt: Wegen der Stetigkeit von ϕ' gibt es ein offenes Intervall I um ξ, in dem $|\phi'| \leq C < 1$ gilt. Für $x, y \in I$ gilt nach dem Mittelwertsatz der Differentialrechnung

$$\phi(x) - \phi(y) = (x - y)\phi'(\eta) \quad \text{für ein } \eta \in I \; .$$

Damit ist auch

$$|\phi(x) - \phi(y)| \leq C|x - y| \; , \quad C < 1$$

Eine Kurzfassung dieser Aussage:

Abbildung 7 illustriert das Konvergenzverhalten der Fixpunkt-Iteration für verschiedene ϕ.

1.13 Konvergenzordnung

Wir haben lineare, superlineare und quadratische Konvergenz bereits erwähnt. Hier fassen wir den Begriff der Konvergenzordnung genauer.

Konvergenzordnung
Sei ξ Fixpunkt von ϕ, und es gelte für alle Startwerte aus einem Intervall um ξ und die zugehörige Folge $\{x^{(k)}\}$ aus der Vorschrift $x^{(k+1)} = \phi(x^{(k)}), k = 0, 1, 2 \ldots$

$$|x^{(k+1)} - \xi| \leq C|x^{(k)} - \xi|^p$$

mit $p \geq 1$ und $C < 1$, falls $p = 1$.
Das Iterationsverfahren heißt dann ein Verfahren von mindestens p-ter Ordnung

Für das lokale Konvergenzverhalten einer Fixpunkt-Iteration ist der Wert der ersten Ableitung am Fixpunkt maßgeblich. Für $|\phi'(\xi)| < 1$ ist lineare Konvergenz gesichert; je kleiner der Betrag der Ableitung, desto schneller konvergiert das Verfahren, wobei $C \approx |\phi'(\xi)|$. Ganz besonders rasche, nämlich superlineare Konvergenz tritt auf, wenn $|\phi'(\xi)| = 0$.

Mit Hilfe der Taylorentwicklung lässt sich zeigen: Ist $\phi(x)$ in einer Umgebung von ξ genügend oft differenzierbar und

$$\phi'(\xi) = 0,\ \phi''(\xi) = 0, \ldots, \phi^{(p-1)}(\xi) = 0, \text{ und } \phi^{(p)}(\xi) \neq 0\ ,$$

dann liegt für $p = 2, 3, \ldots$ ein Verfahren p-ter Ordnung vor. Ein Verfahren erster Ordnung liegt vor, wenn zu $p = 1$ gilt: $|\phi'(\xi)| < 1$.

1.14 Konvergenz des Newton-Verfahrens

Das Newtonverfahren, angewandt auf die Funktion f, entspricht einem Fixpunkt-Verfahren für die Funktion ϕ,

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

Nun ist

$$\phi'(x) = \frac{f''(x)f(x)}{(f'(x))^2}\ ,$$

und da an einer einfachen Nullstelle $f(x) = 0, f'(x) \neq 0$ gilt, verschwindet $\phi'(x)$ dort. Man überzeugt sich leicht, dass $\phi''(x) \neq 0$ gilt, sofern $f''(x) \neq 0$. Daraus folgt die quadratische Konvergenz des Newtonverfahrens bei einfachen Nullstellen. Bei mehrfachen Nullstellen lässt sich lineare Konvergenz nachweisen.

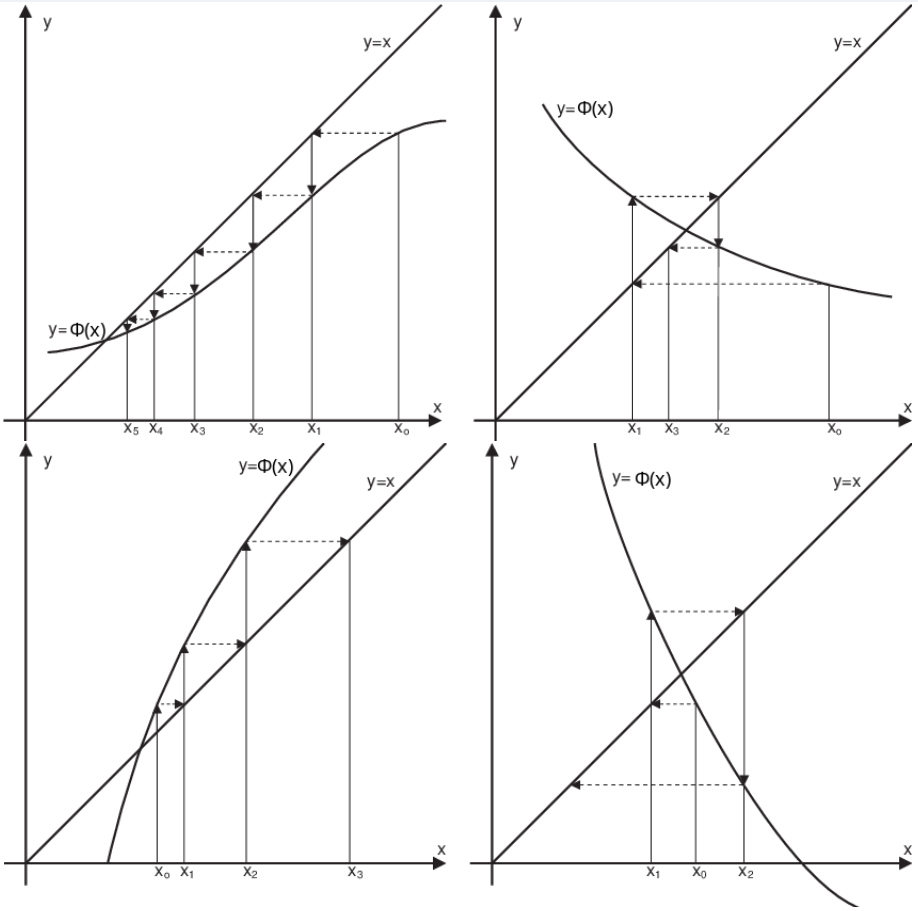


Abbildung 7: Fixpunkt-Iteration in graphischer Darstellung für verschiedene Funktionen ϕ. Mögliche Fälle: Einseitige Annäherung an den Fixpunkt, falls in einer Umgebung des Fixpunktes $0 < \phi' < 1$; alternierende Konvergenz, falls $-1 < \phi' < 0$, Divergenz falls $\phi' > 1$ oder $\phi' < -1$.

2 Systeme nichtlinearer Gleichungen

Abschnitt 1.2 definiert die Begriffe *Lösung, Nullstelle, Fixpunkt* für skalare Funktionen $\mathbb{R} \to \mathbb{R}$. Diese Begriffe lassen sich problemlos auf vektorwertige Funktionen $\mathbb{R}^n \to \mathbb{R}^n$ übertragen. Auch hier lassen sich Gleichungen auf verschiedene Weise formulieren.

Schreibweise für Vektoren und vektorwertige Funktionen: Fettdruck

Reellwertige Funktionen, Skalare: $f : \mathbb{R} \to \mathbb{R}$, $y = f(x)$
Vektorwertige Funktionen, Vektoren: $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{y} = \mathbf{f}(\mathbf{x})$

Komponenten eines Vektors $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{oder } \mathbf{x}^T = [x_1, x_2, \ldots, x_n]$$

Normalerweise ist mit $\mathbf{x}$ ein Spalten-, mit $\mathbf{x}^T$ ein Zeilenvektor gemeint.

Iterationsindizes werden hier (um sie von Vektorkomponenten zu unterscheiden) hochgestellt und in Klammern gesetzt: $\mathbf{x}^{(k)}, k = 0, 1, 2 \ldots$

2.1 Lösung, Nullstelle und Fixpunkt: mehrdimensionaler Fall

Aufgabentypen im $\mathbb{R}^n$

Es seien $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{\Phi}$ Funktionen $\mathbb{R}^n \to \mathbb{R}^n$ und $\mathbf{x} \in \mathbb{R}^n$

Problemstellung: gesucht ist ein $\mathbf{x}$, für das gilt...

$\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x})$, (Finden einer *Lösung* des Gleichungssystems)
$\mathbf{f}(\mathbf{x}) = 0$, (Finden einer *Nullstelle* der Funktion $\mathbf{f}$)
$\mathbf{x} = \mathbf{\Phi}(\mathbf{x})$, (Finden eines *Fixpunktes* der Funktion $\mathbf{\Phi}$)

Im Vergleich zu den Definitionen von Abschnitt 1.2 hat fast nichts geändert außer der Schreibweise.

Beispiel: ein *nichtlineares Gleichungssystem* mit zwei Unbekannten

$$\begin{aligned} 4x_1 - x_2 + x_1x_2 &= 1 \\ -x_1 + 6x_2 &= 2 - \log(x_1x_2) \end{aligned}$$

hat die Form $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x})$ mit

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 4x_1 - x_2 + x_1x_2 \\ -x_1 + 6x_2 \end{bmatrix}, \qquad \mathbf{h}(\mathbf{x}) = \begin{bmatrix} h_1(x_1, x_2) \\ h_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 - \log(x_1x_2) \end{bmatrix}$$

Das Gleichungssystem lässt sich umformulieren:

$$\begin{aligned} 4x_1 - x_2 + x_1x_2 - 1 &= 0 \\ -x_1 + 6x_2 + \log(x_1x_2) - 2 &= 0 \end{aligned}$$

In dieser Form lautet die Aufgabe: gesucht sind *Nullstellen der vektorwertigen Funktion* $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$, also Lösungen von $\mathbf{f}(\mathbf{x}) = 0$ mit

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 4x_1 - x_2 + x_1x_2 - 1 \\ -x_1 + 6x_2 + \log(x_1x_2) - 2 \end{bmatrix}$$

Eine andere, äquivalente Umformung liefert

$$\begin{aligned} x_1 &= \frac{1}{4}(x_2 - x_1x_2 + 1) \\ x_2 &= \frac{1}{6}(x_1 - \log(x_1x_2) + 2) \end{aligned}$$

Hier sind *Fixpunkte der vektorwertigen Funktion* $\mathbf{\Phi} : \mathbb{R}^2 \to \mathbb{R}^2$ gesucht, also Lösungen von $\mathbf{x} = \mathbf{\Phi}(\mathbf{x})$ mit

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(x_1, x_2) \\ \phi_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(x_2 - x_1x_2 + 1) \\ \frac{1}{6}(x_1 - \log(x_1x_2) + 2) \end{bmatrix}$$

Noch ein Hinweis zur Schreibweise: Wenn wir einen speziellen Fixpunkt gefunden haben, dann bezeichnen wir den im Folgenden mit $\boldsymbol{\xi}$, um ihn von anderen, allgemeinen Werten $\mathbf{x}$ zu unterscheiden.

2.2 Mehrdimensionale Fixpunkt-Iteration

Fixpunkt-Iterationen sind auch im mehrdimensionalen Fall möglich. Ein Fixpunkt einer Abbildung $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n$ ist – völlig analog zur eindimensionalen Definition – ein Wert $\boldsymbol{\xi} \in \mathbb{R}^n$, für den gilt:

$$\boldsymbol{\xi} = \mathbf{\Phi}(\boldsymbol{\xi}) \, .$$

Genauso wie im eindimensionalen Fall findet Fixpunkt-Iteration (falls sie konvergiert) einen Fixpunkt. Noch einmal: Wir setzen hier Vektoren aus dem $\mathbb{R}^n$ und vektorwertige Funktionen in fetter Schrift ($\mathbf{\Phi}, \boldsymbol{\xi}, \mathbf{x} \ldots$), zum Unterschied von Variablen und reellwertigen Funktionen ($\phi, \xi, x, \ldots$). Sonst ändert sich nichts am Schema der Fixpunkt-Iteration.

Fixpunkt-Iteration, mehrdimensional
Gegeben sei eine Abbildung $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n, \quad \mathbf{x} \to \mathbf{\Phi}(\mathbf{x})$.
Gesucht ist ein Fixpunkt $\boldsymbol{\xi}$ von $\mathbf{\Phi}$.

$\mathbf{x}^{(0)}$ als Startwert gegeben.
Iterationsvorschrift
$\mathbf{x}^{(k+1)} = \mathbf{\Phi}(\mathbf{x}^{(k)})$ für $k = 0, 1, 2 \ldots$

Beachte die Konvergenzbedingungen (Abschnitt 2.4)!

Beispiel: Fixpunkt-Iteration für ein System zweier nichtlinearer Gleichungen

Gegeben sei das nichtlineare Gleichungssystem (log ist natürlich der natürliche Logarithmus)

$$\begin{aligned} 4x - y + xy - 1 &= 0 \\ -x + 6y + \log(xy) - 2 &= 0 \end{aligned}$$

Ausgehend von der Näherungslösung $x_0 = 1$ und $y_0 = 1$ bestimme man durch geeignete Fixpunkt-Iteration verbesserten Näherungen.

In der Nähe des Startwertes hängt die erste Gleichung am stärksten vom Term $4x$ ab; die zweite Gleichung von $6y$. Vorgangsweise: löse die beiden Gleichungen jeweils nach diesen Termen auf.

$$\begin{aligned} x &= \frac{1}{4}(y - xy + 1) \\ y &= \frac{1}{6}(x - \log(xy) + 2) \end{aligned}$$

Die Funktion $\mathbf{\Phi}$ ist hier ein Vektor aus zwei reellwertigen Funktionen ϕ und ψ, der Vektor $\mathbf{x}$ hat zwei Komponenten x und y.

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \phi(x,y) \\ \psi(x,y) \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(y - xy + 1) \\ \frac{1}{6}(x - \log(xy) + 2) \end{bmatrix}$$

Iteration liefert die Folge $(1;1)$, $(1/4;1/2)$, $(0{,}343\,75; 0{,}721\,574)$, $(0{,}368\,383; 0{,}622\,985)$, …, die gegen den Fixpunkt $(0{,}353\,443\,88; 0{,}639\,968\,47)$ konvergiert.

2.3 Normen

Exakte Lösung, Näherungslösung und Fehler sind bei Gleichungssystemen jeweils Vektoren im $\mathbb{R}^n$. Wir brauchen ein Maß für die Größe oder Länge des Fehlervektors, oder für den Abstand der Näherung von der exakten Lösung. Im eindimensionalen Fall messen wir die „Größe“ von x mit dem Absolutbetrag $|x|$, und den Abstand zweier Werte x und y auf der reellen Achse durch $|y - x|$.

Während es aber in $\mathbb{R}$ nur eine sinnvolle Definition für den Absolutbetrag gibt, stehen im $\mathbb{R}^n$ mehrere Möglichkeiten offen. Da ist zunächst einmal die „übliche“ Definition für die Länge eines Vektors, auch *euklidische* Länge oder *2-Norm* genannt. Oft lässt sich aber mit anderen Normen einfacher arbeiten. Wir verwenden noch die *1-Norm* und die *∞-Norm*.

Normen im $\mathbb{R}^n$ für einen Vektor $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \quad , \quad \text{Einsnorm, Summennorm}$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} (x_i)^2} \quad , \quad \text{Zweinorm, euklidische Norm}$$

$$\|\mathbf{x}\|_\infty = \max_i |x_i| \quad , \quad \text{Unendlich-Norm, Maximums-Norm}$$

Erinnern Sie sich an die Definition einer Norm aus Mathematik 2?

Eine Norm im $\mathbb{R}^n$ ist eine Funktion, die jedem Vektor $\mathbf{x} \in \mathbb{R}^n$ eine nichtnegative reelle Zahl $\|\mathbf{x}\| \in \mathbb{R}_0^+$ zuordnet, wobei $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\forall \alpha \in \mathbb{R}$ drei Bedingungen gelten müssen:

- Nur der Nullvektor hat Norm 0

$$\|\mathbf{x}\| = 0 \quad \Rightarrow \quad \mathbf{x} = 0$$

- Skalar α läst sich als Betrag herausheben

$$\|\alpha \cdot \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$$

- Die Dreiecksungleichung gilt

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$$

Norm und Distanz

Eine Norm kann auch die *Distanz* zwischen zwei Punkten $\mathbf{x}$ und $\mathbf{y}$ messen:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$$

- Taxis in Manhattan messen Strecken in der 1-Norm.

  Deswegen heißt die 1-Norm auch Taxi- oder Manhattan-Norm (engl. *taxicab norm*)
- Abstand in der Luftlinie entspricht der 2-Norm.
- Größter Unterschied in den Komponenten: ∞-Norm.

Matrixnormen

- Der Hauptberuf von Matrizen ist, Vektoren zu multiplizieren.
- Das Ergebnis einer Matrix-Vektor-Multiplikation ist wieder ein Vektor; der ist gewöhnlich länger oder kürzer und verdreht gegenüber Ausgangsvektor.
- Eine *Matrixnorm* misst als „Größe“ einer Matrix, wie stark sie die Länge von Vektoren ändern kann.
- Eine gegebene Matrix kann Vektoren nicht beliebig stark verlängern. Es gibt für jede Matrix einen „Maximal-Verlängerungs-Faktor“

Der „Maximal-Verlängerungs-Faktor“ ist eine Matrixnorm

Verschiedene Matrixnormen

Die 1-, 2- und ∞-Normen lassen sich von den entsprechenden Vektornormen ableiten: Sie geben für die Rechenoperation $\mathbf{y} = A \cdot \mathbf{x}$ an, um wieviel $\mathbf{y}$ gegenüber $\mathbf{x}$ *maximal vergrößert* wird. Eins- und ∞-Norm lassen sich aus den Matrixelementen einfach berechnen:

$\|A\|_1$	*Einsnorm*: maximale Spaltenbetragssumme
$\|A\|_\infty$	*Unendlich-Norm*: maximale Zeilenbetragssumme

Für die Matrix-Zweinorm lässt sich keine so einfache Rechenvorschrift angeben, obwohl gerade sie häufig verwendet wird.

MATLAB kann alle Normen leicht berechnen: $\|A\|_1$ = `norm(A,1)`, $\|A\|_2$ = `norm(A)`, $\|A\|_\infty$ = `norm(A,Inf)`.

Matrixnorm, allgemeine Definition

Matrizen lassen sich addieren und mit Skalaren multiplizieren. In diesem Sinn verhalten sie sich genauso wie Vektoren des $\mathbb{R}^n$. Alles, was sich wie ein Vektor verhält, können wir als „Vektor“ interpretieren: Die $m \times n$-Matrizen bilden einen *Vektorraum*. Der Begriff „Norm“ wird genau so definiert wie die Norm von Vektoren des $\mathbb{R}^n$. Vergleichen Sie die Definition einer Norm im $\mathbb{R}^n$ auf Seite 23 – sie wird hier nahezu wörtlich übernommen.

Eine *Norm* im $\mathbb{R}^m \times \mathbb{R}^n$ ist eine Funktion, die jeder $m \times n$-Matrix A eine nichtnegative reelle Zahl $\|A\| \in \mathbb{R}_0^+$ zuordnet, wobei $\forall A, B \in \mathbb{R}^m \times \mathbb{R}^n$, $\forall \alpha \in \mathbb{R}$ drei Bedingungen gelten müssen:

- Nur die Nullmatrix hat Norm 0:

$$\|A\| = 0 \quad \Rightarrow \quad A = 0$$

- Skalar α läst sich als Betrag herausheben:

$$\|\alpha \cdot A\| = |\alpha| \cdot \|A\|$$

- Die Dreiecksungleichung gilt:

$$\|A + B\| \leq \|A\| + \|B\|$$

Diese drei Grundregeln muss jede Norm erfüllen. Aber es gibt für die 1-, 2- oder ∞-Norm noch Bonus-Features. Für diese Matrix-Normen gelten nämlich noch folgende Rechenregeln:

$$\|A \cdot B\| \leq \|A\| \cdot \|B\| \tag{8}$$

$$\|A \cdot \mathbf{x}\| \leq \|A\| \cdot \|\mathbf{x}\| \tag{9}$$

Vergleiche Absolutbetrag: $|a \cdot b| = |a| \cdot |b|$

Frobeniusnorm:

Noch eine weitere Norm; Die Frobenius-Norm $\|A\|_F$ wird so ähnlich berechnet wie die Vektor-Zweinorm: *Quadrieren, summieren, Wurzel ziehen*

Frobenius-Norm: $\qquad \|A\|_F = \sqrt{\sum a_{ij}^2}$

Die Frobeniusnorm lässt sich leichter berechnen als die Matrix-Zweinorm und dient zu deren Abschätzung:

$$\|A\|_2 \leq \|A\|_F$$

Auch für $\|A\|_F$ gelten neben den Norm-Axiome noch die Rechenregeln

$$\|A \cdot B\|_F \leq \|A\|_F \cdot \|B\|_F \quad , \quad \|A \cdot \mathbf{x}\|_2 \leq \|A\|_F \|\mathbf{x}\|_2$$

MATLAB: $\|A\|_F =$ `norm(A,'fro')`.

Matrixnormen – das Kleingedruckte[6]

[6] Was hier dasteht, ist nicht wichtig,
wenn 's nicht dastünd', wär's nicht richtig.

Die lockere Erklärung *„Matrixnorm ist maximaler Verlängerungsfaktor“* ist mathematisch korrekt für 1-, 2- und ∞-Norm, wenn Vektorlängen in den jeweiligen Normen gemessen werden. Die Frobeniusnorm überschätzt aber gewöhnlich den maximal auftretenden Verlängerungsfaktor, wenn Vektorlängen in der 2-Norm gemessen werden. Immerhin liefert sie eine obere Schranke für den Verlängerungsfaktor.
Auch die Vorschrift $\|A\| = \max_{i,j} |a_{ij}|$ erfüllt die drei Bedingungen einer Norm, ist aber nicht immer eine obere Schranke für den Verlängerungsfaktor.

2.4 Konvergenz

Die Konvergenz der mehrdimensionalen Fixpunkt-Iteration hängt wie im eindimensionalen Fall mit dem Begriff der kontrahierenden Abbildung zusammen.

Konvergenz der Fixpunkt-Iteration im $\mathbb{R}^n$
Die Funktion $\mathbf{\Phi}(x)$ besitze einen Fixpunkt $\boldsymbol{\xi}$: $\mathbf{\Phi}(\boldsymbol{\xi}) = \boldsymbol{\xi}$. Sei ferner B eine offene Umgebung um den Fixpunkt in der Form $B = \{\mathbf{x} : \|\boldsymbol{\xi} - \mathbf{x}\| < r\}, \quad r > 0$. Wenn $\mathbf{\Phi}$ in B eine *kontrahierende Abbildung* in (irgend-) einer Norm $\|\cdot\|$ ist, d. h.,

$$\|\mathbf{\Phi}(\mathbf{x}) - \mathbf{\Phi}(\mathbf{y})\| \leq C\|\mathbf{x} - \mathbf{y}\| \,, \quad C < 1 \text{ für alle } \mathbf{x}, \mathbf{y} \in B \,,$$

dann konvergiert die Fixpunkt-Iteration $\mathbf{x}^{(k+1)} = \mathbf{\Phi}(\mathbf{x}^{(k)})$ mindestens linear gegen $\boldsymbol{\xi}$ für alle $\mathbf{x}^{(0)} \in B$.

Der Beweis erfolgt analog zu der eindimensionalen Form des Konvergenzsatzes. Auch der Begriff der Konvergenzordnung läßt sich unter Verwendung von Normen geradewegs auf den mehrdimensionalen Fall übertragen.

Kontraktion und Jacobi-Matrix

Das Konvergenzkriterium $|\phi'(\xi)| < 1$ im eindimensionalen Fall (vergleiche Seite 17) lässt sich auf den mehrdimensionalen Fall übertragen. Dazu fasst man die partiellen Ableitungen von $\mathbf{\Phi}$ in einer Matrix D_ϕ, genannt die *Jacobi -Matrix*, zusammen.

Jacobi-Matrix D_ϕ einer Funktion $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^n$

$$D_\phi = \begin{bmatrix} \frac{\partial \phi_1}{\partial x_1} & \frac{\partial \phi_1}{\partial x_2} & \cdots & \frac{\partial \phi_1}{\partial x_n} \\ \frac{\partial \phi_2}{\partial x_1} & \frac{\partial \phi_2}{\partial x_2} & \cdots & \frac{\partial \phi_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \phi_n}{\partial x_1} & \frac{\partial \phi_n}{\partial x_2} & \cdots & \frac{\partial \phi_n}{\partial x_n} \end{bmatrix}$$

Dann lässt sich ganz ähnlich wie im eindimensionalen Fall aussagen:

Das Fixpunktverfahren konvergiert lokal,
falls in der 1-,2-, Frobenius- oder ∞-Matrixnorm gilt

$$\|D_\phi\| < 1$$

2.5 Newton-Verfahren für Systeme

Gegeben sei eine vektorwertige Funktion $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$. Gesucht sei eine Nullstelle von $\mathbf{f}$. Das ist ein Vektor $\mathbf{x} \in \mathbb{R}^n$ als Lösung von

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Dies ist die allgemeine Formulierung eines Systems von n linearen oder nichtlinearen Gleichungen in n Unbekannten. Und noch einmal sei darauf hingewiesen: wir setzen Vektoren aus dem $\mathbb{R}^n$ und vektorwertige Funktionen in fetter Schrift $(\mathbf{x}, \mathbf{f}(\mathbf{x}), \ldots)$, zum Unterschied von Variablen und reellwertigen Funktionen $(x, f(x), \ldots)$.

Komponentenweise ausgeschrieben mit

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \quad \text{und} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{lautet das System} \quad \begin{array}{l} f_1(x_1, x_2, \ \ldots \ , x_n) = 0 \\ f_2(x_1, x_2, \ \ldots \ , x_n) = 0 \\ \qquad \ldots \\ f_n(x_1, x_2, \ \ldots \ , x_n) = 0 \end{array} .$$

Das Newton-Verfahren für Systeme führt die Lösung eines nichtlinearen Systems auf die Lösung einer Folge von linearen Gleichungssystemen zurück. Die Lösung von Systemen linearer Gleichungen ist vergleichsweise einfach gegenüber nichtlinearen Gleichungssystemen. Wir behandeln lineare Gleichungssysteme später noch ausführlich, aber einstweilen nehmen wir an, dass Sie aus der Mittelschule damit hinreichend vertraut sind.

Sofern die entsprechenden partiellen Ableitungen existieren, definieren wir die *Jacobi-Matrix* D_f von $\mathbf{f}$ durch

$$D_f = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Angenommen, ein Punkt $\mathbf{x}^{(0)}$ ist als Startwert in der Nähe einer Nullstelle gegeben. Dann lässt sich $\mathbf{f}$ in der Umgebung von $\mathbf{x}^{(0)}$ in linearisierter Näherung schreiben als (Taylorscher Lehrsatz für Funktionen mehrerer Veränderlicher)

$$\mathbf{f}(\mathbf{x}^{(0)} + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(0)}) + D_f(\mathbf{x}^{(0)}) \cdot \Delta\mathbf{x} + \mathbf{R}$$

mit einem Restglied $\mathbf{R}$, das im Limes $\Delta\mathbf{x} \to 0$ mit höherer Ordnung verschwindet. Wir vernachlässigen das Restglied und fordern $\mathbf{f}(\mathbf{x}^{(0)} + \Delta\mathbf{x}) = 0$. Aus der daraus entstandenen Gleichung

$$0 = \mathbf{f}(\mathbf{x}^{(0)}) + D_f(\mathbf{x}^{(0)}) \cdot \Delta\mathbf{x}$$

lässt sich der Korrekturvektor $\Delta\mathbf{x}$ bestimmen und damit eine verbesserte Näherung $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}$.

Das Newton-Verfahrens für Systeme lässt sich so formulieren:

Newton-Verfahren für Systeme
Gegeben eine differenzierbare vektorwertige Funktion $\mathbf{f}$ und ein Startwert $\mathbf{x}^{(0)}$. Gesucht eine Nullstelle von $\mathbf{f}$.

Iterationsvorschrift

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$$

mit $\Delta\mathbf{x}^{(k)}$ als Lösung von $D_f(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$

Auch dieses Verfahren ist ein Fixpunktverfahren, und zwar für die Funktion

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{x} - D_f^{-1}(\mathbf{x})\mathbf{f}(\mathbf{x}).$$

Notwendig für die Durchführbarkeit ist, dass D_f^{-1} existiert.

Man kann zeigen: Sofern D_f^{-1} an der Nullstelle existiert, konvergiert das Verfahren für genügend genaue Startwerte quadratisch.

Da es oft sehr mühsam ist, immer alle Elemente von D_f an jedem Punkt $\mathbf{x}^{(k)}$ zu berechnen, geht man manchmal so vor, daß man D_f an einem einzigen Punkt $\mathbf{x}^{(0)}$ berechnet und für den weiteren Verlauf des Verfahrens fix lässt. Dieses Verfahren heißt vereinfachtes Newton-Verfahren. Dafür muss $x^{(0)}$ bereits eine brauchbare Näherung sein. Das vereinfachte Newton-Verfahren konvergiert allerdings nur linear.

Das Newton-Verfahren für Systeme erfordert also in jedem Schritt die Lösung eines linearen Gleichungssystems. Das nächste Kapitel bringt die systematische Behandlung linearer Gleichungssysteme.

Beispiel: nichtlineares Gleichungssystem aus Abschnitt 2.2

Die Funktion $\mathbf{f}$ und ihre Jacobi-Matrix D_f sind hier

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 4x - y + xy - 1 \\ -x + 6y + \log(xy) - 2 \end{bmatrix}, \qquad D_f = \begin{bmatrix} 4+y & -1+x \\ -1+\frac{1}{x} & 6+\frac{1}{y} \end{bmatrix}.$$

Startwert $(1;1)$ eingesetzt liefert

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \qquad D_f = \begin{bmatrix} 5 & 0 \\ 0 & 7 \end{bmatrix}.$$

Zu lösen ist also das Gleichungssystem

$$\begin{bmatrix} 5 & 0 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Es liefert den Korrekturvektor und die verbesserte Lösung

$$\Delta\mathbf{x}^{(0)} = \begin{bmatrix} -0,6 \\ -0,428571 \end{bmatrix}, \quad \mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)} = \begin{bmatrix} 0,4 \\ 0,571429 \end{bmatrix}.$$

Der nächste Schritt wertet zuerst $\mathbf{f}$ und D_f für die neuen Werte von $\mathbf{x}$, löst das Gleichungssystem für den Korrekturterm $\Delta\mathbf{x}^{(1)}$ und errechnet daraus die verbesserte Näherung $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \Delta\mathbf{x}^{(1)}$. Die Matrix D_f hat aber hier nicht mehr so „schöne“ Einträge; das Gleichungssystem ist deswegen nicht so unmittelbar lösbar wie im ersten Schritt. Das vereinfachte Newtonverfahren würde zwar $\mathbf{f}$ neu auswerten, die Matrix D_f des ersten Schrittes beibehalten. Einfacherer Rechengang, aber langsamere (nur lineare statt quadratischer) Konvergenz!

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. Wenn das System aus n Gleichungen und Unbekannten besteht, dann ist A eine $n \times n$-Matrix.

Lösungsverfahren für lineare Gleichungssysteme lassen sich in zwei Hauptgruppen unterteilen: direkte und iterative Verfahren.

- Direkte Verfahren berechnen (rundungsfehlerfreie Rechnung vorausgesetzt) eine exakte Lösung. Eliminations- und Substitutionsverfahren sowie die Cramersche Regel fallen in diese Kategorie. Wenn Sie mit Papier und Stift Systeme mit zwei oder drei Unbekannten lösen wollen, sind solche direkte Verfahren die Methoden der Wahl. Computer können heutzutage problemlos für mehrere zehntausend Gleichungen und Unbekannten direkte Lösungsmethoden verwenden.
- Iterative Verfahren berechnen schrittweise immer bessere Näherungslösungen. Diese Methoden eigenen sich aber nur für Gleichungssysteme mit spezieller Matrix-Struktur. Computer lösen damit riesig große Gleichungssysteme (mehrere Millionen Unbekannte), wie sie zum Beispiel bei numerischer Strömungssimulation oder Festigkeitsberechnungen auftreten.

Dieses Kapitel behandelt direkte Verfahren und wiederholt (was aus Mathematik 1 bekannt sein sollte) theoretische Aussagen zur Existenz und Eindeutigkeit der Lösung; iterative Methoden behandelt Kapitel 4.

Software in anerkannt hoher Qualität ist in der Programmbibliothek LAPACK (`http://www.netlib.org/lapack/`) frei verfügbar. Sie werden auch in kommerziell angebotenen Paketen nichts Besseres finden. Auch MATLAB enthält die LAPACK-Algorithmen. (Übrigens ist MATLAB ursprünglich als einfache Benutzerschnittstelle zu diesem Programmpaket entstanden).

3.1 Dreiecksmatrizen

Wenn A eine untere oder obere *Dreiecksmatrix* ist, kann man das Gleichungssystem $A\mathbf{x} = \mathbf{b}$ einfach durch schrittweises Einsetzen lösen (Vorwärts- oder Rückwärtssubstitution).

Andernfalls transformiert man das Gleichungssystem auf Dreiecksgestalt, wie es Abschnitt 3.2 beschreibt. Andere Möglichkeit: man *faktorisiert* A zuerst in ein Produkt von Dreiecksmatrizen. Das entsprechende Verfahren beschreibt Abschnitt 3.5.

Wir bezeichnen Dreiecksmatrizen mit L und R. In der üblichen Notation sind in L 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 $\mathbf{b}$ auf einem Vektor `b[i]` gespeichert sind, löst das folgende Java-Programmsegment das Gleichungssystem $L\mathbf{x} = \mathbf{b}$ schrittweise durch *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 Dreiecksmatrixformulieren. Unterschiede zu vorhin: Die *Rückwärts-Substitution* beginnt in der letzten Zeile und schreitet von unten nach oben fort; durch das Hauptdiagonalelement wird dividiert; weiters ist hier die rechte Seite anfangs bereits auf `x[ ]` gespeichert, der Algorithmus überschreibt `x[ ]` mit dem Lösungsvektor.

```
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 allein der quadratische Term ausschlaggebend. Wir sagen daher

Rechenaufwand bei Vorwärts- oder Rückwärts-Substitution
Lösen eines $n \times n$-Dreieckssystem erfordert $O(n^2)$ Punktoperationen.

Der Rechenaufwand wächst also quadratisch mit der Anzahl der Gleichungen.

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).

Darum geht es im nächsten Abschnitt.

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. Es wird sich später, in Abschnitt 3.5, herausstellen, dass diese Einträge eine wichtige Bedeutung haben.

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.

Als JAVA Code sieht Gauß-Elimination verblüffend einfach aus. Zu Beginn muss in `x[ ]` die rechte Seite gespeichert sein. In drei geschachtelte Schleifen wird schließlich die Lösung auf `x[ ]` geschrieben.

```
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 wächst kubisch mit der Anzahl der Gleichungen.
Gauß-Elimination löst ein $n \times n$-System $A\mathbf{x} = \mathbf{b}$ mit $O(n^3)$ Rechenoperationen.

(Genau nachgezählt sind es $n^3/3 + n^2 - n/3 = n^3/3 + O(n^2)$ Punktoperationen.)

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,...,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)[7]. Die Empfindlichkeit gegenüber Rundungsfehlern ist bei Zeilen-Pivotsuche etwas höher als bei vollständiger Pivot-Suche.

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

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

sehen Sie hoffentlich mit freiem Auge: Beim ersten und beim zweiten ist $x = 1, y = 1$ eine Lösung. Das dritte System ist unlösbar. Beim ersten System gibt es allerdings noch unendlich viele weitere Lösungen. Diese beispiele illustrieren den allgemeinen Fall.

Lösbarkeit linearer Systeme
Für ein lineares Gleichungssystem gilt genau eine von drei Aussagen: Es gibt

- unendlich viele Lösungen;
- eine eindeutige Lösung;
- keine Lösung.

(Sie haben das in der Mathematik-Grundvorlesung gelernt!)

> Dieser Abschnitt behandelt nur Systeme mit genausoviel Gleichungen wie Unbekannten, aber die obige Aussage gilt auch für lineare Systeme mit mehr Gleichungen als Unbekannten. Bei weniger Gleichungen als Unbekannten sind nur die zwei Fälle möglich: keine Lösung oder unendlich viele Lösungen.

Das Gaußsche Eliminationsverfahren kann entscheiden, welcher Fall vorliegt und mögliche Lösungen berechnen.

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*: Von jeder Zeile zur nächsten nimmt (von links her gesehen) die Zahl der führenden Nullen um mindestens eins zu.

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: *eine eindeutige Lösung.*

[7] Sie finden in Wikipedia unter dem Stichwort *Gaußsches Eliminationsverfahren* Pseudocode in mehreren Varianten.

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.

Wenn ein Computer die Elimination in Gleitkomma-Arithmetik durchführt, 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. Es ist überhaupt nicht trivial, eine Schranke anzugeben, ab der Einträge als Null anzusehen sind. Dubiose Grenzfälle, in denen das Eliminationsverfahren gerade noch eine Lösung findet, obwohl Matrixzeilen schon fast null sind, heißen *numerisch singulär*.

3.4.2 Rang der Matrix und der erweiterten Matrix

Der *Rang einer $m \times n$-Matrix* ist die Anzahl ihrer linear unabhängigen Zeilen- oder Spaltenvektoren.

> Auch wenn die Matrix unterschiedlich viele Zeilen und Spalten hat, gilt: es gibt immer genau so viele linear unabhängige Zeilen wie Spalten; kurz: Zeilenrang = Spaltenrang.

Der MATLAB-Befehl `rank(A)` bestimmt den Rang der $n \times n$-Matrix A, und `rank([A,b])` den Rang der *erweiterten Koeffizientenmatrix* (der Matrix des Gleichungssystems mit rechter Seite als letzte Spalte angefügt).

Rang und Lösbarkeit: Fallunterscheidungen

- `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 mit „schöneren“ Zahlen bei einfachen Beispielmatrizen. Allerdings rechnet diese Variante numerisch weniger zuverlässig. (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, $x_3 = 1 - 2\mu, x_1 = -2 - 2\lambda + 2\mu$. In Vektorform:

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

Bei so einer einfachen Matrix kann MATLAB eine partikuläre Lösung auch in der Form `x=A\b` und den Nullraum mittels `null(A,'r')` berechnen. Dieses Vorgehen ist bei praktischen Problemen mit realen Daten nicht zu empfehlen, aber hier liefert es (abgesehen von einer Warnmeldung) das „schöne“ Ergebnis

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

`pinv(A)*b` und `null(A)`liefern die aus numerischer Sicht vorteilhafteste Darstellung,

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

MATLAB berechnet dieses Resultat mit aufwändigen und zuverlässigen numerischen Verfahren. Aber anders als bei den vorigen Darstellungen der Lösung ergibt probeweises Einsetzen in den Ausdruck $A\mathbf{x} - \mathbf{b}$ nicht exakt Null, sondern aufgund der Rundungsfehler Werte im Bereich 1×10^{-15}. (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[8] $\det A = -9480{,}580$ 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. MATLABs Wert für die Determinante ist ziemlich falsch.

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 $\det A = 5{,}3673 \times 10^{-18}$, und das könnte man mit voller Berechtigung gerundet für $\det A = 0$ 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 Programm auf Seite 30 zum Eliminationsverfahren ist, sich die Zwischenresultate `p` zu merken. 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.

[8] mit Version 2022b. Der Wert mit Version 2021b war $\det A = -10\,611$. Ältere Versionen so um 2018 lieferten $\det A = -9478{,}9$; die 2015-Version bringt $-9448{,}8$; vor 2013 war $\det A = -13\,017$. Es sollte Sie beunruhigen, dass ein Rechenergebnis je nach Programmversion so unterschiedlich ausfällt!

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.

Siehe Übungsunterlagen für weitere Informationen!

Für die LR-Zerlegung braucht man keine rechte Seite. Die kommt erst später ins Spiel. Zur Lösung des Systems $A\mathbf{x} = \mathbf{b}$, wenn $A = LR$ bereits gegeben ist, 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. Die *Cholesky-Zerlegung*

$$A = LL^T$$

ist für symmetrisch positiv definiter Matrizen das Standardverfahren.

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 30.)

$n = 3$; 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 für x_3 in zweiter Zeile

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

Einsetzen für x_2 und x_3 in erster 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[9] 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.

[9] Dem deutschen Fachbegriff Links-Rechts-Zerlegung entspricht auf Englisch *lower–upper (LU) decomposition* oder *factorization*. Deswegen heißt der entsprechende MATLAB-Befehl `lu` und nicht `lr`.

Zerlegung mit dem Befehl `[L, U]=lu(A)` liefert

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

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 für eine $n \times n$-Matrix: $n^3/3 + 2n/3 - 1$ 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. Ist etwa der Vektor $\mathbf{x} = A^{-1}\mathbf{y}$ gefragt, so erhält man $\mathbf{x}$ auch genauso gut als Lösung des Gleichungssystems $A\mathbf{x} = \mathbf{y}$, 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 alle 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 $(4n^3 - n)/3$.

Der *Gauß-Jordan-Algorithmus* ist eine speziell günstig organisierte Form des Eliminationsverfahrens; er eignet sich gut zur Berechnung mit Stift und Papier. (Sie kennen dieses Verfahren aus Mathematik 1.)

3.7.3 Symmetrisch positiv definite Matrizen

Einfache Argumente der linearen Algebra zeigen: Für symmetrisch positiv definite Matrizen bricht die einfache Gauß-Elimination nie wegen $a_{kk} = 0$ 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 (wie bereits auf Seite 36 erwähnt) 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, können die Faktoren L und R deutlich mehr Nichtnull-Einträge enthalten. Durch Gauß-Elimination werden zusätzliche *Fällterme* erzeugt (also Elemente $\neq 0$ an Stellen, wo die Ausgangsmatrix Elemente $= 0$ hatte). Wenn man alle (oder kleine) Fällterme 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 (mehr Informationen im Übungsteil): 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 Fällterme 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 spärlich 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 leicht veränderten Systems wird um ein (hoffentlich nicht zu großes) $\delta\mathbf{x}$ 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* $\kappa(A)$ misst, wie empfindlich in einem System $A\mathbf{x} = \mathbf{b}$ der relative Fehler von $\mathbf{x}$ von kleinen relativen Änderungen in A und $\mathbf{b}$ abhängt.

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right)$$

In der obigen Ungleichung bezeichnet $\|\cdot\|$ sowohl eine Vektornorm (zum Beispiel 1-, 2- oder ∞-Norm) als auch die entsprechende Matrixnorm. Aus den Rechenregeln für Vektor- und Matrixnormen (vergleiche Abschnitt 2.3) lässt sich herleiten:

$$\kappa(A) = \|A\|\|A^{-1}\|$$

Beweisskizze: Beginne mit dem gestörten System

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

löse Klammern auf

$$A\mathbf{x} + A \cdot \delta\mathbf{x} + \delta A \cdot \mathbf{x} + \delta A \cdot \delta\mathbf{x} = \mathbf{b} + \delta\mathbf{b}\,;$$

weil $A\mathbf{x} = \mathbf{b}$, können wir links $A\mathbf{x}$ und rechts $\mathbf{b}$ streichen. Für kleine $\delta\mathbf{b}$ und δA ist das Produkt $\delta A \cdot \delta\mathbf{x}$ von höherer Ordnung klein; wir vernachlässigen ihn hier. Somit bleibt

$$A \cdot \delta\mathbf{x} + \delta A \cdot \mathbf{x} = \delta\mathbf{b}\ .$$

Daraus lässt sich $\delta\mathbf{x}$ ausdrücken:

$$\delta\mathbf{x} = A^{-1}\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\ ,$$

wende auf beiden Seiten eine Vektornorm an,

$$\|\delta\mathbf{x}\| = \|A^{-1}\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\|\,,$$

nütze eine Eigenschaft der Matrixnorm, Ungleichung (9),

$$\|\delta\mathbf{x}\| \le \|A^{-1}\| \cdot \|\left(\delta\mathbf{b} - \delta A \cdot \mathbf{x}\right)\|\,,$$

verwende die Dreiecksungleichung,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\|\left(\|\delta\mathbf{b}\| + \|\delta A \cdot \mathbf{x}\|\right)\ ,$$

erweitere die Terme in der Klammer,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\|\left(\frac{\|A\mathbf{x}\|}{\|\mathbf{b}\|}\|\delta\mathbf{b}\| + \frac{\|A\|}{\|A\|}\|\delta A \cdot \mathbf{x}\|\right)\ ,$$

vewende noch einmal eine Ungleichung für die Matrixnormen und hebe $\|A\|$ heraus,

$$\|\delta\mathbf{x}\| \le \|A^{-1}\| \cdot \|A\|\left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}\|\mathbf{x}\| + \frac{\|\delta A\|}{\|A\|}\|\mathbf{x}\|\right)\ ,$$

eine letzte Division durch $\|\mathbf{x}\|$, und wir sind fertig:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \|A^{-1}\| \cdot \|A\|\left(\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\delta A\|}{\|A\|}\right)\ .$$

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.

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

$$\kappa(A) \ge \frac{\max|\lambda|}{\min|\lambda|}$$

(Verhältnis von größtem zu kleinsten Eigenwert-Bertag; Eigenwerte werden in Abschnitt 9 behandelt).

4 Iterative Verfahren für lineare Gleichungssysteme

Gegeben sei ein lineares Gleichungssystem in n Gleichungen und Unbekannten.

$$\begin{array}{lcl} a_{11}x_1 + a_{12}x_2 + & \ldots & + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + & \ldots & + a_{2n}x_n = b_2 \\ \vdots & & \vdots \quad\; \vdots \\ a_{n1}x_1 + a_{n2}x_2 + & \ldots & + a_{nn}x_n = b_n \end{array} \tag{10}$$

In Matrixschreibweise

$$A\mathbf{x} = \mathbf{b}\,. \tag{11}$$

Das klassische Lösungsverfahren dafür, jedenfalls für Systeme von zwei bis einigen hundert Gleichungen, ist Gauß-Elimination. Sie wird in Kapitel 3 systematisch behandelt. Aus vielen Anwendungen (Strömungssimulation, Seismik, Computertomographie, Festigkeitsrechnungen mit finiten Elementen...) resultieren Gleichungssysteme mit vielen tausend oder sogar Millionen Unbekannten. Solche Systeme werden meist iterativ gelöst. Sie lernen hier nur einige Basis-Verfahren kennen, auf denen die leistungsfähigeren Methoden aufbauen.

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

Angenommen, die Diagonalelemente a_{ii} einer $n \times n$-Matrix A sind alle ungleich null. Dann wäre folgendes Rezept zur Lösung von $A\mathbf{x} = \mathbf{b}$ möglich (ein Fixpunkt-Verfahren):

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

Ausführlicher in der komponentenweisen Schreibweise (10) formuliert: 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

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

Vielleicht finden Sie Matrix-Schreibweise übersichtlicher. Dazu definieren wir 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:

$$A = D + E \text{ mit } D = [d_{ij}], \quad d_{ij} = \begin{cases} a_{ii} & \text{falls } i = j, \\ 0 & \text{sonst.} \end{cases} \qquad E = A - D\,. \tag{12}$$

Das Gleichungssystem (11) lässt sich dann äquivalent umformen zu

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

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

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{13}$$

heißt *Jacobi-Verfahren* .is called *Jacobi method* .

Iterationsschritt des Jacobi-Verfahrens
In Matrix-Schreibweise für Zerlegung $A = D + E$:

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

Komponentenweise geschrieben: 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 nützt nicht die aktuellste Information zur Berechnung von $x_i^{(k+1)}$. Beispielsweise verwendet es $x_1^{(k)}$ zur Berechnung von $x_2^{(k+1)}$, obwohl $x_1^{(k+1)}$ bereits verfügbar ist. Wenn wir das Verfahren so formulieren, dass es immer die aktuellsten Näherungswerte an die x_i verwendet, erhalten wir das *Gauß-Seidel-Verfahren* .

Für die Matrix-Schreibweise des Gauß-Seidel-Verfahrens definieren wir eine Matrix $C = [c_{ij}]$ mit den gleichen Elementen wie A in und unterhalb der Hauptdiagonale und Null oberhalb der Hauptdiagonale. Die restlichen Elemente von A schreiben wir in eine Matrix E:

$$A = C + E \text{ mit } C = [c_{ij}], \quad c_{ij} = \begin{cases} a_{ij} & \text{falls } i \geq j, \\ 0 & \text{sonst.} \end{cases} \qquad E = A - C \ . \tag{14}$$

Dieselben Schritte, die für das Jacobi-Verfahren zur Fixpunkt-Gleichung 13 geführt haben, können wir mit der Matrix C statt D wiederholen und erhalten die Iterationsvorschrift für das Gauß-Seidel-Verfahren:

$$\mathbf{x}^{(k+1)} = C^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{15}$$

Iterationsschritt des Gauß-Seidel-Verfahrens
einfach formuliert
Löse der Reihe nach jede Gleichung nach ihrem Diagonal-Term auf, setze Startwerte ein, iteriere, verwende jeweils neuerste Näherungswerte.

Matrix-Schreibweise für Zerlegung $A = C + E$

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

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

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

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*)

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.

Iterationsschritt des SOR-Verfahrens
einfach 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: $x_i^{(k+1)} = \omega y_i^{(k+1)} + (1 - \omega) x_i^{(k)}$

Die Komponenten-Schreibweise sieht hier bereits etwas unübersichtlich aus.

für $i = 1, \ldots, n$

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

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

Auch dieses Verfahren lässt sich mit einer Zerlegung $A = B + E$ ähnlich wie die Gleichungen 13 und 15 anschreiben:

$$\mathbf{x}^{(k+1)} = B^{-1}(\mathbf{b} - E\mathbf{x}^{(k)}) \tag{16}$$

Darin ist B eine Kombination der Matrizen C und D von vorhin (12, 14), und zwar

$$B = C + \left(\frac{1}{\omega} - 1 \right) D$$

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 $A = [a_{ij}]$ heißt *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 eindeutigen Lösung.

Beweis: Wir zeigen, dass die zur Iterationsvorschrift (13) 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. Laut Abschnitt 2.4 ist damit Konvergenz gewährleistet. 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

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

Die Summe der Elementbeträge in dieser Zeile ist < 1 (Begründung: Aus der Summe $1/|a_{ii}|$ herausheben, Diagonaldominanz ausnützen). Damit ist auch für die Zeilensummen- oder Unendlich-Norm der Matrix $D^{-1}E$ gezeigt:

$$\|D^{-1}E\|_\infty < 1 \ .$$

Aus einer Eigenschaft der Matrixnorm, Ungleichung (9), folgt sofort die Kontraktionseigenschaft

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

mit $C = \|D^{-1}E\|_\infty < 1$.

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.

Es gibt aber auch Matrizen, für die ein Verfahren konvergiert, das andere aber nicht.

Wir zitieren hier ohne Beweis zwei weitere Sätze, die Konvergenzbedingungen formulieren.

> 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 (*Satz von Stein und Rosenberg*).
>
> Ist A symmetrisch 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*, englisch *sparse matrix*). 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.

Diese Unterlagen können nur eine einführende Übersicht auf einige Prinzipien geben, die moderne iterative Gleichungslöser verwenden.

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.

Allgemein gilt, je besser $\tilde{A}$ die Original-Matrix approximiert, um so rascher konvergiert ein solches iterative Verfahren. Besonders gute Splittings entstehen aus *unvollständiger LR-Zerlegung*. Diese Methoden werden bei den Eliminationsverfahren in Abschnitt 3.7.4 kurz 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 sollte. Eine algebraisch gleichwertige, aber für Rechner geeignete Form lautet

Iterativer Gleichungslöser, 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*. 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 tatsächlich 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 $\mathbf{x}_{\text{exc}} - \mathbf{x}^{(k)}$ zwischen exakter und genäherter Lösung klein ist. Es gilt beispielsweise, falls A symmetrisch ist, die Abschätzung

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

wobei $\lambda_{\max}$ und $\lambda_{\min}$ 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, zwar in eine günstige Richtung zeigen, aber nicht die passende Länge haben. Anstatt den Näherungsvektor $\mathbf{x}^{(k)}$ pro Iterationsschritt nur um den Vektor $\mathbf{d}^{(k+1)}$ zu korrigieren, kann man daher versuchen, gleich ein Vielfaches ω dieser Korrektur anzubringen. (Eine ähnliche Idee verwendet auch schon das SOR-Verfahren.)

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 in jedem Schritt 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 alle diese Ideen (Präkonditionierun, Minimierung, Orthogonalisierung) verwendet. Inzwischen wurde eine Fülle von Verfahren entwickelt, die auf ähnlichen Prinzipien beruhen und heute alle als *Krylov-Unterraum-Verfahren* bezeichnet werden.

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.

Für unsymmetrische Matrizen sind GMRES (für *generalized minimal residual method*), BiCG (für *biconjugate gradients*) oder CGS (für *conjugate gradient squared*) gängige Verfahren.

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 rechteckige $n \times m$-Matrix mit $n > m$, hat also mehr Zeilen als Spalten. In der Regel hat ein solches System keine exakte Lösung, aber eine eindeutig bestimmte „am wenigsten falsche“ *Ausgleichslösung nach der Methode der kleinsten Quadrate*.

Regelfall: m linear unabhängige Spalten in A, $m+1$ linear unabhängige Spalten in der erweiterten Koeffizientenmatrix $[A, \mathbf{b}]$.
Sonderfälle:

- $\operatorname{rank} A = \operatorname{rank}[A, \mathbf{b}] = m \rightarrow$ eindeutige exakte Lösung.

Systeme ohne vollen Spaltenrang; die nachfolgend beschriebenen Normalengleichungen sind nicht eindeutig lösbar.

- $\operatorname{rank} A = \operatorname{rank}[A, \mathbf{b}] < m \rightarrow$ unendlich viele exakte Lösungen;
- $\operatorname{rank} A < \operatorname{rank}[A, \mathbf{b}] < m \rightarrow$ keine exakte Lösung, unendlich viele gleichberechtigte Ausgleichslösungen.

Überbestimmte Systeme entstehen beispielsweise bei der Auswertung von Messergebnissen, wenn mehr Messungen vorliegen als Parameter gesucht sind.

5.1 Normalengleichungen

Überbestimmte Systeme
Ausgleichslösung 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 $(\|\mathbf{r}\|_2)^2 = \mathbf{r}^T \cdot \mathbf{r}$ nach den einzelnen Komponenten von $\mathbf{x}$; Nullsetzen der Ableitung.

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

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

Das heißt, für keinen anderer Vektor gibt es einen kleineres Residuum[10]. In diesem Sinn ist $\hat{\mathbf{x}}$ eine optimale Ausgleichslösung 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

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

[10] Im Regelfall sind alle Spaltenvektoren in A linear unabhängig. Dann gilt sogar strikt $\|\mathbf{b} - A\mathbf{x}\|_2 > \|\mathbf{b} - A\hat{\mathbf{x}}\|_2$ und $\hat{\mathbf{x}}$ ist die eindeutig bestimmte beste Ausgleichslösung.

Damit berechnen wir das innere Produkt $\mathbf{r}^T \cdot \mathbf{r}$ und den Zusammenhang mit $\hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}}$.

$$\begin{aligned}\mathbf{r}^T \cdot \mathbf{r} &= (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (\hat{\mathbf{r}} + A(\hat{\mathbf{x}} - \mathbf{x})) \\ &= \hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}} + \hat{\mathbf{r}}^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x})) + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot \hat{\mathbf{r}} + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x})) \\ &= \hat{\mathbf{r}}^T \cdot \hat{\mathbf{r}} + (\hat{\mathbf{r}}^T A) \cdot (\hat{\mathbf{x}} - \mathbf{x})) + (\hat{\mathbf{x}} - \mathbf{x})^T \cdot (A^T \hat{\mathbf{r}}) + (A(\hat{\mathbf{x}} - \mathbf{x}))^T \cdot (A(\hat{\mathbf{x}} - \mathbf{x}))\end{aligned}$$

Weil $\hat{\mathbf{x}}$ laut Voraussetzung die Normalengleichungen erfüllt, ist $A^T \hat{\mathbf{r}} = 0$, ebenso gilt $\hat{\mathbf{r}}^T A = 0$. Es bleibt also

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

Der letzte Term ist (als inneres Produkt eines Vektors mit sich selbst) immer ≥ 0[11]. Damit ist die Aussage bewiesen.

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 (wenn die Spalten von A linear unabhängig sind) 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

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

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 das klassische Standardverfahren, 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. Programmpakete und Rechenumgebungen (wie MATLAB) verwenden zumeist 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 lineare Gleichungen in zwei Unbekannten,

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

Das Gleichungssystem in Matrixschreibweise lautet

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

[11] Sind alle Spaltenvektoren in A linear unabhängig, dann ist mit $\mathbf{x} \neq \hat{\mathbf{x}}$ auch $A(\hat{\mathbf{x}} - \mathbf{x}) \neq 0$; der letzte Term ist dann echt > 0.

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):

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$, 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.

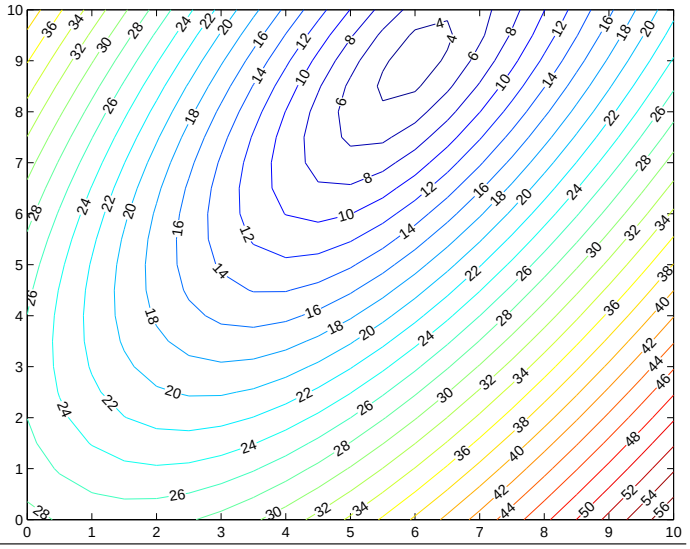
Ändert man beispielsweise den Vektor $\mathbf{x}$ geringfügig auf

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

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 optimal.

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

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

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

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

Das transformierte System lautet in diesem Fall

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

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 $a_0 + a_1x_1 + a_2x_2 = y$. 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).

6 Approximation von Daten, Ausgleichsrechnung

6.1 Lineare Datenmodelle

Ein Beispiel dazu hat schon Kapitel 5.4 gebracht. Die Vorlesungsfolien und die Übungen bringen weitere Beispiele dazu. Das best-angepasste Modell ergibt sich dabei immer aus der kleinste-Quadrate-Näherung an ein überbestimmtes Gleichungssystem.

Aber nicht immer liefert die Methode der kleinsten Fehlerquadrate eine plausible Anpassung. Einige wenige grob falsche Werte in den Daten („Ausreißer“) können das Ergebnis gewaltig verzerren. Im Kapitel 6.6 wird eine robuste Methode vorgestellt. MATLAB bietet in seinen Toolboxen verschiedene Methoden zur robusten Anpassung (*robust fit*) an.

6.2 Polynomiale Regression

Hier handelt es sich um einen wichtigen Spezialfall der linearen Datenmodelle, für den sich die allgemeinen Formeln etwas vereinfachen.

Eine typische Aufgabe zu diesem Abschnitt könnte lauten: Gegeben sind Messergebnisse für einen Satz von Temperaturwerten T und die entsprechenden Widerstandswerte R eines Temperaturfühlers. Der Zusammenhang zwischen R und T lässt sich näherungsweise in der Form $R = a + bT + cT^2$ beschreiben. Wenn für genau drei (verschiedene) T-Werte Daten vorliegen, lassen sich die drei Parameter a, b und c eindeutig bestimmen. Es ist aber sinnvoll, mehr Messungen durchführen, damit Messfehler der Einzelmessungen nicht so stark ins Gewicht fallen. Die Parameter a, b und c werden dann durch *Ausgleichsrechnung* (man sagt auch *Regression*) bestimmt.

Polynomiale Regression (Ausgleich durch ein Polynom)
Gegeben: $m + 1$ Wertepaare $(x_i, y_i), i = 0, \ldots, m$
Gesucht: $p(x)$, ein Polynom n-ten Grades, $n < m$, so dass die Summe der Fehlerquadrate

$$\sum_{i=0}^{m} (p(x_i) - y_i)^2$$

minimal wird. Locker formuliert: $y = p(x)$ approximiert möglichst gut die Datenpunkte.

Nicht immer ist ein Polynomansatz ein geeignetes Modell, aber oft lässt sich ein scheinbar komplizierteres Modell auf ein polynomiales Modell zurückführen (Beispiele in den Übungen).

Direkter Lösungsweg: Ansatz des Polynoms mit unbestimmten Koeffizienten,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x_n \ .$$

Einsetzen der gegebenen Wertepaare führt auf ein System von $m + 1$ linearen Gleichungen in den $n + 1$ unbekannten Koeffizienten $a_0, a_1, \ldots, a_n$.

Sofern $n < m$, liegt ein *überbestimmtes System* vor.

- klassische Lösung: Näherung nach der Methode der Normalengleichungen: lässt sich bei kleinen Beispielen mit Papier und Stift rechnen; bei großen Datenmengen Gefahr von Rundungsfehlern.

  Die Normalengleichungen sind aber nur dann eindeutig lösbar, wenn in den insgesamt $m + 1$ Wertepaaren mindestens $n + 1$ der x-Werte verschieden sind.

- moderner Lösungsweg: QR-Zerlegung. Praktisch nur am Rechner durchführbar. Bessere Konditionszahl, weniger anfällig für Rundungsfehler.

Dieser Lösungsweg (Ansatz mit unbestimmten Koeffizienten, Aufstellen des überbestimmten Systems, Bilden der Normalengleichungen) lässt sich für die polynomiale Regression etwas abkürzen. Setzt man

$$s_0 = m + 1, \quad t_0 = \sum_{i=0}^{m} y_i$$

und

$$s_k = \sum_{i=0}^{m} x_i^k, \quad t_k = \sum_{i=0}^{m} x_i^k y_i \quad \text{für } k > 0,$$

so lassen sich, wie man leicht herleiten kann, die Normalengleichungen in folgender Gestalt schreiben:

$$\begin{array}{lllllll}
s_0 a_0 & + & s_1 a_1 & + & \ldots\ldots & + & s_n a_n \quad = t_0 \\
s_1 a_0 & + & s_2 a_1 & + & \ldots\ldots & + & s_{n+1} a_n \quad = t_1 \\
 & & & & \ldots\ldots & & \\
s_n a_0 & + & s_{n+1} a_1 & + & \ldots\ldots & + & s_{2n} a_n \quad = t_n
\end{array}$$

Die Normalengleichungen können für größere n ziemlich schlecht konditioniert sein. Bessere Resultate lassen sich in solchen Fällen durch Entwicklung nach orthogonalen Polynomen erzielen.
Verwendet man beispielsweise als Datensatz die Punkte

$$(x, \exp(x)) \quad \text{für } x = 0;\, 0,01;\, 0,02; \ldots; 3,99; 4$$

und approximiert die Daten durch Regressionspolynome verschieden hohen Grades (Rechnung mit vierzehnstelliger Genauigkeit), so wird die Güte der Approximation vorerst mit steigendem Grad besser. Ab dem dreizehnten Grad aber wachsen die Fehler wieder an. Das aus den Normalengleichungen berechnete Polynom 25-ten Grades hat kaum mehr Ähnlichkeit mit der approximierten Funktion. Verwendet man orthogonale Polynome (Tschebyscheff-Polynome), treten diese numerischen Probleme nicht auf.

6.3 Ausgleichsgerade

Ein wichtiger Spezialfall der obigen Problemstellung: An $m + 1$ (mehr als zwei) gegebene Datenpunkte soll eine Gerade mit Gleichung $y = a + bx$ so angepasst werden, dass die Summe der Fehlerquadrate minimal wird.

$$a = \frac{s_2 t_0 - s_1 t_1}{s_0 s_2 - s_1^2}$$

$$b = \frac{s_0 t_1 - s_1 t_0}{s_0 s_2 - s_1^2}$$

Das Finden einer Ausgleichsgeraden wird oft auch als „lineare Regression“ bezeichnet. Das ist aber ein mißverständlicher Terminus, weil er leicht zur Verwechslung mit „linearen Datenmodellen“ führt.

6.4 Nichtlineare Datenmodelle

Dazu gibt es Material auf den Vorlesungsfolien und ausführlicher in den Übungsunterlagen. Kurzfassung: Jacobi-Matrix bilden und iterieren. Im Unterschied zum Newton-Verfahren, das Sie schon kennen, ist das lineare Gleichungssystem mit der Jacobimatrix nun überbestimmt und wird näherungsweise im Sinn der kleinsten Fehlerquadrate gelöst.

Dieses Verfahren wird als *Gauß-Newton-Verfahren* bezeichnet.

Bei nichtlinearen Ausgleichsproblemen gibt es neben dem Gauß-Newton-Verfahren noch weitere Verfahren (z. B. Levenberg-Marquardt-Algorithmus). Damit beschäftigt sich die Optimierung als Teilgebiet der Angewandten Mathematik. Dieses Skript kann nicht näher darauf eingehen.

6.5 Warum „kleinste Quadrate“

Die Methode der kleinsten Quadrate minimiert die euklidische Länge des Residuenvektors. Man kann aber die Größe des Residuenvektors durchaus auch anders messen und entsprechend andere Minimalbedingungen fordern. Wichtige Beispiele: Die Summe der *Absolutbeträge* der Fehler soll minimal werden, oder der *maximale Fehler* soll minimal werden („Minimax-Approximation“). Zwei Gründe sprechen für die kleinsten Quadrate:

- Einfache Herleitung und Durchführung: die Minimalwert-Aufgabe lässt sich mit elementarer Differentialrechnung lösen und führt auf ein einfaches algebraisches Problem

- Statistische Überlegungen: Wenn die Daten mit unabhängigen, zufälligen, normalverteilten Fehlern mit gleicher Standardabweichung behaftet sind, sind kleinste Quadrate in gewissem Sinn optimal (genauer: Die Methode liefert eine *maximum likelihood*-Schätzung der Parameter). Umgekehrt gilt aber: wenn die Fehler in den Daten *nicht* normalverteilt etc. sind, dann sind kleinste Quadrate unter Umständen ziemlich schlecht; siehe unten.

- Weitere statistische Eigenschaften: Wenn man eine Abschätzung für die Genauigkeit der Daten hat, kann man auf die Genauigkeit der berechneten Modellparameter schließen.

  > Angenommen, die Daten sind so skaliert, dass die Varianz der Messfehler gleich 1 ist. Sei $C = (A^T A)^{-1}$ die inverse Matrix des Systems der Normalengleichungen. Die Diagonalelemente von C sind die Varianzen der entsprechenden Modellparameter; die Elemente außerhalb der Hauptdiagonale sind die entsprechenden Kovarianzen.

Und was spricht dagegen?

- Die Methode reagiert empfindlich auf „Ausreißer“ in den Daten. Das Quadrieren der Fehler bestraft grosse Abweichungen streng. Deswegen ist die Methode der kleinsten Quadrate bereit, eine Kurve wild zu verzerren, um ein paar weit außen liegende Datenpunkte auch noch annähernd zu erreichen. Ein paar Ausreißer in ansonsten vernünftigen Daten können so eine völlig unsinnige Approximation bewirken.

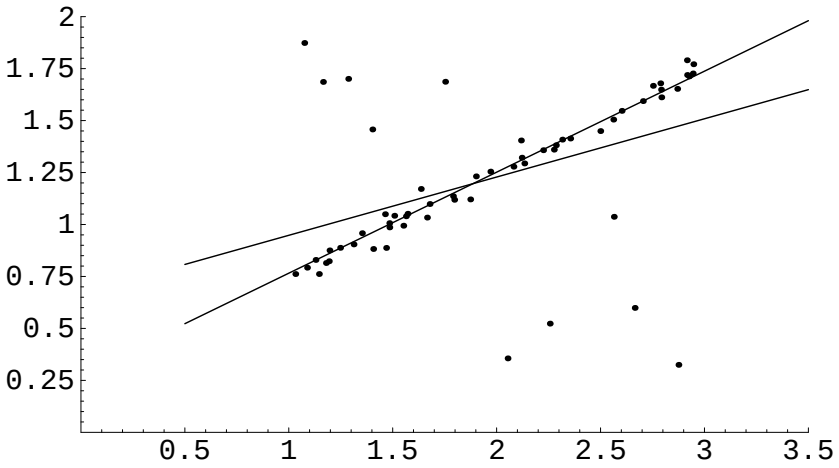


Abbildung 8: Anpassen einer Geraden an Datenpunkte. Die Ausgleichsgerade nach der Methode der kleinsten Quadrate lässt sich von den wenigen Ausreißern stark ablenken. Minimieren des absoluten Fehlers legt eine wesentlich plausiblere Gerade durch die Daten.

6.6 Ausgleichsgerade mit Minimieren der absoluten Fehler (L_1-Norm)

Gegeben: m Wertepaare $(x_i, y_i), i = 1, \ldots, m$
Gesucht: Eine Gerade in der Form $y = a + bx$, so dass die Summe der absoluten Fehler

$$\sum_{i=1}^{m} |p(x_i) - y_i|$$

minimal wird.
Die Lösung lautet: Für gegebenes b ergibt sich a als Median eines Datenfeldes,

$$a = \text{median}\{y_i - bx_i\}$$

Den Parameter b findet man als Lösung der Gleichung

$$0 = \sum_{i=1}^{m} x_i \,\text{sgn}(y_i - a - bx_i)$$

(wobei sgn(0) als Null interpretiert werden soll). Wenn man für a in dieser Gleichung die durch die vorigen Gleichung bestimmte Funktion $a(b)$ einsetzt, bleibt eine Gleichung in einer Unbekannten übrig. Intervallhalbierung (siehe Kapitel 1.7) ist die geeignete Lösungsmethode dafür.

6.7 Ausgleichsgerade mit Minimieren der Normalabstände

Dazu gibt es unter dem Titel *Total Least Squares* Material auf den Vorlesungsfolien und den Übungsunterlagen.