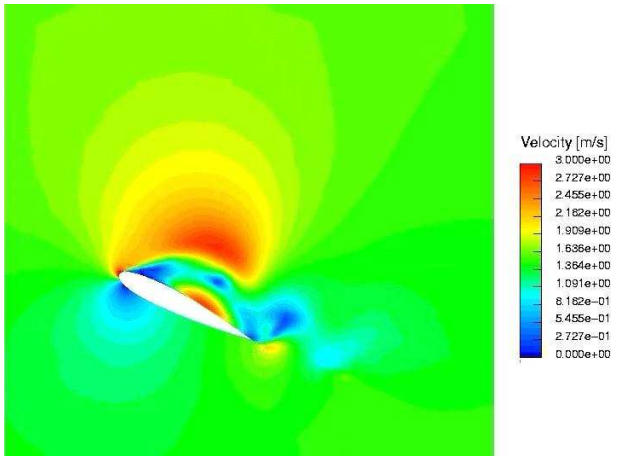
Stefan A. Funken, Dirk Lebiedz, Karsten Urban

Angewandte Numerik 1



SKRIPT, UNIVERSITÄT ULM, SOMMERSEMESTER 2014

Vorwort. Dieses Manuskript ist entstanden aus Mitschriften und Skripten verschiedener Vorlesungen, die wir seit 2002 an der Universität Ulm gehalten haben. Es ist der Sinn des vorliegenden Dokumentes, den Studierenden unserer Vorlesungen einen einheitlichen Stoffumfang für die Vorlesung **Angewandte Numerik I** zu geben, unabhänging davon, wer von uns tatsächlich die Vorlesung hält. In diesem Sinne bietet das vorliegende Manuskript einen Rahmen für die Nachbearbeitung der Vorlesungen und der Vorbereitung auf Prüfungen. Dieses Manuskript kann keinesfalls das Studium von Lehrbüchern ersetzen. Eine entsprechende Liste von Lehrbüchern findet sich im Literaturverzeichnis und auf der Internet–Seite der Vorlesung.
Jedes Manuskript weist Fehler auf, sicher auch dieses. Wenn Sie Fehler, Druckfehler, sprachliche Unzulänglichkeiten oder inhaltliche Flüchtigkeiten finden, würden wir uns über einen entsprechenden Hinweis per Email freuen. Sie helfen damit zukünftigen Studierenden. Vielen Dank im Voraus.

Danksagung. Einige Vorgängerversionen dieses Manuskriptes wurden aus Mitteln der Studiengebühren finanziert. Eine Reihe von Personen haben bei der Erstellung geholfen. Wir danken Theresa und Julia Springer, Markus Bantle und Judith Rommel für zahlreiche Hinweise. Frau Kristin Kirchner und Herrn Moritz Reinhard sind wir für das sorgfältige Lesen des Manuskripts zu besonderem Dank verpflichtet. Ihre zahlreichen Kommentare, Vorschläge, Korrekturen und Hinweise haben die Qualität des Textes wesentlich verbessert. Weiterhin möchten wir Katharina Becker-Steinberger, Sebastian Kestler, Dr. Michael Lehn und Moritz Reinhard für die Ausarbeitung der Aufgaben und zugehöriger Lösungen danken.

Ganz besonderer Dank gebührt auch Frau Petra Hildebrand, die unsere handschriftlichen Aufzeichnungen in LaTeX umgesetzt und zahlreiche Grafiken erstellt hat. Frau Brandner, Frau Serbine und Herrn Weithmann möchten wir für das LaTeX'en der Lösungen danken.

Stand. Ulm, März 2014, Stefan A. Funken, Dirk Lebiedz, Karsten Urban

Inhaltsverzeichnis

1 Einleitung: Was ist Numerik? 1

2 Zahlendarstellung, Fehlerarten und Kondition 5

2.1 Zahlendarstellung 5
2.2 Rundung und Maschinengenauigkeit 15
2.3 Gleitkommaarithmetik 17
2.4 Fehlerverstärkung bei elementaren Rechenoperationen 19
2.5 Kondition eines Problems 21
2.6 Numerische Stabilität eines Algorithmus 23

3 Direkte Lösung linearer Gleichungssysteme 27

3.1 Einführung, Cramersche Regel 28
3.2 Gestaffelte Systeme 31
3.3 Gaußsche Eliminationsmethode 35
3.4 Pivot-Strategien 38
3.5 Nachiteration 44
3.6 Cholesky-Verfahren 45
3.7 Bandgleichungen 48
3.8 Fehlerabschätzung mittels Kondition 50

4 Lineare Ausgleichsprobleme 55

4.1 Die Methode der kleinsten Quadrate 55
4.2 Die **QR**-Zerlegung 57
4.3 **Givens**-Rotationen 59
4.4 **Householder**–Spiegelungen 62
4.5 Die Singulärwertzerlegung 67

5 Nichtlineare Gleichungen 71

5.1 Bisektionsmethode 74
5.2 Regula-Falsi[1] 75
5.3 Die Sekantenmethode 77
5.4 Das Verfahren von **Newton** 79
5.5 Effizienz 82
5.6 Fixpunkt–Iteration 82
5.7 Newton–Verfahren für Systeme 87
5.8 Hinweise zur Durchführung in der Praxis 90

6 Interpolation 93

6.1 Das allgemeine Interpolationsproblem 93
6.2 Polynominterpolation 94

7 Numerische Integration und Differenziation 109
7.1 Quadraturformeln 110
7.2 Klassische interpolatorische Quadraturformeln 118
7.3 **Gauß**-Quadratur 121
7.4 Numerische Differenziation 136

A Landau-Symbole 139
A.1 Definition 139
A.2 Beispiele 139

B Normen 141
B.1 Vektornorm-Eigenschaften 141
B.2 Matrixnormen 142

C Einführung in MATLAB 145
C.1 Grundlegende MATLAB-Befehle 145
C.2 Mathematik mit Matrizen 149
C.3 Datenverwaltung 154
C.4 Ausgabe von Text 155
C.5 Kontrollbefehle 156
C.6 Graphische Darstellung 158
C.7 Fortgeschrittenes 161

Literaturverzeichnis 163

Stichwortverzeichnis 169

1 EINLEITUNG: WAS IST NUMERIK?

Wenn man z.B. in der Öffentlichkeit über Mathematik diskutiert, stellt man oft fest, dass viele eine vollkommen falsche Interpretation des Begriffes „Numerik“ (oder besser „Numerische Mathematik“) haben. Das Wort scheint eine fachliche Nähe zur Zahlentheorie anzudeuten und man mag vermuten, dass die Numerik ein Teilgebiet der Reinen Mathematik ist. Dies ist jedoch nicht richtig.

Numerik ist ein Teilgebiet der Angewandten Mathematik und bezeichnet allgemein die mathematische Untersuchung von Berechnungsverfahren, Algorithmen oder Methoden zur näherungsweisen Berechnung bestimmter Größen auf dem Computer.

Zu berechnende Größen können z.B. sein:

- Die Auswertung von Funktionen wie z.B. $\sin(1), e^2$ und ähnliche, die nicht unmittelbar angegeben werden können. Solche Berechnungen geschehen auch in jedem Taschenrechner. Dort sind die entsprechenden Berechnungsverfahren in der Regel durch Schaltkreise oder spezielle Microchips realisiert, also in der Hardware fest verdrahtet.

- Die Lösung von Gleichungen z.B. lineare $Ax = b$ oder nichtlineare Gleichungen. d.h. $f(x) = 0$, wobei die gesuchte Lösung $x^* \in \mathbb{R}^n$ oft aus sehr vielen Komponenten besteht, also $n > 1.000.000$ gilt! Andere Typen von Gleichungen sind z.B. Differentialgleichungen etwa $u''(x) = f(x),\ x \in (0,1)$, wobei die Funktion $u : [0,1] \to \mathbb{R}$ bei gegebenem $f : [0,1] \to \mathbb{R}$ gesucht ist. Ein Beispiel hierfür ist etwa die gesuchte Auslenkung u eines Werkstücks unter der äußeren Last f.

- Die Näherung von Größen, die nicht exakt berechnet werden können (z.B. Ableitungen – Sensitivitäten, Integrale – Mittel- oder Erwartungswerte). Manchmal will man solche Größen auch gar nicht exakt bestimmen, etwa, wenn die Berechnung zu lange dauern würde. Andere Beispiele wären optimale Steuerungen oder optimale Strategien.

- Computer-gestützte Simulationen komplexer Vorgänge.
  In vielen Bereichen sind Experimente sehr teuer, aufwändig, zu gefährlich oder gar nicht möglich. Beispiele sind
  - Wettervorhersage: Dies bedeutet die Simulation der turbulenten Wolkenströmungen.
  - Strömungsmechanik, also etwa Aerodynamik, Flugzeug-, Automobil- oder Schiffsbau.
  - Bauingenieurwesen, z.B. die Simulation der Statik oder der Eigenschwingung von Brücken und anderen Bauwerken.
  - Medizin: Simulation der Knochenheilung oder die Therapie von Gehirn–Tumoren.
  - Wirtschaftswissenschaften: Simulation von Aktienkursen, Bewertung von komplexen Finanz- oder Versicherungsprodukten, Bestimmung optimaler Anlage–Strategien.

Speziell beschäftigt sich die Numerik mit

- der Konstruktion „geeigneter“ Lösungsverfahren, die insbesondere
  - schnell („effizient“) sind, teilweise in oder sogar schneller als in „Echtzeit“,
  - zuverlässig, also mit **beweisbarer** Abschätzung z.B. der folgenden Form $\|x_{\text{numerisch}} - x_{\text{exakt}}\| \leq$ Toleranz (wobei x_{exakt} unbekannt ist), und
  - robust gegenüber Störungen wie z.B. Messfehlern, Modell–Unsicherheiten etc.

  sind;

- mit der mathematischen Analyse dieser Verfahren (Konvergenz, Geschwindigkeit, Aufwand, Robustheit etc.) und

- deren effizienter Realisierung (Implementierung).

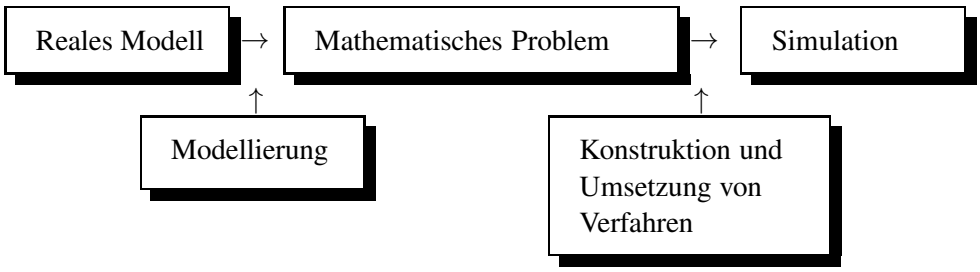
In diesem Sinne liegt die Numerik an der Schnittstelle von Mathematik und Informatik.

Die Numerische Mathematik selbst hat eine Reihe von Teilgebieten bzw. Gebiete, die unmittelbar an sie angrenzen, etwa:

- Numerische Lineare Algebra (Numerik 1),
- Numerische Analysis (Numerik 2),
- Numerische Optimierung (Numerik 3),
- Numerik von Differenzialgleichungen (Numerik 4 und Numerik von Partiellen Differenzialgleichungen)
- Numerical Finance
- Computational Physics, Computational Science
- CFD: Computational Fluid Dynamics
- Wissenschaftliches Rechnen
- ...

Insbesondere wird klar, dass die Numerik einen starken interdisziplinären Bezug aufweist. Dies ist auch der Grund, warum einführende Vorlesungen in Numerik Grundbestandteil des Studiums auch in Fächern wie Physik, Informatik oder Ingenieurwissenschaften sind.

Das Vorgehen kann man (vereinfacht) so darstellen:



Wir beschreiben dieses abtraget Vorgehen an einem konkreten Beispiel.

Beispiel 1.0.1 (Bestimmung des Abraums bei der Braunkohleförderung) Wir wollen nun den Weg vom realen Modell zur Simulation an einem Beispiel verdeutlichen. Nehmen wir an, dass wir den Abraum bestimmen wollen, der durch die Förderung von Braunkohle entstanden ist. Würde man die Form des entstandenen Lochs exakt kennen, so würde sich der Abraum als der Wert des Volumenintegrals ergeben. Da man aber die Form nicht exakt kennt, fliegt z.B. ein Flugzeug über die Braunkohle-Halde und macht Tiefenmessungen an einzelnen Punkten. Daraus ergibt sich dann folgendes Vorgehen:

1) **Reales Modell:** Volumen des Abraums; mit konkreten **Messungen (Experiment)** in Form von Tiefenmessungen durch Stereofotographien aus Flugzeugen, Satelliten;

2) **Mathematisches Problem:** Bestimme das Volumen des Lochs $\rightsquigarrow$ Berechnung von Volumenintegralen; verwende dabei die Mess-Ergebnisse als Daten (**Modellierung**);

3) **Konstruktion und Umsetzung von Verfahren:** Verwende geeignete Formeln zur näherungsweisen Berechnung der 3D–Integrale;
Simulation: Programmiere das o.g. Verfahren und erzeuge so entsprechende Simulationen.

Oftmals muss man Simulationswerte im Nachhinein anhand von Messungen validieren — wenn dies möglich ist.

Was bedeutet nun "Angewandte" Numerik? Dies wirft insbesondere die Frage auf, wie das Verhältnis der Numerik-Vorlesungen für Ingenieure, Naturwissenschaftler und Informatiker im Vergleich zu denen der Mathematiker ist. Man könnte an zwei Extreme denken (die natürlich **beide** nicht zutreffen):

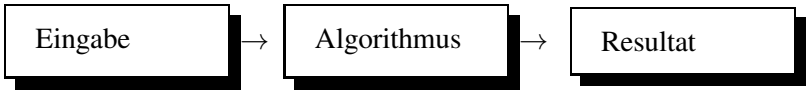
1. Angewandte Numerik ist genau wie Numerik für Mathematiker, nur dass man sich zusätzlich auf die Anwendung numerischer Verfahren konzentriert.

2. Angewandte Numerik beschränkt sich alleine auf die Anwendung numerischer Methoden ohne jede mathematische Theorie (Kochrezepte).

Wie schon gesagt, beides ist falsch — die Wahrheit liegt irgendwo dazwischen. Die Anwendung numerischer Verfahren steht im Vordergrund, gewiss. Um aber die Handwerkszeuge sachgemäß einsetzen zu k̈nnen, muss man sie –bis zu einem gewissen Grad– auch verstehen. Dies gilt spätestens dann, wenn es gilt, Fehler in einem numerischen Verfahren zu finden und zu beseitigen. Dies bedeutet, dass die Angewandte Numerik nicht so tief eindringt wie man es von Mathematikern erwartet, es werden keine neuen Methoden entwickelt. Daher sind an vielen Stellen die mathematische Theorie und auch Beweise weggelassen (oder klein geschrieben, was auf Zusatzmaterial hindeutet). Diese k̈nnen in der angegebenen Literatur oder im entsprechenden Skript für Mathematiker nachgelesen werden.

2 Zahlendarstellung, Fehlerarten und Kondition

Auf einer endlichen Maschine kann man niemals exakt rechnen, als Beispiel nehme man die Darstellung reeller Zahlen mit unendlicher Dezimaldarstellung. Dies ist auf einer endlichen Maschine unmöglich. Wir werden noch sehen, dass viele Rechenoperationen ebenfalls nicht exakt ausgeführt werden können. Man hat also **immer** Fehler. Diese gilt es zu analysieren und zu kontrollieren.
Nehmen wir an, wir wollten die Lösung $x \in \mathbb{R}^n$ eines linearen Gleichungssystems $Ax = b$ mit gegebener regulärer Matrix $A \in \mathbb{R}^{n\times n}$ und gegebener rechten Seite $b \in \mathbb{R}^n$ mittels eines Computerprogramms berechnen. Wir wollen also aus den Eingabegrößen (A, b) das Resultat $f(A, b) := A^{-1}b$ berechnen.



Bei exakter Arithmetik würden wir mit dem Gauß-Verfahren (aus der Linearen Algebra, siehe auch Kapitel 3) ein exaktes Ergebnis erhalten. Im Allgemeinen treten jedoch sowohl Fehler in der Eingabe als auch im Algorithmus auf, z.B. lässt sich die Zeit nur auf eine Genauigkeit von $10^{-14}s$ und die Masse auf $10^{-8}g$ genau bestimmen[1], Zahlen nur näherungsweise im Rechner als Gleitkommazahl (auch Fließkommazahl genannt, engl. floating point number) darstellen und z.B. auch die Darstellung

$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k\, x^{2k+1}}{(2k+1)!} \tag{2.1}$$

nur näherungsweise auswerten. Welchen Einfluss haben nun beide Fehlerarten

- Eingabefehler (lassen sich nicht vermeiden) und
- Fehler im Algorithmus (wobei keine algorithmischen Fehler gemeint sind)

auf den Fehler im Resultat? Die Unterscheidung beider Arten von Fehlern führt uns zu den Begriffen

Kondition eines Problems und

Stabilität eines Algorithmus.

2.1 Zahlendarstellung

Ein erstes Beispiel soll motivieren, wieso wir uns mit der Frage, wie Zahlen im Rechner dargestellt werden, beschäftigen sollten.

Beispiel 2.1.1 Eine einfache Überlegung zeigt, dass das Ergebnis von

$$4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2$$

[1] Siehe www.ptb.de: Physikalisch-technische Bundesanstalt.

ungerade ist, eine erste Rechnung, dass an der letzten Stelle eine 1 oder 9 stehen muss. (Dabei könnte sich die 9 z.B. durch $-1 \cdot 10 + 1$ ergeben.)

$$\begin{aligned} & 4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 = 4(1386 \cdot 10)^4 - (1960 \cdot 10 + 1)^4 + 2(1960 \cdot 10 + 1)^2 \\ = \; & 4 \cdot 1386^4\, 10^4 - (1960^4\, 10^4 + 4 \cdot 1960^3\, 10^3 + 6 \cdot 1960^2\, 10^2 + 4 \cdot 1960\, 10 + 1) + \\ & \quad +2(1960^2\, 10^2 + 2 \cdot 1960 \cdot 10 + 1) \\ = \; & 10^4(\cdots) + 10^3(\cdots) + 10^2(\cdots) + 10^1(\cdots) + 10^0(-1 + 2). \end{aligned}$$

Rechnet man auch die anderen Stellen aus (per Hand oder mit Maple) so ergibt sich

$$4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 = 1\,.$$

MATLAB-Beispiel:

Überraschenderweise liefert Matlab aber ein anderes Resultat.

```
>> myfun = @(x,y) 4*x^4-y^4+2*y^2;
>> myfun(13860,19601)
ans =
     2
```

Auch die folgende Realisierung mittels quadratischer Ergänzung ($u = 4x^2 + 1$, $v = y^2 - 1$, $u - v^2 = 4x^4 - y^4 + 2y^2$) liefert ein (anderes) falsches Ergebnis.

```
>> x = 13860; y = 19601;
>> u = 4 * x * x * x * x + 1;
>> v = y * y - 1;
>> u - v * v
ans =
     0
```

Und Vorsicht beim Umgang mit Maple! Schreibt man dort versehentlich nach einer Zahl noch einen Punkt, so ergibt es das scheinbar überraschende folgende Ergebnis. (Man beachte dabei, dass Maple, wenn man den Punkt hinzufügt, nicht mehr mit exakter Arithmetik rechnet, sondern so genannte Gleitkommazahlen (Maschinenzahlen, wird später noch im Detail erklärt) verwendet, deren Genauigkeit man einstellen kann. Die Anweisung `Digits:=18` führt dazu, dass Maple Gleitkommazahlen mit 18 Stellen verwendet.

```
> restart:
  4  *13860^4-19601^4+2*19601^2;

  4. *13860^4-19601^4+2*19601^2;
  Digits:=15:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=16:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=17:
  4. *13860^4-19601^4+2*19601^2;
  Digits:=18:
  4. *13860^4-19601^4+2*19601^2;

                          1
                   68398402.
                      402.
                       2.
                       2.
                       1.
```

MATLAB-Beispiel:

Aus dem Topf der mathematischen Mirakel ziehen wir noch folgendes Beispiel:

```
>> floor(114)
ans =
   114
>> floor(1.14*100)
ans =
   113
```

Die Routine `floor` rundet dabei auf die nächstkleinere ganze Zahl. Die Zahl 114 auf die nächstkleinere ganze Zahl gerundet sollte also in beiden Fällen 114 ergeben!

Ein Rechner verwendet als einfachste Form der Zahlendarstellung **Festkommazahlen**. Dabei wird eine (meistens) begrenzte Ziffernfolge gespeichert und an festgelegter Stelle das Komma angenommen. Bei vielen Anwendungen reicht der darstellbare Zahlenbereich jedoch nicht aus, so dass es naheliegend ist, bei jedem Wert die genaue Stelle des Kommas zusätzlich zu speichern. Das bedeutet mathematisch nichts anderes als die Darstellung der Zahl x mit zwei Werten, der **Mantisse** m und dem **Exponenten** e. Wir bekommen somit die Darstellung $x = m \cdot 10^e$. Diese Darstellung ist jedoch nicht eindeutig, siehe z.B. $1014 = 1.014 \cdot 10^3 = 10140 \cdot 10^{-1}$. Legt man jedoch den Wertebereich der Mantisse m geschickt fest, z.B. $1 \leq m < 10$, so erhält man eine eindeutige Darstellung (wenn man endlich viele Stellen hat). Diesen Schritt bezeichnet man als **Normalisierung** der Mantisse. Wir haben soeben stillschweigend vorausgesetzt, dass wir die Basis 10 verwenden[2]. Darauf ist man jedoch nicht festgelegt. Pascal[3] erkannte, dass man jede Basis $b \geq 2$ verwenden kann. Heutige Rechner verwenden meist binäre Zahlendarstellungen, d.h. zur Basis 2.
Betrachten wir nun die Zahlendarstellung noch etwas allgemeiner.

Definition 2.1.2 (b-adischer Bruch) *Es sei $b \geq 2$ eine natürliche Zahl, $n \in \mathbb{N}_0$, $a_k \in \mathbb{N}_0$ mit $0 \leq a_k < b$ und $a_n \neq 0$. Eine Reihe der Gestalt*

$$\pm \sum_{k=-\infty}^{n} a_k b^k$$

*wird als b-**adischer Bruch** bezeichnet, b heißt **Basis**. Falls die Basis festgelegt ist, kann man einen b-adischen Bruch auch durch die Reihung der Ziffern a_k angeben:*

$$\pm a_n a_{n-1} \cdots a_1 a_0 \,.\, a_{-1} a_{-2} a_{-3} \cdots .$$

Bemerkungen 2.1.3

i) Für $b = 10$ spricht man von **Dezimalbrüchen**, *für $b = 2$ von* **dyadischen Brüchen** *und die Babylonier[4] verwendeten ein System zur Basis $b = 60$, was sich heute noch im Verhältnis von Sekunde, Minute und Stunde wieder findet.*

*ii) Die Basis wird auch als Grundzahl bezeichnet, weswegen man auch die Bezeichnung g-**adisch** findet.*

*iii) Vorsicht, nicht b-**adisch** mit p-**adisch** verwechseln ($p \sim$ Primzahl)!* △

[2] **Dezimalsystem**: Unter dem Einfluss der Schrift „De Thiende“ (Leiden 1585) von Simon Stevin (1548-1620) setzte sich im 16. Jahrhundert das Rechnen mit Dezimalbrüchen in Westeuropa durch.

[3] **Blaise Pascal**, 1623-1662.

[4] **Sexagesimalsystem**: Zählt man mit den Fingern der linken Hand wie üblich von 1 bis 5 und registriert die an der linken Hand gezählten 5-er Gruppen durch das Drücken des Daumens der rechten Hand auf das passende Fingerglied an der rechten Hand, so kommt man zum $5 \times 12 = 60$-System. Diese Zählweise wurde schon von den Vorfahren der Sumerer im 4. Jahrtausend v. Chr. angewandt. Die Babylonier (1750 v. Chr.) verwendeten dann das Sexagesimalsystem in dem Sinne, dass sie 60 Herzschläge zu einer Minute zusammen fassten und 60 Minuten zu einer Stunde.

iv) Der Null kommt immer eine Sonderrolle zu. Sie wird gesondert betrachtet, um die Darstellung einfacher zu halten. Ohne den Sonderfall der Null jeweils aufzuführen, schließen wir ihn indirekt mit ein.

Die nächsten beiden Sätze besagen, dass sich jede reelle Zahl durch einen b-adischen Bruch darstellen lässt und umgekehrt.

Satz 2.1.4 (Konvergenz b-adischer Bruch) *Jeder b-adische Bruch konvergiert gegen eine reelle Zahl.*

Beweis. Wir beschränken uns auf den Fall eines nicht-negativen b-adischen Bruchs $\sum_{k=-\infty}^{n} a_k b^k$. Wir definieren für $-\ell \le n$ die Partialsummen $s_\ell := \sum_{k=-\ell}^{n} a_k b^k$. Es ist nun zu zeigen, dass die Folge $(s_\ell)_{-\ell \le n}$ eine Cauchy-Folge ist. Sei $\varepsilon > 0$ und $L \in \mathbb{N}$ so groß, dass $b^{-L} < \varepsilon$ gelte. Dann gilt für $\ell \ge m \ge L$

$$\begin{aligned} |s_\ell - s_m| &= \sum_{k=-\ell}^{n} a_k b^k - \sum_{k=-m}^{n} a_k b^k = \sum_{k=-\ell}^{-m-1} a_k b^k \le \sum_{k=-\ell}^{-m-1} (b-1) b^k \\ &= (b-1) b^{-m-1} \sum_{k=-\ell+m+1}^{0} b^k = (b-1) b^{-m-1} \sum_{k=0}^{\ell-m-1} b^{-k} \\ &< (b-1) b^{-m-1} \frac{1}{1-b^{-1}} = b^{-m} \le b^{-L} < \varepsilon \end{aligned}$$

und der Satz ist bewiesen. □

Satz 2.1.5 (Reelle Zahl als b-adischer Bruch) *Sei $b \ge 2$ eine natürliche Zahl. Dann lässt sich jede reelle von Null verschiedene Zahl in einen b-adischen Bruch entwickeln.*

Beweis. Es genügt wieder, nur den Fall $x \ge 0$ zu zeigen. Zuerst zeigen wir, dass es zu jeder Basis $b \ge 2$ mindestens eine Zahl $m \in \mathbb{N}$ gibt mit $x < b^m$. Nach dem Archimedischen Axiom gibt es ein $m \in \mathbb{N}$ mit $m > (x-1)/(b-1) \in \mathbb{R}$. Mit diesem m und der Bernoullischen Ungleichung erhalten wir $b^m = (1+(b-1))^m \ge 1 + m(b-1) > 1 + x - 1 = x$. Sei $n := \min\{k \in \mathbb{N}_0 \mid 0 \le x < b^{k+1}\}$. Durch vollständige Induktion konstruieren wir nun eine Folge $(a_{-\ell})_{\ell \ge -n}$ natürlicher Zahlen $0 \le a_{-\ell} < b$, so dass für alle $m \ge -n$ gilt $x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m}$ mit $0 \le \nu_{-m} < b^{-m}$. Da $\lim_{m\to\infty} \nu_{-m} = 0$ gilt, folgt $x = \sum_{\ell=-\infty}^{n} a_\ell b^\ell$ und somit die Behauptung.

Induktionsanfang $-m = n$: Es gilt $0 \le x b^{-n} < b$. Somit existiert ein $a_n \in \{0, 1, \ldots, b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $x b^{-n} = a_n + \delta$ gilt. Mit $\nu_n := \delta b^n$ gewinnt man $x = a_n b^n + \nu_n$ mit $0 \le \nu_n < b^n$.

Induktionsschritt $(-m) \to (-m-1)$: Es gilt $0 \le \nu_{-m} b^{m+1} < b$, also gibt es ein $a_{-m-1} \in \{0, 1, \ldots, b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $\nu_{-m} b^{m+1} = a_{-m-1} + \delta$ gilt. Mit $\nu_{-m-1} := \delta b^{-m-1}$ gewinnt man

$$x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m} = \sum_{\ell=-m}^{n} a_\ell b^\ell + (a_{-m-1} + \delta) b^{-m-1} = \sum_{\ell=-m-1}^{n} a_\ell b^\ell + \nu_{-m-1}\,,$$

wobei $0 \le \nu_{-m-1} < b^{-m-1}$ gilt. □

Bemerkung 2.1.6 *Die b-adische Darstellung ist in der gewählten Form nicht eindeutig, siehe z. B. $1 = 0.\overline{9} = 0.999\ldots$, falls $b = 10$ gewählt ist. Durch die zusätzliche Bedingung „$a_k < b-1$ für unendlich viele $k \le n$“ erhält man eine eindeutige Darstellung. Somit wäre z.B. für $b = 10$ die Darstellung $0.\overline{9} = 0.999\ldots$ für $x = 1$ nicht zugelassen.*

Auf einem Rechner können wir nur eine endliche Entwicklung speichern, also werden wir den b-adischen Bruch durch eine endliche Entwicklung approximieren.

Beispiel 2.1.7 i) Binärdarstellung ($b = 2$) von $x = (14.625)_{10} = (1110.101)_2$

$$\begin{aligned} 14 = \quad & 7 \cdot 2 \quad +0 & & & \Rightarrow a_0 = 0 \\ & 7 \quad = \quad & 3 \cdot 2 \quad +1 & & \Rightarrow a_1 = 1 \\ & & 3 = \quad & 1 \cdot 2 + 1 & \Rightarrow a_2 = 1 \\ & & & 1 = 0 \cdot 2 + 1 & \Rightarrow a_3 = 1 \end{aligned}$$

Im Folgenden bezeichnet $\lfloor \cdot \rfloor$ die untere Gaußklammer, d.h.

$$\lfloor x \rfloor := \max\{j \in \mathbb{Z} : j \le x\}\,.$$

$$\begin{aligned} a_{-1} &= \lfloor 2 * 0.625 \rfloor = \lfloor 1.25 \rfloor = 1 \\ a_{-2} &= \lfloor 2 * 0.25 \rfloor = \lfloor 0.5 \rfloor = 0 \\ a_{-3} &= \lfloor 2 * 0.5 \rfloor = \lfloor 1 \rfloor = 1 \end{aligned}$$

ii) Binärdarstellung von $x = (0.2)_{10} = (0.\overline{0011})_2$

$$\begin{aligned} a_{-1} &= \lfloor 2 * 0.2 \rfloor = \lfloor 0.4 \rfloor = 0 \\ a_{-2} &= \lfloor 2 * 0.4 \rfloor = \lfloor 0.8 \rfloor = 0 \\ a_{-3} &= \lfloor 2 * 0.8 \rfloor = \lfloor 1.6 \rfloor = 1 \\ a_{-4} &= \lfloor 2 * 0.6 \rfloor = \lfloor 1.2 \rfloor = 1 \\ a_{-5} &= \lfloor 2 * 0.2 \rfloor = \dots \\ &\vdots \end{aligned}$$

Bemerkung 2.1.8 *Da es unmöglich ist, reelle Zahlen als unendliche b-adische Brüche zu speichern, werden reelle Zahlen auf Rechnern durch eine endliche Entwicklung approximiert.*

Definition 2.1.9 (Festpunktzahl) *Zu gegebener Basis b ist*

$$F(\ell, n) := \left\{ \pm \sum_{k=-\ell}^{n} a_k b^k = \pm (a_n \dots a_1 a_0 \,.\, a_{-1} \dots a_{-\ell})_b \right\}$$

*die Menge der **Festpunktzahlen** mit $(n+1)$ Vor- und ℓ Nachkommastellen.*

Bemerkungen 2.1.10 *i) Typischerweise werden ganze Zahlen (engl. integers) auf Computern dargestellt mit $\ell = 0$ und $b = 2$.*

ii) In modernen Rechnern werden negative Zahlen häufig als 2-er-Komplement[5] abgespeichert. Dabei ist x das Komplement zu y, wenn in der jeweiligen Arithmetik $x+y = 0$ gilt. Beispiel: Für $b = 2$ und $n = 3$ ergibt sich

$\lvert x \rvert$	x			$-x$		
0	0	0	0			
1	0	0	1	1	1	1
2	0	1	0	1	1	0
3	0	1	1	1	0	1
4	1	0	0	1	0	0

also darstellbarer Bereich: $-4, \dots, 3$

Bemerkung 2.1.11 *Da im 2-er-Komplement der Wert 0 den positiven Zahlen zugeordnet wird,*

[5] **Zweierkomplement** ist eine Möglichkeit, negative Zahlen im Dualsystem darzustellen. Dabei werden keine zusätzlichen Symbole wie + und - benötigt. Da im Zweierkomplement der Wert 0 den positiven Zahlen zugerechnet wird, umfasst der Wertebereich bei n binären Stellen allgemein den Bereich $-2^{n-1}, \dots, 0, \dots 2^{n-1} - 1$. Zum Beispiel bei 4 Bit von -8 bis 7. Negative Zahlen gewinnt man aus positiven Zahlen, indem sämtliche binären Stellen negiert werden und zu dem Ergebnis der Wert 1 addiert wird. Somit wird aus $(4)_{10} = (0100)_2$ durch Invertieren $Not(0100) = 1011$ und Addition der 1 $(1011)_2 + (0001)_2 = (1100)_2$. Addiert man nun 4 und -4, so gilt $(4)_{10} + (-4)_{10} = (0100)_2 + (1100)_2 = (0000)_2 = (0)_{10}$, wobei die vorderste Stelle verworfen wurde.

erhält man für 8*-Bit*[6]*,* 16*-Bit und* 32*-Bit, bzw.* 1*-,* 2*- und* 4*-Byte*[7] *Darstellungen die Zahlenbereiche*

$$\begin{aligned} &\textit{8-Bit, } n=7: \quad 0,\dots,255 \textit{ ohne Vorzeichen} \\ &\qquad\qquad \textit{bzw. } -128,\dots,127 \ (8 \textit{ Stellen}) \end{aligned}$$

$$\begin{aligned} &\textit{16-Bit, } n=15: \quad 0,\dots,65535 \textit{ ohne Vorzeichen} \\ &\qquad\qquad \textit{bzw. } -32768,\dots,32767 \ (16 \textit{ Stellen}) \end{aligned}$$

$$\begin{aligned} &\textit{32-Bit, } n=31: \quad 0,\dots,4.294.967.295 \textit{ ohne Vorzeichen} \\ &\qquad\qquad \textit{bzw. } -2.147.483.648,\dots,2.147.483.647 \ (32 \textit{ Stellen}) \end{aligned}$$

Bemerkung 2.1.12 *Als Problem stellt sich dabei heraus, dass damit nur ein kleiner Zahlenbereich darstellbar ist.*

Die Verwendung eines Festpunktes schränkt also die auf dem Rechner darstellbaren Werte stark ein. Gibt man stattdessen nur die Gesamtanzahl an Vor- und Nachkommastellen vor und lässt die Position des Punktes variabel, skaliert also mit unterschiedlichen Potenzen der Basis b, so ist die Menge der darstellbaren Zahlen wesentlich größer. Dies führt zur Definition der Gleitpunkt-Darstellung einer reellen Zahl x.

Definition 2.1.13 (Gleitpunkt-Darstellung) *Die* **normalisierte** *Gleitpunkt-Darstellung einer reellen Zahl* $x \in \mathbb{R}$ *hat die Form* $x = f \cdot b^{\ell}$*, wobei*

- *die* **Mantisse** f *ist eine feste Zahl, die* m *Stellen hat und*

  $$b^{-1} \le |f| < 1, \textit{ falls } x \neq 0 \textit{ und } f = 0 \textit{ falls, } x = 0$$

  genügt, d.h.

  $$f = \begin{cases} \pm\,(0.d_1 \dots d_m)_b = \pm \sum_{j=1}^{m} d_j b^{-j} & , d_1 \neq 0 \\ 0 & , d_1 = 0 \end{cases}$$

- *der* **Exponent** ℓ *ist eine ganze Zahl innerhalb fester Schranken*

  $$r \le \ell \le R\,,$$

  welche sich in der Form

  $$\ell = \pm\,(\ell_{n-1} \cdots \ell_0)_b = \pm \sum_{j=0}^{n-1} \ell_j b^j$$

  darstellen lässt. Hierbei braucht nicht $\ell_{n-1} \neq 0$ *zu gelten.*
  (Für 8-stelligen binären Exponenten wird z.B. $r = -128$ *und* $R = 127$ *gewählt.)*

Die Menge der Maschinenzahlen mit Basis b*,* m*-stelliger Mantisse und* n*-stelligem Exponenten wird als* $\mathbb{M}(b, m, n)$ *bezeichnet.*

[6] **Bit** $\sim$ (engl. binary digit, lat. digitus $\sim$ Finger).
[7] **Byte** $\sim$ Maßeinheit für eine Datenmenge von 8 Bit.

Beispiel 2.1.14 $\mathbb{M}(2,3,1)$ besteht aus den Zahlen

$$x = \pm(d_{-1}\cdot 2^{-1} + d_{-2}\cdot 2^{-2} + d_{-3}\cdot 2^{-3})\cdot 2^{\pm \ell_0},$$

wobei $d_{-1} \neq 0 \vee x = d_{-1} = d_{-2} = d_{-3} = 0$ gilt. Somit

$$\mathbb{M}(2,3,1) = \left\{0, \pm\frac{1}{4}, \pm\frac{5}{16}, \pm\frac{3}{8}, \pm\frac{7}{16}, \pm\frac{1}{2}, \pm\frac{5}{8}, \pm\frac{3}{4}, \pm\frac{7}{8}, \pm 1, \pm\frac{5}{4}, \pm\frac{3}{2}, \pm\frac{7}{4}\right\}.$$

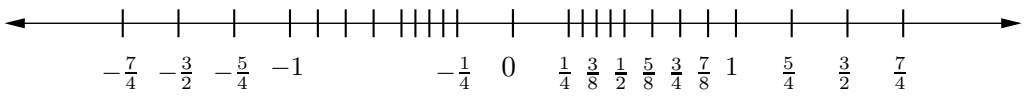


Abb. 2.1: Graphische Darstellung von $\mathbb{M}(2,3,1)$.

Man beachte den kleiner werdenden Abstand zur Null hin und den Abstand zwischen den betragsmäßig kleinsten Zahlen und der Null.

Bemerkung 2.1.15 *Die betragsmäßig kleinste und größte Zahl in* $\mathbb{M}(b,m,n)$ *bezeichnen wir mit* $x_{\min}$ *bzw.* $x_{\max}$*,*

$$\begin{aligned} x_{\min} &:= \min\{|z| \in \mathbb{M}(b,m,n)\}, \quad |x| < x_{\min} \Rightarrow x = 0 \\ x_{\max} &:= \max\{|z| \in \mathbb{M}(b,m,n)\}, \quad |x| > x_{\max} \Rightarrow \textbf{overflow}. \end{aligned}$$

Beispiel 2.1.16

$$\begin{aligned} &\mathbb{M}(10,4,2), \quad x = -51.34 \quad &&\rightsquigarrow -0.5134\cdot 10^2 \\ &\mathbb{M}(2,4,2), \quad x = 3.25 = (11.01)_2 \quad &&\rightsquigarrow (0.1101)_2 \cdot 2^{(10)_2} \\ &\mathbb{M}(2,4,2), \quad x = 14.625 = (1110.101)_2 \quad &&\rightsquigarrow (0.1110101)_2 \cdot 2^{(100)_2} \notin \mathbb{M}(2,4,2) \end{aligned}$$

Bemerkung 2.1.17 *Üblicherweise werden 8 Byte (64 Bit) Speicherplatz für eine Gleitpunktzahl wie folgt aufgeteilt:*

- *1 Bit für das Vorzeichen;*
- *11 Bit für den Exponenten, d.h. die Werte sind darstellbar im Bereich -1022 ... 1023 (1 ... 2046 – Bias*[8]*-Wert 1023). Die Werte -1023 und 1024, bzw. 0 und* $2047 = 2^{11} - 1$ *als Charakteristik sind reserviert für die speziellen Zahlenwerte „Null“, „Unendlich“ und „NaN“;*
- *52 Bit für die Mantisse. Dies sind eigentlich Werte zwischen* $(0\ldots01)_2$ *und* $(1\ldots1)_2$*, da die Null gesondert abgespeichert wird. Ist man in der normalisierten Darstellung, so ist die erste Null redundant und man gewinnt noch eine Stelle hinzu. Also muss auch noch irgendwie festgehalten werden, ob eine normalisierte Darstellung vorliegt oder nicht.*

Also ergibt sich $b = 2$*,* $m = 52$*,* $n = 11$ *und damit* $r = -1023$ *und* $R = 1024$*.*

Bemerkung 2.1.18 *Der Standardtyp in Matlab, wenn nichts Weiteres angegeben wird, ist double.*

Bemerkungen 2.1.19 *i) Die Werte von* b, m, ℓ *hängen in der Regel vom Typ des jeweiligen Computers ab, ebenso* r, R*. Der IEEE*[9]*-Standard versucht dies zu vereinheitlichen.*

ii) Selbst bei der Umwandlung „exakter“ Eingaben in Maschinenzahlen können Fehler auftreten, z.B. bei

$$x = 0.1 = \left(0.000\overline{1100}\right)_2.$$

[8] Hier zu verstehen als ein konstanter Wert, der zu etwas hinzuaddiert bzw. subtrahiert wird. Wird auch als Offset bezeichnet.

[9] IEEE∼ Institute of Electrical and Electronics Engineers.

Tab. 2.1: Interpretation des Zahlenformats nach IEEE-754

Exponent e	Mantisse	Bedeutung	Bezeichnung
$e = 0$	$m = 0$	± 0	Null
$e = 0$	$m > 0$	$\pm 0, m \times 2^{1-Bias}$	denormalisierte Zahl
$0 < e < e_{max}$	$m \geq 0$	$\pm 1, m \times 2^{e-Bias}$	normalisierte Zahl
$e = e_{max}$	$m = 0$	$\pm \infty$	Unendlich (Inf)
$e = e_{max}$	$m > 0$	keine Zahl	**N**ot **a** **N**umber (NaN)

Tab. 2.2: Zahlenformat nach IEEE-754

Typ	Größe	Exponent	Mantisse	Werte des Exponenten (e)
single double	32 Bit 64 Bit	8 Bit 11 Bit	23 Bit 52 Bit	-126 $\leq$ e $\leq$ 127 -1022 $\leq$ e $\leq$ 1023
	rel. Abstand zweier Zahlen	Dezimalstellen	betragsmäßig kleinste Zahl	größte Zahl
single double	$2^{-(23+1)}$ $2^{-(52+1)}$	7-8 15-16	$2^{-23} \cdot 2^{-126}$ $2^{-52} \cdot 2^{-1022}$	$(1-2^{-24})2^{128}$ $(1-2^{-53})2^{1024}$

Mit Hilfe der folgenden mex-Routine `digits.c` lassen sich auf einem 32-Bit Rechner unter Matlab die einzelnen Bits einer double precision-Variablen auslesen. (Kompilieren der c-Routine auf der Kommandozeilenebene von Matlab mit `mex digits.c`.)

C-Funktion: digits.c

```
#include "mex.h"
bool get_bit_at(double y[], int pos)
{
  unsigned long bitmask;
  unsigned long *cy = (unsigned long *) y;

  if (pos >= 0){
    if (pos < 32){
        bitmask = 1 << pos;       // bitmask  setzen
        return (cy[0] & bitmask)!= 0;
    } else if (pos < 64){
        bitmask = 1 << (pos-32); // bitmask  setzen
                                 // unsigned long ist nur
                                 // 4 Byte = 32 Bit lang
        return (cy[1] & bitmask)!= 0;
    }
  }
  mexErrMsgTxt("No valid position for Bit.");
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double *x,*y;
  mwSize mrows,ncols;
  int k;
```

```
27    /* Check for proper number of arguments. */
28    if(nrhs!=1) {
29      mexErrMsgTxt("One input required.");
30    } else if(nlhs>1) {
31      mexErrMsgTxt("Too many output arguments");
32    }
33    /* The input must be a noncomplex scalar double.*/
34    mrows = mxGetM(prhs[0]);
35    ncols = mxGetN(prhs[0]);
36    if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
37        !(mrows==1 && ncols==1) ) {
38      mexErrMsgTxt("Input must be a noncomplex scalar double.");
39    }
40    /* Create matrix for the return argument. */
41    plhs[0] = mxCreateDoubleMatrix(1,64, mxREAL);
42    /* Assign pointers to each input and output. */
43    x = mxGetPr(prhs[0]);
44    y = mxGetPr(plhs[0]);
45    /* Call the digit subroutine. */
46    for (k=0; k<64; k++) y[63-k] = (double) get_bit_at(x,k);
47  }
```

Versuchen Sie sich die Ergebnisse der folgenden Matlab-Ausgaben zu erklären bzw. testen Sie es auf Ihrem eigenen Rechner.

```
>> A=digits(-1);A(1)
ans =
     1
>> A=digits(1);A(1)
ans =
     0
>> A=digits(2^(1023)*(1.9999999999999998));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     1  1  1  1  1  1  1  1  1  1  0
ans =
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
ans =
     1
>> A=digits(2^(-971)*(2-1.9999999999999995));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     0  0  0  0  0  0  0  0  0  0  1
ans =
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
ans =
     0
>> A=digits(2^(-972)*(2-1.9999999999999995));
>> A(1),A(2:12),[A(13:29); A(30:46); A(47:63)],A(64)
ans =
     0
ans =
     0  0  0  0  0  0  0  0  0  0  0
ans
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
ans =
     0
>> A=digits(realmax)        % try by yourself
>> A=digits(realmin)
>> A=digits(realmin/2^52)
>> A=digits(2^0+2^(-1)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)+2^(-3)); A(13:29)
>> A=digits(2^0+2^(-1)+2^(-2)+2^(-3)+2^(-17))A(13:29)1
```

2.2 Rundung und Maschinengenauigkeit

Aufgabe: Approximiere $x \in \mathbb{R}$ durch $fl(x) \in \mathbb{M}(b,m,n)$ („float“)

$$fl : \mathbb{R} \to \mathbb{M}(b,m,n)$$

bzw.

$$fl : [-x_{\max}, x_{\max}] \to \mathbb{M}(b,m,n)$$

Dies wird durch geeignete Rundungsstrategien gewährleistet. Wir betrachten ein konkretes Beispiel:

Sei $x \in \mathbb{R}$ mit $|x| \in [x_{\min}, x_{\max}]$ und b-adischer Darstellung

$$x = \pm b^{\ell} \sum_{j=1}^{\infty} d_j \, b^{-j},$$

wobei ℓ ein n-stelliger Exponent sei. Es gelte $d_1 \neq 0$ sowie $d_j < b-1$ für unendlich viele $j \in \mathbb{N}$. Für eine vorgegebene Mantissenlänge m ist die **Standardrundung** definiert durch

$$fl(x;b,m,n) = fl(x) := \pm \sum_{j=1}^{m} d_j \, b^{\ell-j} + \begin{cases} 0 & \text{, falls } d_{m+1} < \frac{b}{2} \\ b^{\ell-m} & \text{, falls } d_{m+1} \geq \frac{b}{2} \end{cases}$$

und für die anderen Fälle gilt

$$|x| < x_{\min} \Rightarrow fl\,(x) := 0$$
$$|x| > x_{\max} \Rightarrow \textbf{overflow}.$$

Bemerkung 2.2.1 *Man mache sich klar, dass die Rundung im Falle $d_{m+1} = b/2$ (b gerade, $b/2$ ungerade) rein willkürlich ist.*

Beispiel 2.2.2 Betrachte $\mathbb{M}(10,4,2)$. Dann gilt für die Eingabe $x = 0.21576 \cdot 10^3$ die Rundung $fl(x) = 0.2158 \cdot 10^3$ und für $y = 216.53$ ergibt sich $fl(y) = 0.2165 \cdot 10^3$, entscheidend sind also die signifikanten Stellen, nicht die Größe der Eingabe.

Definition 2.2.3 (absoluter/relativer Fehler) *Es sei $x \in \mathbb{R}$ und $\widetilde{x} \in \mathbb{R}$ eine Approximation.*

(i) Die Größe $|x - \widetilde{x}|$ heißt **absoluter Fehler**.

(ii) Die Größe $\frac{|x-\widetilde{x}|}{|x|}$ heißt **relativer Fehler**.

Der relative Fehler ist normalerweise die interessante Größe; er ist dimensionslos, während die Größe des absoluten Fehlers von der gewählten Maßeinheit abhängt.

Lemma 2.2.4 *Die Basis b sei gerade. Es sei $x = \pm b^{\ell} \sum_{k=1}^{\infty} d_k \, b^{-k}$ ($0 < d_1 < b$) und $fl(x)$ die Standardrundung von x auf m Stellen. Dann gilt*

$$|fl(x) - x| \leq \frac{b^{1-m}}{2} \cdot b^{\ell-1} = \frac{1}{2} b^{\ell-m}$$

und

$$\frac{|fl(x) - x|}{|x|} \leq \frac{b^{1-m}}{2}.$$

Beweis. O.B.d.A sei x positiv. Sei $n \in \mathbb{N}$. Für $b > 1$ und $0 \le d_k \le b-1$ $(k = n, n+1, \ldots)$ gilt

$$\sum_{k=n}^{\infty} d_k\, b^{-k} \le \sum_{k=n}^{\infty} (b-1) b^{-k} = \sum_{k=n-1}^{\infty} b^{-k} - \sum_{k=n}^{\infty} b^{-k} = b^{-(n-1)}. \tag{2.2}$$

Betrachten wir zuerst die Situation des Abrundens bei der Standardrundung, was der Situation $d_{m+1} < b/2$ entspricht. Da b nach Voraussetzung gerade ist, gilt somit $b/2 \in \mathbb{Z}$. Somit folgt aus $d_{m+1} < b/2$, dass auch $d_{m+1} + 1 \le b/2$ gilt. Da $0 < d_1 < b$ vorausgesetzt wurde (was der normierten Gleitpunktdarstellung 2.1.13 entspricht), hat man $b^{-1} \le \sum_{k=1}^{\infty} d_k\, b^{-k} \le 1$ und somit $b^{\ell-1} \le x \le b^{\ell}$. Beachtet man $x - fl(x) = b^{\ell} \sum_{m+1}^{\infty} d_k\, b^{-k}$, so schließt man nun für $d_{m+1} < b/2$ mit (2.2)

$$\begin{aligned} |fl(x) - x| &= b^{\ell} \left[d_{m+1}\, b^{-(m+1)} + \sum_{k=m+2}^{\infty} d_k\, b^{-k} \right] \le b^{\ell} \left[d_{m+1}\, b^{-(m+1)} + b^{-(m+1)} \right] \\ &= b^{\ell-(m+1)} \left(d_{m+1} + 1 \right) \le b^{\ell-(m+1)} \cdot \frac{b}{2} = \frac{b^{1-m}}{2} \cdot b^{\ell-1}. \end{aligned} \tag{2.3}$$

Für das Aufrunden, d.h. für den Fall $d_{m+1} \ge \frac{b}{2}$ gilt nun $d_{m+1}\, b^{-1} \ge \frac{1}{2}$ und analog zu (2.3)

$$\begin{aligned} |f\ell(x) - x| &= b^{\ell} \left[b^{-m} - \sum_{k=m+1}^{\infty} d_k\, b^{-k} \right] \le b^{\ell} \left[b^{-m} - d_{m+1}\, b^{-(m+1)} \right] \\ &= b^{\ell-m} \underbrace{|\, 1 - \underbrace{d_{m+1}\, b^{-1}}_{\ge 1/2} \,|}_{\le 1/2} \le \frac{b^{1-m}}{2} \cdot b^{\ell-1}. \end{aligned}$$

Damit ist die Abschätzung für den absoluten Fehler bewiesen und die Abschätzung für den relativen Fehler erhält man sofort mit $b^{\ell-1} \le x$. □

Definition 2.2.5 *Die Zahl* $\mathrm{eps} := \frac{b^{1-m}}{2}$ *heißt (relative)* **Maschinengenauigkeit**.

Definition 2.2.6 *Wir definieren* $\mathbb{M}' := \{x \in \mathbb{R} : x_{\min} \le |x| \le x_{\max}\} \cup \{0\}$.

Bemerkungen 2.2.7 *i) Es gilt* $\mathrm{eps} = \inf \{\delta > 0 : fl(1+\delta) > 1\}$.

ii) In Matlab liefert die Anweisung eps die Maschinengenauigkeit, die sich aber auch durch folgende Routine mineps.m bestimmen lässt. Unter anderen Programmiersprachen geht dies ähnlich, man muss sich jedoch vergewissern, dass der Compiler die Anweisung `while 1 < 1 + value` *nicht zu* `while 0 < value` *optimiert!*

iii) $\forall x \in \mathbb{M}', x \neq 0\ \exists \varepsilon \in \mathbb{R}$ *mit* $|\varepsilon| < \mathrm{eps}$ *und* $\frac{|fl(x)-x|}{|x|} = \varepsilon$, *d.h.*

$$fl(x) = x(1+\varepsilon). \tag{2.4}$$

MATLAB-Funktion: mineps.m

```
1 function value = mineps
2 value = 1;
3 while 1 < 1 + value
4   value = value / 2;
5 end
6 value = value * 2;
```

MATLAB-Beispiel:

Die Funktion `mineps` liefert das gleiche Ergebnis wie die Matlab-Konstante `eps`. Ergänzend sei noch die Matlab-Konstante `realmin` wiedergegeben. Sie ist die kleinste positive darstellbare Maschinenzahl.

```
>> eps
ans =
    2.220446049250313e-016
>> mineps
ans =
    2.220446049250313e-016
>> realmin
ans =
    2.225073858507201e-308
```

2.3 Gleitkommaarithmetik

Ein Algorithmus ist eine Folge von arithmetischen Operationen, mit denen Maschinenzahlen verknüpft werden. Ein Computer rechnet also mit Maschinenzahlen durch die Realisierung (Implementierung) eines Algorithmus.

Beispiel 2.3.1 Betrachte $\mathbb{M}(10,3,3)$. Dann gilt:

$$\underbrace{0.346 \cdot 10^2}_{\in \mathbb{M}(10,3,3)} + \underbrace{0.785 \cdot 10^3}_{\in \mathbb{M}(10,3,3)} = 0.1131 \cdot 10^3 \notin \mathbb{M}(10,3,3).$$

Aus diesem einfachen Beispiel sieht man, dass das Ergebnis einer arithmetischen Operation auf Maschinenzahlen **keine** Maschinenzahl sein muss!

Voraussetzung 2.3.2 *Wir nehmen wir an, dass für die Addition $\oplus$ im Rechner gilt*

$$x \oplus y = fl(x+y) \quad \text{für alle } x, y \in \mathbb{M} \text{ mit } |x+y| \in \mathbb{M}'$$

und Entsprechendes auch für die anderen Grundrechenarten: $\nabla \in \{+,-,*,\div\}$ wird durch eine Gleitpunktoperation $\overline{\nabla}$ ersetzt mit

$$x \overline{\nabla} y = fl(x \nabla y)\ .$$

Man sagt $\oplus$ (bzw. $\overline{\nabla}$) ist **exakt gerundet**.

Folgerung 2.3.3 *Für alle $x, y \in \mathbb{M}$ mit $x+y \in \mathbb{M}'$ gilt*

$$x \oplus y = (x+y)(1+\varepsilon) \tag{2.5}$$

für ein geeignetes $|\varepsilon| \leq \text{eps}$, und Entsprechendes auch für die anderen Grundrechenarten: $x \overline{\nabla} y = (x \nabla y)(1+\varepsilon)$ mit $|\varepsilon| \leq \text{eps}$. Dies folgt aus $fl(x) = x(1+\varepsilon)$.

Bemerkung 2.3.4 *(a) Maschinenoperationen sind nicht assoziativ, wie folgendes Beispiel in $\mathbb{M}(10,3,1)$ zeigt. Für $x = 6590 = 0.659 \cdot 10^4$, $y = 1 = 0.100 \cdot 10^1$ und $z = 4 = 0.400 \cdot 10^1$ ergibt sich bei exakter Rechnung $(x+y)+z = x+(y+z) = 6595$. Für die Maschinenoperationen gilt hingegen auf der einen Seite*

$$x \oplus y = 0.659 \cdot 10^4 = 6590 \Rightarrow (x \oplus y) \oplus z = 6590,$$

aber

$$y \oplus z = 0.500 \cdot 10 = 5 \Rightarrow x \oplus (y \oplus z) = 6660$$

Dies zeigt, dass die Reihenfolge der Verknüpfungen wesentlich ist!
Als „Faustregel“ (die aber auch nicht immer hilft, s.u.) kann man sagen, dass man die Summation in der Reihenfolge aufsteigender Beträge ausführen sollte, um Rundungsfehler zu minimieren.

(b) *Ebenso gilt das Distributivgesetz nicht mehr. Als Beispiel betachte folgende Subtraktion:* $0.73563 - 0.73441 = 0.00122$. *Bei 3-stelliger Rechnung:* $0.736 - 0.734 = 0.002 = 0.2 \cdot 10^{-2}$, *d.h., der absolute Fehler ist in der Größenordnung des Rundungsfehlers, für den relativen Fehler gilt jedoch*

$$\frac{0.002 - 0.00122}{0 - 00122} = 0.64\hat{=}64\%$$

also sehr groß. Dieser unangenehme Effekt der Gleitpunktarithmetik heißt ***Auslöschung*** *(genaueres später).*

Beispiel 2.3.5 (Auslöschung) (a) Für $b = 10$ und $m = 2$ gilt

$$(100 \oplus 4) \oplus 4 = 100 \neq 110 = 100 \oplus (4 \oplus 4)\,.$$

Man beachte dabei $100 \oplus 4 = 100$, da $100 + 4$ auf 2 Stellen gerundet 100 ergibt, und dass $100+8$ auf 2 Stellen gerundet 110 ergibt. Wir sehen also wieder den Effekt der Auslöschung.

(b) Wir betrachten die Rechnung

$$0.1236 + 1.234 - 1.356 = 0.0016 = 0.1600 \cdot 10^{-2}.$$

Gleitpunktrechnung mit $b = 10, m = 4$ liefert

$$(0.1236 \oplus 1.234) \ominus 1.356 = 1.358 \ominus 1.356 = 0.2000 \cdot 10^{-2},$$

d.h. schon die erste Stelle des berechneten Ergebnisses ist falsch! Der Rundungsfehler der ersten Addition wird durch die nachfolgende Subtraktion extrem verstärkt.

Bemerkung 2.3.6 *Auch wenn alle Summanden positiv sind, kann etwas schiefgehen: Sei wie im Beispiel 2.3.5* $b = 10$ *und* $m = 4$. *Wir wollen*

$$\sum_{n=0}^{N} a_n \text{ mit } a_0 = 1 \tag{2.6}$$

berechnen. Falls $0 \leq a_n < 10^{-4}$ *für alle* $n > 0$, *und falls wir der Reihe nach summieren, ist das berechnete Ergebnis 1, egal wie groß die Summe wirklich ist. Eine erste Abhilfe ist es nach der obigen Faustregel, der Größe nach zu summieren (zuerst die kleinste Zahl). In unserem Beispiel (2.6) hilft das allerdings nicht wirklich, das berechnete Ergebnis ist dann höchstens* 2. *Besser ist jedoch folgendes Vorgehen: Man ersetzt die zwei kleinsten Zahlen durch ihre Summe und wiederholt diesen Vorgang so lange, bis nur noch eine Zahl, nämlich die gesuchte Summe, übrigbleibt.*

Beispiel 2.3.7 (Guard Digit) Sei $\mathbb{M} = \mathbb{M}(10, 3, 1)$ und betrachte $\ominus$. Sei weiter $x = 0.215 \cdot 10^9$ und $y = 0.125 \cdot 10^{-9}$. Naive Realisierung von $x \ominus y = fl(x - y)$ erfordert **schieben** von y auf den größeren Exponenten $y = 0.125 \cdot 10^{-18} \cdot 10^9$ und **subtrahieren** der Mantissen:

$$\begin{array}{rcl} x & = & 0.215000000000000000000 \cdot 10^9 \\ y & = & 0.000000000000000000125 \cdot 10^9 \\ \hline x - y & = & 0.214999999999999999875 \cdot 10^9 \end{array} \tag{2.7}$$

Runden auf drei Stellen liefert dann $x \ominus y = 0.215 \cdot 10^9$. Dies erfordert einen Addierer mit $2(b^n - 1) + m = 21$ Stellen. In diesem Fall hätten wir das Ergebnis auch durch die Abfolge **Schieben**, **Runde** y, **Subtrahiere** erhalten. Im Allgemeinen ist das aber nicht gut wie folgendes Beispiel zeigt:

$$\begin{array}{rcl} x & = & 0.101 \cdot 10^1 \\ y & = & 0.993 \cdot 10^0 \end{array} \quad \longrightarrow \quad \begin{array}{rcll} x & = & 0.101 & \cdot 10^1 \\ y & = & 0.099 & \cdot 10^1 \\ \hline x \ominus y & = & 0.002 & \cdot 10^1 \end{array} .$$

Für den relativen Fehler im Ergebnis gilt dann

$$\frac{(x \ominus y) - (x - y)}{(x - y)} = \frac{0.02 - 0.017}{0.017} \approx 0.176 \approx 35 \text{ eps},$$

wobei $\text{eps} = 1/2 \cdot 10^{-3+1} = 0.005$ sei.
Verwenden wir nun einen $m+1$-stelligen Addierer, dann erhalten wir

$$\begin{array}{rcl} x & = & 0.1010 \cdot 10^1 \\ y & = & 0.0993 \cdot 10^1 \\ \hline x - y & = & 0.0017 \cdot 10^1 \end{array} .$$

Das Ergebnis $x \ominus y = 1.7 \cdot 10^{-2}$ ist exakt!

Bemerkung 2.3.8 *Allgemein kann man zeigen (siehe [Knuth]): Mit einer zusätzlichen Stelle (sogenannter* **Schutzziffer** *oder* **Guard Digit***) gilt*

$$\frac{(x \ominus y) - (x - y)}{(x - y)} \leq 2 \text{ eps} .$$

Mit nur 2 Guard Digits erhält man sogar genau das gerundete Ergebnis der exakten Rechnung, d.h. das gerundete Ergebnis aus (2.7) *ohne einen* $2(b^n - 1) + m$*-stelligen Addierer.*

Völlig unkalkulierbar sind Gleitkommaoperationen aber nicht. Es gilt z.B. folgendes Resultat: Sind u, v Gleitkommazahlen und

$$\begin{array}{rclcrcl} u' & = & (u \oplus v) \ominus v, & & v' & = & (u \oplus v) \ominus u, \\ u'' & = & (u \oplus v) \ominus v', & & v'' & = & (u \oplus v) \ominus u', \end{array}$$

dann folgt

$$u + v = (u \oplus v) + ((u \ominus u') \oplus (v \ominus v'')) .$$

Dies erlaubt eine Berechnung des Fehlers mittels Gleitkommaarithmetik. (Siehe [Knuth, 4.2.2, Theorem B].)

2.4 Fehlerverstärkung bei elementaren Rechenoperationen

In einem Programm/Algorithmus werden **viele** Rechenoperationen druchgeführt. Es stellt sich also die Frage, was mit Fehlern im weiteren Programmverlauf passiert. Hier beschäftigen wir uns zunächst mit **exakten Operationen**, d.h. ohne weitere Rundung.
Seien also x, y die exakten Eingaben und $\tilde{x}, \tilde{y}$ die gestörten (realen) Eingaben, mit den relativen Fehlern

$$\delta_x := \frac{\tilde{x} - x}{x} , \ \delta_y := \frac{\tilde{y} - y}{y} ,$$

d.h.

$$\tilde{x} = x(1 + \delta_x), \qquad \tilde{y} = y(1 + \delta_y), \tag{2.8}$$

und wir nehmen an, dass $|\delta_x| , \ |\delta_y| \leq \varepsilon < 1$ gilt, also, dass die Eingabefehler „klein“ sind.
Da ein Algorithmus aus den Grundrechenarten zusammengesetzt ist, betrachten wir die Fehlerverstärkung bei diesen elementaren Rechenoperationen.

Bemerkung 2.4.1 (Multiplikation) *Offenbar gilt*

$$\tilde{x} * \tilde{y} \quad = \quad (x(1+\delta_x)) * (y(1+\delta_y)) = (x*y)(1+\delta_x+\delta_y+\delta_x\delta_y) =: (x*y)(1+\delta)$$

mit $|\delta| = |\delta_x+\delta_y+\delta_x\delta_y| \leq \varepsilon(2+\varepsilon)$*, d.h. bei Vernachlässigung des quadratischen Terms ist dies von der Ordnung* 2ε*. Damit ergibt sich für den relativen Fehler (falls* $|\delta_x|, |\delta_y| \leq \mathrm{eps}$ *und* $\varepsilon < 1$*)*

$$\left|\frac{\tilde{x}*\tilde{y} - x*y}{x*y}\right| \quad = \quad |\delta| = 2\varepsilon + \varepsilon\cdot\varepsilon < 3\varepsilon \leq 3\,\mathrm{eps}\,.$$

Also bleibt der relative Fehler im Rahmen der Maschinengenauigkeit.

Bemerkung 2.4.2 (Division) *Bei der Division gilt*

$$\frac{\tilde{x}}{\tilde{y}} = \frac{x(1+\delta_x)}{y(1+\delta_y)}.$$

Mit Hilfe der geometrischen Reihe ($\frac{1}{1-q} = \sum_{j=1}^{\infty} q^j$*, verwende* $q = -\delta y$*) gilt*

$$\frac{1}{1+\delta_y} = 1 + \sum_{j=1}^{\infty}(-1)^j\delta_y^j,$$

und damit folgt

$$\frac{1+\delta_x}{1+\delta_y} = 1 + \delta_x + (1+\delta_x)\sum_{j=1}^{\infty}(-1)^j\delta_y^j =: 1+\delta.$$

Für den Fehlerverstärkungsfaktor δ *gilt dann unter der Annahme* $|\delta_x|, |\delta_y| \leq \varepsilon$

$$|\delta| \leq \varepsilon + (1+\varepsilon)\varepsilon \underbrace{\sum_{j=0}^{\infty}\varepsilon^j}_{=\frac{1}{1-\varepsilon}} = \varepsilon\left(1 + \frac{1+\varepsilon}{1-\varepsilon}\right).$$

Falls z.B. $\varepsilon \leq \frac{1}{2}$ *gilt, folgt* $\frac{1+\varepsilon}{1-\varepsilon} \leq 3$ *und damit* $|\delta| \leq 4\,\varepsilon$*, also zeigt die Division ein ähnliches Verhalten wie die Multiplikation, d.h. der relative Fehler bleibt im Rahmen der Maschinengenauigkeit.*

Bemerkung 2.4.3 (Addition) *Im Fall der Addition gilt*

$$\begin{aligned}\tilde{x}+\tilde{y} &= x(1+\delta_x) + y(1+\delta_y) = x+y+x\delta_x+y\delta_y \\ &= (x+y)\left(1 + \frac{x}{x+y}\delta_x + \frac{y}{x+y}\delta_y\right) =: (x+y)(1+\delta)\end{aligned}$$

mit der folgenden Abschätzung für den Fehlerverstärkungsfaktor

$$|\delta| \quad \leq \quad (|\delta_x| + |\delta_y|)\max\left\{\left|\frac{x}{x+y}\right|, \left|\frac{y}{x+y}\right|\right\} \leq 2\varepsilon\max\left\{\left|\frac{x}{x+y}\right|, \left|\frac{y}{x+y}\right|\right\}$$

falls $|\delta_x|, |\delta_y| \leq \varepsilon$*. Man beachte, dass der Term*

$$\max\left\{\left|\frac{x}{x+y}\right|, \left|\frac{y}{x+y}\right|\right\}$$

nicht gegen ε *abgeschätzt werden kann. Insbesondere für* $x \not\approx -y$ *kann dieser Faktor beliebig groß werden, der Ausgabefehler ist also* ***nicht*** *notwendigerweise in der Größe des Eingabefehlers.*

Dies ist die Erklärung der Auslöschung und kann durch folgende Überlegung verdeutlicht werden. Nehmen wir an, wir wollten $x + y$ berechnen für zwei Eingaben mit folgender Darstellung

$$x = 0.d_1 \dots d_r d_{r+1} \dots d_m * b^e, \qquad y = -0.d_1 \dots d_r \tilde{d}_{r+1} \dots \tilde{d}_m * b^e,$$

d.h., die ersten r Stellen von x und y sind identisch. Wir nehmen der Einfachheit halber an $d_j \geq \tilde{d}_j$, $j \geq r+1$, $d_{r+1} > \tilde{d}_{r+1}$. Daraus folgt

$$x + y = 0.0 \dots 0\hat{d}_{r+1} \dots \hat{d}_m * b^e = 0.\hat{d}_{r+1} \dots \hat{d}_m * b^{e-r}$$

mit $\hat{d}_j \geq 0$ und damit für den relativen Fehler

$$\left| \frac{x}{x+y} \right| = \left| \frac{0.d_1 \dots d_m * b^e}{0.\hat{d}_{r+1} \dots \hat{d}_m * b^{e-r}} \right| \leq \frac{b \cdot b^e}{b^{-1} b^{e-r}} = b^{r+2},$$

da $0.d, \dots d_m \leq b$ und $0.\hat{d}_{r+1} \dots \hat{d}_m * b^{-1}$. Man verliert also mindestens r Stellen an Genauigkeit, diese werden „ausgelöscht". Intern (im Computer) werden diese Stellen zufällig aufgefüllt mit Information, die sich in den entsprechenden Speicherzellen befindet. Egal, was dort steht, diese Stellen sind im Prinzip immer falsch, da bezüglich x und y keine Information jenseits der m-ten Stelle vorliegt.

2.5 Kondition eines Problems

Bislang haben wir einzelne Operationen untersucht, nun werden diese Operationen zu einem Algorithmus verkettet. Ein Algorithmus wird dann verwendet, um ein mathematisches Problem zu lösen. Wir wollen nun untersuchen, welche Probleme „einfach" bzw. „schwer" sind und welche Algorithmen „gut" bzw. „schlecht" sind. Wir beginnen mit der Untersuchung der Problemstellungen.

Beispiel 2.5.1 Diskutieren wir am Anfang zuerst folgendes geometrische Problem: die zeichnerische Bestimmung des Schnittpunkts S zweier Geraden g und h in der Ebene. Schon beim Zeichnen haben wir Schwierigkeiten, die Geraden ohne Fehler darzustellen. Die Frage, die wir näher untersuchen wollen, lautet, wie stark der Schnittpunkt S (Output) von den Zeichenfehlern (Fehler im Input) abhängt.

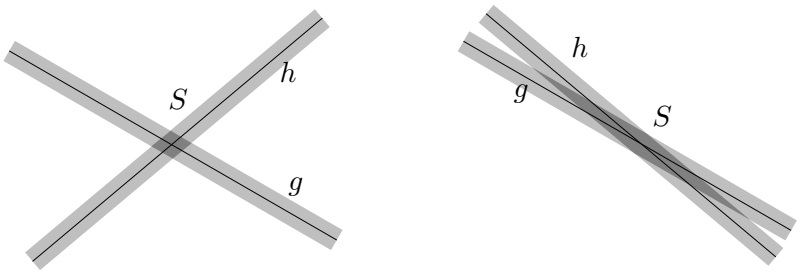


Abb. 2.2: Schnittpunkt S zweier Geraden g und h. Links gut konditioniert, rechts schlecht konditioniert.

Wie wir der Grafik direkt entnehmen können, hängt der Fehler in der Ausgabe stark davon ab, in welchem Winkel $\measuredangle(g, h)$ sich die Geraden schneiden. Stehen g und h annähernd senkrecht aufeinander, so variiert der Schnittpunkt S etwa im gleichen Maße wie der Fehler beim Zeichnen von g und h. In diesem Fall bezeichnet man das Problem, den Schnittpunkt S zeichnerisch zu bestimmen, als **gut konditioniert**.

Ist jedoch der Winkel $\measuredangle(g,h)$ sehr klein (g und h sind fast parallel), so kann man schon mit dem Auge keinen genauen Schnittpunkt S ausmachen, und eine kleine Lageänderung von g oder h liefert einen gänzlich anderen Schnittpunkt. Man spricht dann von einem **schlecht konditionierten** Problem.

Kommen wir nun zu einer mathematischen Präzisierung des Konditionsbegriffs. Dazu brauchen wir einen etwas allgemeineren Rahmen und werden später sehen, dass dieser Rahmen für die Untersuchung sehr gut geeignet ist.

Problem 2.5.2 *Seien X, Y Mengen und $\varphi : X \to Y$. Wir betrachten das Problem:*

$$\textit{Gegeben sei } x \in X, \textit{ gesucht } y = \varphi(x).$$

Wir untersuchen also, wie sich Störungen in den Daten x auf das Ergebnis y auswirken. Man beachte, dass dies nichts mit der Realisierung auf dem Computer (dem Algorithmus) zu tun hat, sondern einzig eine Eigenschaft der Problemstellung ist.

Beispiel 2.5.3 (Noch einmal der Geradenschnittpunkt) Die beiden Geraden seien in folgender Form gegeben

$$G_1 = \{(x_1,x_2) \in \mathbb{R}^2 : a_{1,1}x_1 + a_{1,2}x_2 = b_1\}, \quad G_2 = \{(x_1,x_2) \in \mathbb{R}^2 : a_{2,1}x_1 + a_{2,2}x_2 = b_2\},$$

wobei $b = (b_1,b_2)^T \in \mathbb{R}^2$ und die Koeffizienten $a_{i,j}$ für $i,j = 1,2$ gegeben sind. Mit $A := (a_{ij})_{i,j=1,2} \in \mathbb{R}^{2\times 2}$ ist der gesuchte Schnittpunkt $x = (x_1,x_2)^T$ von G_1, G_2 gegeben durch

$$Ax = b\,,$$

also die Lösung eines linearen Gleichungssystems. Falls A regulär ist können wir schreiben $x = A^{-1}b$. Also sind A und b die Eingaben und x die Ausgabe, d.h. $X = \mathbb{R}^{2\times 2}, \quad Y = \mathbb{R}^2$ und $\varphi(A,b) = A^{-1}b = x$.

Beispiel 2.5.4 Betrachten wir das Beispiel 1.0.1 des Kohleaushubs. Wir gehen davon aus, dass wir aus den Messungen bereits eine Höhenfunktion $h(x) = z$, $h : \mathbb{R}^2 \to \mathbb{R}$, gewonnen haben, die jedem Punkt $x \in \mathbb{R}^2$ einer Karte die Höhe zuweist. Weiterhin sei das Kohlerevier in einem Rechteck $R := [a,b] \times [c,d] \subset \mathbb{R}^2$ enthalten. Dann lautet die Formel für den Kohleaushub

$$f(h) = \int_a^b \int_c^d h(x)\, dx_2\, dx_1,$$

also ein Doppelintegral. Damit ist klar, dass $Y = \mathbb{R}$, aber für welche Eingaben ist φ definiert? Offenbar muss die Funktion h, die hier als Eingabe fungiert, integrierbar sein, etwa $h \in \mathcal{R}(\mathbb{R}) = X$, wobei $\mathcal{R}(\mathbb{R})$ die Menge der Rieman-integrierbaren Funktionen auf $\mathbb{R}$ bezeichnet. Man beachte, dass X hier (im Gegensatz zu den beiden ersten Beispielen) unendlich-dimensional ist.

Dies motiviert den Begriff der Konditionszahlen.

Definition 2.5.5 (Konditionszahlen) *Sei $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ differenzierbar in $x \in \mathbb{R}^n$ sowie $\varphi_i(x) \neq 0,\ 1 \le i \le m$. Die Zahlen*

$$\kappa_{ij}(x) = \frac{|x_j|}{|\varphi_i(x)|}\left|\frac{\partial \varphi_i}{\partial x_j}(x)\right|, \qquad 1 \le i \le m,\ 1 \le j \le n, \tag{2.9}$$

*heißen die **Konditionszahlen** von φ in x.*

Beispiel 2.5.6 1. Multiplikation: $\varphi : \mathbb{R}^2 \to \mathbb{R}, \varphi(x_1, x_2) = x_1 \cdot x_2$,

$$\kappa_1(x) = \frac{|x_1|}{|x_1 x_2|}\left|\frac{\partial \varphi}{\partial x_1}(x)\right| = 1, \quad \kappa_2(x) = 1.$$

Keine Verstärkung des relativen Fehlers; die Multiplikation ist „gut konditioniert".

2. Addition: $\varphi : \mathbb{R}^2 \to \mathbb{R}, \varphi(x_1, x_2) = x_1 + x_2$,

$$\kappa_1(x) = \frac{|x_1|}{|x_1 + x_2|}\left|\frac{\partial \varphi}{\partial x_1}(x)\right| = \frac{|x_1|}{|x_1 + x_2|}, \quad \kappa_2(x) = \frac{|x_2|}{|x_1 + x_2|}.$$

Im Falle der Auslöschung, d.h. wenn $|x_j| \gg |x_1 + x_2|$, große Verstärkung des relativen Fehlers; die Addition ist in diesem Fall „schlecht konditioniert".

3. Lösen der quadratischen Gleichung $x^2 + 2px - q = 0$ im Fall $p, q > 0$: Berechnung der größeren der beiden Nullstellen (Mitternachtsformel):

$$\varphi(p, q) = -p + \sqrt{p^2 + q},\ \varphi : \mathbb{R}^2 \to \mathbb{R}, \tag{2.10}$$

Eine Rechnung ergibt

$$\kappa_p = \frac{p}{\sqrt{p^2 + q}},\ \ \kappa_q = \frac{p + \sqrt{p^2 + q}}{2\sqrt{p^2 + q}}, \tag{2.11}$$

also $\kappa_p \leq 1,\ \kappa_q \leq 1$; das Problem ist ebenfalls „gut konditioniert". Im Fall $q < 0$, $q \approx -p^2$ hingegen wäre das Problem „schlecht konditioniert".

2.6 Numerische Stabilität eines Algorithmus

Jeder Algorithmus zur Lösung von Problem 2.5.2 lässt sich auffassen als eine Abbildung $\widetilde{\varphi} : X \to Y$. Von einem guten Algorithmus wird man erwarten, dass er die relativen Fehler nur unwesentlich mehr verstärkt als die Konditionszahlen $\kappa_{ij}(x)$ des Problems φ es erwarten lassen. Für $\widetilde{\varphi} : \mathbb{R}^n \to \mathbb{R}^m$ sollte also eine Abschätzung der Form

$$\left|\frac{\widetilde{\varphi}_i(\widetilde{x}) - \widetilde{\varphi}_i(x)}{\widetilde{\varphi}_i(x)}\right| \leq C_{i1} \underbrace{\sum_{j=1}^{n} \kappa_{ij}(x)\frac{|\widetilde{x}_j - x_j|}{|x_j|}}_{\geq \frac{|\varphi_i(\widetilde{x}) - \varphi_i(x)|}{|\varphi_i(x)|} + o(\|\tilde{x} - x\|)} + C_{i2}\, n\ \text{eps} \quad (i = 1 \ldots, m), \tag{2.12}$$

(oder so ähnlich) gelten mit Konstanten $C_{i1}, C_{i2} \geq 0$, welche nicht viel größer als 1 sind.

Definition 2.6.1 (Numerische Stabilität eines Algorithmus) *Ein Algorithmus $\widetilde{\varphi} : \mathbb{R}^n \to \mathbb{R}^m$ zur Lösung eines Problems $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ heißt* ***numerisch stabil*** *(oder* ***gut konditioniert****), falls (2.12) oder etwas Ähnliches gilt mit vernünftigen Konstanten C_{i1}, C_{i2}. Andernfalls heißt der Algorithmus* ***numerisch instabil*** *(oder* ***schlecht konditioniert****).*

Bemerkung 2.6.2 (Konsistenz eines Algorithmus) *Aus* (2.12) *ist nicht ersichtlich, was das exakte Datum $\varphi(x)$ mit $\widetilde{\varphi}(x)$ zu tun hat. (Z.B. ist $\widetilde{\varphi}(x) = 0$ ein stabiles Verfahren, aber im Allgemeinen nutzlos.) Für diesen Zusammenhang hat man in der Numerik den Begriff der* **Konsistenz***. Ein numerisches Verfahren heißt* ***konsistent****, wenn der Algorithmus (in einer noch näher zu bestimmenden Art und Weise) tatsächlich das gegebene Problem löst und nicht ein anderes. Wir werden dies später (in Numerik 4) noch insbesondere im Zusammenhang mit der numerischen Lösung von gewöhnlichen Differentialgleichungen untersuchen, aber man mag schon jetzt festhalten, dass Stabilität und Konsistenz für die Konvergenz einer numerischen Näherungslösungsfolge gegen die exakte Lösung wichtig sind.*

Bemerkung 2.6.3 (Standardfehler) *Typisch für das Entstehen numerischer Instabilität ist, dass man das Ausgangsproblem φ in zwei Teilschritte $\varphi = \varphi^{(2)} \circ \varphi^{(1)}$ zerlegt, von denen einer erheblich schlechter konditioniert ist als das Ausgangsproblem.*

Beispiel 2.6.4 (Quadratische Gleichung) Wir untersuchen zwei verschiedene Verfahren zur Lösung von (2.10) in Beispiel 2.5.6

$$\varphi(p,q) = -p + \sqrt{p^2+q} \text{ löst } x^2 + 2px - q = 0.$$

Methode 2.6.5

$$u = \sqrt{p^2+q}, \quad y = \chi_1(p,u) = -p + u. \tag{2.13}$$

Falls $u \approx p$, d.h. falls $p \gg q$, sind die Konditionszahlen von χ_1 erheblich größer als 1 (Auslöschung).

Methode 2.6.6

$$u = \sqrt{p^2+q}, \quad y = \chi_2(p,q,u) = \frac{q}{p+u}. \tag{2.14}$$

Die Konditionszahlen von χ_2 sind kleiner als 1. (Das Verfahren beruht darauf, dass $-p-u$ die andere Lösung und das Produkt der beiden Lösungen gleich $-q$ ist (Satz von Viëta).)

Es stellt sich heraus, dass Algorithmus 2.6.5 numerisch instabil, aber Algorithmus 2.6.6 numerisch stabil ist.

Bemerkung 2.6.7 *Man beachte beim Algorithmus 2.6.5 in Beispiel 2.6.4, dass die numerische Auswertung der Funktion χ_1 für sich genommen (trotz Auslöschung) nicht numerisch instabil ist! Schlecht konditioniert ist hier das Problem, $\chi_1(p, \sqrt{p^2+q})$ zu berechnen.*
Umgekehrt kann man daran sehen, dass das Zusammensetzen zweier gut konditionierter Algorithmen sehr wohl einen schlecht konditionierten Algorithmus für das Gesamtproblem ergeben kann. Diese Tatsache steht in unangenehmem, aber unvermeidlichem Kontrast zu dem Wunsch, ein Problem in unabhängig voneinander zu bearbeitende Teilprobleme zu zerlegen.

Die bislang eingeführen Konditionszahlen erlauben die Untersuchung der Fehlerverstärkung für jede einzelne Komponente von φ. Manchmal ist man ist aber nur an einem Gesamtfehler (im Sinne eines Fehlermaßes, einer Norm) interessiert. Dann sind folgende Begriffe sinnvoll.

Definition 2.6.8 *(a) Es sei $\widetilde{x} \in X$ eine Approximation von $x \in X$. Als* **Fehler** *bezeichnet man $\Delta x := x - \widetilde{x}$ und zu einer gegebenen Norm $\|\cdot\|_X$ ist*

$$\|x - \widetilde{x}\|_X$$

der **absolute Fehler** *von $\widetilde{x}$. Es sei $x \neq 0$, dann beschreibt*

$$\frac{\|x-\widetilde{x}\|_X}{\|x\|_X} = \frac{\|\Delta x\|_X}{\|x\|_X}$$

den **relativen Fehler** *von $\widetilde{x}$.*

(b) Seien $\|\cdot\|_X$, $\|\cdot\|_Y$ geeignete Normen auf X bzw. Y und

$$\delta_x := \frac{\|\Delta x\|_X}{\|x\|_X}\ ,\ \delta_y := \frac{\|\Delta y\|_Y}{\|y\|_Y}$$

die relativen Ein- bzw. Ausgabefehler mit $\Delta x := x - \tilde{x}$, $\Delta y := y - \tilde{y}$. Dann heißt

$$\kappa_\varphi := \frac{\delta_y}{\delta_x} \tag{2.15}$$

*die **(relative) Kondition** des Problems $\varphi(x)$ beschrieben durch die Funktion $\varphi : X \to Y$. Die Größe*

$$\kappa_{\varphi,\text{abs}} := \frac{\|\Delta y\|_Y}{\|\Delta x\|_X} \tag{2.16}$$

*heißt **absolute Kondition** von $y = \varphi(x)$.*

(c) *Ein Problem heißt **gut konditioniert**, wenn „kleine" Schranken für κ_φ für $\delta_x \to 0$ existieren. Offenbar wäre $\kappa_\varphi = 1$ (bzw. $\kappa_\varphi \approx 1$) optimal.*

Bemerkung 2.6.9 *Relative Fehler in der ∞-Norm können durch eine Aussage über die Anzahl korrekter Stellen von $\widetilde{x}$ ausgedrückt werden, d.h.*

$$\frac{\|x - \widetilde{x}\|_\infty}{\|x\|_\infty} \approx 10^{-p},$$

falls die betragsgrößte Komponente von $\widetilde{x}$ näherungsweise p korrekte signifikante Stellen hat.

3 Direkte Lösung linearer Gleichungssysteme

Lineare Gleichungssysteme (LGS) sind zwar ein verhältnismäßig einfaches Problem, bedürfen aber gerade für große Dimensionen sehr guter numerischer Lösungsverfahren. Hinzu kommt, dass LGS in extrem vielen Anwendungen vorkommen. Oft kann man das Wissen über die „Herkunft“ eines Gleichungssystems zu dessen schnellem Lösen nutzen.

Beispiel 3.0.1 (Schwingungsgleichung) Gegeben sei eine elastische Saite der Länge 1, die an beiden Enden fixiert ist. Die Saite wird nun durch eine äußere Kraft f ausgelenkt (angezupft). Wir wollen die Auslenkung u der Saite aus ihrer Ruhelage als Funktion von $x \in [0,1]$ berechnen. Die

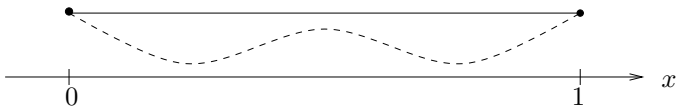


Abb. 3.1: Elastische, an beiden Enden fixierte Saite.

gesuchte Auslenkung $u : [0,1] \to \mathbb{R}$ ist Lösung des folgenden linearen Randwertproblems zweiter Ordnung

$$-u''(x) + \lambda(x)u(x) = f(x)\ ,\ x \in (0,1), \qquad u(0) = u(1) = 0 \tag{3.1}$$

mit gegebenen $f : (0,1) \to \mathbb{R}$ und $\lambda \in \mathbb{R}$. Genaueres zur Modellierung findet man z.B. in [Arendt/Urban].
Wir wollen (3.1) näherungsweise mit Hilfe eines numerischen Verfahrens lösen. Dazu unterteilen wir $[0,1]$ in Teilintervalle gleicher Länge. Die Anzahl der Intervalle sei $N > 1, N \in \mathbb{N}$ und $h = \frac{1}{N}$ die Schrittweite. Dann setzt man $x_i := ih\ ,\ i = 0, \ldots, N \quad (x_0 = 0, x_N = 1)$, die x_i werden

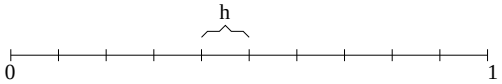


Abb. 3.2: Unterteilung des Intervalls in Teilintervalle der Länge $h > 0$.

als **Knoten** bezeichnet. Die **Schrittweite** ist $h := x_{i+1} - x_i$ für alle i, man spricht von einem **äquidistanten Gitter**. Wir wollen die Lösung an den Knoten x_i approximieren und ersetzen hierzu (wie aus der Analysis bekannt) die zweite Ableitung durch den zentralen Differenzenquotienten

$$u''(x_i) \approx \frac{1}{h^2}\big(u(x_{i-1}) - 2u(x_i) + u(x_{i+1})\big) =: D_c^2 u(x_i).$$

Es gilt bekanntlich $\|u'' - D_c^2 u\| = \mathcal{O}(h^2)$, falls $u \in C^4[0,1]$. Damit erhält man für die Näherung $u_i \approx u(x_i)$ also folgende Bedingungen ($\lambda_i = \lambda(x_i), f_i = f(x_i)$):

$$\begin{cases} \frac{1}{h^2}(-u_{i-1} + 2u_i - u_{i+1}) + \lambda_i u_i = f_i,\ 1 \le i \le N-1, \\ u_0 = u_N = 0, \end{cases}$$

also ein lineares Gleichungssystem der Form

$$\underbrace{\begin{bmatrix} (2+h^2\lambda_1) & -1 & & 0 \\ -1 & (2+h^2\lambda_2) & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & (2+h^2\lambda_{n-1}) \end{bmatrix}}_{=:A_h} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix}}_{=:u_h} = \underbrace{\begin{bmatrix} h^2 f_1 \\ \vdots \\ \vdots \\ h^2 f_{n-1} \end{bmatrix}}_{=:f_h},$$

d.h. $A_h u_h = f_h$. Für $N \to \infty$ ($h \to 0$) konvergiert die „**diskrete Lösung**“ u_h gegen die Lösung u von (3.1). Allerdings wächst die Dimension der Matrix A_h mit kleiner werdendem h. Bei mehrdimensionalen Problemen führt dies leicht zu sehr großen LGS. Wir nennen diese Matrix auch **Standardmatrix**.

3.1 Einführung, Cramersche Regel

Wir beginnen mit dem klassischen Verfahren zur Lösung eines linearen Gleichungssystems (LGS), der Gaußschen[1] Eliminationsmethode.

Zu lösen ist ein System von n linearen Gleichungen mit n Unbekannten $x_1, \ldots, x_n \in \mathbb{R}$

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{3.2}$$

oder kurz

$$Ax = b, \tag{3.3}$$

wobei $A \in \mathbb{R}^{n\times n}$ eine reelle $n \times n$-Matrix (also insbesondere eine quadratische Matrix) ist und $b, x \in \mathbb{R}^n$ reelle (Spalten-)Vektoren sind.

Wann ist ein lineares Gleichungssystem überhaupt lösbar? Aus der Linearen Algebra (siehe z.B. [Wille]) kennen wir das folgende Resultat, das die Lösbarkeit mit Hilfe der Determinante der Matrix A charakterisiert.

Satz 3.1.1 (Lösbarkeit) *Sei $A \in \mathbb{R}^{n\times n}$ mit $\det A \neq 0$ und $b \in \mathbb{R}^n$. Dann existiert genau ein $x \in \mathbb{R}^n$, so dass $Ax = b$.*

Falls $\det A \neq 0$, so lässt sich die Lösung $x = A^{-1}b$ mit der **Cramerschen Regel** berechnen, d.h.

$$x_i = \frac{1}{\det A} \begin{vmatrix} a_{11} & \ldots & b_1 & \ldots & a_{1n} \\ a_{21} & \ldots & b_2 & \ldots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{nn} & \ldots & b_n & \ldots & a_{nn} \end{vmatrix} = \frac{D_i}{D} \qquad (i = 1, \ldots, n).$$

Dabei geht die im Zähler stehende Determinante D_i dadurch aus $D := \det A$ hervor, dass man die i-te Spalte der Matrix A durch den Vektor b der rechten Seite ersetzt.

[1] **Carl Friedrich Gauß**, 1777 - 1855. Lagrange hatte 1759 die Methode schon vorweggenommen und in China war sie schon vor dem ersten Jahrhundert bekannt. Näheres zu Gauß, Lagrange und weiteren Mathematikern findet man im Internet unter `www-groups.dcs.st-andrews.ac.uk/~history`.

Man beachte hier die Verbindung von Existenz- und Eindeutigkeitsaussage mit dem Rechenverfahren, was einen „guten" Algorithmus ausmacht. Dieser braucht dabei nicht unbedingt optimal zu sein!
Wenn wir die Lösung eines linearen Gleichungssystems mit Hilfe der Cramerschen Regel bestimmen wollen, müssen wir die verschiedenen auftretenden Determinanten berechnen.

Die Determinanten von $n \times n$-Matrizen lassen sich mittels der **Leibnizschen Darstellung** berechnen, d.h.

$$\det A := \sum_{\pi} (-1)^{j(\pi)} a_{1i_1}\, a_{2i_2} \cdots a_{ni_n}\,,$$

wobei die Summe über alle möglichen $n!$ Permutationen π der Zahlen $1, 2, \ldots, n$ zu berechnen ist. Der Wert des Ausdrucks $(-1)^{j(\pi)}$ ergibt sich aus der Anzahl $j(\pi)$ der Inversionen der Permutation $\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$.

Bemerkung 3.1.2 (Rechenoperationen) *Im Folgenden werden die Verknüpfungen Multiplikation, Addition, Division und Subtraktion in ihrem Rechenaufwand nicht unterschieden und unter dem Begriff* **Gleitkommaoperation** *zusammengefasst (1 Gleitkommaoperation $\simeq$ 1 FLOP[2]). Systematische Multiplikationen mit ± 1 bleiben im Allgemeinen unberücksichtigt. Anderweitige Operationen wie z.B. Wurzelziehen werden gesondert betrachtet.*

Satz 3.1.3 (Aufwand Leibnizsche Darstellung) *Sei $A \in \mathbb{R}^{n\times n}$. Der Aufwand zur Berechnung von $\det A$ mit der Leibnizschen Darstellung, d.h. als Summe über alle Permutationen der Menge $\{1, \ldots, n\}$, beträgt*

$$\text{FLOP}(\det A) = n\, n! - 1\,.$$

Beweis. Der Aufwand zur Berechnung von $\det A$ für $A \in \mathbb{R}^{n\times n}$ in der Leibnizschen Darstellung (unter Vernachlässigung der Multiplikation mit $(-1)^{j(\pi)}$) ergibt sich wie folgt:

$$\begin{aligned} \text{FLOP}(\det A) &= \text{„}(n! - 1) \text{ Additionen } + n! \text{ Produkte mit } n \text{ Faktoren "} \\ &= n! - 1 + n!(n-1) = n\, n! - 1\,. \end{aligned}$$

Damit ist die Aussage des Satzes bewiesen. □

Alternativ lässt sich die Determinante rekursiv mit Hilfe des **Laplaceschen Entwicklungssatzes** berechnen. Dieser lautet für eine quadratische Matrix $A \in \mathbb{R}^{n\times n}$

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}),$$

wobei A_{1j} diejenige Matrix ist, die aus A durch Streichen der ersten Zeile und der j-ten Spalte entsteht.

Satz 3.1.4 (Aufwand Laplacescher Entwicklungssatz) *Sei $A \in \mathbb{R}^{n\times n}$ ($n \geq 2$). Der Aufwand zur Berechnung von $\det A$ mit dem Laplaceschen Entwicklungssatz beträgt*

$$\text{FLOP}(\det A) = \sum_{k=0}^{n} \frac{n!}{k!} - 2 < e\, n! - 2,$$

wobei $e = \exp(1)$ die Eulersche Zahl bezeichnet.

[2] **FLOP** (Floating point operation) ist nicht zu verwechseln mit **FLOP/s** oder **flops** (floating point operations per second), welches als Maßeinheit für die Geschwindigkeit von Computersystemen verwendet wird.

Beweis. Die Ungleichung ist klar. Die Gleichung lässt sich induktiv beweisen:

Induktionsanfang ($n = 2$): Die Determinante der Matrix $A = (a_{ij}) \in \mathbb{R}^{2\times 2}$ lautet $\det(A) = a_{11}a_{22} - a_{21}a_{12}$, d.h. der Aufwand beträgt 2 Multiplikationen und 1 Addition, also

$$\text{FLOP}\left(\det(\mathbb{R}^{2\times 2})\right) = 3 = \frac{2!}{0!} + \frac{2!}{1!} + \frac{2!}{2!} - 2\,.$$

Induktionsschritt ($n \to n+1$): Für $n \geq 2$ gilt

$$\begin{aligned}
\text{FLOP}\left(\det(\mathbb{R}^{(n+1)\times(n+1)})\right) &= \text{„}n \text{ Additionen } + (n+1) \text{ Multiplikationen} \\
&\quad + (n+1) \text{ Berechnungen von } \det(\mathbb{R}^{n\times n})\text{“} \\
&= n + (n+1) + (n+1)\cdot \text{FLOP}(\det(\mathbb{R}^{n\times n})) \\
&\overset{\text{IV}}{=} 2n + 1 + (n+1)\cdot\left(\sum_{k=0}^{n}\frac{n!}{k!} - 2\right) \\
&= 2n + 1 + \sum_{k=0}^{n}\frac{(n+1)!}{k!} - 2(n+1) \\
&= \sum_{k=0}^{n}\frac{(n+1)!}{k!} - 1 = \sum_{k=0}^{n+1}\frac{(n+1)!}{k!} - 2,
\end{aligned}$$

womit der Satz bewiesen ist. □

Beispiel 3.1.5 Ein einfaches Beispiel soll zeigen, dass man die Determinante überhaupt nur für sehr kleine allgemeine Matrizen der Dimension $n \ll 23$ mit dem Laplaceschen Entwicklungssatz berechnen kann.
Ein Jahr hat $365 \cdot 24 \cdot 60 \cdot 60 \approx 3 \cdot 10^7$ Sekunden. Geht man nun von einem schnellen Rechner[3] mit 3000 TFlops aus, so benötigt man zur Berechnung der Determinante einer 21×21-Matrix mit dem Laplaceschen Entwicklungssatz

$$\frac{\text{Anzahl der Operationen}}{\text{Gleitkommaoperationen pro Sekunde}}\,[s] = \frac{\sum_{k=0}^{21}\frac{21!}{k!} - 2}{3000\cdot 10^{12}}\,[s] \approx 46293\,[s] \approx 12.9\,[h]$$

bzw. für eine 25×25 -Matrix

$$\frac{\sum_{k=0}^{25}\frac{25!}{k!} - 2}{3000\cdot 10^{12}}\,[s] \approx 1.406\cdot 10^{10}\,[s] \approx 445.7 \text{ Jahre}\,.$$

Bemerkung 3.1.6 *Die Steigerung der Leistungsfähigkeit moderner Computer (Moore'sches Gesetz) ist bei weitem geringer als das Wachstum der Problemgröße mit steigendem n. Das Hoffen auf immer schnellere Rechner hilft nicht, um immer größere Probleme lösen zu können. Zum Vergleich: Im Jahr 2008 war der Supercomputer JUGENE vom Forschungszentrum Jülich weltweit Platz sechs und in Europa Platz eins mit 180 Flops. Für $n = 21$ brauchte man damals 214.3h, also immerhin eine Beschleunigung um den Faktor 16 in 4 Jahren. Trotzdem bleibt $n = 25$ auch in naher Zukunft unerreichbar.*

[3] Der weltweit auf Platz vier rangierende und zugleich schnellste europäische Supercomputer SuperMUC am Leibniz Rechenzentrum Garching bei München hat ca. 3100 TFlops (Stand Juni 2012). Ein aktueller PC hat eine Leistung von ca. 300 GFlops. T = Tera = 10^{12}, G = Giga = 10^9.

Bemerkung 3.1.7 *Bei der Cramerschen Regel ist zum Lösen eines linearen Gleichungssystems $Ax = b$ mit $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$ die Berechnung von $n+1$ Determinanten und n Quotienten notwendig. Der Aufwand zur Lösung eines linearen Gleichungssystems lässt sich somit zusammenfassen, wobei wir auf die Komplexität des Gauß-Verfahrens (siehe Seite 37) erst später eingehen werden.*

Cramersche Regel		Gauß-Elimination
Leibniz	Laplace	
$n\,(n+1)! - 1$	$\sum_{k=0}^{n} \frac{(n+1)!}{k!} - n - 2$	$\frac{4n^3 + 9n^2 - n}{6}$

Tab. 3.1: Aufwand für die Lösung eines linearen Gleichungssystems mit verschiedenen direkten Verfahren.

Bemerkung 3.1.8 *Für das Lösen eines LGS mit 21 Unbekannten mit der Cramerschen Regel und dem Laplaceschen Entwicklungssatz benötigt man auf einem der schnellsten Rechner mehr als 10 Tage, d.h.* $22 \cdot 12.9\ [h] = 283.8\ [h] = 11.825\ [d]$ *(22 verschiedene Determinanten im Zähler und eine im Nenner, vgl. Beispiel 3.1.5). Ein System mit 24 Unbekannten ist mit einem heutigen Supercomputer und der Cramerschen Regel nicht in einem Menschenleben zu lösen.*

Beispiel 3.1.9 (Vergleich Rechenaufwand) Aufwand zum Lösen eines linearen Gleichungssystems $Ax = b$ via Cramerscher Regel und Gauß-Verfahren.

Für einige n sei die Anzahl der notwendigen FLOPs wiedergegeben.

	Cramer/Leibniz	Cramer/Laplace	Gauß
n	$= n\,(n+1)! - 1$	$= \sum_{k=0}^{n} \frac{(n+1)!}{k!} - n - 2$	$= (4n^3 + 9n^2 - n)/6$
$n = 2$	11	11	11
$n = 3$	71	59	31
$n = 4$	479	319	66
$n = 5$	3599	1949	120
$n = 8$	2903039	986399	436
$n = 10$	399167999	108505099	815

Im Folgenden werden wir nun Verfahren beschreiben, die bereits für $n \geq 3$ effektiver als die Cramersche Regel sind. Mit Hilfe dieser Verfahren lassen sich auch Determinanten in polynomieller Zeit bestimmen. Daher wird die Cramersche Regel im Allgemeinen nur für $n = 2,\ 3$ verwendet. Der Vorteil der Cramerschen Regel ist jedoch die explizite Schreibweise, d.h. sie ist eine Formel für alle Fälle. Man spart sich bei ähnlichem Rechenaufwand (d.h. $n = 2, 3$) aufwendige Fallunterscheidungen (z.B. Pivotwahl) im Programm.

3.2 Gestaffelte Systeme

Betrachten wir zuerst besonders einfach zu lösende Spezialfälle. Am einfachsten ist sicherlich der Fall einer diagonalen Matrix A.

Diagonalmatrix

Man bezeichnet eine quadratische Matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ als **Diagonalmatrix**, falls $a_{ij} = 0$ für $i \neq j$, $i, j = 1, \ldots, n$ gilt. Häufig schreibt man eine Diagonalmatrix A mit Diagonaleinträgen $a_{11}, \ldots, a_{nn}$ auch einfach $A = \text{diag}(a_{11}, \ldots, a_{nn})$. Die Anwendung der Inversen einer Diagonalmatrix A mit Diagonalelementen $a_{ii} \neq 0$, $i = 1, \ldots, n$, auf einen Vektor b lässt sich als Pseudocode wie folgt schreiben:

Algorithmus 3.2.1: Lösen von $Ax = b$ mit Diagonalmatrix A

Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Diagonalmatrix und $b \in \mathbb{R}^n$

Input $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$

for $j = 1, \ldots, n$

$x_j = b_j / a_{jj}$

end

Output $x = (x_1, \cdots, x_n)^T$

Eine einfache Matlab Realisierung ist im Folgenden dargestellt.

Man löse $Ax = b$ mit

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \; b = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}.$$

MATLAB-Beispiel:

```
>> A=diag([1,2,4]);
>> b=[5;3;2];
>> for j=1:3, x(j)=b(j)/A(j,j); end
>> x
x =
    5.0000    1.5000    0.5000
```

Dreiecksmatrix

Der nächstschwierigere Fall ist der einer **Dreiecksmatrix** A. Man spricht von einer (quadratischen) oberen Dreiecksmatrix $A = (a_{ij})$, falls $a_{ij} = 0$ für $i > j$, $i, j = 1, \ldots, n$ und von einer unteren Dreiecksmatrix $A = (a_{ij})$, falls $a_{ij} = 0$ für $i < j, i, j = 1, \ldots, n$, gilt. Häufig verwenden wir auch R für eine obere Dreiecksmatrix und L für eine untere Dreiecksmatrix.
Betrachten wir nun das „gestaffelte“ Gleichungssystem

$$\begin{array}{ccccccccc} r_{11}x_1 & + & r_{12}x_2 & + & \cdots & + & r_{1n}x_n & = & z_1 \\ & & r_{22}x_2 & + & \cdots & + & r_{2n}x_n & = & z_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & r_{nn}x_n & = & z_n \end{array} \tag{3.4}$$

oder in Matrix-Vektor-Schreibweise

$$Rx = z, \tag{3.5}$$

wobei $R = (r_{ij}) \in \mathbb{R}^{n\times n}$ gilt. Offenbar erhalten wir x durch sukzessive Auflösung des „gestaffelten“ Gleichungssystems, beginnend mit der n-ten Zeile:

$$\begin{array}{lcll} x_n & := & z_n/r_{nn} & , \text{ falls } \quad r_{nn} \neq 0 \\ x_{n-1} & := & (z_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1} & , \text{ falls } \quad r_{n-1,n-1} \neq 0 \\ \vdots & & \vdots & \vdots \\ x_k & := & (z_k - \sum_{i=k+1}^{n} r_{ki}x_i)/r_{kk} & , \text{ falls } \quad r_{kk} \neq 0 \\ \vdots & & \vdots & \vdots \\ x_1 & := & (z_1 - r_{12}x_2 - \cdots - r_{1n}x_n)/r_{11} & , \text{ falls } \quad r_{11} \neq 0. \end{array} \tag{3.6}$$

Für die obere Dreiecksmatrix R gilt, dass

$$\det R = r_{11} \cdot r_{22} \cdot \cdots \cdot r_{nn} \tag{3.7}$$

und daher

$$\det R \neq 0 \Longleftrightarrow r_{ii} \neq 0 \quad (i = 1, \ldots, n). \tag{3.8}$$

Der angegebene Algorithmus ist also wiederum genau dann anwendbar, wenn det $R \neq 0$ (d.h., wenn R regulär ist), also unter der Bedingung des Existenz- und Eindeutigkeitssatzes (Satz 3.1.1).

Analog zum obigen Vorgehen lässt sich auch ein gestaffeltes lineares Gleichungssystem der Form

$$Lx = z$$

mit einer unteren Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ lösen. In diesem Fall beginnt man in der ersten Zeile mit der Berechnung von x_1 und arbeitet sich dann bis zur letzten Zeile zur Bestimmung von x_n vor.

Bemerkung 3.2.1 (Vorwärts-, Rückwärtssubstitution) *Das Lösen eines LGS mit oberer Dreiecksmatrix nennt man auch Rückwärtseinsetzen/-substitution (Index läuft „rückwärts“ von n nach 1), bzw. mit einer unteren Dreiecksmatrix auch Vorwärtseinsetzen/-substitution (Index läuft „vorwärts“ von 1 nach n).*

Satz 3.2.2 (Rechenaufwand Ax und $A^{-1}b$ – Dreiecksmatrix) *Sei $A \in \mathbb{R}^{n\times n}$ eine reguläre obere oder untere Dreiecksmatrix und x, $b \in \mathbb{R}^n$. Dann gilt*

$$\text{FLOP}(Ax) = n^2 \quad \textit{und} \quad \text{FLOP}(A^{-1}b) = n^2\,.$$

Bei der Berechnung von $A^{-1}b$ ist im Gegensatz zu Ax die Reihenfolge, in der die Zeilen „abgearbeitet” werden, festgelegt.

Beweis. Es genügt den Fall einer regulären oberen Dreiecksmatrix $R \in \mathbb{R}^{n\times n}$ zu betrachten. Für den Rechenaufwand zur Lösung von $Rx = b$ (oder die Anwendung von R^{-1} auf einen Vektor b) ergibt sich:

i) für die i-te Zeile: je $(n - i)$ Additionen und Multiplikationen sowie eine Division

ii) insgesamt für die Zeilen n bis 1: Betrachten wir zuerst die Summenformel

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

Mit i) erhält man insgesamt einen Aufwand von $2\,\frac{n^2-n}{2} + n = n^2$ Operationen.

Der zweite Teil der Aussage bleibt Ihnen als Übungsaufgabe überlassen. □

Man beachte allerdings, dass man allgemein für die Berechnung von $A^{-1}b$ zunächst die Inverse A^{-1} berechnen muß. Für ganz allgemeine reguläre Matrizen ist dies sehr aufwändig, weswegen wir unter der Schreibweise $A^{-1}b$ stets die Lösung des LGS mit gegebener rechten Seite b verstehen wollen und für diesen Fall schnelle Verfahren konstruieren wollen.

Aufgabe 3.2.3 Es sei $A \in \mathbb{R}^{n\times n}$ eine obere (oder untere) Dreiecksmatrix und $x \in \mathbb{R}^n$. Man zeige, dass das Matrix-Vektor-Produkt Ax mit n^2 Operationen berechnet werden kann.

Eine Matlab Realisierung zur Bestimmung von $R^{-1}b$, bei der die Matrix R zeilenweise durchlaufen wird, und ein Anwendungsbeispiel sind im Folgenden dargestellt.

MATLAB-Funktion: Rinvb.m

```
function x = Rinvb(R,b)
% compute solution of R * x = b with upper triangular matrix R
n = size(R,1);
x = zeros(n,1);
for j = n:-1:1
  for k=j+1:n
    b(j) = b(j) - R(j,k) * x(k);
  end
  x(j) = b(j) / R(j,j);
end
```

MATLAB-Beispiel:

Man löse $Rx = b$ mit

$$R = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 9 \\ 12 \\ 3 \end{pmatrix}.$$

```
>> R = [4,1,1;0,3,2;0,0,1];
>> b = [9;12;3];
>> x = Rinvb(R,b);
>> x'
ans =
     1     2     3
```

Eine alternative Realisierung, bei der die Matrix R spaltenweise durchlaufen wird, ist mit `Rinvb2.m` gegeben.

MATLAB-Funktion: Rinvb2.m

```
function x = Rinvb2(R,b)
% compute solution of R * x = b with upper triangular matrix R
n = size(R,1);
x = zeros(n,1);
for j = n:-1:1
  x(j) = b(j) / R(j,j);
  for k=1:j-1
    b(k) = b(k) - R(k,j) * x(j);
  end
end
```

Was ist der Unterschied? Machen Sie sich dies klar!

3.3 Gaußsche Eliminationsmethode

Gäbe es nun zu einer beliebigen Matrix A eine Zerlegung

$$A = L \cdot R\,, \tag{3.9}$$

so könnte ein beliebiges lineares Gleichungssystem $Ax = b$ durch eine Rückwärts- und Vorwärtssubstitution mittels der beiden Schritte

i) löse $Lz = b$,

ii) löse $Rx = z$,

gelöst werden. Die folgende Gaußsche Eliminationsmethode liefert gerade eine solche Zerlegung. Betrachten wir ein allgemeines lineares Gleichungssystem $Ax = b$ ($A \in \mathbb{R}^{n\times n}$ regulär, $b \in \mathbb{R}^n$)

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{3.10}$$

und versuchen dies in ein gestaffeltes System umzuformen.
Durch die folgenden drei Äquivalenzoperationen

1. Vertauschung von Zeilen,
2. Multiplikation einer Zeile mit einem Skalar $\neq 0$,
3. Addition eines Vielfachen einer Zeile zu einer anderen,

wird die Lösungsmenge des Gleichungssystems (3.10) nicht verändert.
Wir setzen zunächst $a_{11} \neq 0$ voraus. Um (3.10) nun in ein gestaffeltes System umzuformen, muss die erste Zeile nicht verändert werden. Die restlichen Zeilen werden so modifiziert, dass in einem ersten Schritt die Koeffizienten vor x_1 verschwinden, d.h. die Variable x_1 aus den Gleichungen in

den Zeilen 2 bis n eliminiert wird.
So entsteht ein System der Art

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}^{'}x_2 & + & \cdots & + & a_{2n}^{'}x_n & = & b_2^{'} \\ & & \vdots & & & & \vdots & & \vdots \\ & & a_{n2}^{'}x_2 & + & \cdots & + & a_{nn}^{'}x_n & = & b_n^{'}. \end{array} \tag{3.11}$$

Haben wir dies erreicht, so können wir das selbe Verfahren auf die letzten $(n-1)$ Zeilen anwenden und so rekursiv ein gestaffeltes System erhalten. Mit den Quotienten

$$l_{i1} = a_{i1}/a_{11} \qquad (i = 2, 3, \ldots, n) \tag{3.12}$$

sind die Elemente in (3.11) gegeben durch

$$\begin{aligned} a_{ik}^{'} &= a_{ik} - l_{i1}a_{1k}\,, \\ b_i^{'} &= b_i - l_{i1}b_1. \end{aligned} \tag{3.13}$$

Damit ist der erste Eliminationsschritt unter der Annahme $a_{11} \neq 0$ ausführbar. Die Matrix, die sich aus diesem ersten Eliminationsschritt ergibt, bezeichnen wir mit $A^{(2)}$.

Wenden wir auf diese Restmatrix die Eliminationsvorschrift erneut an, so erhalten wir eine Folge

$$A = A^{(1)} \to A^{(2)} \to \cdots \to A^{(n)} =: R$$

von Matrizen der speziellen Gestalt

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \ldots & & a_{2n}^{(2)} \\ & & \ddots & & \\ & & & a_{kk}^{(k)} & \ldots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k)} & \ldots & a_{nn}^{(k)} \end{pmatrix} \tag{3.14}$$

mit einer $(n-k+1, n-k+1)$-Restmatrix, auf die wir den Eliminationsschritt

$$\begin{array}{lcll} l_{ik} & := & a_{ik}^{(k)}/a_{kk}^{(k)} & \text{für} \quad i = k+1, \ldots, n \\ a_{ij}^{(k+1)} & := & a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)} & \text{für} \quad i,j = k+1, \ldots, n \\ b_i^{(k+1)} & := & b_i^{(k)} - l_{ik}b_k^{(k)} & \text{für} \quad i = k+1, \ldots, n \end{array} \tag{3.15}$$

ausführen können, wenn das **Pivotelement** (Dreh- und Angelpunkt) $a_{kk}^{(k)}$ nicht verschwindet. Da jeder Eliminationsschritt eine lineare Operation auf den Zeilen von A ist, lässt sich der Übergang von $A^{(k)}$ und $b^{(k)}$ zu $A^{(k+1)}$ und $b^{(k+1)}$ als Multiplikation mit einer Matrix $L_k \in \mathbb{R}^{n\times n}$ von links darstellen, d.h.

$$A^{(k+1)} = L_k A^{(k)}, \quad b^{(k+1)} = L_k b^{(k)}. \tag{3.16}$$

Die Matrix

$$L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & 1 \end{pmatrix} = I - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{pmatrix} \cdot e_k^T =: I - \ell_k \cdot e_k^T\,, \tag{3.17}$$

wobei e_k der k-te Einheitsvektor sei, hat die Eigenschaft, dass die Inverse L_k^{-1} aus L_k durch einen Vorzeichenwechsel in den Einträgen l_{ik} $(i > k)$ entsteht und für das Produkt der L_k^{-1} gilt

$$L := L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ l_{21} & 1 & \ddots & & \vdots \\ l_{31} & l_{32} & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{n1} & \ldots & & l_{n,n-1} & 1 \end{pmatrix}. \tag{3.18}$$

Zusammengefasst erhalten wir auf diese Weise das zu $Ax = b$ äquivalente gestaffelte System $Rx = z$ mit der oberen Dreiecksmatrix

$$R = L^{-1}A \qquad \text{und der rechten Seite} \qquad z = L^{-1}b. \tag{3.19}$$

Das **Gaußsche Eliminationsverfahren** schreibt sich dann wie folgt:
Sei $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$. Berechne nacheinander

i) $L \cdot R = A$ (Zerlegung mit R obere und L untere Dreiecksmatrix),

ii) $Lz = b$ (Vorwärtseinsetzen bzw. -substitution),

iii) $Rx = z$ (Rückwärtseinsetzen bzw. -substitution).

Definition 3.3.1 (unipotente Matrix) *Eine untere oder obere Dreiecksmatrix, deren Diagonalelemente alle gleich eins sind, heißt unipotent.*

Definition 3.3.2 (Gaußsche Dreieckszerlegung, LR-Zerlegung) *Die oben genannte Darstellung $A = L \cdot R$ der Matrix A als Produkt einer unipotenten unteren Dreiecksmatrix L und einer oberen Dreiecksmatrix R heißt* **Gaußsche Dreieckszerlegung** *oder LR-***Zerlegung** *von A.*

Satz 3.3.3 *Existiert die LR-Zerlegung einer quadratischen Matrix A, so sind L und R eindeutig bestimmt.*

Beweis. Übung. □

Satz 3.3.4 (Rechenaufwand Gaußsche Dreieckszerlegung, Gaußsche Elimination)
Sei $A \in \mathbb{R}^{n\times n}$ regulär. Existiert die LR-Zerlegung, so gilt

$$\text{FLOP}(LR - Zerlegung) = \frac{4n^3 - 3n^2 - n}{6}.$$

Des Weiteren sei $b \in \mathbb{R}^n$. Dann gilt für die Gaußsche Elimination

$$\text{FLOP}(A^{-1}b) = \frac{4n^3 + 9n^2 - n}{6},$$

d.h. die Lösung von $Ax = b$ kann mit $(4n^3 + 9n^2 - n)/6$ FLOPs berechnet werden.

Bemerkung 3.3.5 *Die Aufwandsberechnung unterscheidet sich teilweise in der Literatur. Neuerdings werden $+$ und $\cdot$ Operationen nicht mehr unterschieden!*

Beweis von Satz 3.3.4. Für den Rechenaufwand der Gaußschen Dreieckszerlegung gilt Folgendes:

i) für den k-ten Eliminationsschritt:
je $(n-k)$ Divisionen zur Berechnung der ℓ_{ik} und je $(n-k)^2$ Multiplikationen und Subtraktionen zur Berechnung der $a_{ij}^{(k+1)}$, d.h. $2\,(n-k)^2 + (n-k)$ Operationen

ii) für alle $n-1$ Eliminationsschritte erhalten wir durch Umindizierung und mit Summenformeln aus Analysis 1

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

iii) insgesamt benötigt man zum Lösen von $Ax = b$ mittels Gauß-Elimination, d.h. LR-Zerlegung und Vorwärts-, Rückwärtssubstitution, d.h. mit dem Lösen von zwei Gleichungssystemen (siehe Satz 3.2.2)

$$\frac{4n^3 - 3n^2 - n}{6} + 2n^2 = \frac{4n^3 + 9n^2 - n}{6} \quad \text{Operationen.}$$

Somit sind die Behauptungen des Satzes bewiesen. □

Das Speicherschema für die Gauß-Elimination orientiert sich an der Darstellung von $A^{(k)}$. In die erzeugten Nullen des Eliminationsschritts können die l_{ik} eingetragen werden. Da die Elemente mit den Werten 0 oder 1 natürlich nicht eigens gespeichert werden müssen, kann die LR-Zerlegung mit dem Speicherplatz der Matrix A bewerkstelligt werden:

$$A \to \left(\begin{array}{c|ccc} a_{11}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ \hline l_{21} & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & & \\ l_{n1} & a_{n2}^{(2)} & & a_{nn}^{(2)} \end{array} \right) \to \dots \to \left(\begin{array}{cccc} a_{11}^{(1)} & \dots & & a_{1n}^{(1)} \\ l_{21} & a_{22}^{(2)} & & a_{2n}^{(2)} \\ \vdots & \ddots & & \\ \vdots & & \ddots & \\ l_{n1} & \dots & l_{n,n-1} & a_{nn}^{(n)} \end{array} \right)$$

$$\Longrightarrow L = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ l_{21} & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ l_{n1} & \dots & \dots & l_{n,n-1} & 1 \end{pmatrix}, \quad R = \begin{pmatrix} a_{11}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_{nn}^{(n)} \end{pmatrix}.$$

3.4 Pivot-Strategien

Bisher haben wir uns noch nicht mit der Frage beschäftigt, ob eine LR-Zerlegung immer existiert und ob eine Vertauschung von Zeilen bei $Ax = b$ zu einem anderen Ergebnis führt, wenn der Algorithmus mit endlicher Rechengenauigkeit durchgeführt wird. (Bei exakter Arithmetik sind beide Ergebnisse gleich, bzw. Vertauschung von Zeilen in A führt zur gleichen Lösung). Betrachten wir hierzu zwei Beispiele:

Beispiel 3.4.1 i) **Permutation liefert LR–Zerlegung:** Sei $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Wie wir leicht sehen, gilt für die Eigenwerte $\lambda_{1,2} = \pm 1$ und somit $\det A \neq 0$. Falls eine Zerlegung existieren würde, gäbe es $a, b, c, d \in \mathbb{R}$ mit

$$A = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} b & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} b & c \\ ab & ac + d \end{pmatrix}.$$

Ein einfacher Vergleich der Koeffizienten zeigt jedoch (zuerst nach b und dann nach a), dass keine LR–Zerlegung existiert, obwohl nach Vertauschen der Zeilen trivialerweise eine LR–Zerlegung sofort gegeben ist. Die Frage, die sich nun stellt, lautet: Kann man für jede reguläre Matrix A durch Vertauschen ihrer Zeilen eine Matrix finden, für die dann eine LR–Zerlegung existiert?

ii) **Permutation liefert höhere Genauigkeit:** Berechnen wir die Lösung des folgenden linearen Gleichungssystems $(\epsilon > 0)$

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 && (3.20) \\ x_1 + x_2 &= 2\,. && (3.21) \end{aligned}$$

Bei exakter Arithmetik erhalten wir

$$\tfrac{1}{\epsilon - 1} \begin{pmatrix} 1 & -1 \\ -1 & \epsilon \end{pmatrix} \begin{matrix} 11 \\ 2 \end{matrix} = \tfrac{1}{1-\epsilon} \begin{matrix} 11 \\ 1 - 2\epsilon \end{matrix}$$

d.h. $x_1 = \frac{1}{1-\epsilon}$, $x_2 = \frac{1-2\epsilon}{1-\epsilon}$. Für $2\epsilon <$ Rechengenauigkeit (d.h. $1 \pm 2\epsilon$ und 1 werden im Rechner durch dieselbe Zahl dargestellt) gilt nun

$$x_1 = 1, \quad x_2 = 1.$$

Für die Gauß-Elimination erhalten wir, wenn wir den Koeffizienten vor x_1 in (3.21) eliminieren, d.h. das $1/\epsilon$-fache von (3.20) subtrahieren

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 \\ \left(1 - \tfrac{1}{\epsilon}\right) x_2 &= 2 - \tfrac{1}{\epsilon}\,. \end{aligned}$$

Somit ergibt sich für $2\epsilon <$ Rechengenauigkeit aus der letzten Gleichung $x_2 = 1$ und damit aus der ersten Gleichung $x_1 = 0$. Vertauschen wir jedoch 1. und 2. Zeile (bzw. eliminieren den Koeffizienten vor x_1 in (3.20)) so erhalten wir

$$\begin{aligned} x_1 + x_2 &= 2 \\ (1 - \epsilon)\, x_2 &= 1 - 2\epsilon\,. \end{aligned}$$

Dies liefert $x_1 = x_2 = 1$ bei $2\epsilon <$ Rechengenauigkeit.

MATLAB-Beispiel:

Dieses scheinbar theoretische Ergebnis aus Beispiel 3.4.1 können wir direkt in Matlab umsetzen. Da die Recheneinheiten heutiger Rechner meist 2 Stellen genauer rechnen als unsere bisherigen theoretischen Untersuchungen vermuten lassen, muss hier $\epsilon \leq$ Maschinengenauigkeit$/8$ gelten.

```
>> e = eps/8;  % eps/2 gen\"ugt nicht!
>> x(2) = (2 - 1/e) / (1 - 1/e);
>> x(1) = (1 - x(2)) / e
x =
     0     1
>> x(2) = (1 - 2*e) / (1 - e);
>> x(1) = 2 - x(2)
x =
     1     1
```

Das Vertauschen kann folglich nötig sein. Zum einen, damit eine LR-Zerlegung überhaupt existiert, zum anderen, um den „Rechenfehler“ bedingt durch Rundungsfehler möglichst klein zu halten. Daraus leiten sich besondere Strategien zur Wahl des Pivotelements ab.

Betrachten wir nun die Gauß-Elimination mit **Spaltenpivotstrategie**, welche theoretisch nur dann nicht zielführend sein kann, falls die Matrix A singulär ist.

Algorithmus 3.4.1: Gauß-Elimination mit Spaltenpivotstrategie

a) Wähle im Eliminationsschritt $A^{(k)} \to A^{(k+1)}$ ein $p \in \{k, \ldots, n\}$, so dass

$$|a_{pk}^{(k)}| \geq |a_{jk}^{(k)}| \qquad \text{für } j = k, \ldots, n\,.$$

Die Zeile p soll die Pivotzeile werden.

b) Vertausche die Zeilen p und k

$$A^{(k)} \to \widetilde{A}^{(k)} \quad \text{mit} \quad \widetilde{a}_{ij}^{(k)} = \begin{cases} a_{kj}^{(k)} & \text{falls} \quad i = p \\ a_{pj}^{(k)} & \text{falls} \quad i = k \\ a_{ij}^{(k)} & \text{sonst.} \end{cases}$$

Nun gilt

$$|l_{ik}| = \left| \frac{\widetilde{a}_{ik}^{(k)}}{\widetilde{a}_{kk}^{(k)}} \right| = \left| \frac{\widetilde{a}_{ik}^{(k)}}{a_{pk}^{(k)}} \right| \leq 1.$$

c) Führe den nächsten Eliminationsschritt angewandt auf $\widetilde{A}^{(k)}$ aus,

$$\widetilde{A}^{(k)} \to A^{(k+1)}.$$

Bemerkung 3.4.2 *Anstelle der Spaltenpivotstrategie mit Zeilentausch kann man auch eine Zeilenpivotstrategie mit Spaltentausch durchführen. Beide Strategien benötigen im schlimmsten Fall $\mathcal{O}(n^2)$ zusätzliche Operationen. In der Praxis werden im Gegensatz zum oben genannten Algorithmus die Zeile bzw. Spalte nicht umgespeichert, sondern besondere Indexvektoren verwendet (siehe Matlabfunktion* `mylu.m`*). Die Kombination von Spalten- und Zeilenpivotstrategie führt zur vollständigen Pivotsuche, bei der die gesamte Restmatrix nach dem betragsgrößten Eintrag durchsucht wird. Wegen des Aufwands $\mathcal{O}(n^3)$ wird dies jedoch so gut wie nie angewandt.*

MATLAB-Funktion: mylu.m

```
function [L,R,P] = mylu(A)
n = size(A,1);                          % get leading dimension of A
p = 1 : n;                              % pivot element vector
for k = 1 : n-1                         % consider k-th column
  [m,mptr] = max(abs(A(p(k:end),k)));   % find pivot element
  tmp = p(k);                           % interchange in vector p
  p(k) = p(k-1+mptr);
  p(k-1+mptr) = tmp;

  for j = k+1 : n                       % modify entries in
    A(p(j),k) = A(p(j),k)/A(p(k),k);    % compute  l_jk, store in A
    for i = k+1 : n                     % (n-k-1)*(n-k-1) submatrix
      A(p(j),i) = A(p(j),i) ...
                  - A(p(j),k)*A(p(k),i);
    end
  end
end
L = tril(A(p,:),-1)+eye(n);             % these lines could be
R = triu(A(p,:));                       % neglected, all information
P = eye(n);                             % already in A and p
P(:,p) = P;
```

Satz 3.4.3 *Es sei $A \in \mathbb{R}^{n\times n}$ regulär. Dann existiert vor dem k-ten Eliminationsschritt des Gauß-Algorithmus stets eine Zeilen-/Spaltenpermutation derart, dass das k-te Diagonalelement von Null verschieden ist. Bei Zeilenpermutation, d.h. Spaltenpivotisierung, sind alle Einträge von L vom Betrag kleiner oder gleich eins.*

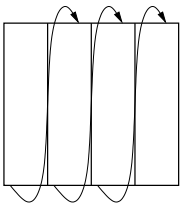
Beweis. Nach Voraussetzung gilt $\det A \neq 0$. Angenommen, es sei $a_{11} = 0$. Dann existiert eine Zeilenvertauschung, so dass das erste Pivotelement $a_{11}^{(1)}$ von Null verschieden und das betragsgrößte Element in der Spalte ist, d.h. $0 \neq |a_{11}^{(1)}| \geq |a_{i1}^{(1)}|$ für $i = 1, \ldots, n$, denn andernfalls wäre die Determinante von A in Widerspruch zur Voraussetzung gleich Null. Die Zeilenvertauschung hat dabei nur einen Vorzeichenwechsel in der Determinante zur Folge. Die Überlegungen für den ersten Eliminationsschritt übertragen sich sinngemäß auf die folgenden, reduzierten Systeme, bzw. ihre zugehörigen Determinanten. Diese Schlussfolgerungen gelten analog bei Zeilenpivotisierung. □

Eine unmittelbare Folge aus der Beweisführung des letzten Satzes ist das folgende Lemma.

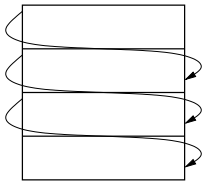
Lemma 3.4.4 *Erfolgen im Verlauf des Gauß-Algorithmus total m Zeilenvertauschungen (Spaltenvertauschungen), so ist die Determinante von A gegeben durch*

$$\det A = (-1)^m \prod_{k=1}^{n} r_{kk}.$$

Bemerkung 3.4.5 *Numerisch sind Spalten- und Zeilenpivotisierungen äquivalent. Die Auswahl hängt von der Speichermethode der Matrix A ab, d.h. man favorisiert die Spaltenpivotisierung, wenn die Einträge der Matrix spaltenweise im Speicher abgelegt sind, z.B. bei den Programmiersprachen Matlab, Fortran und die Zeilenpivotisierung u.a. bei C, denn hier sind die Matrizen als Folge von Zeilenvektoren abgelegt.*



Spaltenpivotisierung, bzw.



Zeilenpivotisierung.

Die genannten Pivotisierungen lassen sich formal durch die Multiplikation mit einer geeigneten Permutationsmatrix P beschreiben. Diese ist eine quadratische Matrix, welche in jeder Zeile und in jeder Spalte genau eine Eins und sonst Nullen enthält. Die Determinante ist $\det P = \pm 1$ und die Inverse ist durch $P^{-1} = P^T$ gegeben. Die Zeilenvertauschungen bei Spaltenpivotisierung entsprechen dabei der Multiplikation $P \cdot A$, analog lassen sich Spaltenpermutationen bei Zeilenpivotisierung durch $A \cdot P$ ausdrücken. Aus dem letzten Satz ergibt sich folgendes Resultat.

Satz 3.4.6 (Existenz einer Zerlegung $PA = LR$) *Zu jeder regulären Matrix A existiert eine Permutationsmatrix P, so dass $P \cdot A$ in ein Produkt $L \cdot R$ zerlegbar ist, d.h.*

$$P \cdot A = L \cdot R\,.$$

Ist nun immer eine Pivotisierung notwendig? Wir nennen im Folgenden ein Kriterium, welches leicht zu überprüfen ist.

Definition 3.4.7 (Diagonaldominanz) *Eine Matrix heißt diagonaldominant, falls gilt*

$$|a_{ii}| \geq \sum_{\substack{k=1 \\ k \neq i}}^{n} |a_{ik}| \qquad \forall\, i = 1, \ldots, n\,.$$

Sie heißt strikt diagonaldominant, falls für alle i „$>$" anstatt „$\geq$" gilt.

Satz 3.4.8 *Es sei $A \in \mathbb{R}^{n \times n}$ eine diagonaldominante und reguläre Matrix. Dann existiert eine Zerlegung*

$$A = L \cdot R.$$

Beweis. Einen Beweis dieser Aussage (in leicht abgewandelter Form) findet man z.B. bei [Schwarz]. Nach Voraussetzung ist entweder $|a_{11}| \geq \sum_{k=2}^{n} |a_{1k}| > 0$ oder es gilt $\sum_{k=2}^{n} |a_{1k}| = 0$ und $|a_{11}| > 0$; dies folgt aus der Regularität von A. Somit ist $a_{11} \neq 0$ ein zulässiges Pivotelement für den ersten Eliminationsschritt. Man muss nun zeigen, dass sich die Eigenschaft der diagonalen Dominanz auf das reduzierte Gleichungssystem überträgt. □

Zum Schluss dieses Abschnittes betrachten wir noch folgendes Beispiel zur konkreten Berechnung der LR-Zerlegung einer Matrix A.

Beispiel 3.4.9 (Berechnung von $PA = LR$) Sei

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

Diese Matrix ist offensichtlich nicht diagonaldominant, also wenden wir die Gauß-Elimination mit Spaltenpivotisierung an. Hierzu sei $P(i,j)$ diejenige Permutationsmatrix, die aus der Einheitsmatrix durch Vertauschen der i-ten und j-ten Zeile entsteht. Das Vertauschen der ersten mit der zweiten Zeile von A entspricht der Multiplikation der Matrix $P_1 := P(1,2)$ mit A von links, also

$$\widetilde{A}^{(1)} = P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 1 & 1 & 1 & 1 \\ -2 & 1 & -2 & 1 \end{pmatrix}.$$

Der erste Schritt der LR-Zerlegung liefert

$$L_1 = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ -\frac{1}{2} & & 1 & \\ 1 & & & 1 \end{pmatrix}, \qquad L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 2 & -1 & \frac{3}{2} \\ 0 & -1 & 2 & 0 \end{pmatrix}.$$

Bei der nächsten Spaltenpivotisierung sind keine Zeilenvertauschungen notwendig, also $P_2 = I$. Es folgt

$$L_2 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -1 & 1 & \\ & \frac{1}{2} & & 1 \end{pmatrix}, \qquad L_2 P_2 L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & 0 & \frac{7}{2} \\ 0 & 0 & \frac{3}{2} & -1 \end{pmatrix}.$$

Um nun auf eine obere Dreiecksgestalt zu kommen, müssen lediglich noch die dritte und vierte Zeile vertauscht werden, also formal $P_3 = P(3,4)$, $L_3 = I$ und damit

$$L_3 P_3 L_2 P_2 L_1 P_1 A = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & \frac{3}{2} & -1 \\ 0 & 0 & 0 & \frac{7}{2} \end{pmatrix} =: R\,.$$

Aber wie sehen nun P und L aus, sodass

$$P \cdot A = L \cdot R \quad ?$$

Wir betrachten hierzu im allgemeinen Fall mit $A \in \mathbb{R}^{n\times n}$ invertierbar für $k = 1, \ldots, n-1$ die Matrizen

$$\widetilde{L}_k := P_n \cdot \ldots \cdot P_{k+1} L_k P_{k+1} \cdot \ldots \cdot P_n\,,$$

wobei $P_n := I$ gesetzt wird und $P_1, \ldots, P_{n-1}$ analog zu Beispiel 3.4.9 gewählt werden. Dann gilt

$$\begin{aligned} \widetilde{L}_k &= P_n \cdot \ldots \cdot P_{k+1} L_k P_{k+1} \cdot \ldots \cdot P_n \\ &\overset{(3.17)}{=} P_n \cdot \ldots \cdot P_{k+1} (I - \ell_k e_k^T) P_{k+1} \cdot \ldots \cdot P_n \\ &= I - \underbrace{P_n \cdot \ldots \cdot P_{k+1} \ell_k}_{=:\widetilde{\ell}_k} \underbrace{e_k^T P_{k+1} \cdot \ldots \cdot P_n}_{=e_k^T} \\ &= I - \widetilde{\ell}_k e_k^T\,. \end{aligned}$$

Da die Multiplikation mit den Matrizen $P_{k+1}, \ldots, P_n$ von links bzw. rechts nur Zeilen- bzw. Spaltenvertauschungen innerhalb der Zeilen bzw. Spalten $k+1, \ldots, n$ bewirkt, besitzt die Matrix $\widetilde{L}_k$ dieselbe Struktur wie L_k. Aus der Definition der $\widetilde{L}_k$ folgt nun

$$\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1 P_{n-1} \cdot \ldots \cdot P_1 = L_{n-1} P_{n-1} \cdot \ldots \cdot L_2 P_2 L_1 P_1\,.$$

Nach Konstruktion der L_k und P_k gilt schließlich

$$\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1 P_{n-1} \cdot \ldots \cdot P_1 A = R\,.$$

Somit ergibt sich mit

$$L := \left(\widetilde{L}_{n-1} \cdot \ldots \cdot \widetilde{L}_1\right)^{-1} = \widetilde{L}_1^{-1} \cdot \ldots \cdot \widetilde{L}_{n-1}^{-1}$$

und

$$P := P_{n-1} \cdot \ldots \cdot P_1$$

eine Zerlegung $PA = LR$.

Bemerkung 3.4.10 *Man beachte, dass mit den obigen Überlegungen ein alternativer, konstruktiver Beweis des Satzes 3.4.6 vorliegt.*

Führen wir nun Beispiel 3.4.9 weiter fort:

Beispiel 3.4.11 Zu der Matrix A aus Beispiel 3.4.9 sollen die Matrizen L und P der Zerlegung $PA = LR$ bestimmt werden. Wir erhalten

$$P = P_3P_2P_1 = P(3,4)\,I\,P(1,2) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\widetilde{L}_1 = P_3P_2L_1P_2P_3 = P(3,4)\begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ -\frac{1}{2} & & 1 & \\ 1 & & & 1 \end{pmatrix} P(3,4) = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ 1 & & 1 & \\ -\frac{1}{2} & & & 1 \end{pmatrix},$$

$$\widetilde{L}_2 = P_3L_2P_3 = P(3,4)\begin{pmatrix} 1 & & & \\ & 1 & & \\ & -1 & 1 & \\ & \frac{1}{2} & & 1 \end{pmatrix} P(3,4) = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{1}{2} & 1 & \\ & -1 & & 1 \end{pmatrix},$$

$$\widetilde{L}_3 = P_4L_3P_4 = I\,,$$

$$L = \widetilde{L}_1^{-1}\widetilde{L}_2^{-1}\widetilde{L}_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 0 & 1 \end{pmatrix}$$

und somit

$$PA = \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ -2 & 1 & -2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 4 & -1 \\ 0 & 2 & -1 & -2 \\ 0 & 0 & \frac{3}{2} & -1 \\ 0 & 0 & 0 & \frac{7}{2} \end{pmatrix} = LR\,.$$

3.5 Nachiteration

Die oben diskutierten Pivotstrategien schließen offenbar nicht aus, dass die so berechnete Lösung x immer noch „ziemlich ungenau" ist. Wie kann man nun x ohne großen Aufwand verbessern?

Häufig lässt sich dies durch eine **Nachiteration** mit Hilfe einer expliziten Auswertung des **Residuums** $r := b - A\widetilde{x}$ erreichen. Dabei bezeichne $\widetilde{x}$ die durch ein numerisches Verfahren berechnete Näherung an x. Ausgehend von $\widetilde{x}$ soll die exakte Lösung mittels des Korrekturansatzes

$$x = \widetilde{x} + s$$

ermittelt werden. Der Korrekturvektor ist so zu bestimmen, dass die Gleichungen erfüllt sind, d.h.

$$Ax - b = A(\widetilde{x} + s) - b = A\widetilde{x} + As - b = 0.$$

Der Korrekturvektor s ergibt sich somit als Lösung von

$$As = b - A\widetilde{x} = r. \tag{3.22}$$

Bei der numerischen Lösung dieser Korrekturgleichung erhalten wir im Allgemeinen eine wiederum fehlerhafte Korrektur $\widetilde{s} \neq s$. Trotzdem erwarten wir, dass die Näherungslösung

$$\widetilde{x} + \widetilde{s}$$

„besser“ ist als $\widetilde{x}$.
Das Gleichungssystem (3.22) unterscheidet sich nur in der rechten Seite von dem ursprünglichen Problem $Ax = b$, sodass die Berechnung des Korrekturvektors s relativ wenig Aufwand erfordert, da beispielsweise bei Anwendung einer Nachiteration in Kombination mit der Gauß-Elimination schon die LR-Zerlegung der Koeffizientenmatrix A bekannt ist.

Bemerkung 3.5.1 *In der Praxis genügen bei Spalten- oder Zeilenpivotisierung meist wenige Nachiterationen, um eine Lösung auf eine dem Problem angepasste Genauigkeit zu erhalten.*

3.6 Cholesky-Verfahren

Wir wollen nun die Gauß-Elimination auf die eingeschränkte Klasse von Gleichungssystemen mit symmetrisch positiv definiten Matrizen anwenden. Es wird sich herausstellen, dass die Dreieckszerlegung in diesem Fall stark vereinfacht werden kann und dass dieser vereinfachte Algorithmus für große Matrizen nur den halben Aufwand an Operationen benötigt. Wir erinnern hier nochmals an die folgende Definition:

Definition 3.6.1 *Eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ heißt **positiv definit**, wenn $x^T Ax > 0$ für alle $x \in \mathbb{R}^n$ mit $x \neq 0$ ist.*

Bemerkung 3.6.2 *Die Bedingung in Definition 3.6.1 macht auch für nicht-symmetrische Matrizen Sinn. Deshalb fordert man oft zusätzlich zur positiven Definitheit die Symmetrie und nennt solche Matrizen **symmetrisch positiv definit**, abgekürzt **s.p.d.***

Positiv definite Matrizen haben folgende Eigenschaften.

Satz 3.6.3 (Eigenschaften s.p.d. Matrizen) *Für jede s.p.d. Matrix $A \in \mathbb{R}^{n\times n}$ gilt:*

i) A *ist invertierbar,*

ii) $a_{ii} > 0 \qquad$ *für* $\quad i = 1, \ldots, n\,,$

iii) $\max\limits_{i,j=1,\ldots,n} |a_{ij}| = \max\limits_{i=1,\ldots,n} a_{ii}\,,$

iv) Bei der Gauß-Elimination ohne Pivotsuche ist jede Restmatrix wiederum positiv definit.

Beweis. Wäre A nicht invertierbar, gäbe es einen Eigenwert $\lambda = 0$, da $\det A = 0$ nach Annahme. Dies steht aber im Widerspruch zur Voraussetzung, dass A positiv definit ist, da es ansonsten einen Eigenvektor $x \neq 0$ zu $\lambda = 0$ mit $Ax = \lambda x$ gäbe und somit dann auch $x^T Ax = 0$ gälte. Somit ist i) bewiesen. Setzt man für x einen kanonischen Einheitsvektor e_i ein, so folgt gerade $a_{ii} = e_i^T Ae_i > 0$ und daher gilt die Aussage ii). Behauptung iii) sei als Hausübung überlassen. Um die verbleibende Behauptung iv) zu beweisen, schreiben wir $A = A^{(1)}$ in der Form

$$A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline z & B^{(1)} \end{array}\right),$$

wobei $z = (a_{12}, \ldots, a_{1n})^T$ sei. Nach einem Eliminationsschritt ergibt sich

$$A^{(2)} = L_1 A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline 0 & \\ \vdots & B^{(2)} \\ 0 & \end{array}\right) \quad \text{mit} \quad L_1 = \begin{pmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{pmatrix}.$$

Multipliziert man nun $A^{(2)}$ von rechts mit L_1^T, so wird auch z^T in der ersten Zeile eliminiert und die Teilmatrix $B^{(2)}$ bleibt unverändert, d.h.

$$L_1 A^{(1)} L_1^T = \left(\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right).$$

Damit ist bewiesen, dass die Restmatrix $B^{(2)}$ symmetrisch ist. Können wir nun noch zeigen, dass $B^{(2)}$ auch wiederum positiv definit ist, so können wir unsere Argumentation sukzessive fortsetzen. Es sei $y \in \mathbb{R}^{n-1}$ mit $y \neq 0$. Da L_1 regulär ist, gilt $x^T := (0 \mid y^T) L_1 \neq 0$. Nach Voraussetzung ist A positiv definit, also gilt

$$0 < x^T Ax = (0 \mid y^T) L_1 A L_1^T \left(\frac{0}{y}\right) = (0 \mid y^T) \left(\begin{array}{c|c} a_{11} & 0 \\ \hline 0 & B^{(2)} \end{array}\right) \left(\frac{0}{y}\right) = y^T B^{(2)} y\,.$$

Somit haben wir gezeigt, dass auch $B^{(2)}$ wieder positiv definit ist. □

Mit Hilfe des letzten Satzes über die LR-Zerlegung können wir jetzt die **rationale** Cholesky-Zerlegung für symmetrische, positiv definite Matrizen herleiten.

Satz 3.6.4 (rationale Cholesky-Zerlegung) *Für jede symmetrisch positiv definite Matrix A existiert eine eindeutig bestimmte Zerlegung der Form*

$$A = \bar{L} D \bar{L}^T,$$

wobei $\bar{L}$ eine unipotente untere Dreiecksmatrix und D eine positive Diagonalmatrix ist.

Beweis. Wir setzen die Konstruktion im Beweis von Satz 3.6.3(iv) für $k = 2, \ldots, n-1$ fort und erhalten so unmittelbar $\bar{L}$ als Produkt der $L_1^{-1}, \ldots, L_{n-1}^{-1}$ und D als Diagonalmatrix der Pivotelemente. □

Bemerkung 3.6.5 *Man beachte die Eigenschaften der L_k bei der tatsächlichen Berechnung von L, d.h. das Produkt $L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1}$ ist nicht wirklich durch Multiplikation auszurechnen (vgl.* (3.18)*).*

Definition und Bemerkung 3.6.6 (Cholesky-Zerlegung) *Da alle Einträge der Diagonalmatrix D positiv sind, existiert $D^{1/2} = \mathrm{diag}(\pm\sqrt{d_i})$ und daher die Cholesky-Zerlegung*

$$A = L L^T,$$

wobei L die untere Dreiecksmatrix $L := \bar{L} D^{1/2}$ mit $\bar{L}$ und D aus der rationalen Cholesky-Zerlegung ist.

Bemerkung 3.6.7 *Die Cholesky-Zerlegung ist nicht eindeutig, da $D^{1/2}$ nur bis auf Vorzeichen der Elemente in der Diagonalen eindeutig bestimmt ist und somit auch $\overline{L} = LD^{1/2}$ nicht eindeutig ist.*

Zur Herleitung des Algorithmus zur Berechnung einer Cholesky-Zerlegung betrachten wir folgende Gleichung

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix} =$$

$$\begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} & \cdots & l_{11}l_{n1} \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{32} & \cdots & l_{21}l_{n1} + l_{22}l_{n2} \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 & \cdots & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} \\ \vdots & \vdots & \vdots & & \vdots \\ l_{11}l_{n1} & l_{21}l_{n1} + l_{22}l_{n2} & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} & \cdots & \sum_{k=1}^{n} l_{nk}^2 \end{pmatrix}$$

d.h.

$$\begin{aligned} &\text{für } i = k \qquad \text{gilt } a_{kk} = \sum_{j=1}^{k} l_{kj}^2 \quad \text{und} \\ &\text{für } i \neq k \qquad \text{gilt } a_{ik} = \sum_{j=1}^{k} l_{ij} \cdot l_{kj} \qquad (k+1 \leq i \leq n) \end{aligned}$$

und werten diese in einer geschickten Reihenfolge aus.

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$
Spaltenweise berechnet man für $k = 1, 2, \ldots, n$

$$\begin{aligned} l_{kk} &= \Big(a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2\Big)^{1/2} \\ l_{ik} &= \frac{1}{l_{kk}}\Big(a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot l_{kj}\Big) \qquad (k+1 \leq i \leq n). \end{aligned}$$

Bemerkungen 3.6.8 *i) Aus der Cholesky-Zerlegung $A = L \cdot L^T$ ergibt sich für $1 \le k \le n$ die Abschätzung*

$$\sum_{j=1}^{k} l_{kj}^2 \le \max_{1\le j\le n} |a_{jj}|.$$

Folglich sind alle Elemente der Matrix L betragsweise durch $\max_{1\le j\le n} \sqrt{|a_{jj}|}$ beschränkt. Die Elemente können damit nicht allzu stark anwachsen, was sich günstig auf die Stabilität des Verfahrens auswirkt.

ii) Da A symmetrisch ist, wird nur Information oberhalb und einschließlich der Hauptdiagonalen benötigt. Wenn man die Diagonalelemente l_{kk} separat in einem Vektor der Länge n speichert, und die Elemente $l_{jk},\ k < j$ unterhalb der Diagonale, so kann man die Information der Matrix A bewahren.

iii) Bei der algorithmischen Durchführung der Cholesky-Zerlegung liefert das Verfahren auch die Information, ob die Matrix positiv definit ist. Man mache sich dies als Übungaufgabe klar!

Satz 3.6.9 (Rechenaufwand Cholesky-Zerlegung) *Sei $A \in \mathbb{R}^{n\times n}$ positiv definit. Dann gilt*

$$\text{FLOP}(\textit{Cholesky-Zerl. von } A) = \frac{2n^3 + 3n^2 - 5n}{6} + n\,\text{Wurzelberechnung} = \frac{1}{3}n^3 + \mathcal{O}(n^2)\,.$$

Beweis. Untersuchen wir nun die Komplexität der Cholesky-Zerlegung zuerst für einen k-ten Zerlegungsschritt und betimmen dann den Gesamtaufwand.

i) Für den k-ten Zerlegungsschritt, d.h. die Berechnung der Elemente l_{ik} für festen Spaltenindex, sind jeweils $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Wurzelberechnung zur Berechnung des Diagonalelements nötig. Für jedes Nebendiagonalelement werden $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Division benötigt, d.h. bei $(n-k)$ Elementen unterhalb des k-ten Diagonalelements sind dies in der Summe

$$\begin{aligned} & \big(2(k-1) + (n-k)(2k-1)\big) \text{ FLOP und 1 Wurzelberechnung} \\ = \quad & \big(-2k^2 + (3+2n)k - (n+2)\big) \text{ FLOP und 1 Wurzelberechnung.} \end{aligned}$$

ii) Insgesamt ergibt sich dann für die Summe über alle Zerlegungschritte

$$\begin{aligned} & n \text{ Wurzelberechnungen} + \sum_{k=1}^{n} \big(-2k^2 + (3+2n)k - (n+2)\big) \\ = \quad & n \text{ Wurzelberechnungen} + \Big(-2\,\frac{(2n+1)(n+1)n}{6} + (3+2n)\,\frac{(n+1)n}{2} - (n+2)n\Big) \\ = \quad & n \text{ Wurzelberechnungen} + \frac{-(4n^3+6n^2+2n) + (6n^3+15n^2+9n) - (6n^2+12n)}{6} \\ = \quad & n \text{ Wurzelberechnungen} + \frac{2n^3+3n^2-5n}{6}\,. \end{aligned}$$

Da eine einzelne Wurzelberechnung an Aufwand ein konstantes Vielfaches einer Gleitkommaoperation benötigt, kann man den Term n Wurzelberechnungen $+\,(3n^2-5n)/6$ zu $\mathcal{O}(n^2)$ Operationen zusammenfassen. □

Bemerkung 3.6.10 *Für große n ist der Aufwand des Cholesky-Verfahrens im Vergleich zum Gauß-Algorithmus ungefähr halb so groß.*

3.7 Bandgleichungen

Eine weitere Klasse spezieller Matrizen, welche beim Lösen linearer Gleichungssysteme eine besondere Rolle spielen, sind Bandmatrizen. Man spricht von einer Bandmatrix A, falls alle von Null verschiedenen Elemente a_{ik} in der Diagonale und in einigen dazu benachbarten Nebendiagonalen stehen. Für die Anwendungen sind insbesondere die symmetrischen, positiv definiten Bandmatrizen wichtig.

Definition 3.7.1 *Unter der* **Bandbreite** m *einer symmetrischen Matrix* $A \in \mathbb{R}^{n\times n}$ *versteht man die kleinste natürliche Zahl* $m < n$*, so dass gilt*

$$a_{ik} = 0 \qquad \text{für alle } i \text{ und } k \text{ mit } |i-k| > m.$$

Bemerkung 3.7.2 *Die Bandbreite* m *gibt somit die Anzahl der Nebendiagonalen unterhalb bzw. oberhalb der Diagonalen an, welche die im Allgemeinen von Null verschiedenen Matrixelemente enthalten.*

Bemerkung 3.7.3 *In verallgemeinerter Form spricht man auch von unterer und oberer Bandbreite, welche sich auf suggestive Weise definieren.*

Satz 3.7.4 *Die untere Dreiecksmatrix* L *der Cholesky-Zerlegung* $A = LL^T$ *einer symmetrischen, positiv definiten Bandmatrix mit der Bandbreite* m *besitzt dieselbe Bandstruktur, denn es gilt* $l_{ik} = 0$ *für alle* i, k *mit* $i - k > m$*.* □

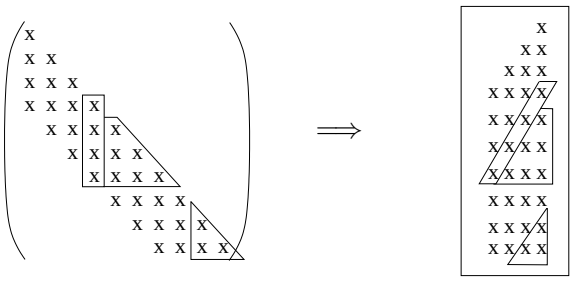


Abb. 3.3: Reduktion und Speicherung einer symmetrischen, positiv definiten Bandmatrix.

Analog zur Herleitung des Cholesky-Verfahrens auf Seite 46 erhalten wir die Rechenvorschrift des Cholesky-Verfahrens für Bandmatrizen, wobei nun die Koeffizienten zeilenweise bestimmt werden. Für $n = 5$ und $m = 2$ sieht die Situation wie folgt aus:

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ 0 & l_{42} & l_{43} & l_{44} & 0 \\ 0 & 0 & l_{53} & l_{54} & l_{55} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & 0 & 0 \\ 0 & l_{22} & l_{32} & l_{42} & 0 \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}^2 & & \text{sym.} & & \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & & & \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 & & \\ & l_{42}l_{22} & l_{42}l_{32} + l_{43}l_{33} & l_{42}^2 + l_{43}^2 + l_{44}^2 & \\ & & l_{53}l_{33} & l_{53}l_{43} + l_{54}l_{44} & l_{53}^2 + l_{54}^2 + l_{55}^2 \end{pmatrix}.$$

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$, wobei $A \in \mathbb{R}^{n\times n}$
mit Bandbreite $0 \le m < n$.
Zeilenweise berechnet man für $k = 1, 2, \ldots, n$

$$l_{ki} = \frac{1}{l_{ii}}\Big(a_{ki} - \sum_{j=\max\{1,k-m\}}^{i-1} l_{kj} \cdot l_{ij}\Big) \qquad (\max\{1, k-m\} \le i \le k-1)$$

$$l_{kk} = \Big(a_{kk} - \sum_{j=\max\{1,k-m\}}^{k-1} l_{kj}^2\Big)^{1/2}$$

Satz 3.7.5 *Es sei $A \in \mathbb{R}^{n\times n}$ eine positiv definite Bandmatrix mit Bandbreite $1 \le m \le n$. Dann beträgt der Aufwand zur Berechnung der Cholesky-Zerlegung von A*

$$\text{FLOP}(\textit{Cholesky-Zerl. von } A) = n(m^2+2m) - \frac{4m^3+9m^2+5m}{6} + n \text{ Wurzelberechnungen}\,.$$

Beweis. Wendet man die Cholesky-Zerlegung auf eine positiv definite Bandmatrix A an, so ist die resultierende untere Dreiecksmatrix L mit $A = LL^T$ wieder eine Bandmatrix mit der gleichen Bandbreite, wie A sie hat.

i) Gehen wir davon aus, dass die Matrix L zeilenweise berechnet wird und betrachten zuerst den Aufwand für die ersten m Zeilen. Hier kann man das Ergebnis aus Satz 3.6.9 verwenden. Es sind hierfür somit

$$m \text{ Wurzelberechnungen} + \frac{2m^3+3m^2-5m}{6}$$

Operationen notwendig.

ii) In den verbleibenden $n-m$ Zeilen, in denen jeweils $m+1$ Koeffizienten zu bestimmen sind, ergibt sich Folgendes:
Um den j-ten von Null verschiedenen Nebendiagonaleintrag von L in der k-ten Zeile $m+1 \le k \le n$ zu berechnen, sind jeweils $j-1$ Multiplikationen, $j-1$ Subtraktionen und eine Division notwendig. Zur Berechnung des Diagonalelements werden m Multiplikationen, m Subtraktionen und eine Wurzelberechnung benötigt. Bei m Nebendiagonalelementen sind dies in der Summe

$$\begin{aligned} & 1 \text{ Wurzelberechnung} + 2m + \sum_{j=1}^{m}(2j-1) = 1\sqrt{\cdot} + m + 2\sum_{j=1}^{m} j \\ = \quad & 1\sqrt{\cdot} + m + m(m+1) = 1\sqrt{\cdot} + m(m+2)\,. \end{aligned}$$

iii) Der gesamte Aufwand beträgt also

$$\begin{aligned} & n \text{ Wurzelberechnungen} + \frac{2m^3+3m^2-5m}{6} + (n-m)m(m+2) \\ = \quad & n \text{ Wurzelberechnungen} + n(m^2+2m) - \frac{4m^3+9m^2+5m}{6}\,. \end{aligned}$$

□

Bemerkung 3.7.6 *Ist für eine Klasse von positiv definiten Bandmatrizen aus $\mathbb{R}^{n\times n}$ für beliebiges n die Bandbreite beschränkt, so wächst der Aufwand zur Berechnung der Cholesky-Zerlegung asymptotisch nur **linear** in n.*

Aufgabe 3.7.7 Man leite mit Hilfe des Cholesky-Verfahrens für symmetrische Bandmatrizen eine Rekursionsformel für s.p.d. Tridiagonalmatrizen her und gebe den Aufwand an.

3.8 Fehlerabschätzung mittels Kondition

Wir wollen nun zwei wichtige Fragestellungen untersuchen, welche bei dem numerischen Lösen von linearen Gleichungssystemen (LGS) auftreten. Zum einen betrachten wir eine Näherungslösung $\widetilde{x}$ zu der Lösung x von $Ax = b$. Welche Rückschlüsse kann man von der Größe des **Residuums** $r := b - A\widetilde{x}$ auf den Fehler $e = x - \widetilde{x}$ ziehen? Des Weiteren rechnet man im Allgemeinen mit endlicher Genauigkeit auf einem Rechner. Welche Einflüsse haben dabei Störungen der Ausgangsdaten auf die Lösung x?

Zu einer Matrix $A \in \mathbb{R}^{n\times n}$ und einem Vektor $x \in \mathbb{R}^n$ sei $\|A\|$ eine beliebige Matrixnorm[4] und $\|x\|$ eine dazu verträgliche Vektornorm (d.h. $\|Ax\| \le \|A\|\,\|x\|$). Nach Definition des Fehlers e und Residuums r gilt

$$Ae = Ax - A\widetilde{x} = b - A\widetilde{x} = r.$$

Daraus folgt

$$\|e\| = \|A^{-1}r\| \le \|A^{-1}\|\,\|r\|.$$

Mit

$$\|b\| = \|Ax\| \le \|A\|\,\|x\|$$

[4] Vgl. hierzu Anhang B.

folgt für den relativen Fehler die Abschätzung

$$\frac{\|e\|}{\|x\|} \le \frac{\|A^{-1}\|\,\|r\|}{\|b\| \cdot \frac{1}{\|A\|}} = \|A^{-1}\|\,\|A\| \cdot \frac{\|r\|}{\|b\|}\,. \tag{3.23}$$

Dies motiviert den Begriff der Konditionszahl.

Definition 3.8.1 (Konditionszahl) *Man bezeichnet*

$$\kappa(A) = \|A\|\,\|A^{-1}\| \tag{3.24}$$

als die **Konditionszahl** *der Matrix A bezüglich der verwendeten Matrixnorm.*

Beispiel 3.8.2 Man betrachte das LGS $Ax = b$ mit

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix} \qquad b := (b_j), \qquad b_j := \sum_{k=1}^{5} \frac{1}{k+j-1} \qquad (1 \le j \le 5)\,.$$

Für die Matrixnormen und die Konditionszahlen erhält man in den Fällen $p = 1, 2, \infty$

$$\begin{array}{lcllcllcl} \|A\|_1 &=& 2.28\overline{3}\,, & \|A^{-1}\|_1 &=& 413280\,, & \kappa_1(A) &=& 943656\,, \\ \|A\|_2 &\approx& 1.5670507\,, & \|A^{-1}\|_2 &\approx& 304142.84\,, & \kappa_2(A) &\approx& 476607.25\,, \\ \|A\|_\infty &=& 2.28\overline{3}\,, & \|A^{-1}\|_\infty &=& 413280\,, & \kappa_\infty(A) &=& \kappa_1(A)\,. \end{array}$$

Sei die exakte Lösung $x = (1, \ldots, 1)^T$. Ist nun $\widetilde{x} = (1-\varepsilon)x$, dann gilt

$$\frac{\|e\|}{\|x\|} = \frac{\|\varepsilon x\|}{\|x\|} = |\varepsilon|$$

und

$$r = b - A\widetilde{x} = Ax - A\widetilde{x} = A(x - (1-\varepsilon)x) = \varepsilon Ax = \varepsilon b,$$

d.h.

$$\frac{\|r\|}{\|b\|} = |\varepsilon|\,.$$

Obwohl die Konditionzahl sehr groß ist, verhält sich bei dieser Störung $\frac{\|e\|}{\|x\|}$ wie $\frac{\|r\|}{\|b\|}$. Obige Abschätzung ist offensichtlich eine „worst case“ Abschätzug. Betrachten wir nun △

$$\widetilde{x} = (0.993826,\ 1.116692,\ 0.493836,\ 1.767191,\ 0.623754)^T. \tag{3.25}$$

A habe die Eigenwerte $0 \le \lambda_1 < \cdots < \lambda_5$ mit den zugehörigen Eigenvektoren $\varphi_1, \ldots, \varphi_5$, $\|\varphi_j\|_2 = 1$. Man beachte, dass $\widetilde{x}$ in (3.25) so gewählt ist, dass $\widetilde{x} - x \approx \varphi_1$ gilt und

$$x \approx -0.004\,\varphi_1 - 0.042\,\varphi_2 + 0.244\,\varphi_3 + 0.972\,\varphi_4 + 1.998\,\varphi_5\,,$$

d.h. x von φ_5 dominiert wird. Dann erhalten wir

$$\frac{\|e\|_1\|b\|_1}{\|x\|_1\|r\|_1} \approx 0.41\ \kappa_1(A), \quad \frac{\|e\|_2\|b\|_2}{\|x\|_2\|r\|_2} \approx 0.89\ \kappa_2(A), \quad \frac{\|e\|_\infty\|b\|_\infty}{\|x\|_\infty\|r\|_\infty} \approx 0.74\ \kappa_\infty(A)$$

und schätzen dann in der richtigen Größenordung ab!

Untersuchen wir nun, welchen Einfluss kleine Störungen in den Ausgangsdaten A, b auf die Lösung x des linearen Gleichungssystems haben können, d.h. wir sind interessiert an der **Empfindlichkeit** der Lösung x auf Störungen in den Koeffizienten. Die genaue Frage lautet:
Wie groß kann die Änderung δx der Lösung x von $Ax = b$ sein, falls die Matrix um δA und b durch δb gestört sind? Dabei seien δA und δb kleine Störungen, so dass $A + \delta A$ immer noch regulär ist. Es sei $x + \delta x$ die Lösung zu

$$(A + \delta A)(x + \delta x) = (b + \delta b).$$

Mit etwas Mathematik erhält man folgendes Resultat:

Satz 3.8.3 *Es sei $A \in \mathbb{R}^{n\times n}$ regulär und $x \in \mathbb{R}^n$ die exakte Lösung von $Ax = b$. Die rechte Seite b sei um δb gestört und für die Störung von A gelte $\kappa(A)\,\|\delta A\|/\|A\| = \|A^{-1}\|\,\|\delta A\| < 1$ und $A + \delta A$ ist immer noch regulär. Dann gilt für die Lösung $\widetilde{x}$ des gestörten Systems mit Koeffizientenmatrix $A + \delta A$ und rechter Seite $b + \delta b$*

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right). \tag{3.26}$$

Bemerkung 3.8.4 *Die Konditionszahl $\kappa(A)$ der Koeffizientenmatrix A ist folglich die entscheidende Größe, welche die Empfindlichkeit der Lösung bzgl. der Störungen δA und δb beschreibt.*

Es bleibt folgendes Lemma zu beweisen:

Lemma 3.8.5 (Neumann-Reihe) *Sei $A \in \mathbb{R}^{n\times n}$, $\|\cdot\|$ eine submultiplikative Matrixnorm und $\|A\| < 1$. Dann ist $(I - A)$ regulär und*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k \qquad \textit{(Neumann-Reihe)} \tag{3.27}$$

mit

$$\|(I - A)^{-1}\| \leq \frac{1}{1 - \|A\|}. \tag{3.28}$$

Beweis. Mit $\|A\| < 1$, der Submultiplikativität und der Summenformel für die geometrische Reihe erhält man

$$\sum_{k=0}^{m} \|A^k\| \leq \sum_{k=0}^{m} \|A\|^k \leq \sum_{k=0}^{\infty} \|A\|^k = \frac{1}{1 - \|A\|} < \infty \quad (m \in \mathbb{N}). \tag{3.29}$$

Der Raum $\mathbb{R}^{n\times n}$ ist isomorph zu $\mathbb{R}^{n^2}$ (siehe Anhang B.2). In [Analysis II, Beispiel 8.4.6] wurde gezeigt, dass $\mathbb{R}^{n^2}$ vollständig ist bezüglich irgendeiner Norm auf $\mathbb{R}^{n^2}$. Somit ist $\mathbb{R}^{n\times n}$ vollständig bezüglich $\|\cdot\|$ und aus der absoluten Konvergenz folgt die Konvergenz von $\sum_{k=0}^{\infty} A^k$. Ebenso folgt aus $\|A^k\| \leq \|A\|^k \to 0$ für $k \to \infty$ die Konvergenz von $\lim_{k\to\infty} A^k = 0$. Weiter gilt die Gleichung („Teleskopsumme“)

$$\left(\sum_{k=0}^{m} A^k \right) (I - A) = I - A^{m+1} \quad (m \in \mathbb{N}). \tag{3.30}$$

Der Grenzübergang von (3.30) führt zur Gleichung

$$\left(\sum_{k=0}^{\infty} A^k \right) (I - A) = I. \tag{3.31}$$

Das bedeutet, $I - A$ ist regulär und $(I - A)^{-1} = \sum_{k=0}^{\infty} A^k$. Mit der Summenformel für die geometrische Reihe erhält man schließlich

$$\|(I - A)^{-1}\| \leq \lim_{N\to\infty} \sum_{k=0}^{N} \|A^k\| \leq \lim_{N\to\infty} \sum_{k=0}^{N} \|A\|^k = \frac{1}{1 - \|A\|}, \tag{3.32}$$

womit der Satz bewiesen ist. □

Zum Ende des Kapitels wollen wir nun den praktischen Nutzen der Abschätzung (3.26) in einer Faustregel festhalten.

Bei einer d-stelligen dezimalen Gleitkommarechnung können die relativen Fehler der Ausgangsgrößen für beliebige, kompatible Normen von der Größenordnung

$$\frac{\|\delta A\|}{\|A\|} \approx 5 \cdot 10^{-d} \qquad \frac{\|\delta b\|}{\|b\|} \approx 5 \cdot 10^{-d}$$

sein. Ist die Konditionszahl $\kappa(A) \approx 10^{\alpha}$ mit $5 \cdot 10^{\alpha-d} \ll 1$, so ergibt die Abschätzung (3.26)

$$\frac{\|\delta x\|}{\|x\|} \leq 10^{\alpha-d+1}.$$

Das heißt, dass $\|\delta x\|$ maximal in der Größenordnung der $(d-\alpha-1)$-ten Dezimalstelle von $\|x\|$ liegen kann und dies motiviert folgende Daumenregel:

Bemerkung 3.8.6 (Daumenregel zur Genauigkeit) *Wird $Ax = b$ mit d-stelliger dezimaler Gleitkommarechnung gelöst, und beträgt die Konditionszahl $\kappa(A) \approx 10^{\alpha}$, so sind, bezogen auf die betragsgrößte Komponente, nur $(d-\alpha-1)$ Dezimalstellen sicher.*

MATLAB-Beispiel:

Es ist bekannt, dass die Gauß-Elemination selbst mit Spaltenpivotstrategie zu überraschend ungenauen Ergebnissen beim Lösen von linearen Gleichungssystemen führen kann, obwohl die Matrix gut konditioniert ist.

Betrachten wir hierzu die von Wilkinson angegebene pathologische Matrix

$$A = \begin{pmatrix} 1 & & & 1 \\ -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ -1 & \cdots & -1 & 1 \end{pmatrix}.$$

```
>> A=toeplitz([1,-ones(1,59)], ...
              [1,zeros(1,59)]);
>> A(:,60)=1;
>> cond(A)
ans =
   26.8035  % rel. gut konditioniert
>> randn('state', 3383)
>> x=randn(60,1);
>> b=A*x;
>> x1=A\b;
>> norm(x-x1)/norm(x)
ans =
    0.3402  % großer rel. Fehler
```

Bemerkung 3.8.7 *Das Beispiel lässt vermuten, dass das Gauß-Verfahren über die ganze Menge der invertierbaren Matrizen betrachtet nicht stabil ist. Für die in der Praxis auftretenden Matrizen, ist das Gauß-Verfahren mit Spaltenpivotierung jedoch „in der Regel" stabil. Für eine weitere Stabilitätsanalyse des Gauß-Verfahrens sei auf [Deuflhard/Hohmann], die grundlegenden Artikel von Wilkinson [Wilkinson65, Wilkinson69] sowie auf die Übersichtsartikel [Higham, Discroll/Maki] verwiesen.*

4 LINEARE AUSGLEICHSPROBLEME

Beispiel 4.0.1 Bestimmung eines unbekannten Widerstands x aus Messungen für die Stromstärke t und die Spannung b. Angenommen, es liegen m Messungen (b_i, t_i), $i = 1, \ldots, m$, mit $m \gg 1$,

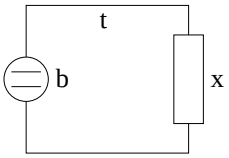


Abb. 4.1: Widerstandsbestimmung aus Messungen für Spannung und Stromstärke.

für Spannung und Stromstärke vor. Das **Ohm'sche Gesetz** aus der Physik besagt: $b = tx$, also

$$b_i = t_i x \,, \quad i = 1, \ldots, m. \tag{4.1}$$

Da die Messdaten in der Regel (Mess-)Fehler beinhalten, kann man (4.1) nicht exakt für alle i erfüllen. Gesucht ist jetzt also ein Widerstand x, der „möglichst gut" zu den Messdaten passt.

Motivation: Gesucht ist eine Gerade $g(x) = \alpha x + b$, deren y-Werte den kleinsten Quadratsummenabstand zu den vorgegebenen Daten $(x_i, f(x_i))$, $i = 1, \ldots, n$ haben, die sogenannte **Ausgleichsgerade**. Die geometrische Veranschaulichung dazu ist in der nachfolgenden Abbildung dargestellt.

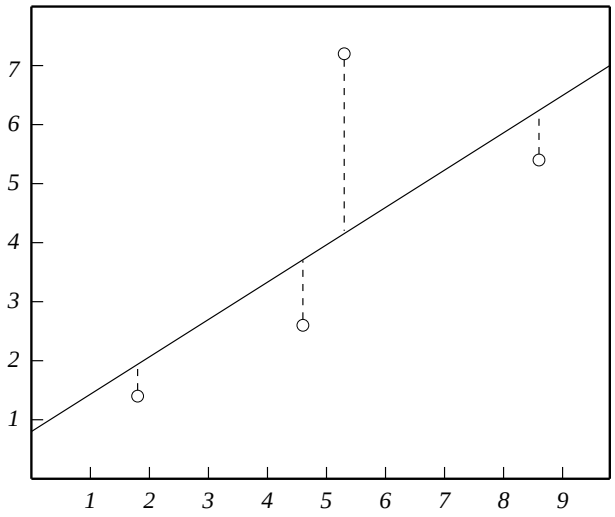


Abb. 4.2: Die Ausgleichsgerade, die die Quadratsumme der Abstände minimiert

4.1 Die Methode der kleinsten Quadrate

Gemäß obiger Motivation gilt es zu gegebenen Punktepaaren $(x_1, f_1), (x_2, f_2), \ldots, (x_n, f_n)$ die in der Geradengleichung $g(x) = mx + b$ freien Parameter m und b so zu bestimmen, dass gilt

$$F(m, b) := \sum_{i=1}^{n} \big(g(x_i) - f_i\big)^2 \to \min,$$

oder in alternativer Formulierung

$$\min_{(b,m)\in\mathbb{R}^2} \left\| A\begin{pmatrix} b \\ m \end{pmatrix} - d \right\|_2^2,$$

wobei wir nachfolgende Definitionen verwenden

$$A = \begin{pmatrix} 1\ x_1 \\ \vdots\ \ \vdots \\ 1\ x_n \end{pmatrix}, \quad d = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}.$$

Hierzu bilden wir alle partiellen Ableitungen bzgl. m und b, woraus sich dann die kritischen Punkte der Funktion $F(m,b)$ wie folgt berechnen

$$\nabla F(m,b) = \Big(\frac{\partial}{\partial b}\, F, \frac{\partial}{\partial m}\, F\Big) = \overrightarrow{0}.$$

Im vorliegenden Fall erhalten wir damit

$$\begin{aligned} \frac{\partial}{\partial b}\, F(m,b) &= 2\sum_{i=1}^{n}(mx_i + b - f_i)\cdot 1 \stackrel{!}{=} 0, \\ \frac{\partial}{\partial m}\, F(m,b) &= 2\sum_{i=1}^{n}(mx_i + b - f_i)\cdot x_i \stackrel{!}{=} 0. \end{aligned}$$

Diese beiden Bestimmungsgleichungen können auch wie folgt geschrieben werden

$$\begin{aligned} 1^T\Big(A\begin{pmatrix} b \\ m \end{pmatrix} - d\Big) &= 0, \\ x^T\Big(A\begin{pmatrix} b \\ m \end{pmatrix} - d\Big) &= 0, \end{aligned}$$

wobei $1 = (1,\ldots,1)^T \in \mathbb{R}^n$ und $x = (x_1,\ldots,x_n) \in \mathbb{R}^n$ seien.
Es gilt also

$$\begin{aligned} \nabla F(m,b) = 0 \quad \Rightarrow \quad & 1^T\Big(A\begin{pmatrix} b \\ m \end{pmatrix} - d\Big) = 0, \\ & x^T\Big(A\begin{pmatrix} b \\ m \end{pmatrix} - d\Big) = 0. \end{aligned}$$

Zusammengefasst folgt für potentielle Minimalstellen

$$\begin{pmatrix} 1\ x_1 \\ \vdots\ \ \vdots \\ 1\ x_n \end{pmatrix}^T \Big(A\begin{pmatrix} b \\ m \end{pmatrix} - d\Big) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Leftrightarrow \quad A^TA\begin{pmatrix} b \\ m \end{pmatrix} = A^Td$$

Bemerkung 4.1.1 *Falls in einem linearen Gleichungssystem die Anzahl der Gleichungen die der Unbekannten übersteigen sollte (wie es im vorliegenden Fall bei $Ax = d$ ist), so nennen wir das System* **überbestimmt** *und können es in der Regel nicht exakt lösen. Deshalb betrachten wir sinnvollerweise das Ausgleichsproblem der* **Gaußschen Normalengleichung**

$$A^TAx = A^Td \tag{4.2}$$

und erhalten dafür genau dann eine eindeutige Lösung, falls A vollen Rang besitzt.

Bemerkung 4.1.2 *Als einführendes Beispiel haben wir uns auf die Betrachtung von linear unabhängigen Basisvektoren des $\mathbb{P}_1$ in der Darstellung von g beschränkt, d.h. $1, x$. Als Nächstes werden wir zu allgemeineren Funktionen in der Funktionsvorschrift von g übergehen, wie etwa*

$$g(x) = a\exp(x) + b\exp(-x) + c\log(x).$$

Damit erhalten wir folgendes Minimierungsproblem bei bekannten Daten (x_i, f_i) $(i = 1, \dots, n)$

$$\sum_{i=1}^{n} \big(g(x_i) - f_i\big)^2 \to \min,$$

welches wir völlig analog zu obigem Verfahren als Normalengleichung in der Form

$$A^T A \begin{pmatrix} a \\ b \\ c \end{pmatrix} = A^T d, \text{ mit } A = \begin{pmatrix} e^{x_1} & e^{-x_1} & \log x_1 \\ \vdots & \vdots & \vdots \\ e^{x_n} & e^{-x_n} & \log x_n \end{pmatrix} \text{ und } d = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}$$

interpretieren können. Je komplexer also die Funktionsvorschrift von g, desto aufwendiger wird auch das Lösen der Gaußschen Normalengleichung (4.2)*. Im Folgenden werden wir ein Verfahren bereitstellen, das eine numerisch stabile und effiziente Lösung des Ausgleichsproblems garantiert.*

4.2 Die QR-Zerlegung

Definition 4.2.1 *Eine quadratische Matrix $Q \in \mathbb{R}^{n\times n}$ heißt **orthogonal**, falls gilt:*

$$QQ^T = Q^T Q = I.$$

Weiterhin bezeichne $\|x\|$ die vom **Euklidschen** Skalarprodukt im $\mathbb{R}^n$ induzierte Norm, vgl. Anhang B. Darüber hinaus halten wir noch fest, dass orthogonale Abbildungen stets längen- und winkeltreu sind, dass also gilt

$$\|Qx\|^2 = (Qx)^T Qx = x^T Q^T Qx = x^T x = \|x\|^2, \text{ d.h. } \|Qx\| = \|x\|.$$

Motivation: Zur Lösung des Minimierungsproblems $\min_{x\in\mathbb{R}^n} \|Ax - b\|^2$ setzen wir voraus, dass zu $A \in \mathbb{R}^{m\times n}$ (rg $A = n < m$) stets eine Zerlegung $A = QR$ existiert, wobei $Q \in \mathbb{R}^{m\times m}$ orthogonal ist und $R \in \mathbb{R}^{m\times n}$ folgende Gestalt besitzt

$$R = \begin{pmatrix} * & * & * \\ & * & * \\ & & * \\ 0 & & \end{pmatrix} = \left(\begin{array}{ccc} & \widehat{R} & \\ \hline & 0 & \end{array}\right),$$

$\widehat{R}$ stellt dabei eine reguläre obere Dreiecksmatrix dar. Damit können wir unser bisheriges Minimierungsproblem wie folgt modifizieren (beachte die Faktorisierung $A = QR$):

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|^2 = \min_{x\in\mathbb{R}^n} \|Q^T(Ax - b)\|^2 = \min_{x\in\mathbb{R}^n} \|Rx - Q^T b\|^2.$$

Hierbei weisen wir vor allen Dingen nochmals auf die Gestalt von R hin:

$$Rx = \left(\begin{array}{ccc} * & * & * \\ & * & * \\ & & * \\ \hline 0 & & \end{array}\right) \cdot x = \left(\begin{array}{c} b_1 \\ \hline b_2 \end{array}\right) = Q^T b$$

Dieser Faktorisierungsansatz führt uns nun zu folgenden Fragen:

- Zu welchen Matrizen $A \in \mathbb{R}^{m\times n}$ existiert eine derartige **QR**-Zerlegung?
- Wie bestimmen wir eine solche Zerlegung möglichst effizient?

Satz 4.2.2 *Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ mit Maximalrang $n < m$ existiert eine orthogonale Matrix $Q \in \mathbb{R}^{m\times m}$ derart, dass*

$$A = QR \text{ mit } R = \begin{pmatrix} \widehat{R} \\ 0 \end{pmatrix},\ \widehat{R} \in \mathbb{R}^{n\times n}$$

gilt, wobei $\widehat{R}$ eine reguläre obere Dreiecksmatrix darstellt.

Beweis. Der Kern des Beweises liegt in der Verwendung von orthogonalen Drehmatrizen der Form

$$\mathcal{U}(p,q;\varphi) := \begin{pmatrix} 1 & & & & & & & & 0 \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\varphi & & \sin\varphi & & & \\ & & & & \ddots & & & & \\ & & & -\sin\varphi & & \cos\varphi & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ 0 & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \xleftarrow{p-te\ Zeile} \\ \\ \xleftarrow{q-te\ Zeile} \\ \\ \\ \end{matrix} \tag{4.3}$$

(mit $p-te\ Spalte$ und $q-te\ Spalte$ über den entsprechenden Spalten)

Die Orthogonalität von $\mathcal{U}(p,q;\varphi)$ ergibt sich unmittelbar, denn es gilt $\mathcal{U}^T(p,q;\varphi)\cdot\mathcal{U}(p,q;\varphi) = I$. Das Produkt $\widetilde{A} := \mathcal{U}^T(p,q;\varphi)\cdot A$ liefert dagegen für $j = 1,\ldots,n$

$$\begin{pmatrix} 1 & & & & & & & & 0 \\ & \ddots & & & & & & & \\ & & \cos\varphi & & & -\sin\varphi & & & \\ & & & & \ddots & & & & \\ & & \sin\varphi & & & \cos\varphi & & & \\ & & & & & & \ddots & & \\ 0 & & & & & & & & 1 \end{pmatrix} \cdot A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{p-1,1} & \cdots & a_{p-1,n} \\ a_{p1}\cos\varphi - a_{q1}\sin\varphi & \cdots & a_{pn}\cos\varphi - a_{qn}\sin\varphi \\ a_{p+1,1} & \cdots & a_{p+1,n} \\ \vdots & & \vdots \\ a_{q-1,1} & \cdots & a_{q-1,n} \\ a_{p1}\sin\varphi + a_{q1}\cos\varphi & \cdots & a_{pn}\sin\varphi + a_{qn}\cos\varphi \\ a_{q+1,1} & \cdots & a_{q+1,n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

(mit $p-te\ Spalte$ und $q-te\ Spalte$ über den entsprechenden Spalten)

Demzufolge werden die Zeilen p und q von A durch Linearkombinationen der cos- und sin-Terme ersetzt.
Die sukzessive Multiplikation der Matrix A von links mit Rotationsmatrizen $\mathcal{U}^T(p,q;\varphi)$ (φ dabei so gewählt, dass $\widetilde{a}_{q,p} = 0$) mit den Rotationsindexpaaren

$$(1,2),(1,3),\ldots,(1,m),(2,3),(2,4),\ldots,(2,m),(3,4),\ldots,(n,m)$$

eliminiert dann die Matrixelemente $a_{2,1}, a_{3,1}, \ldots, a_{m,n}$. Sukzessive werden dabei unterhalb des Diagonalelements Nullen erzeugt. Nach $k := \frac{1}{2}n(2m-n-1)$ Transformationsschritten gilt dann $A = QR$ mit

$$\mathcal{U}_k^T \cdot \ldots \cdot \mathcal{U}_2^T \mathcal{U}_1^T A = Q^T A = R \quad \text{oder} \quad A = QR.$$

Da die orthogonale Matrix Q regulär ist, ist der Rang von A und der Rang von R gleich n und folglich ist die Rechtecksdreiecksmatrix $\widehat{R}$ regulär, da A nach Voraussetzung Maximalrang hat. □

Der soeben erbrachte Existenzbeweis ist rein konstruktiv und liefert damit eine erste Möglichkeit eine Zerlegung QR zu bestimmen. Wir unterscheiden die folgenden beiden Vorgehensweisen:

- **Givens**[1]-Rotation,
- **Householder**[2]-Spiegelung.

4.3 Givens-Rotationen

Es bezeichne im Folgenden stets $c := \cos\varphi$ und $s := \sin\varphi$. Die Grundidee der **Givens**-Rotation besteht darin, durch die Multiplikation mit einer geeigneten orthogonalen (Dreh-)Matrix einen Vektor $x \in \mathbb{R}^n$ so zu drehen, dass möglichst viele seiner Komponenten verschwinden.
Im Falle $(a, b) \in \mathbb{R}^2$ werden wir also $c, s \in \mathbb{R}$ so bestimmen, dass gilt

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}.$$

Setzen wir zuerst $a,\ b \neq 0$ voraus. Wir erhalten damit folgende Gleichung

$$as = bc \tag{4.4}$$

und mit $s^2 + c^2 = 1$ gilt

$$r^2 = (ac+bs)^2 = a^2c^2 + abcs + abcs + b^2s^2 \overset{(4.4)}{=} a^2c^2 + a^2s^2 + b^2c^2 + b^2s^2 = a^2 + b^2\ .$$

Weiter ergibt sich für $a \neq 0$

$$\begin{aligned} ac + bs = r \Rightarrow c = \frac{1}{a}(r - bs) \quad &\Rightarrow \quad as - \frac{b}{a}(r - bs) = 0 \\ &\Leftrightarrow \quad a^2 s - br + b^2 s = 0 \\ &\Leftrightarrow \quad br = (a^2 + b^2)s \\ &\Leftrightarrow \quad s = \frac{br}{a^2+b^2} \overset{(r^2 = a^2+b^2)}{=} \frac{b}{r}. \end{aligned}$$

Und damit erhalten wir dann auch für $b \neq 0$

$$bc = as \ \Rightarrow\ c = \frac{a}{b}s = -\frac{a}{r}.$$

Offen ist noch die Wahl des Vorzeichens des Vorzeichens $r = \pm\sqrt{a^2+b^2}$. Setzen wir $r := \operatorname{sign}(a)\sqrt{a^2+b^2}$, so folgt $c > 0$, falls $a \neq 0$, und für $a = 0$ setzen wir $c = 0$ und $s = -1$. Somit sind auch s und c für die anderen Spezialfälle $b = 0$ und $a = b = 0$ wohldefiniert. Damit

[1] Givens, W.
[2] Householder, A.

gilt (obwohl der Winkel bisher gar nicht explizit genannt wurde) $\varphi \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ und somit lässt sich $\cos\varphi$ eindeutig aus $\sin\varphi$ mit $\cos\varphi = \sqrt{1-\sin^2\varphi}$ bestimmen.
Seien p und q zwei Zeilenindizes von A. Zur Eliminierung des Matrixelements $a_{q,j}$ gehen wir daher wie folgt vor:

$$\begin{aligned} \text{Falls} \quad a_{p,j} \neq 0: \quad \cos\varphi &= \frac{|a_{p,j}|}{\sqrt{a_{p,j}^2 + a_{q,j}^2}}, \\ \sin\varphi &= \frac{-\operatorname{sign}(a_{p,j})\, a_{q,j}}{\sqrt{a_{p,j}^2 + a_{q,j}^2}}, \\ a_{p,j} = 0: \quad \cos\varphi &= 0, \ \ \sin\varphi = 1. \end{aligned}$$

Da sich $\cos\varphi$ aus $\sin\varphi$ bestimmen lässt, ermöglicht das Verfahren eine effiziente Speicherung, denn es genügt $\sin\varphi$ an den entsprechenden freiwerdenden Stellen in A abzulegen:

$$A = \begin{pmatrix} a_{1,1}^{(0)} & \cdots & a_{1,n}^{(0)} \\ \vdots & & \vdots \\ a_{m,1}^{(0)} & \cdots & a_{m,n}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{1,1}^{(1)} & \cdots & \cdots & a_{1,n}^{(1)} \\ \sin\varphi_{2,1} & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ a_{3,1}^{(0)} & a_{3,2}^{(0)} & \cdots & a_{3,n}^{(0)} \\ a_{4,1}^{(0)} & a_{4,2}^{(0)} & \cdots & a_{4,n}^{(0)} \\ \vdots & & & \vdots \\ a_{m,1}^{(0)} & \cdots & \cdots & a_{m,n}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{1,1}^{(2)} & \cdots & \cdots & a_{1,n}^{(2)} \\ \sin\varphi_{2,1} & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \sin\varphi_{3,1} & a_{3,2}^{(2)} & \cdots & a_{3,n}^{(2)} \\ a_{4,1}^{(0)} & a_{4,2}^{(0)} & \cdots & a_{4,n}^{(0)} \\ \vdots & & & \vdots \\ a_{m,1}^{(0)} & \cdots & \cdots & a_{m,n}^{(0)} \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} a_{1,1}^{(3)} & \cdots & \cdots & a_{1,n}^{(3)} \\ \sin\varphi_{2,1} & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \sin\varphi_{3,1} & a_{3,2}^{(2)} & \cdots & a_{3,n}^{(2)} \\ \sin\varphi_{4,1} & a_{4,2}^{(3)} & \cdots & a_{4,n}^{(3)} \\ \vdots & & & \vdots \\ a_{m,1}^{(0)} & \cdots & \cdots & a_{m,n}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{1,1}^{(m-1)} & \cdots & \cdots & a_{1,n}^{(m-1)} \\ \sin\varphi_{2,1} & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \sin\varphi_{3,1} & a_{3,2}^{(2)} & \cdots & a_{3,n}^{(2)} \\ \sin\varphi_{4,1} & a_{4,2}^{(3)} & \cdots & a_{4,n}^{(3)} \\ \vdots & & & \vdots \\ \sin\varphi_{m,1} & a_{m,2}^{(m-1)} & \cdots & a_{m,n}^{(m-1)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ddots & & \widehat{R} \\ & \ddots & \\ \text{„}Q\text{“} & & \ddots \end{pmatrix}.$$

Die QR**–Zerlegung** liefert also QR in kompakter Form, gespeichert in A.
Der Aufwand dieser Vorgehensweise liegt bei ungefähr $\mathcal{O}(n^3)$ Operationen.

Bemerkung 4.3.1 *Die Multiplikation eines gegebenen Vektors $x \in \mathbb{R}^n$ mit der Drehmatrix $\mathcal{U}(p,q;\varphi)$ ist äquivalent zur Drehung von x um einen Winkel φ entgegen dem Uhrzeigersinn in der Koordinatenebene. Vergleiche hierzu die nachfolgende Abbildung, die diesen geometrischen Sachverhalt verdeutlicht:*

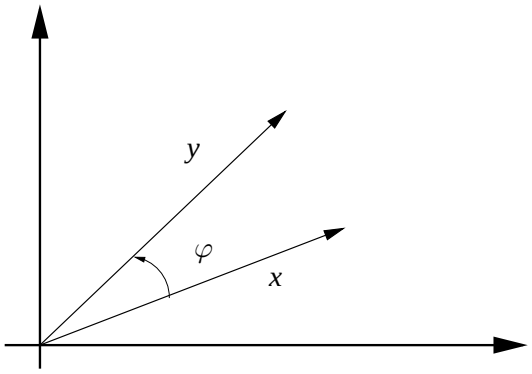


Abb. 4.3: Drehung um einen Winkel φ in der Ebene

MATLAB-Funktion: GivensRotMat.m

```
function rot = GivensRotMat(ap,aq)
if ap == 0
  s = 1; c = 0;
else
  dist2 = sqrt(ap^2+aq^2);
  c = abs(ap)/dist2;
  s = -sign(ap)*aq/dist2;
end
rot = [c,s;-s,c];
```

MATLAB-Funktion: Givens.m

```
function [Q,A] = Givens(A)
Q = eye(size(A,1));
for j=1:size(A,2)
  for i=j+1:size(A,1)
    rot = GivensRotMat(A(j,j),A(i,j));
    A([j,i],j:end) = rot' * A([j,i],j:end);
    Q(:,[j,i]) = Q(:,[j,i]) * rot;
  end
  A(j+1:end,j) = 0;
end
```

MATLAB-Beispiel:

Alternativ zu der Matlab-Funktion `qr` können wir auch die Routine `Givens` verwenden, die eine QR-Zerlegung, wie oben diskutiert, berechnet.

```
>> A = [1,2;3,4;5,6];
> [Q,R] = Givens(A);
>> Q*R
ans =
    1.0000    2.0000
    3.0000    4.0000
    5.0000    6.0000
```

4.4 Householder–Spiegelungen

Definition 4.4.1 *Sei $\omega \in \mathbb{R}^n$. Die $n \times n$- Matrix*

$$P := I - 2\frac{\omega\omega^T}{\omega^T\omega} \tag{4.5}$$

*wird als **Householder-Transformation** und ω als **Householder-Vektor** bezeichnet.*

Satz 4.4.2 *Eine **Householder**-Transformation $P = I - 2(\omega\omega^T)/(\omega^T\omega)$ ist symmetrisch und orthogonal, d.h. $P^{-1} = P^T$.*

Beweis. Da $\omega\omega^T$ symmetrisch ist $((\omega\omega^T)^T = \omega\omega^T)$, folgt

$$P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)^T = I - 2\frac{\omega\omega^T}{\omega^T\omega} = P\,.$$

Des Weiteren ergibt sich

$$P\,P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)\left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right) = I - 2\frac{\omega\omega^T}{\omega^T\omega} - 2\frac{\omega\omega^T}{\omega^T\omega} + 4\frac{\omega\omega^T\omega\omega^T}{(\omega^T\omega)^2}$$

und mit $\omega\omega^T\omega\omega^T = (\omega^T\omega)\,(\omega\omega^T)$ somit die Behauptung. □

Betrachten wir nun folgende Frage: Sei $x \in \mathbb{R}^n$. Wie müsste ein $\omega \in \mathbb{R}^n$ aussehen, so dass $Px = \alpha e_1$ gelte ($\alpha \in \mathbb{R}$, e_1 erster kanonischer Einheitsvektor)?
Mit $Px = x - 2\frac{\omega\omega^T}{\omega^T\omega}x = x - \lambda\omega \stackrel{!}{=} \alpha e_1$ folgt $\omega \in \operatorname{span}\{x - \alpha e_1\}$.
Wir setzen nun $\omega = x - \alpha e_1$ in $Px = \alpha e_1$ ein und erhalten

$$Px = x - 2\frac{\omega^T x}{\omega^T\omega}\,\omega = \left(1 - \frac{2(x-\alpha e_1)^T x}{\|x - \alpha e_1\|^2}\right)x + \alpha\,\frac{2(x-\alpha e_1)^T x}{\|x-\alpha e_1\|^2}e_1 = \alpha e_1\,.$$

Damit in der letzten Gleichung der Faktor vor x verschwindet, muss

$$1 = \frac{2(x-\alpha e_1)^T x}{\|x-\alpha e_1\|^2} \Leftrightarrow (x-\alpha e_1)^T(x-\alpha e_1) = 2x^Tx - 2\alpha x_1 \Leftrightarrow \alpha = \pm\sqrt{x^Tx}$$

gelten.
Wie ist nun das Vorzeichen zu wählen, $\alpha = \pm\sqrt{x^Tx}$? Die Wahl $\alpha = \|x\|_2$ hat die schöne Eigenschaft, dass Px ein positives Vielfaches von e_1 ist. Aber das Rezept ist gefährlich, wenn x annähernd ein positives Vielfaches von e_1 ist, da Auslöschungen auftreten können. Berechnet man ω_1 mittels $\omega_1 = x_1 - \|x\|_2$ treten für $x_1 \leq 0$ keine Auslöschungen auf und mit der Umformung

$$\omega_1 = x_1 - \|x\|_2 = \frac{x_1^2 - \|x\|_2^2}{x_1 + \|x\|_2} = \frac{-(x_2^2 + \ldots + x_n^2)}{x_1 + \|x\|_2}$$

treten auch für $x_1 > 0$ keine Auslöschungen auf. Man beachte außerdem, dass $(\omega_2, \dots \omega_n) = (x_2, \dots, x_n)$.
Normiert man den Vektor ω so, dass $\omega_1 = 1$ gilt, d.h.

$$\omega := \frac{x - \alpha e_1}{x_1 - \alpha} \text{ mit } \alpha^2 = \|x\|^2 \quad (\omega^T e_1 = 1),$$

dann folgt

$$\omega^T\omega = \frac{(x-\alpha e_1)^T(x-\alpha e_1)}{(x_1-\alpha)^2} = \frac{\|x\|^2 - 2\alpha x^T e_1 + \alpha^2}{(x_1-\alpha)^2} = \frac{2\alpha(\alpha - x_1)}{(x_1-\alpha)^2} = \frac{2\alpha}{\alpha - x_1}\,.$$

Des Weiteren bleibt festzuhalten, dass bei der Speicherung der Matrix $P = I - 2(\omega\omega^T)/(\omega^T\omega)$, $P \in \mathbb{R}^{n\times n}$, auch nur der Vektor $(\omega_2, \dots, \omega_n)^T$ zu berücksichtigen ist, der sich auf den $n-1$ freigewordenen Einträgen speichern lässt.

Algorithmus 4.4.1: Berechnung Householder-Vektor:

Zu gegebenem $x \in \mathbb{R}^n$ berechnet die Funktion ein $\omega \in \mathbb{R}^n$ mit $\omega(1) = 1$ und $\beta \in \mathbb{R}$, so dass $P = I_n - \beta\omega\omega^T$ orthogonal ist und $Px = \|x\|_2 e_1$.

function: $[\omega, \beta] = housevector(x)$
$n = length(x)$
$\sigma = x(2:n)^T\, x(2:n)$
$\omega = \begin{bmatrix} 1 \\ x(2:n) \end{bmatrix}$
`if` $\sigma = 0$
  $\beta = 0$
`else`
  $\mu = \sqrt{x(1)^2 + \sigma}$
  `if` $x(1) \le 0$
    $\omega(1) = x(1) - \mu$
  `else`
    $\omega(1) = -\sigma/(x(1) + \mu)$
  `end`
  $\beta = 2\,\omega(1)^2/(\sigma + \omega(1)^2)$
  $\omega = \omega/\omega(1)$
`end`

MATLAB-Funktion: HouseholderVektor.m

```
function [v,beta] = HouseholderVector(x)
n = length(x);
if n>1
  sigma = x(2:end)'*x(2:end);
  if sigma==0
    beta = 0;
  else
    mu = sqrt(x(1)^2+sigma);
    if x(1)<=0
      tmp = x(1) - mu;
    else
      tmp = -sigma / (x(1) + mu);
    end
    beta = 2*tmp^2/(sigma + tmp^2);
    x(2:end) = x(2:end)/tmp;
  end
  v = [1;x(2:end)];
else
  beta = 0;
  v = 1;
end
```

Wir haben gerade konstruktiv gezeigt, dass sich mit Hilfe der **Householder**-Transformation eine Matrix $A \in \mathbb{R}^{m\times n}$ in eine Matrix der folgenden Form transformieren können:

$$H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}.$$

Gehen wir nun davon aus, dass wir nach einigen Schritten die Ausgangsmatrix A auf folgende Gestalt gebracht haben:

$$H_2 H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \end{pmatrix}.$$

Im nächsten Schritt soll nun die nächste Subspalte unterhalb der Diagonalen eliminiert werden. Es ist also eine Householder-Matrix zu bestimmen, so dass

$$\widetilde{H}_3 \begin{pmatrix} \boxplus \\ \boxplus \\ \boxplus \\ \boxplus \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

gilt. Definiert man H_3 als Blockdiagonalmatrix

$$H_3 := \begin{pmatrix} I & 0 \\ 0 & \widetilde{H}_3 \end{pmatrix}$$

so erhalten wir

$$H_3H_2H_1A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}.$$

Mit dieser Vorgehensweise erhalten wir folgende Faktorisierung:

$$R = H_{n-1}H_{n-2}\cdot\ldots\cdot H_1A \;\Leftrightarrow\; A = (H_1\cdot\ldots\cdot H_{n-1})R \rightsquigarrow Q = H_1\cdot\ldots\cdot H_{n-1}.$$

Stellen wir nun dar, wie A überschrieben wird. Es sei

$$\omega^{(j)} = (\underbrace{0,\ldots,0}_{j-1},1,\omega_{j+1}^{(j)},\ldots,\omega_m^{(j)})^T$$

der j-te Householder-Vektor, dann erhält man nach Durchführung der QR-Zerlegung

$$A = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ \omega_2^{(1)} & r_{22} & r_{23} & r_{24} & r_{25} \\ \omega_3^{(1)} & \omega_3^{(2)} & r_{33} & r_{34} & r_{35} \\ \omega_4^{(1)} & \omega_4^{(2)} & \omega_4^{(3)} & r_{44} & r_{45} \\ \omega_5^{(1)} & \omega_5^{(2)} & \omega_5^{(3)} & \omega_5^{(4)} & r_{55} \\ \omega_6^{(1)} & \omega_6^{(2)} & \omega_6^{(3)} & \omega_6^{(4)} & \omega_6^{(5)} \end{pmatrix}.$$

Algorithmus 4.4.2: Berechnung Householder-QR:

Gegeben sei $A \in \mathbb{R}^{m\times n}$ mit $m \geq n$. Der folgende Algorithmus bestimmt die Householder-Matrizen $H_1,\ldots,H_n$, so dass mit $Q = H_1\cdots H_n$ die Matrix $R = Q^TA$ eine obere Dreiecksmatrix ist.
Der obere Dreieckteil von A wird mit R überschrieben und unterhalb der Diagonalen werden die nichttrivialen Komponenten der Householder-Vektoren gespeichert.

function: $A = housematrix(A)$
for $j = 1:n$
  $[\omega,\beta] =$ housevector$(A(j:m,j))$
  $A(j:m,j:n) = (I_{m-j+1} - \beta\omega\omega^T)A(j:m,j:n)$
  if $j < m$
    $A(j+1:m,j) = \omega(2:m-j+1)$
  end
end

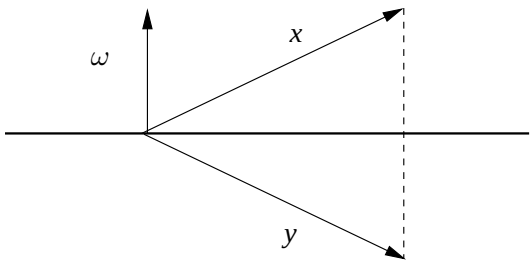


Abb. 4.4: Spiegelung an der zu ω orthogonalen Ebene

Bemerkung 4.4.3 *Analog zur **Givens**–Rotation führen wir auch an dieser Stelle kurz die geometrische Anschauung der **Householder**–Spiegelung an. Für einen gegebenen Vektor $x \in \mathbb{R}^n$ ist der Vektor $y = Px$ die Spiegelung von x an der Ebene $span\{\omega\}^\perp$. Hierzu nachfolgende Abbildung:*

Aufgabe 4.4.4 Man zeige, dass die QR-Zerlegung mittels Givens-Rotation $\approx 50\%$ mehr Rechenoperationen benötigt als mittels Householder-Transformation.

MATLAB-Funktion: Householder.m

```
function [A,R] = Householder(A)
for j = 1:size(A,2)
  [v,beta(j)] = HouseholderVector(A(j:end,j));
  A(j:end,j:end) = A(j:end,j:end)- v * (beta(j) * v'*A(j:end,j:end)
     );
  if j<size(A,1)
    A(j+1:end,j) = v(2:end);
  end
end
if nargout == 2
  R = A;
  A = eye(size(A,1));
  for j = size(R,2):-1:1
    v = [1;R(j+1:end,j)];
    A(j:end,j:end) = A(j:end,j:end) - v*(beta(j)*v'*A(j:end,j:end))
       ;
  end
  R = triu(R);
end
```

MATLAB-Funktion: HouseholderMult.m

```
function y = prod_rx(A,x)
% extract R from A and multiply with x
y = triu(A) * x;

function y = prod_qx(A,x)
y = x;
for j = size(A,2)-1:-1:1
```

```
8       v = [1;A(j+1:end,j)];
9       beta = 2/(v'*v);
10      y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
11  end
12
13  function y = prod_qtx(A,x)
14  y = x;
15  for j = 1:size(A,2)-1
16      v = [1;A(j+1:end,j)];
17      beta = 2/(v'*v);
18      y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
19  end
```

MATLAB-Beispiel:

Mit der Funktion `Householder` wird zum einen die Matrix A kompakt überschrieben, d.h. der Rückgabewert benötigt nur den Speicherplatz der ursprünglichen Matrix; zum anderen kann man sich auch Q und R mit $A = QR$ ausgeben lassen. Die Berechnung von Rx, Qx und $Q^T x$, wenn die kompakte Form vorliegt, ist in den Routinen `prod_qx`, etc. realisiert.

```
>> A = rand(3);
>> x = rand(3,1);
>> [Q,R] = Householder(A);
>> A-Q*R
ans =
  1.0e-015 *
   -0.1110         0   -0.1110
   -0.0035   -0.1110         0
   -0.1110         0    0.2220
>> C = Householder(A);
>> Q'*x - prod_qtx(C,x)
ans =
  1.0e-015 *
    0.2220
    0.0555
    0.5551
```

4.5 Die Singulärwertzerlegung

Die Singulärwertzerlegung (engl. Singular Value Decomposition, SVD) ist eine weitere Möglichkeit, ein lineares Ausgleichsproblem zu lösen. Sie hat aber auch zahlreiche andere Anwendungen z.B. in der Statistik (u.a. bei der Hauptkomponentenanalyse, Hauptachsentransformation).

Satz 4.5.1 *Zu jedem $A \in \mathbb{R}^{m\times n}$ existieren orthogonale Matrizen $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ und eine Diagonalmatrix*

$$\Sigma := \mathit{diag}\,(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}\ ,\ p = \min\{m, n\}$$

*mit $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, (den **Singulärwerten**), so dass*

$$A = U\Sigma V^T. \tag{4.6}$$

Beweis: Es genügt zu zeigen, dass

$$U^T AV = \begin{pmatrix} \sigma & 0 \\ 0 & B \end{pmatrix} \text{ mit } \sigma \geq 0\,,\ B \in \mathbb{R}^{(m-1)\times(m-1)}\,,$$

der Rest folgt dann induktiv. Sei $\sigma = \|A\|_2 = \sup_{\|x\|_2=1} \|Ax\|_2$. Da $B_1(0) = \{x \in \mathbb{R}^n : \|x\|_2 = 1\}$ kompakt ist, existiert ein $v \in B_1(0)$ mit

$$\|Av\|_2 = \|A\|_2 \|v\|_2 = \sigma.$$

Also erfüllt $u := \frac{1}{\sigma} Av$ die Gleichungen $Av = \sigma u$, $\|u\|_2 = \frac{1}{\sigma}\|Av\|_2 = 1$.
Ergänze nun u und v zu Orthogonalbasen

$$\{u = U^1, U^2, \ldots, U^m\}\ ,\ \{v = V^1, \ldots, V^n\}$$

des $\mathbb{R}^m$ bzw. $\mathbb{R}^n$. Daraus folgt, dass $U = (U^1, \ldots, U^m),\ V = (V^1, \ldots, V^n)$ orthogonal sind und

$$\begin{aligned} A_1 : &= U^T A V = (U^T A V^1, \ldots, U^T A V^n) \\ &= (\sigma e^1, U^T A V^2, \ldots, U^T A V^n) = \begin{pmatrix} \sigma & w^T \\ 0 & B \end{pmatrix},\ w \in \mathbb{R}^{n-1}. \end{aligned}$$

Nun gilt

$$\left\| A_1 \begin{pmatrix} \sigma \\ w \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} \sigma^2 + w^T w \\ Bw \end{pmatrix} \right\|_2^2 \geq (\sigma^2 + \|w\|_2^2)^2 = \left\| \begin{pmatrix} \sigma \\ w \end{pmatrix} \right\|_2^2,$$

also $\|A_1\|_2^2 \geq \sigma^2 + \|w\|_2^2$.

Daraus folgt $\sigma^2 = \|A\|_2^2 = \|U^T A V\|_2^2 = \|A_1\|_2^2 \geq \sigma^2 + \|w\|_2^2 \geq \sigma^2 > 0$, was nur gelten kann, wenn $w = 0$. □

Bemerkung 4.5.2 *(a) Mit $A = U\Sigma V^T$ folgt $A^T = V\Sigma^T U^T$, also*

$$A^T A = V\Sigma^T U^T U \Sigma V^T = V\Sigma^2 V^T$$

mit $\Sigma^2 =$ diag $(\sigma_1^2, \ldots, \sigma_n^2)$, also der Diagonalisierung von $A^T A$.

(b) Der SVD–Algorithmus ist etwas kompliziert. Er besteht

- *aus einer Bi-Diagonalisierung von A z.B. mit Givens und*
- *einem symmetrischen QR für $A^T A$.*

Details findet man z.B. in [Golub/Loan, 427-433].

Die Lösung des linearen Ausgleichsproblems mittels der SVD erfolgt nun über die sogenannte Pseudo–Inverse.

Definition 4.5.3 *Angenommen $A \in \mathbb{R}^{m\times n}$ habe Rang r und die SVD $A = U\Sigma V^T$, dann heißt die Matrix*

$$A^+ = V\Sigma^+ U^T \tag{4.7}$$

mit $\Sigma^+ :=$ diag $(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0)$ ***Pseudo–Inverse*** *(oder* ***Moore–Penrose–Inverse****).*

Lemma 4.5.4 *Es gilt $A^+ = (A^T A)^{-1} A^T$.*

Beweis. Nach Bemerkung 4.5.2 (a) gilt

$$(A^T A)^{-1} = (V\Sigma^2 V^T)^{-1} = V\Sigma^{-2} V^T = V(\Sigma^T\Sigma)^{-1} V^T,$$

also

$$(A^T A)^{-1} A^T = V(\Sigma^T\Sigma)^{-1} V^T V \Sigma U^T = V(\Sigma^T\Sigma)^{-1} U^T$$

sowie $\Sigma^T\Sigma =$ diag $(\sigma_1^2, \ldots, \sigma_r^2)$ und damit $(\Sigma^T\Sigma)^{-1}\Sigma = \Sigma^+$. □

Lemma 4.5.5 *Die Pseudo–Inverse A^+ von A besitzt folgende Eigenschaften*

(a) $(A^+A)^T = A^+A,\ (AA^+)^T = AA^+,$

(b) $A^+AA^+ = A^+,$

(c) $AA^+A = A,$

(d) Falls A vollen Rang hat, gilt $A^+A = Id$.

Bemerkung 4.5.6 *Sowohl die Eigenschaft in Lemma 4.5.4 als auch diejenigen in Lemma 4.5.5 (gemeinsam) können als äquivalente Definition von A^+ verwendet werden.*

Satz 4.5.7 *Sei $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $b \in \mathbb{R}^m$, dann ist $x = A^+b$ diejenige Lösung von*

$$\|Ax - b\|_2 \longrightarrow \textit{Min}!$$

*mit der **kleinsten** euklidischen Norm.*

Beweis. Für $x = A^+b$ gilt nach Lemma 4.5.4

$$x = A^+b = (A^TA)^{-1}A^Tb,$$

also $A^TAx = A^Tb$, also erfüllt x die Normalengleichungen und löst damit das lineare Ausgleichsproblem. Seien nun $x_1, x_2 \in \mathbb{R}^n$ zwei Lösungen, dann gilt $A^TA(x_1 - x_2) = 0$, also

$$0 = (x_1 - x_2)^TA^TA(x_1 - x_2) = \|A(x_1 - x_2)\|_2^2,$$

d.h. $A(x_1 - x_2)$ bzw. $x_1 - x_2 \in$ Ker (A). Damit hat also jede Lösung $\tilde{x}$ die Form

$$\tilde{x} = x + y \text{ mit } x = A^+b,\ y \in \text{ Ker } (A).$$

Sei nun $y \in$ Ker (A), dann gilt wegen Lemma 4.5.5 (b)

$$y^TA^+b = y^TA^+AA^+b = ((A^+A)^Ty)^TA^+b = (A^+Ay)^TA^+b = 0,$$

da $y \in$ Ker (A) und wegen Lemma 4.5.5 (a). Also gilt

$$x = A^+b \perp \text{ Ker } (A).$$

Damit folgt nun für jede Lösung $\tilde{x} = x + y$

$$\|\tilde{x}\|_2^2 = \|x + y\|_2^2 = \|x\|_2^2 + 2(x, y)_2 + \|y\|_2^2 = \|x\|_2^2 + \|y\|_2^2,$$

da $(x, y)_2 = x^Ty = 0$ und der obige Ausdruck wird minimal genau dann, wenn $y = 0$. □

Bemerkung 4.5.8 *Nun sei $p :=$ Rang $(A) \leq \min\{m, n\}$. Dann liefert die QR–Zerlegung nach p Schritten*

$$QA = \left[\begin{array}{c|c} R & S \\ \hline 0 & 0 \end{array}\right] \tag{4.8}$$

mit $R \in \mathbb{R}^{p\times p}$ und $S \in \mathbb{R}^{p\times(n-p)}$. Damit kann man also die QR–Zerlegung auch zur numerischen Rangbestimmung verwenden.

Lemma 4.5.9 *(a) $x \in \mathbb{R}^n$ ist genau dann Lösung des linearen Ausgleichsproblems, falls $x = (x_1, x_2)^T$ mit $x_1 \in \mathbb{R}^p$, $x_2 \in \mathbb{R}^{n-p}$ und*

$$x_1 = R^{-1}b_1 - R^{-1}Sx_2,$$

wobei $(b_1, b_2)^T = Qb$, $b_1 \in \mathbb{R}^p$, $b_2 \in \mathbb{R}^{m-p}$. Hier wird nichts über den Rang vorausgesetzt.

*(b) Sei $p < n$ (der sogenannte **Rang-defekte Fall**),*

$$V := R^{-1}S \in \mathbb{R}^{p\times(n-p)}, \quad u := R^{-1}b_1 \in \mathbb{R}^p.$$

Dann ist die Lösung x des linearen Ausgleichsproblems mit der kleinsten Norm $\|x\|_2$ gegeben durch $x = (x_1, x_2) \in \mathbb{R}^p \times \mathbb{R}^{n-p}$ mit

$$(Id + V^TV)x_2 = V^Tu, \quad x_1 = u - Vx_2.$$

Beweis. (a) Es gilt

$$\|b - Ax\|_2^2 = \|Qb - QAx\|_2^2 = \|Rx_1 + Sx_2 - b_1\|_2^2 + \|b_2\|_2^2,$$

also die Bedingung $Rx_1 = b_1 - Sx_2$.

(b) Nach (a) gilt $x_1 = R^{-1}(b_1 - Sx_2) = u - Vx_2$ und damit

$$\begin{aligned} \|x\|_2^2 &= \|x_1\|_2^2 + \|x_2\|_2^2 = \|u - Vx_2\|_2^2 + \|x_2\|_2^2 \\ &= \|u\|_2^2 - 2\langle u, Vx_2\rangle + \langle Vx_2, Vx_2\rangle + \|x_2\|_2^2. \end{aligned}$$

Weiter gilt $-2(u, Vx_2) + \langle Vx_2, Vx_2\rangle = \langle x_2, V^TVx_2 - 2V^Tu\rangle$, also

$$\begin{aligned} \|x\|_2^2 &= \|u\|_2^2 + \langle x_2, V^TVx_2 - 2V^Tu\rangle + \|x_2\|_2^2 \\ &= \|u\|_2^2 + \langle x_2, (Id + V^TV)x_2 - 2V^Tu\rangle =: \varphi(x_2). \end{aligned}$$

Letztere Funktion gilt es zu minimieren. Es gilt

$$\begin{aligned} \varphi'(x_2) &= -2V^Tu + 2(Id + V^TV)x_2 \\ \varphi''(x_2) &= 2(Id + V^TV) > 0, \end{aligned}$$

da diese Matrix s.p.d. ist. Also ist $\varphi(x_2)$ genau dann minimal, wenn

$$(Id + V^TV)x_2 = V^Tu,$$

also die Behauptung. □

Damit kann man die Pseudo–Inverse auch ohne SVD direkt aus QR bestimmen.

Algorithmus 4.5.1: Pseudo–Inverse über QR

Sei $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Bestimme $x = A^+b$ gemäß:

1.) Bestimme die QR-Zerlegung von A gemäß (4.8) und berechne p, R und S sowie Q.

2.) Berechne $V \in \mathbb{R}^{p\times(n-p)}$ aus $RV = S$, löse also $n - p$ gestaffelte Gleichungssysteme der Dimension p.

3.) Bestimme die Cholesky-Zerlegung $Id + V^TV = LL^T$.

4.) Setze $Qb =: (b_1, b_2)^T$ mit $b_1 \in \mathbb{R}^p$, $b_2 \in \mathbb{R}^{m-p}$.

5.) Bestimme u aus dem gestaffelten System $Ru = b_1$.

6.) Berechne $x_2 \in \mathbb{R}^{n-p}$ aus zwei gestaffelten Systemen $LL^Tx_2 = V^Tu$.

7.) Setze $x_1 = u - Vx_2$.

Ausgabe: $x = (x_1, x_2)^T = A^+b$.

---

5 NICHTLINEARE GLEICHUNGEN

Nachdem wir uns in der Vorlesung Numerik I mit dem Lösen linearer Gleichungssysteme beschäftigt haben, wenden wir uns nun nichtlinearen Gleichungen (in einer oder mehreren Variablen) zu. Diese Gleichungen treten häufig als Teilaufgabe bei der Behandlung komplexerer Probleme auf.

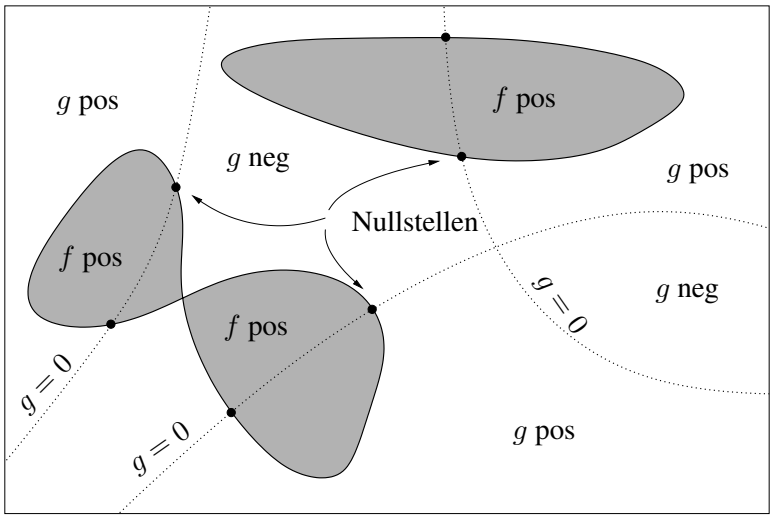


Abb. 5.1: Lösung von zwei Gleichungen in zwei Unbekannten. Die durchgezogenen Linien sind Niveaulinien zu $f(x, y) = 0$, die gestrichelten Linien zu $g(x, y) = 0$. Die gesuchten Lösungen sind die Schnittpunkte der völlig unabhängigen Nulllinien. Die Anzahl der Nullstellen ist im Allgemeinen a-priori nicht bekannt.

Ist ein System von n nichtlinearen Gleichungen in n Unbekannten gegeben, d.h. mit einer stetigen, nichtlinearen Funktion

$$\tilde{f} : \mathbb{R}^n \to \mathbb{R}^n,$$

so können wir das gegebene Problem $\tilde{f}(x) = b$ in eine **Nullstellenaufgabe** transformieren, so dass Lösungen $x \in \mathbb{R}^n$ der Gleichung

$$f(x) := \tilde{f}(x) - b = 0 \tag{5.1}$$

zu bestimmen sind. Noch allgemeiner: $f : \Omega \to W$ mit Mengen Ω, W. Wir beschränken uns aber in dieser Vorlesung auf den Fall $W = \mathbb{R}^n$. Für $n = 1$ spricht man von einer **skalaren** Gleichung, für $n > 1$ von einem **System**. Mehrere Modellbeispiele, bei denen nichtlineare Gleichungen auftreten, wollen wir hier vorstellen.

Beispiel 5.0.1 Wir beginnen mit einigen einfachen Beispielen, die aber bereits einen Eindruck von der Vielfalt nichtlinearer Gleichungssysteme vermitteln.

(a) Löse $f(x) = 0$ für $f(x) = x^2 - 2px + q$ mit$p, q \in \mathbb{R}$. Bekanntermaßen gibt es 2 Lösungen $x_{1,2} = p \pm \sqrt{p^2 - q}$. Falls $p^2 - q < 0$ gibt es keine reelle Lösung. An diesem ganz einfachen Beispiel sieht man bereits, dass Existenz und Eindeutigkeit im nichtlinearen Fall nicht trivial sind.

(b) Es müssen aber nicht immer Zahlen (oder Vektoren) die gesuchten LÖsungen sein. Als Beispiel betrachte man nichtlineare Differentialgleichungen, z.B. die Transportgleichung

(„Burgers–Gleichung“):

$$\frac{d}{dt}u(t,x) + u(t,x)\frac{d}{dx}u(t,x) = 0,\ (u_t + uu_x = 0)$$

wobei $u(t,x)$ die Geschwindigkeit eines Teilchens zur Zeit t am Ort x ist. Die Unbekannte ist also eine **Funktion**, man sucht also in einem **unendlich-dimensionalen** Raum (den Raum aller in t und x stetig differenzierteren Funktionen). Anwendungen sind z.B. Simulationen des Straßenverkehrs oder Informationsausbreitung im Internet.

(c) Numerische Transformation von Wahrscheinlichkeits–Verteilungsfunktionen und die Berechnung von Quantilen. Aus der Wahrscheinlichkeitsrechnung kennen Sie Tabellen von Quantilen z.B. der Standardnormalverteilung. Diese können nicht exakt berechnet werden, sondern ergeben sich als Lösung eines nichtlinearen Problems.

(d) Fast alle „realen“ Phänomene sind nichtlinear — werden oft zur Vereinfachung linearisiert.

Wie man schon an obigem Beispiel sieht, hat man hier keine „handlichen“ Kriterien für Existenz und Eindeutigkeit. Dies hat natürlich auch Konsequenzen für die numerischen Lösungsverfahren. Im Folgenden zeigen wir weitere Beispiele.

Beispiel 5.0.2 (Zinssatz bei einem Kredit) Wie hoch darf der Zinssatz sein, wenn man einen Kredit über 10.000 Euro in 10 Jahren abzahlen möchte und man höchstens 400 Euro monatlich aufbringen kann? Oder allgemeiner eine Kreditsumme K_0 in n Jahren mit monatlichen Raten R getilgt habe möchte?
Der Einfachheit halber rechnen wir den jährlichen Zinssatz p in einen monatlichen Zinssatz m um, d.h. verzinst man monatlich mit einem Prozentsatz m oder jährlich mit p, so erhält man am Ende des Jahres jeweils das Gleiche. Die Beziehung zwischen p und m ist also $(1+m)^{12} = 1+p$.
Für den Restbetrag K_i des Kredits nach $i \in \mathbb{N}$ Monaten gilt:

$$\begin{aligned}
K_i &= K_{i-1}(1+m) - R \\
K_{i+1} &= K_i(1+m) - R = [K_{i-1}(1+m) - R](1+m) - R \\
&= K_{i-1}(1+m)^2 - R\,[(1+m)+1] \\
&\vdots \\
K_n &= K_0(1+m)^n - R\,\left[(1+m)^{n-1} + \ldots + (1+m) + 1\right] \\
&= K_0(1+m)^n - R\,[(1+m)^n - 1]\,/m \\
&= (K_0 - R/m)(1+m)^n + R/m
\end{aligned}$$

Zu gegebener Kreditsumme K_0, monatlichem Zinssatz m und Tilgungsdauer (n Monate) berechnet sich die Rate R, um den Kredit vollständig zu tilgen, indem wir $K_n = 0$ setzen, d.h.

$$R = K_0\, m\, (1+m)^n / \left[(1+m)^n - 1\right] .$$

Auch die Tilgungsdauer lässt sich analytisch darstellen

$$n = \frac{\log\left(R/(R - m\,K_0)\right)}{\log(1+m)} .$$

(Wie ist hier das Ergebnis zu interpretieren, falls n nicht ganzzahlig ist?) Wie gewinnt man jedoch m zu gegebenen $K_0 > 0$, $n \in \mathbb{N}$, $K_0 > R > K_0/n$ aus der Gleichung

$$(m\,K_0 - R)(1+m)^n + R = 0\,?$$

Es ist offensichtlich, dass $0 < m < 1$ gilt und $f(m) := (m\,K_0 - R)(1+m)^n + R$ nur eine Nullstelle in $(0,1)$ hat. Aber wie kann man diese einfach, schnell und numerisch stabil bestimmen?

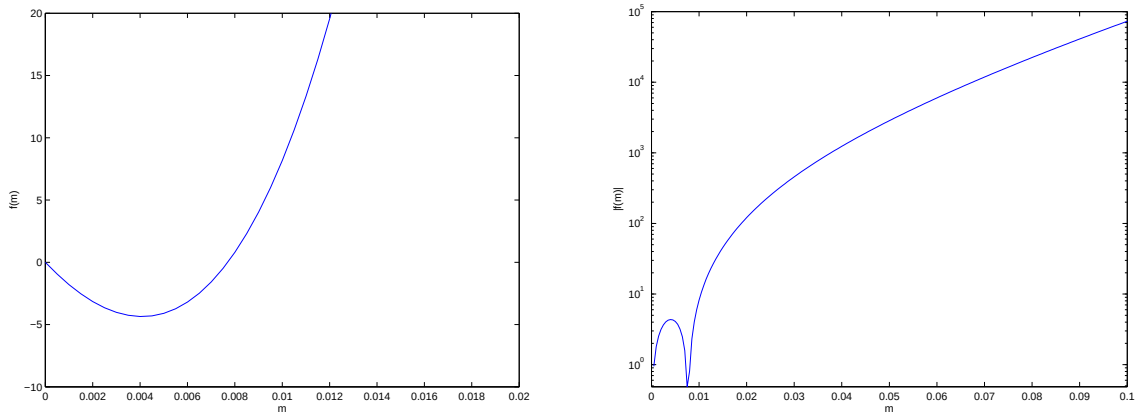


Abb. 5.2: Der Graph von $f(m) := (m\,K_0 - R)(1+m)^n + R$ (links, y-Achse linear skaliert, $x \in [0, 0.02]$) bzw. $|f(m)|$ (rechts, logarithmisch-skaliert, $x \in [0, 0.1]$) für $K_0 = 10000$, $R = 250$ und $n = 48$. Die Funktion f hat eine Nullstelle bei 0 und bei ≈ 0.008.

Beispiel 5.0.3 (Nullstellen von Orthogonalpolynomen) Die **Legendre-Polynome** $P_n \in \mathbb{P}_n$ $(n = 0, 1, 2, \ldots)$ erfüllen die Drei-Term-Rekursion

$$P_0(x) = 1,\ P_1(x) = x,\ (n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x),\ \ n \in \mathbb{N}\,.$$

Die ersten Legendre-Polynome lauten

$$\begin{aligned} P_0 &= 1 & P_3 &= \frac{1}{2}(5x^3 - 3x) \\ P_1 &= x & P_4 &= \frac{1}{8}(35x^4 - 30x^2 + 3) \\ P_2 &= \frac{1}{2}(3x^2 - 1) & P_5 &= \frac{1}{8}(63x^5 - 70x^3 + 15x)\,. \end{aligned}$$

Die Nullstellen der Legendre-Polynome liegen in $(-1, 1)$ und die Nullstellen von P_n trennen die Nullstellen von P_{n+1} $(n \in \mathbb{N})$. Mit Hilfe dieser Eigenschaft lassen sich sukzessive Intervalle finden, in denen Nullstellen liegen, z.B. P_1 hat die Nullstelle $\xi_1^{(1)} = 0$ und somit liegen die Nullstellen $\xi_1^{(2)}$, $\xi_2^{(2)}$ von P_2 in $(-1, 0)$ und $(0, 1)$, die Nullstellen $\xi_1^{(3)}$, $\xi_2^{(3)}$, $\xi_3^{(3)}$ von P_3 in $(-1, \xi_1^{(2)})$, $(\xi_1^{(2)}, \xi_2^{(2)})$ und $(\xi_2^{(2)}, 1)$. Diese Eigenschaft der Legendre-Polynome gilt auch für weitere Orthogonalpolynome. Die Berechnung der Nullstellen ist u.a. wichtig im Zusammenhang mit Gauß-Quadraturformeln.

Beispiel 5.0.4 (Extremalstellen von skalaren Funktionen) Die mehrdimensionale Erweiterung der Rosenbrock[1]-Funktion

$$f(x) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right] \quad (x \in \mathbb{R}^n)$$

hat für $n \geq 4$ mindestens ein lokales Minimum in der Umgebung von $(x_1, x_2, \ldots, x_n) = (-1, 1, \ldots, 1)$ neben dem globalen Minimum $(x_1, \ldots, x_n) = (1, \ldots, 1)$. Diese Extremalstellen erfüllen notwendigerweise die Gleichung $\nabla f = 0$. Wie kann man nun für $n \geq 4$ ein solches

[1] Howard Harry Rosenbrock, 16. Dez. 1920 - 21 Okt. 2010

lokales Minimum bestimmen? Dies bedeutet, ein $x \in \mathbb{R}^n \setminus \{(1,\ldots,1)\}$ ist zu bestimmen mit

$$\begin{aligned} g(x) &= (g_1(x),\ldots,g_n(x))^T = 0 \\ g_1(x) &:= \frac{\partial f}{\partial x_1} = -2(1-x_1) - 400x_1(x_2 - x_1^2), \\ g_i(x) &:= \frac{\partial f}{\partial x_i} = -2(1-x_i) - 400x_i(x_{i+1} - x_i^2) + 200(x_i - x_{i-1}^2) \quad (i = 2,\ldots,n-1), \\ g_n(x) &:= \frac{\partial f}{\partial x_n} = 200(x_n - x_{n-1}^2). \end{aligned}$$

Betrachten wir dazu zunächst die Situation in einer Raumdimension.

5.1 Bisektionsmethode

Herleitung des Algorithmus: Diese Methode, motiviert durch Überlegungen aus der reellen eindimensionalen Analysis (siehe Intervallschachtelung), löst (5.1) dadurch, dass sie die Lösung durch systematisch kleiner werdende Intervalle einschließt. Man geht von der Annahme aus, es sei ein Intervall $I := [a,b]$ bekannt mit $f(a) \cdot f(b) < 0$.
Aus Stetigkeitsgründen existiert eine Lösung x^* im Inneren von I mit $f(x^*) = 0$.
Durch den Mittelpunkt $m = \frac{1}{2}(a+b)$ des Intervalls I wird der Funktionswert $f(m)$ bestimmt. Ist $f(m) \neq 0$ entscheidet nun das Vorzeichen, in welchem der Teilintervalle $[a,m],[m,b]$ die gesuchte Lösung x^* liegt. Wir erhalten damit folgenden Algorithmus:

MATLAB-Funktion: BisektionsMethode.m

```
function Nullstelle = BisektionsMethode(a,b,func,epsilon)
% Initialisierung
temp=[]
fa = func(a); fb = func(b);
while abs(a-b) > epsilon
 m = (a+b)/2;
 fm = func(m);
 if fm == 0
   Nullstelle = m;
   return
 elseif fa*fm < 0
   b = m;
   fb = fm;
 else
   a = m;
   fa = fm;
 end
 temp = [temp;m];
end
% Lösung
Nullstelle = m;
```

MATLAB-Beispiel:

Testen wir nun die Bisektionsmethode anhand von Bsp. 5.0.2. Wie hoch darf der Zinssatz sein, wenn man einen Kredit über 10.000$ in 48 Monaten zurückzahlen möchte, aber nur 250$ monatlich aufbringen kann?

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = Bisektionsmethode(eps,1,f,1e-7)
m =
   0.00770145654678
>> p = 100 * ((1+m)^12-1)
p =
   9.64343564476941
```

Nach der Umrechnung in den jährlichen Zinssatz sehen wir, dass wir uns den Kredit nur erlauben könnten, wenn der Zinssatz niedriger als 9.65% ist.

Definition 5.1.1 (Konvergenzgeschwindigkeit) *Sei (x_k) eine reellwertige konvergente Folge mit $x_k \in \mathbb{R}^n$ $(k \in \mathbb{N})$ und Grenzwert $x \in \mathbb{R}^n$. Man bezeichnet (x_k) als* **linear konvergent**, *wenn es ein $0 < \rho < 1$ gibt mit*

$$\|x - x_{k+1}\| \leq \rho \|x - x_k\| \quad (k = 0, 1, \ldots).$$

Die Zahl ρ wird **Konvergenzfaktor** *(oder auch* **Kontraktionsrate***) genannt. Gibt es eine gegen Null konvergente Folge (ρ_k) mit*

$$\|x - x_{k+1}\| \leq \rho_k \|x - x_k\| \quad (k = 0, 1, \ldots),$$

so heißt (x_k) **superlinear konvergent**. *Gibt es ein $\rho > 0$ und ein $1 < q \in \mathbb{R}$ mit*

$$\|x - x_{k+1}\| \leq \rho \|x - x_k\|^q \quad (k = 0, 1, \ldots),$$

so heißt (x_k) konvergent mit **Konvergenzordnung** *q. Für $q = 2$ spricht man auch von* **quadratischer Konvergenz**.

Bemerkung 5.1.2 (Konvergenz des Bisektionsverfahrens) *Betrachten wir den Mittelpunkt x_k des Intervalls nach der k-ten Intervallhalbierung als Näherung an x^*, so gilt die a-priori Fehlerabschätzung*

$$|x_k - x^*| \leq \frac{b-a}{2^{k+1}} \quad (k = 0, 1, 2, \ldots).$$

Da die Fehlerschranke wie eine geometrische Folge abnimmt, liest man manchmal, dass die „Konvergenzordnung“ 1 sei (also lineare Konvergenz vorliege). Nach Definition 5.1.1 ist das Bisektionsverfahren ***nicht*** *linear konvergent, da auf der rechten Seite der Abschätzung nicht $|x_{k-1} - x^*|$ steht, man also keine monotone Reduktion des Fehlers hat. Es gibt Beispiele, bei denen der Fehler springt, vgl. [QSS1, §6.2].*

5.2 Regula-Falsi[1]

Herleitung des Algorithmus Wiederum gehen wir davon aus, dass ein Intervall $I := [a, b]$ mit $f(a) \cdot f(b) < 0$ bekannt sei. Anstatt nun I durch Hinzufügen eines Mittelpunkts in zwei Intervalle zu zerlegen, wählen wir eine zusätzliche Stelle ξ, die Nullstelle der Geraden durch $(a, f(a))$ und $(b, f(b))$. Mit den beiden Intervallen $[a, \xi]$, $[\xi, b]$ verfahren wir nun analog zur Methode der Intervallhalbierung, wobei gilt:

$$\xi = \frac{af(b) - bf(a)}{f(b) - f(a)}. \tag{5.2}$$

Als Algorithmus ergibt sich somit:

[1] lat.: „Regel des falschen Ansatzes“

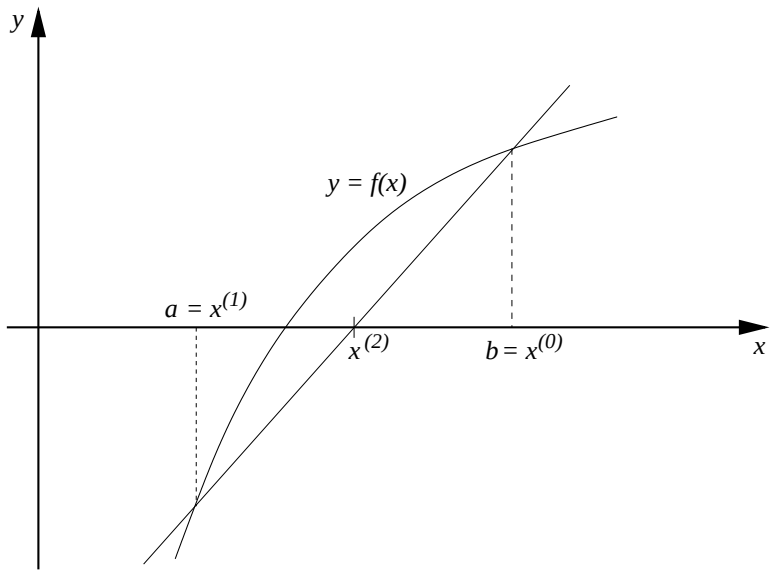


Abb. 5.3: Die geometrische Interpretation der Regula-Falsi-Methode.

MATLAB-Funktion: RegulaFalsi.m

```
function Nullstelle = RegulaFalsi(a,b,func,epsilon)
fa = func(a); fb = func(b); % Initialisierung
m = (a*fb-b*fa)/(fb-fa); fm = func(m);
while abs(fm) > epsilon
 if fa*fm < 0
   b = m;
   fb = fm;
 else
   a = m;
   fa = fm;
 end
 m = (a*fb-b*fa)/(fb-fa);
 fm = func(m);
end
Nullstelle = m; % Lösung
```

Satz 5.2.1 *Es sei x^* einzige Nullstelle von f in $I := [a,b]$, $f \in C^3(I)$ und $f'(x^*) \cdot f''(x^*) \neq 0$. Dann beträgt die Konvergenzordnung der Regula-Falsi-Methode $p = 1$.*

Beweis. Es bezeichne (x_k) die sich aus aus dem Regula-Falsi-Algorithmus ergebende Folge von „Mittelpunkten“, d.h. $x_k \in (a_k, b_k)$ $(k \in \mathbb{N})$. Es seien

$$\varepsilon_k^a := a_k - x^*, \quad \varepsilon_k^b := b_k - x^*.$$

Aus (5.2) und der Voraussetzung $f(x^*) = 0$ folgt

$$\begin{aligned} \varepsilon_k := x_k - x^* &= \frac{(a_k - x^*)f(b_k) - (b_k - x^*)f(a_k)}{f(b_k) - f(a_k)} \\ &= \frac{\varepsilon_k^a f(x^* + \varepsilon_k^b) - \varepsilon_k^b f(x^* + \varepsilon_k^a)}{f(x^* + \varepsilon_k^b) - f(x^* + \varepsilon_k^a)}. \end{aligned}$$

Da $f \in C^3(I)$ nach Voraussetzung gilt, liefert die **Taylor**[2]-Entwicklung:

$$\begin{aligned}
\varepsilon_k &= \frac{\varepsilon_k^a\{\varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \ldots\} - \varepsilon_k^b\{\varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \ldots\}}{\{\varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \ldots\} - \{\varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \ldots\}} \\
&= \frac{\frac{1}{2}\varepsilon_k^a\varepsilon_k^b(\varepsilon_k^b - \varepsilon_k^a)f''(x^*) + \ldots}{(\varepsilon_k^b - \varepsilon_k^a)\{f'(x^*) + \frac{1}{2}(\varepsilon_k^b + \varepsilon_k^a)f''(x^*) + \ldots\}} \\
&= \frac{\frac{1}{2}\varepsilon_k^a\varepsilon_k^b f''(x^*) + \ldots}{f'(x^*) + \ldots}
\end{aligned}$$

Nach der Herleitung des Algorithmus gilt $a_k < x^* < b_k$. Demzufolge besitzen ε_k^a und ε_k^b entgegengesetztes Vorzeichen und es gilt weiter $\varepsilon_k^b - \varepsilon_k^a > 0$. Für hinreichend kleine $|\varepsilon_k^a|$ und $|\varepsilon_k^b|$ folgt damit

$$\varepsilon_k \approx \frac{f''(x^*)}{2f'(x^*)}\varepsilon_k^a\varepsilon_k^b\,. \tag{5.3}$$

Mit $f''(x^*) \neq 0$ gilt aber weiterhin, dass f in einer hinreichend kleinen Umgebung von x^* konvex oder konkav ist, und folglich bleibt ein Intervallende fest und wird im Verfahren nur umbenannt. Deshalb ist ε_k nur direkt proportional zu einem der beiden vorhergehenden ε_k^a oder ε_k^b. Die asymptotische Fehlerkonstante C ist für eine konkave Funktion gegeben durch

$$C = \left|\frac{f''(x^*)}{2f'(x^*)}\varepsilon_k^a\right|, \quad \varepsilon_k^a = \varepsilon_{k-1}^a = \ldots = \varepsilon_1^a = x_1 - x^*,$$

und damit konvergiert die Folge (x_k) linear. Analoges gilt für konvexe Funktionen mit ε_k^b anstatt ε_k^a. □

MATLAB-Beispiel:

Testen wir nun das Regula-Falsi-Verfahren anhand Bsp. 5.0.2 $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = RegulaFalsi(1e-5,0.1,f,1e-7)
m =
    0.00770147244890
```

Bei einem genaueren Vergleich mit dem Bisektionsverfahren stellt man bei diesem Beispiel eine langsamere Konvergenz des Regula-Falsi-Verfahrens fest.

5.3 Die Sekantenmethode

Als Modifikation des Regula-Falsi-Verfahrens verzichtet man bei der Sekantenmethode darauf, die Lösung durch zwei Näherungswerte einzuschließen.

Zu zwei vorgegebenen Näherungswerten x_0 und x_1, welche die Lösung nicht notwendigerweise einzuschließen brauchen, bestimmt man x_2 als Nullstelle der Sekante zu $(x_0, f(x_0))$ und $(x_1, f(x_1))$. Ungeachtet der Vorzeichen bestimmt man aus x_1, x_2 eine nächste Näherung x_3.

Die Iterationsvorschrift der Sekantenmethode ergibt sich damit zu

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \quad (k = 1, 2, \ldots)\,. \tag{5.4}$$

Dieses zweistufige Iterationsverfahren setzt natürlich $f(x_k) \neq f(x_{k-1})$ voraus und gehört nicht zur Klasse der oben betrachteten Iterationsverfahren.

Satz 5.3.1 *Falls $f'(x^*) \cdot f''(x^*) \neq 0$ gilt, ist die Konvergenzordnung der Sekantenmethode $p = \frac{1}{2}(1 + \sqrt{5})$.*

[2] Taylor, Brook (1685-1731)

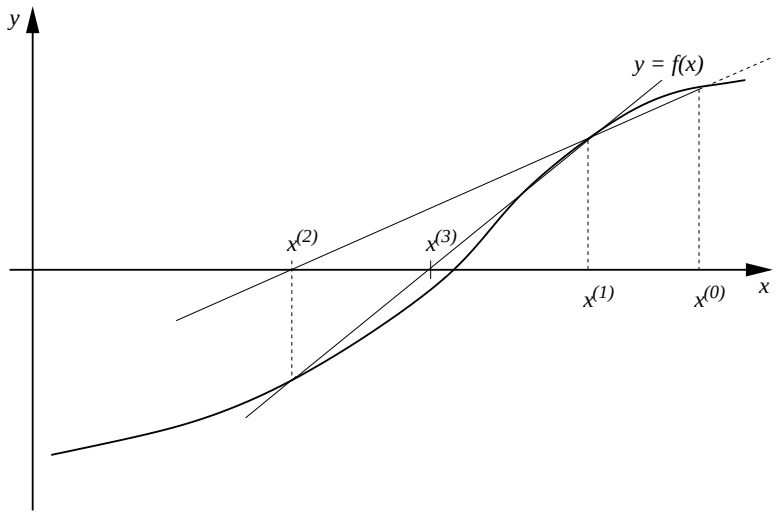


Abb. 5.4: Die geometrische Interpretation der Sekantenmethode.

Beweis. Wir betrachten die Sekantenmethode als Modifikation des Regula-Falsi-Verfahrens. Für hinreichend kleine Fehler $|\varepsilon_{k-1}|$ und $|\varepsilon_k|$ bleibt (5.3) für die Sekantenmethode gültig, bzw. geht über in

$$\varepsilon_{k+1} \approx \frac{f''(x^*)}{2f'(x^*)}\varepsilon_k\varepsilon_{k-1}\,. \tag{5.5}$$

Es besteht jedoch der Unterschied, dass bei Erhöhung von k sich beide Werte ε_{k-1} und ε_k ändern. Mit der Konstanten $C := |f''(x^*)/(2f'(x^*))|$ gilt für hinreichend große k

$$|\varepsilon_{k+1}| \approx C|\varepsilon_k| \cdot |\varepsilon_{k-1}|.$$

Nach unserer Definition der Konvergenzordnung versuchen wir nun diese **Differenzengleichung** mit dem Ansatz

$$|\varepsilon_k| = \rho|\varepsilon_{k-1}|^p,\ \ \rho > 0,\ p \geq 1$$

zu lösen. Einsetzen ergibt

$$\left(|\varepsilon_{k+1}| =\right)\quad \rho|\varepsilon_k|^p = \rho \cdot \rho^p|\varepsilon_{k-1}|^{p^2} \stackrel{!}{=} C \cdot \rho|\varepsilon_{k-1}|^{p+1} \quad \left(= C|\varepsilon_k| \cdot |\varepsilon_{k-1}|\right).$$

Diese letzte Gleichung kann aber für alle (hinreichend großen) k nur dann gelten, falls

$$\rho^p = C \text{ und } p^2 = p + 1$$

erfüllt sind. Die positive Lösung $p = \frac{1}{2}(1 + \sqrt{5})$ der quadratischen Gleichung ist deshalb die Konvergenzordnung der Sekantenmethode. Die asymptotische Fehlerkonstante ist $\rho = C^{1/p} = C^{0.618}$. □

MATLAB-Funktion: SekantenMethode.m

```
function x1 = SekantenMethode(x0,x1,func,epsilon)
% Initialisierung
fx0 = func(x0); fx1 = func(x1);
while abs(fx1) > epsilon && abs(fx0-fx1) > epsilon
  tmp = x1;
  x1 = x1 - fx1 *(x1-x0)/(fx1-fx0);
  x0 = tmp;
  fx0 = fx1;
  fx1 = func(x1);
end
```

MATLAB-Beispiel:

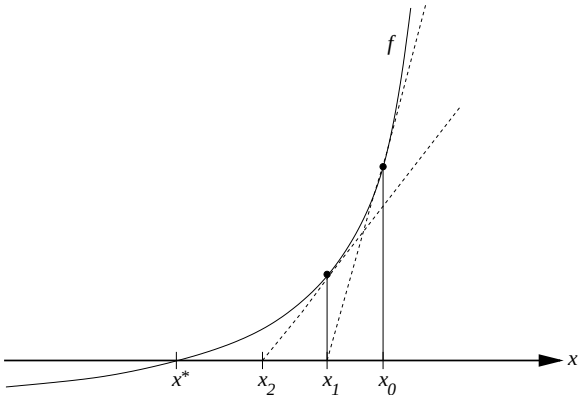


Abb. 5.5: Die geometrische Interpretation des **Newton**-Verfahrens.

Testen wir nun abschließend die Sekantenmethode an Bsp. 5.0.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = SekantenMethode(1e-2,0.1,f,1e-7)
m =
   0.00770147248822
```

Im Vergleich zu den oben gemachten Tests ist hier das Startintervall kleiner zu wählen.

5.4 Das Verfahren von Newton

Herleitung des Algorithmus Ist die gegebene Funktion $f(x)$ der zu lösenden Gleichung $f(x) = 0$ stetig differenzierbar, so wird im Verfahren von **Newton**[3] die Funktion $f(x)$ im Näherungswert x_k linearisiert und der iterierte Wert x_{k+1} als Nullstelle der Tangente in x_k definiert (vgl. Abbildung 5.5).

Aus der Tangentengleichung

$$y(x) = f(x_k) + (x - x_k)f'(x_k)$$

ergibt sich die Iterationsvorschrift

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,. \tag{5.6}$$

Bemerkung 5.4.1 *Die Methode von* ***Newton*** *gehört zur Klasse der Fixpunktiterationen mit der Funktion*

$$\phi(x) = x - \frac{f(x)}{f'(x)} \text{ mit } \phi(x^*) = x^*.$$

Satz 5.4.2 *Es sei $I := [a, b]$ ein echtes Intervall mit $a < x^* < b$ und $f \in C^3(I)$ mit $f'(x^*) \neq 0$, d.h. x^* ist eine einfache Nullstelle von $f(x)$.*

[3] Newton, Isaac (1642-1727)

*Dann existiert ein Intervall $I_\delta = [x^* - \delta, x^* + \delta]$ mit $\delta > 0$, für welches $\phi : I_\delta \to I_\delta$ eine Kontraktion darstellt. Ferner ist für jeden Startwert $x_0 \in I_\delta$ die Konvergenz der Folge (x_k) des **Newton**-Verfahrens mindestens von quadratischer Ordnung.*

Beweis. Für die erste Ableitung von ϕ erhalten wir

$$\phi'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

Da $f(x^*) = 0$, $f'(x^*) \neq 0$ und $f \in C^2(I)$ vorausgesetzt sind, gilt auch $\phi'(x^*) = 0$. Aus Stetigkeitsgründen existiert dann ein $\delta > 0$ derart, dass

$$|\phi'(x)| < 1 \text{ für alle } x \in [x^* - \delta, x^* + \delta] =: I_\delta$$

gilt. Somit ist ϕ eine Kontraktion in I_δ. Weiterhin sind für I_δ die Voraussetzungen des **Banachschen** Fixpunktsatzes erfüllt und damit ist die Konvergenz von (x_k) gezeigt. Zum Beweis der Konvergenzordnung definieren wir $e_{k+1} := x_{k+1} - x^*$. Eine **Taylorentwicklung** von ϕ um x^* und $\phi'(x^*) = 0$ liefert

$$\begin{aligned} e_{k+1} &= x_{k+1} - x^* = \phi(x_k) - \phi(x^*) \\ &= \phi(x^* + e_k) - \phi(x^*) \\ &= \phi(x^*) + e_k\phi'(x^*) + \frac{(e_k)^2}{2}\phi''(x^* + \theta_k e_k) - \phi(x^*) \quad \text{mit } \theta_k \in (0,1) \\ &= \frac{1}{2}(e_k)^2\phi''(x^* + \theta_k e_k). \end{aligned}$$

Damit gilt also

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} \leq \frac{1}{2} \sup_{0<\theta_k<1} |\phi''(x^* + \theta_k e_k)| =: M_k \,.$$

Da aber auch

$$\phi''(x) = \frac{f'(x)^2 f''(x) + f(x)f'(x)f^{(3)}(x) - 2f(x)f''(x)^2}{f'(x)^3}$$

gilt und $f \in C^3(I)$ vorausgesetzt wurde, existiert ein $C \in \mathbb{R}$ mit $M_k \leq C$ (für $k = 0, 1, 2, \ldots$) und somit ist das **Newton**-Verfahren quadratisch konvergent. □

MATLAB-Funktion: NewtonSimple.m

```
function [x,nit] = NewtonSimple(x,f,Df,tol,maxit,param)
nit = 0;
fx = f(x,param);
while norm(fx) > tol && nit <= maxit
  nit = nit+1;
  x = x-Df(x,param)\fx;
  fx = f(x,param);
end
nit
```

Beispiel 5.4.3 Vergleichen wir nun die Bisektionsmethode, das Sekantenverfahren und das Newton-Verfahren angewandt auf das Problem aus Bsp. 5.0.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$ sowie Anfangsbedingungen $a = eps$, $b = 1$ für die Bisektionsmethode, $x_0 = 1/2$, $x_1 = 1$ für das Sekantenverfahren und $x_0 = 0.1$ für das Newton-Verfahren, so erhalten wir die in Abb. 5.4 dargestellte Konvergenz.

Bemerkung 5.4.4 (Merkregel) *Das **Newton**-Verfahren konvergiert lokal quadratisch.*

MATLAB-Funktion: rosenbrock.m

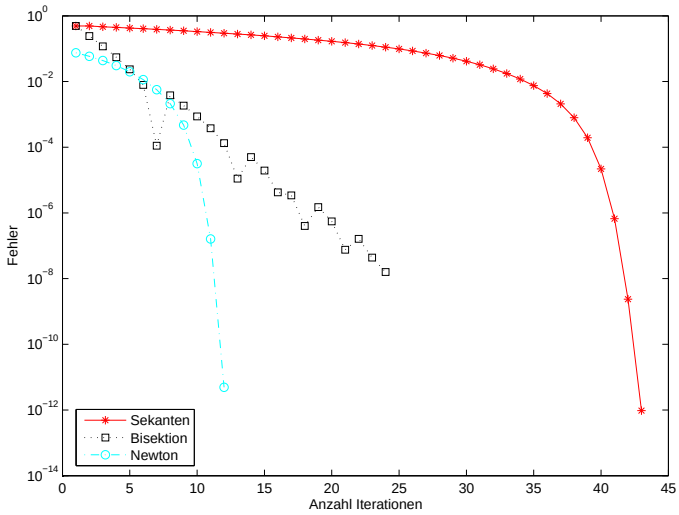


Abb. 5.6: Konvergenz von Bisektions-, Sekanten- und Newton-Verfahren angewandt auf $f(m) = (m \cdot K_0 - R) \cdot (1+m)^n + R$ aus Bsp. 5.0.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

```
function value = rosenbrock(x,param)
n = size(x,1);
value(2:n,1) = 200 * (x(2:end)-x(1:end-1).^2);
value(1:n-1,1) = value(1:n-1,1) - 2 *(1-x(1:end-1)) ...
                 - 400*x(1:end-1).*(x(2:end)-x(1:end-1).^2);
```

MATLAB-Funktion: D_rosenbrock.m

```
function D = D_rosenbrock(x,param)
n = size(x,1);
d0(2:n,1) = 200;
d0(1:n-1) = d0(1:n-1) + 2 + 1200*x(1:n-1).^2-400*x(2:n);
dm1 = -400*x;
dp1 = -400*x([1,1:n-1]);
D = spdiags([dm1,d0,dp1],[-1 0 1],n,n);
```

Man kann z.B. neben dem globalen Minimum $(x_1, \ldots, x_n) = (1, \ldots, 1)$ mit dem Newton-Verfahren ein weiteres lokales Minimum in der Umgebung von $(x_1, x_2, \ldots, x_n) = (-1, 1, \ldots, 1)$ von f in Bsp. 5.0.4 finden.

MATLAB-Beispiel:

```
>> n = 6;
>> x0 = [-1;ones(n-1,1)];
>> [x,nit] = NewtonSimple(x0,@rosenbrock,
   @D_rosenbrock,1e-10,100,[])
x =
  -0.98657497957099
   0.98339822883618
   0.97210667005309
   0.94743743682644
   0.89865118485173
   0.80757395203542
nit =
     5
```

5.5 Effizienz

Was nutzt eine hohe Konvergenzordnung, wenn man diese mit vielen Operationen „erkauft“? Ein solches Verfahren wäre ineffizient. Die „teuersten“ Operationen sind i.d.R. bei der Auswertung der Funktion f versteckt. Dies könnte ja z.B. die Lösung einer partiellen Differentialgleichung bedeuten. Daher verwendet man die Anzahl der Funktionsauswertungen als Synonym für den Aufwand bzw. die Anzahl der Operationen.

Definition 5.5.1 *Sei m die Anzahl von Funktionsauswertungen pro Iterationsschritt. Damit definiert man den* ***Effizienzindex***

$$\text{ind} := p^{\frac{1}{m}} \tag{5.7}$$

Beispiel 5.5.2 Wir fassen p und m für die oben vorgestellten Verfahren zusammen.

Verfahren	p	m	ind
Fixpunktiteration	1	1	1
Newton	2	2*	$\sqrt{2}$
Sekanten	$\frac{1+\sqrt{5}}{2}$	1	$\approx 1.6 (> \sqrt{2})$!
Regula Falsi	1	1	1

Unter der Annahme, dass die Auswertung von f in etwa so aufwendig ist wie die von f'.

Bemerkung 5.5.3 *Bislang haben wir — außer Differenzierbarkeit bei Newton — keine Eigenschaften von f vorausgesetzt. Für speziellere Funktionen kann man durch Ausnutzung der zusätzlichen Informationen effizientere Verfahren konstruieren. Eine Klasse „spezieller“ Funktionen sind Polynome.*

5.6 Fixpunkt–Iteration

Idee: Wie schon bei klassischen Iterationsverfahren zur Lösung linearer Gleichungssysteme besteht die einfache Idee darin, das Nullstellen–Problem $f(x) = 0$, in ein Fixpunkt–Problem um-

zuschreiben. Wir werden schnell sehen, dass diese Idee auch für Systeme von nichtlinearen Gleichungen verwendet werden kann. Die Fixpunkt–Funktion lautet dann

$$\Phi(x) := x - f(x). \tag{5.8}$$

Offensichtlich gilt $f(x^*) = 0$ genau dann, wenn $x^* = \Phi(x^*)$, also wenn x^* Fixpunkt von Φ ist. Nun sind auch andere Definitionen für Φ denkbar, so dass wir eine ganze Klasse von Verfahren erhalten.

Die Konvergenz von Fixpunkt–Iterationen wird durch den Banach'schen Fixpunktsatz geklärt. Wir formulieren und beweisen diesen hier in einem etwas allgemeineren Rahmen als er aus der Analysis bekannt sein dürfte.

Satz 5.6.1 ***(Banach'scher Fixpunktsatz)***
X sei ein linear normierter Raum mit Norm $\|\cdot\|$, $E \subseteq X$ sei eine vollständige Teilmenge von X. Die Abbildung $\Phi : X \to X$ erfülle folgende Bedingungen:

(i) ***Selbstabbildung:*** *$\Phi(E) \subseteq E$, also $\Phi : E \to E$.*

(ii) ***Kontraktion:*** *Es gelte $\|\Phi(x) - \Phi(y)\| \leq L\|x - y\| \quad \forall x, y \in E$ (Lipschitz–Stetigkeit) mit einer Konstanten $L < 1$.*

Dann gilt:

(a) Es existiert genau ein Fixpunkt x^* von Φ in E.

(b) Die Iteration

$$x_{n+1} := \Phi(x_n), \quad n = 0, 1, 2, \ldots \tag{5.9}$$

konvergiert für jedes $x_0 \in E$ gegen den Fixpunkt x^*.

(c) A-priori-Fehlerabschätzung:

$$\|x_n - x^*\| \leq \frac{L^n}{1 - L}\|x_1 - x_0\|. \tag{5.10}$$

(d) A-posteriori-Fehlerabschätzung:

$$\|x_n - x^*\| \leq \frac{L^n}{1 - L}\|x_n - x_{n-1}\|. \tag{5.11}$$

Beweis:

1) Zeige, dass $\{x_n\}_{n=0}^{\infty}$ eine Cauchy–Folge ist: Mit einer Teleskopsumme und der Dreiecks–Ungleichung gilt

$$\begin{aligned} \|x_{n+m} - x_n\| &= \|x_{n+m} - x_{n+m-1} + x_{n+m-1} - + \ldots - x_{n+1} + x_{n+1} - x_n\| \\ &\leq \|x_{n+m} - x_{n+m-1}\| + \ldots + \|x_{n+1} - x_n\| \end{aligned}$$

sowie aufgrund der Kontraktivität:

$$\begin{aligned} \|x_{n+1} - x_n\| &= \|\Phi(x_n) - \Phi(x_{n-1})\| \\ &\leq L\|x_n - x_{n-1}\| \leq \ldots \leq L^n\|x_1 - x_0\|, \end{aligned} \tag{5.12}$$

also wegen $L < 1$ und der (endlichen) geometrischen Reihe

$$\begin{aligned}\|x_{n+m} - x_n\| &\leq (L^{n+m-1} + \ldots + L^n)\|x_1 - x_0\| \\ &= L^n \frac{1-L^n}{1-L}\|x_1 - x_0\| \\ &\leq \frac{L^n}{1-L}\|x_1 - x_0\| \overset{n\to\infty}{\longrightarrow} 0.\end{aligned}$$

Also ist $\{x_n\}_{n=0}^{\infty}$ eine Cauchy–Folge. Da E vollständig ist, existiert ein Grenzwert x^*, d.h.

$$\|x_n - x^*\| \overset{n\to\infty}{\longrightarrow} 0.$$

2) Zeige, dass x^* ein Fixpunkt ist: Mit der Dreiecks–Ungleichung gilt

$$\begin{aligned}\|x^* - \Phi(x^*)\| &= \|x^* - x_n + x_n - \Phi(x^*)\| \\ &\leq \|x^* - x_n\| + \|\Phi(x_{n-1}) - \Phi(x^*)\| \\ &\leq \underbrace{\|x^* - x_n\|}_{\overset{n\to\infty}{\longrightarrow} 0} + L\underbrace{\|x_{n-1} - x^*\|}_{\overset{n\to\infty}{\longrightarrow} 0} \overset{n\to\infty}{\longrightarrow} 0,\end{aligned}$$

also $x^* = \Phi(x^*)$, da $n \in \mathbb{N}$ beliebig war.

3) **Eindeutigkeit:** Seien x^* und x^{**} zwei Fixpunkte, d.h.

$$x^* = \Phi(x^*) \text{ und } x^{**} = \Phi(x^{**}),$$

Dann gilt für $x^* \neq x^{**}$

$$\begin{aligned}\|\Phi(x^*) - \Phi(x^{**})\| &\leq L\|x^* - x^{**}\| \\ &= L\|\Phi(x^*) - \Phi(x^{**})\| \\ &< \|\Phi(x^*) - \Phi(x^{**})\|,\end{aligned}$$

da $L < 1$. Dies ist ein Widerspruch zu $x^* \neq x^{**}$, also folgt $x^* = x^{**}$.

4) **Fehlerabschätzung:** Für jedes $m > 0$ gilt wie in 1)

$$\begin{aligned}\|x_n - x^*\| &\leq \|x_n - x_{n+1}\| + \|x_{n+1} - x_{n+2}\| + \ldots + \|x_{n+m} - x^*\| \\ &= \|\Phi(x_{n-1}) - \Phi(x_n)\| + \|\Phi(x_n) - \Phi(x_{n+1})\| \\ &\quad + \ldots + \|\Phi(x_{n+m-1}) - \Phi(x^*)\| \\ &\leq (L + L^2 + \ldots + L^m)\|x_{n-1} - x_n\| + L\|x_{n+m-1} - x^*\| \\ &= L\Bigg(\underbrace{\frac{1-L^m}{1-L}}_{\overset{m\to\infty}{\longrightarrow} \frac{1}{1-L}}\|x_{n-1} - x_n\| + \underbrace{\|x_{n+m-1} - x^*\|}_{\overset{m\to\infty}{\longrightarrow} 0}\Bigg)\end{aligned}$$

also (d) und mit (5.12) folgt dann (c). □

Bemerkung 5.6.2 *(a) Für $X = \mathbb{R}$, $\|\cdot\| \equiv |\cdot|$ (Betrag) und $E = [a, b]$ oder $E = \mathbb{R}$ erhält man den Satz für skalare Gleichungen.*

(b) Der Satz sagt auch aus, wie viele Iterationen man machen muss, um eine gewünschte Genauigkeit zu erzielen.
Ziel: *$\|x_n - x^*\| \leq \varepsilon$ mit einer Toleranz $\varepsilon > 0$. Wegen (c) ist dies erfüllt, falls $\frac{L^n}{1-L}\|x_1 - x_0\| \leq \varepsilon$, also wegen $L < 1$*

$$L^{-n} \geq \frac{\|x_1 - x_0\|}{\varepsilon(1-L)},$$

und damit

$$n \geq \log\left(\frac{\|x_1 - x_0\|}{\varepsilon(1-L)}\right) \Big/ \log(L^{-1}). \tag{5.13}$$

(c) Man beachte, dass der Satz insbesondere auch für unendlich-dimensionale Räume, wie z.B. Funktionenräume gilt.

Folgerung 5.6.3 *Sei $\Phi \in C^1(\mathbb{R})$ mit $\Phi : [a,b] \to [a,b]$ mit*

$$\max_{x\in[a,b]} |\Phi'(x)| =: L < 1, \tag{5.14}$$

dann besitzt Φ genau einen Fixpunkt und Satz 5.6.1 gilt für $\|\cdot\| \equiv |\cdot|$.

Beweis: Mit dem Mittelwertsatz gilt

$$|\Phi(x) - \Phi(y)| = |\Phi'(\xi)(x-y)| \leq |x-y| \max_{\xi\in(a,b)} |\Phi'(\xi)| = L|x-y|,$$

also ist Φ eine Kontraktion. □

Folgerung 5.6.4 *Sei $\Omega \subseteq \mathbb{R}^n$ abgeschlossen und konvex, $\Phi : \Omega \to \mathbb{R}^n$ stetig differenzierbar mit $\Phi : \Omega \to \Omega$. Falls für eine beliebige Vektornorm $\|\cdot\|$ auf $\mathbb{R}^n$ und die zugehörige Matrixnorm*

$$\max_{x\in\Omega} \|\Phi'(x)\| = L < 1 \tag{5.15}$$

gilt (Φ' ist die Jacobi–Matrix von Φ), dann gelten die Aussagen von Satz 5.6.1.

Beispiel 5.6.5 Betrachte das System

$$6x = \cos x + 2y, \quad 8y = xy^2 + \sin y \tag{5.16}$$

auf $\Omega = [0,1] \times [0,1]$. Gesucht sind also $x, y \in [0,1]$ mit (5.16). Definiere also $\Phi : \mathbb{R}^2 \to \mathbb{R}^2$ durch

$$\Phi(x,y) := \begin{pmatrix} \frac{1}{6}\cos x + \frac{1}{3}y \\ \frac{1}{8}xy^2 + \frac{1}{8}\sin y \end{pmatrix}.$$

Für diese Funktion untersuchen wir die Voraussetzungen des Banach'schen Fixpunktsatzes.

- Wegen $0 \leq \cos x \leq 1$, $0 \leq \sin x \leq 1$ $\forall x, y \in [0,1]$ $(1 < \frac{\pi}{2})$ gilt also

$$\begin{aligned} 0 &\leq \tfrac{1}{6}\cos x + \tfrac{1}{3}y &&\leq \tfrac{1}{6} + \tfrac{1}{3} &&= \tfrac{1}{2} < 1 \\ 0 &\leq \tfrac{1}{8}xy^2 + \tfrac{1}{8}\sin y &&\leq \tfrac{1}{8} + \tfrac{1}{8} &&= \tfrac{1}{4} < 1 \end{aligned}$$

  Also gilt $\Phi : \Omega \to \Omega$, Φ ist also eine Selbstabbildung.

- Wähle $\|\cdot\|_\infty$ (die induzierte Matrixnorm ist die Zeilensummennorm). Wegen

$$\Phi'(x,y) = \begin{pmatrix} -\frac{1}{6}\sin x & \frac{1}{3} \\ \frac{1}{8}y^2 & \frac{1}{4} + \frac{1}{8}\cos y \end{pmatrix}$$

  gilt

$$\|\Phi'\|_\infty = \max\left\{ \underbrace{\frac{1}{6}|\sin x| + \frac{1}{3}}_{\leq\frac{1}{2}}, \underbrace{\frac{1}{8}y^2 + \frac{1}{4}xy + \frac{1}{8}\cos y}_{\leq\frac{1}{2}} \right\} \leq \frac{1}{2} =: L.$$

- Wähle z.B. $(x_0, y_0) = (0, 0)$. Angenommen, wir wollten die Lösung bis auf einen Fehler von $\varepsilon = 0.1$ bestimmen: $(x_1, y_1) = \Phi(x_0, y_0) = (\frac{1}{6}, 0)$

$$\begin{aligned}(x_2, y_2) &= \Phi(x_1, y_1) = \Phi\left(\frac{1}{6}, 0\right) \\ &= \left(\frac{1}{6}\cos\left(\frac{1}{6}\right), 0\right) \doteq (0.164, 0).\end{aligned}$$

Mit der Abschätzung (5.13) gilt mit $\varepsilon = 0.1$, $L = \frac{1}{2}$ und $\|(x_1, y_1) - (x_0, y_0)\|_\infty = \frac{1}{6}$

$$n \geq \log\Bigg(\underbrace{\frac{\frac{1}{6}}{\frac{1}{10}\cdot\frac{1}{2}}}_{=\frac{20}{6}=\frac{10}{3}}\Bigg) / \log(2) = \frac{\log 10 - \log 3}{\log 2} \doteq 1,74,$$

also reichen obige 2 Schritte aus, um die gewünschte Genauigkeit zu erzielen.

Beispiel 5.6.6 Die Eigenschaften von Φ (vor allem die Größe der Konstanten L) beeinflussen die Iteration maßgebich. Daher ist u.U. eine Umformung sinnvoll. Beispiel

$$f(x) := 2x - \tan x \qquad \begin{aligned}\Phi_1(x) &:= \tfrac{1}{2}\tan x \\ \Phi_2(x) &:= \arctan(2x)\end{aligned}$$

Beachte: Φ_1 ist weder Selbstabbildung auf $\Omega = [0, \frac{\pi}{2}]$ noch Kontraktion! Man untersuche dies für Φ_2.

Für Fixpunkt-Iterationen kann man ein einfach zu handhabendes Kriterium herleiten, um die Konvergenzordnung zu bestimmen.

Lemma 5.6.7 *Sei $\Phi : \mathbb{R} \to \mathbb{R}$ $(p+1)$-mal stetig differenzierbar in einer Umgebung von $x^* = \Phi(x^*)$ und es gelte*

$$\Phi^{(j)}(x^*) = 0, \quad j = 1, \ldots, p-1, \quad \Phi^{(p)}(x^*) \neq 0,$$

*dann konvergiert $x_{k+1} := \Phi(x_k)$ **lokal** genau von der Ordnung p.*

Bemerkung 5.6.8 *„Lokal" heißt wiederum für alle Startwerte $x_0 \in \mathbb{R}$, die nahe genug bei x^* liegen!*

Beweis: Für $p = 1$ folgt die Behauptung aus Satz 5.6.1 (Banach'scher Fixpunktsatz).
Sei also $p > 1$: Wir entwickeln $\Phi(x)$ um x^* und erhalten mit einem $\xi_k \in (x_k, x^*)$

$$\begin{aligned}x_{k+1} &= \Phi(x_k) \\ &= \underbrace{\Phi(x^*)}_{=x^*} + \sum_{j=1}^{p} \frac{(x_k - x^*)^j}{j!} \underbrace{\Phi^{(j)}(x^*)}_{=0,\ j=1,\ldots,p-1} + \frac{(x_k - x^*)^{p+1}}{(p+1)!}\Phi^{(p+1)}(\xi_k)\end{aligned}$$

also mit der Dreiecksungleichung

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} \leq \frac{1}{p!}|\Phi^{(p)}(x^*)| + \underbrace{\frac{|x_k - x^*|}{(p+1)!}|\Phi^{(p+1)}(\xi_k)|}_{\substack{k\to\infty \\ \longrightarrow 0}},$$

somit folgt die Behauptung. □

5.7 Newton–Verfahren für Systeme

Wir wollen nun das Nullstellenproblem $f(x) = 0$ für Funktionen

$$f : \mathbb{R}^n \to \mathbb{R}^n (n > 1),\ f \in C^2(\mathbb{R}^n) \tag{5.17}$$

lösen. Man beachte, dass hier

$$f'(x) = \left(\frac{\partial}{\partial x_j} f_i(x) \right)_{i,j=1}^{n} \in \mathbb{R}^{n \times n} \tag{5.18}$$

eine **Matrix** ist, also macht

$$\frac{f(x)}{f'(x)}$$

keinen Sinn! Mit dem Satz von Taylor gilt

$$f(x) = f(x^k) + f'(x^k)(x - x^k) + \mathcal{O}(\|x - x^k\|_2^2). \tag{5.19}$$

Wie in Abschnitt 5.4 bestimmt man also anstelle der Nullstelle von f die Nullstelle x^{k+1} des Taylorpolynoms ersten Grades

$$T(x) = f(x^k) + f'(x^k)(x - x^k). \tag{5.20}$$

Falls die Matrix f' in x^k regulär (invertierbar) ist, folgt $x^{k+1} = x^k - (f'(x^k))^{-1} f(x^k)$. Formal müssten wir also die inverse Matrix $(f'(x^k))^{-1}$ berechnen, was natürlich numerisch völlig unbrauchbar ist. Man kann dies aber leicht umgehen.

Algorithmus 5.7.1 ***(Newton–Iteration)***
Sei $x^0 \in \mathbb{R}^n$ ein "geeigneter" Starwert. Für $k = 0, 1, 2, \ldots$:

1.) Bestimme $s^k \in \mathbb{R}^n$ durch

$$f'(x^k) s^k = -f(x^k) \tag{5.21}$$

2.) ***Newton–Korrektur****: $x^{k+1} = x^k + s^k$*

Bemerkung 5.7.1 *In 1.) hat man also ein lineares Gleichungssystem zu lösen! Je nach Größe von n macht man dies iterativ oder mittels LR/QR.*

Es stellt sich natürlich die Frage, ob man die lokal quadratische Konvergenz des Newton-Verfahrens für skalier Gleichungen auch bei Systemen erhält. Die Antwort ist "ja", falls folgende Voraussetzung erfüllt ist.

Voraussetzung 5.7.2 *Sei $\Omega \subset \mathbb{R}^n$ offen und konvex, $f : \Omega \to \mathbb{R}^n$, $f \in C^1(\Omega)$, $f'(x)$ regulär für alle $x \in \Omega$ und es gelte*

$$\|(f'(x))^{-1}\| \leq \beta \quad \forall x \in \Omega, \tag{5.22}$$

Weiterhin sei f' auf Ω Lipschitz-stetig mit Konstante γ, d.h.,

$$\|f'(x) - f'(y)\| \leq \gamma \|x - y\| \quad \forall x, y \in \Omega \tag{5.23}$$

und es existiere $x^* \in \Omega$ mit $f(x^*) = 0$.

Lemma 5.7.3 *Unter Voraussetzung 5.7.2 gilt*

$$\|f(x) - f(y) - f'(y)(x-y)\| \leq \frac{\gamma}{2}\|x-y\|^2 \quad \forall x, y \in \Omega. \tag{5.24}$$

Beweis: Sei $\varphi(t) := f(y + t(x-y)), t \in [0,1]$. Dann gilt $\varphi \in C^1([0,1]) \quad \forall x, y \in \Omega$ und mit der Kettenregel gilt

$$\varphi'(t) = f'(y + t(x-y))(x-y).$$

Dann gilt

$$\begin{aligned} \|\varphi'(t) - \varphi'(0)\| &= \|[f'(y + t(x-y)) - f'(y)](x-y)\| \\ &\leq \|f'(y + t(x-y)) - f'(y)\| \cdot \|(x-y)\| \\ &\overset{(5.23)}{\leq} \gamma \cdot t\|x-y\|^2. \end{aligned}$$

Auf der anderen Seite gilt:

$$f(x) - f(y) - f'(y)(x-y) = \varphi(1) - \varphi(0) - \varphi'(0) = \int_0^1 (\varphi'(t) - \varphi'(0))\, dt,$$

also

$$\begin{aligned} \|f(x) - f(y) - f'(y)(x-y)\| &\leq \int_0^1 \|\varphi'(t) - \varphi'(0)\|\, dt \\ &\leq \gamma\|x-y\|^2 \underbrace{\int_0^1 t\, dt}_{=1/2} = \frac{\gamma}{2}\|x-y\|^2 \end{aligned}$$

und damit die Behauptung. □

Satz 5.7.4 *Zusätzlich zu Voraussetzung 5.7.2 sei $x^0 \in \Omega$ ein Starwert mit*

$$\|x^* - x^0\| < \omega \tag{5.25}$$

und

$$\frac{1}{2}(\beta\gamma\omega) < 1. \tag{5.26}$$

Dann bleibt die durch das Newton–Verfahren definierte Folge $\{x^k\}_{k=0}^{\infty}$ innerhalb der Kugel

$$K_\omega(x^*) = \{x \in \mathbb{R}^n : \|x^* - x\| < \omega\}$$

*und konvergiert (mindestens) **quadratisch** gegen x^*.*

Beweis: Zunächst gilt:

$$\begin{aligned} x^{k+1} - x^* &= x^k - x^* - (f'(x^k))^{-1}(f(x^k) - \overbrace{f(x^*)}^{=0}) \\ &= (f'(x^k))^{-1}\{f(x^k) - f(x^*) - f'(x^k)(x^k - x^*)\}. \end{aligned}$$

Mit Lemma 5.7.3 folgt dann

$$\begin{aligned} \|x^{k+1} - x^*\| &\leq \underbrace{\|(f'(x^k))^{-1}\|}_{\overset{(5.22)}{\leq} \beta} \underbrace{\|f(x^k) - f(x^*) - f'(x^k)(x^k - x^*)\|}_{\overset{(5.24)}{\leq} \frac{\gamma}{2}\|x^k - x^*\|^2} \\ &= \frac{\beta\gamma}{2}\|x^k - x^*\|^2, \end{aligned}$$

Wir erhalten also quadratische Konvergenz.

Zeige noch $\{x^k\}_{k=0}^{\infty} \subset K_\omega$: für $k = 0$ ist dies gerade (5.25). Weiter induktiv:

$$\|x^{k+1} - x^*\| \leq \frac{\beta\gamma}{2}\|x^k - x^*\|^2 \overset{(I.A.)}{<} \frac{\beta\gamma}{2}\omega^2 \overset{(5.26)}{<} \omega.$$

□

Bemerkung 5.7.5 *(a) Falls $f \in C^2(\Omega)$, dann gilt (5.23) — wir haben also etwas weniger Regularität gefordert.*

*(b) Die Konstanten β, γ sind durch das Problem gegeben. Dabei bedeutet (5.26), dass man **gute Startwerte** benötigt. Dies sichert dann die **lokale** quadratische Konvergenz.*

(c) Unter obigen Voraussetzungen kann man zeigen, dass x^* die einzige Nullstelle in $K_{2/\beta\gamma}(x^*)$ ist.

Bemerkung 5.7.6 (Konvergenztest) *Als Näherung an den Fehler $\|x - x^{(k)}\|$ verwenden wir den Term $\|J^{-1}(x^{(k)})f(x^{(k)})\|$ $(= \|J^{-1}(x^{(k)})(f(x) - f(x^{(k)}))\|)$. Da man erwartet, dass der Fehler monoton fällt, d.h. $\|x - x^{(k+1)}\| \leq \|x - x^{(k)}\|$ gilt, testet man dieses für jedes k durch den* **natürlichen Monotonietest**, *d.h.*

$$\|J^{-1}(x^{(k)})f(x^{(k+1)})\| \leq \overline{\theta}\,\|J^{-1}(x^{(k)})f(x^{(k)})\| \quad \text{sei erfüllt für ein } \overline{\theta} < 1. \tag{5.27}$$

Im Newton-Verfahren berechnen wir zu $x^{(k)}$,

$$J(x^{(k)})s^{(k)} = -f(x^{(k)}) \text{ und } x^{(k+1)} = x^{(k)} + s^{(k)}.$$

Somit ist der Ausdruck $J^{-1}(x^{(k)})f(x^{(k)})$ auf der rechten Seite in (5.27) gleich dem Negativen der Newton-Korrektur $s^{(k)}$, die sowieso berechnet werden muss. Zusätzlich muss nur $\overline{s}^{(k)}$, definiert durch

$$J(x^{(k)})\overline{s}^{(k)} = f(x^{(k+1)})\,,$$

bestimmt werden. Theoretische Untersuchungen und numerische Experimente liefern $\overline{\theta} = 1/2$ als eine gute Wahl. Falls der **natürliche Monotonietest**, *d.h.*

$$\|\overline{s}^{(k)}\| \leq \frac{1}{2}\|s^{(k)}\|$$

*für ein k verletzt ist, so ist das **Newton**-Verfahren abzubrechen und es bleibt nichts Anderes übrig, als einen (hoffentlich) besseren Startwert zu finden.*

Bemerkung 5.7.7 (Dämpfung des Newtonverfahrens) *Eine Möglichkeit die Konvergenz des Newton-Verfahrens zu retten, ist häufig eine Dämpfung in der Form*

$$x^{(k+1)} = x^{(k)} - \lambda_k s^{(k)}, \quad \text{für } \lambda_k \in (0, 1]\,.$$

Für eine einfache Dämpfungsstrategie können wir den Dämpfungsparameter λ_k derart wählen, so dass der natürliche Monotonietest für $\overline{\theta} = 1 - \lambda_k/2$ erfüllt ist, d.h.

$$\|F'(x^{(k)})^{-1}F(x^{(k)} + \lambda_k s^{(k)})\| \leq \left(1 - \frac{\lambda_k}{2}\right)\|F'(x^{(k)})^{-1}F(x^{(k)})\|\,.$$

Dabei wählen wir λ_k aus einer endlichen Folge $\{1, \frac{1}{2}, \frac{1}{4}, \ldots, \lambda_{\min}\}$ und brechen ggf. das Verfahren ab, falls $\lambda_k < \lambda_{\min}$ notwendig wäre. War λ_k erfolgreich, so zeigt die Praxis, dass es effizienter ist, mit $\lambda_{k+1} = \min\{1, 2\lambda_k\}$ fortzufahren anstatt wieder mit $\lambda_{k+1} = 1$ anzufangen. War der Monotonietest mit λ_k verletzt, so testet man erneut mit $\lambda_k/2$.

MATLAB-Funktion: Newton.m

```
1 function [u,nit] = newton(u,F,DF,tol,maxit,param)
```

```
Fu = F(u,param);
DFu = DF(u,param);
s = -DFu\Fu;
lam = 1;
tmp = max(tol,tol*norm(s));
nit = 0;
while norm(s) > tmp && nit <= maxit
  nit = nit + 1;
  u_old = u;
  lam = min(1,2*lam);
  for k=1:30
    u = u_old + lam * s; % Daempfung mit Parameter lam
    Fu = F(u,param);
    if norm(DFu\Fu) <= (1-lam/2) * norm(s)
      break              % Abbruch der for-Schleife, falls
    end                  % Konvergenztest erfuellt
    lam = lam/2;         % lam noch zu groß --> halbieren
  end
  DFu = DF(u,param);
  s  = -DFu\Fu;
end
```

Beispiel 5.7.8 (Extremalstellen der Rosenbrock-Funktion) Es gilt für

$$f(x) = \sum_{i=1}^{n-1} \left[(1-x_i)^2 + 100(x_{i+1} - x_i^2)^2\right] \quad (x \in \mathbb{R}^n)$$

aus Beispiel 5.0.4, dass die Jacobi-Matrix $A := A(x) := J_f(x)$ folgende Einträge enthält

$$\begin{aligned} a_{11} &= 2 + 1200x_1^2 - 400x_2 \\ a_{jj} &= 202 + 1200x_j^2 - 400x_{j+1} \quad (j = 2, \ldots, n-1) \\ a_{nn} &= 200 \\ a_{j,j+1} &= a_{j+1,j} = -400x_j \quad (j = 1, \ldots, n-1) \\ a_{jk} &= 0 \quad (|j-k| \geq 2)\,. \end{aligned}$$

5.8 Hinweise zur Durchführung in der Praxis

Wir betrachten nun einige Aspekte für die Realisierung des Newton-Verfahrens für Systeme

Bemerkung 5.8.1 ***Das vereinfachte Newton–Verfahren***

- *(5.21) bedeutet Aufstellung und Lösung eines linearen Gleichungssystems in* ***jedem*** *Schritt.*
- *Idee: Ersetze $f'(x^k)$ durch $f'(x^0)$, berechne und speichere einmal QR- oder LR–Zerlegung. Dadurch geht die quadratische Konvergenz i.A. verloren.*
- *Man kann die Jacobi–Matrix auch alle paar Schritte aufdatieren (vgl. Deuflhard/Hohmann).*

Bemerkung 5.8.2 ***Auswertung der Jacobi–Matrix***

- *Oftmals kann man $f'(x^k)$ nicht exakt berechnen (oder nur mit riesigem Aufwand)*
- *Auswege:*

- *automatische Differenziation (AD), auch algorithmische Differenziation genannt.*
- *numerische Differenziation: Ersetze Ableitungen durch Differenzenquotienten.*

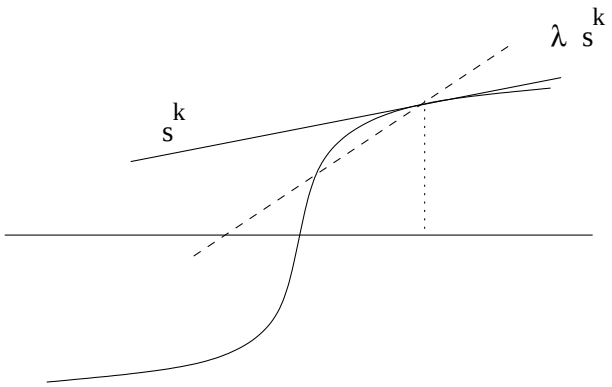
Bemerkung 5.8.3 ***Wahl des Startwertes***

- *Konvergenz oder Divergenz kann sensibel von x^0 abhängen!*
- *Man kann z.B. einige Schritte mit einem anderen Verfahren machen um in den Konvergenzradius zu gelangen.*

Einen Ausweg bedeutet das **Gedämpfte Newton–Verfahren:**

$$x^{k+1} = x^k + \lambda s^k,\ 0 < \lambda \leq 1. \tag{5.28}$$

Hierzu muss ein geeigneter Dämpfungsparameter $\lambda \in (0, 1]$ bestimmt werden!



Algorithmus 5.8.1 ***(Gedämpftes Newton–Verfahren)***
Gegeben: $f = (f_1, \ldots, f_n)^T : \mathbb{R}^n \to \mathbb{R}^n$, $x^0 \in \mathbb{R}^n$ *Startwert*
Gesucht: $x^* \in \mathbb{R}^n$ *mit* $f(x^*) = 0$
Für $k = 0, 1, 2, \ldots$

1.) *Bestimme* $s^k \in \mathbb{R}^n$ *durch* $f'(x^k)s^k = -f(x^k)$. *Setze* $\lambda := 1$

2.) ***Dämfungsschritt:*** *Setze* $x := x^k + \lambda s^k$ *und prüfe*

$$\|f'(x^k)^{-1} f(x)\| \leq \left(1 - \frac{\lambda_k}{2}\right) \|s^k\| \tag{5.29}$$

Falls (5.29) gilt, gehe zu 3.) sonst $\lambda := \frac{\lambda}{2}$ *(falls* $\lambda < \lambda_{\min}$ *ABBRUCH)*

3.) ***Neuer Iterationsschritt***
Falls $k > k_{\max}$ *oder* $\|f(x^{k+1})\| \leq tol$ *STOP, sonst* $k := k + 1$ *und gehe zu 1.)*

6 INTERPOLATION

Häufig sind in der Praxis z.B. durch Marktanalysen, technische Messungen von einer Funktion nur einzelne Punkte bekannt, aber keine analytische Beschreibung der Funktion, um sie an beliebigen Stellen auswerten zu können. Könnte man die diskreten Daten durch eine (eventuell glatte) Kurve verbinden, so wäre es möglich, die unbekannte Funktion an den dazwischenliegenden Stellen zu schätzen. In anderen Fällen will man eine schwierig berechenbare Funktion näherungsweise durch eine einfachere darstellen. Eine Interpolationsfunktion kann diese Anforderung der Einfachheit erfüllen.

Interpolationsaufgabe: Eine gegebene Funktion $f : I \to \mathbb{R}$ sei geeignet zu approximieren unter der Vorgabe, dass f an diskreten (d.h. endlich vielen) Stützstellen die gegebenen Funktionswerte annehmen soll.

Die Interpolation ist somit eine Art der Approximation. Die Approximationsgüte hängt vom Ansatz ab. Um sie zu schätzen, werden Zusatzinformationen (Co-observations) über die Funktion f benötigt. Diese ergeben sich auch bei Unkenntnis von f häufig in natürlicher Weise: Beschränktheit, Stetigkeit oder Differenzierbarkeit lassen sich häufig voraussetzen.

Bei anderen Approximationsverfahren wie z. B. der Ausgleichungsrechnung wird nicht gefordert, dass die Daten exakt wiedergegeben werden; das unterscheidet diese Verfahren von der Interpolation.

Bemerkung 6.0.1 *i) Ist man am gesamten Verlauf von f interessiert, so sollte man eine Interpoliernde If konstruieren, die sich „möglichst wenig" von f unterscheidet.*

ii) Diese **Interpolierende** *If sollte eine leicht berechenbare Funktion sein - hierfür eignen sich* **Polynome**, **trigonometrische Funktionen**, **Exponentialfunktionen** *sowie* **rationale Funktionen**.

6.1 Das allgemeine Interpolationsproblem

Manchmal möchte man nicht nur Daten (also Funktionswerte) interpolieren, sondern z.B. auch Steigungen, Krümmungen oder Richtungsableitungen. Dazu ist folgende allgemeine Problemformulierung hilfreich.

Definition 6.1.1 (Lineares Funktional) *Sei $\mathcal{F}$ ein linearer Raum (z.B. ein Funktionenraum $(C[a,b]$, $C^\infty(\Omega)$, $L_2(\Omega)$, $\ldots)$. Eine Abbildung $\mu : \mathcal{F} \to \mathbb{R}$ heißt* ***lineares Funktional****, falls*

$$\mu(\alpha_1 f_1 + \alpha_2 f_2) = \alpha_1 \mu(f_1) + \alpha_2 \mu(f_2), \quad \alpha_1, \alpha_2 \in \mathbb{R},\ f_1, f_2 \in \mathcal{F}, \tag{6.1}$$

d.h. falls μ reellwertig und linear ist.

Beispiel 6.1.2 Folgende Funktionen sind lineare Funktionale:

(a) $\mathcal{F} = C[a,b],\ x_0 \in [a,b],\ \mu(f) := f(x_0)$, Funktionswerte, Punktauswertungen.

(b) $\mathcal{F} = C^1[a,b],\ x_0 \in [a,b],\ \mu(f) := f'(x_0)$, Steigungen, Ableitungen.

(c) $\mathcal{F} = R[a,b]$,

$$\mu(f) := \int_a^b f(x)\,dx,$$

Integrale.

(d) Mittelwerte, Mediane und andere statistische Größen.

Dann lautet das **allgemeine Interpolationsproblem:** Sei $\mathcal{F}$ ein Funktionenraum und G_n ein $(n+1)$-dimensionaler Teilraum von $\mathcal{F}$. Weiter seien lineare Funktionale $\mu_0, \ldots, \mu_n$ auf G_n gegeben.

$$\left\{ \begin{array}{ll} \text{Zu } f \in \mathcal{F} & \text{finde } g_n \in G_n \text{ mit} \\ & \mu_j(g_n) = \mu_j(f), \quad j = 0, \ldots, n. \end{array} \right. \tag{6.2}$$

Man kann nun die Frage der Existenz und Eindeutigkeit beweisen:

Lemma 6.1.3 (Existenz und Eindeutigkeit der Lösung der allgemeinen Interpolationsaufgabe) *Die allgemeine Interpolationsaufgabe (6.2) hat genau dann für jedes $f \in \mathcal{F}$ eine eindeutige Lösung g_n, wenn*

$$\det[(\mu_i(\varphi_j))_{i,j=0}^n] \neq 0$$

für eine (und damit jede) Basis $\{\varphi_0, \ldots, \varphi_n\}$ von G_n gilt.

Beweis. Sei $g_n \in G_n$, dann existieren eindeutig bestimmte Koeffizienten $\{\alpha_0, \ldots, \alpha_n\}$ mit $g_n = \sum_{j=0}^n \alpha_j \varphi_j$, also bedeutet (6.2) $\mu_i(g_n) = \sum_{j=0}^n \alpha_j \mu_i(\varphi_j) = (G\alpha)_i = \mu_i(f) = b_i$ mit $G = (\mu_i(\varphi_j))_{i,j=0}^n$, $\alpha = (\alpha_j)_{j=0}^n$, $b = (b_j)_{j=0}^n$, d.h. $G\alpha = b$. □

Bemerkung 6.1.4 *Je nach Wahl von $g_n \in G_n$ erhält man unterschiedliche Instanzen der Interpolation:*

- *Ist $G_n = \mathbb{P}_n$ redet man von **Polynominterpolation**.*
- *$G_{2n} := \left\{ F(x) := \frac{P(x)}{Q(x)}, P, Q \in \mathbb{P}_n, Q \neq 0 \right\}$ ergibt die **rationale Interpolation**.*
- *Ist $G_n = \mathcal{S}^k(\mathcal{T})$ so spricht man von **Spline-Interpolation** (Vgl. Kapitel 4).*

6.2 Polynominterpolation

Nach dem aus der Analysis bekannten Approximationssatz von Weierstraßist es möglich, jede stetige Funktion beliebig gut durch Polynome zu approximieren. In diesem Abschnitt sei $G_n = \mathbb{P}_n$, d.h. der Vektorraum der Polynome vom Grad kleiner gleich n.

Wenn man nun in der allgemeinen Interpolationsaufgabe (6.2) nur Funktionswerte f_i interpoliert, spricht man von **Lagrange-Interpolation**. Werden Funktionswerte und Ableitungen interpoliert, spricht man von **Hermite-Interpolation**. Beide Interpolationsaufgaben werden im Folgenden behandelt.

6.2.1 Lagrange-Interpolation

Gegeben seien $(n+1)$ diskrete, paarweise verschiedene **Stützstellen** (auch **Knoten** genannt) $x_0, \ldots, x_n$ und dazugehörige beliebige **Stützwerte** $f_0, \ldots, f_n$.

Gesucht ist nun ein Polynom $P \in \mathbb{P}_n$ vom Grad grad $p \leq n$, d.h.

$$P(x) = a_n x^n + \ldots + a_1 x + a_0, \qquad \text{mit } a_\nu \in \mathbb{R}, \nu = 0, \ldots, n,$$

welches die **Interpolationsbedingungen**

$$P(x_i) = f_i, \qquad i = 0, \ldots, n \tag{6.3}$$

erfüllt.

Sei $\varphi_j \equiv x^j, j = 0, \ldots, n$ die **monomiale Basis** des $\mathbb{P}_n$. Für (6.3) erhalten wir $\mu_i(f) := f(x_i)$ also $\mu_i(\varphi_j) = x_i^j$. Also ist die Matrix $(\mu_i(\varphi_j))_{i,j=0}^n$ die Vandermonde-Matrix, die bekanntermaßen invertierbar ist, falls die x_i paarweise disjunkt sind.

Um eines solches Polynoms $P(x)$ auch explizit zu konstruieren, werden wir zunächst die sogenannte **Lagrange**-Darstellung von Polynomen einführen. Mit dieser Darstellung kann man besonders schön zeigen, dass man immer ein Polynom konstruieren kann, welches die Interpolationsaufgabe erfüllt.

6.2.1.1 Lagrange-Darstellung

Zur Konstruktion der **Lagrange**-Darstellung von Polynomen benötigen wir die folgende Definition:

Definition 6.2.1 (Lagrange-Polynome) *Die Polynome*

$$L_i^{(n)}(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} \in \mathbb{P}_n, \quad i = 0, 1, \ldots, n. \tag{6.4}$$

*zu den $(n+1)$ diskreten, paarweise verschiedenen Knoten $x_0, \ldots, x_n$ werden **Lagrange-Basispolynome** genannt.*

Bemerkung 6.2.2 (Basis-Darstellung) *Als endlich dimensionalen Vektorraum kann man die Elemente des Vektorraums $\mathbb{P}_n$ der Polynome vom Grad kleiner oder gleich n, d.h., die Polynome, eindeutig als Linearkombination endlicher, linear unabhängiger Teilmengen—einer Basis–darstellen. Wie man sich leicht überlegt, ist die Menge der **Lagrange**-Polynome $\{L_i^{(n)}, i = 0, \ldots, n\}$ eine Basis des $\mathbb{P}_n$ (Übungsaufgabe).*

Bemerkung 6.2.3 (Eigenschaft der Lagrange-Polynome) *Per Konstruktion erfüllen die **Lagrange**-Polynome die Knotenbedingung*

$$L_i^{(n)}(x_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k, \\ 0 & \text{für } i \neq k. \end{cases} \tag{6.5}$$

Hieraus folgt, das $L_i^{(n)}, i = 0, \ldots, n$ linear unabhängig sind.

Definition 6.2.4 (Lagrange-Darstellung) *Das Polynom*

$$P(x) := \sum_{i=0}^{n} f_i L_i^{(n)}(x) \tag{6.6}$$

*wird **Lagrange-Darstellung** des Interpolationspolynoms zu den Stützstellen $(x_0, f_0), \ldots, (x_n, f_n)$ genannt.*

Wie angekündigt, liefert die **Lagrange**-Darstellung einen konstruktiven Beweis für die Existenz und Eindeutigkeit eines Polynoms $P(x)$, das die Interpolationsbedingung (6.3) erfüllt.

Satz 6.2.5 (Existenz und Eindeutigkeit der Lagrange-Interpolation) *Zu beliebigen $(n+1)$ Paaren (x_i, f_i), $i = 0, \ldots, n$, mit paarweise verschiedenen Stützstellen $x_0, \ldots, x_n$ existiert genau ein Interpolationspolynom $P \in \mathbb{P}_n$, das (6.3) erfüllt.*

Beweis. **(Existenz:)** Die Existenz des Interpolationspolynoms $P(x)$ zeigen wir auf konstruktive Art. Zu diesem Zweck betrachten wir zu den gegebenen Stützstellen die $(n+1)$ **Lagrange-Polynome** $L_i^{(n)}$, $i = 0, \ldots, n$, aus (6.4). Diese Polynome sind vom echten Grad n und besitzen offensichtlich die Eigenschaft (6.5). Demzufolge besitzt das Polynom

$$P(x) := \sum_{i=0}^{n} f_i L_i^{(n)}(x)$$

wegen (6.5) die geforderten Interpolationseigenschaften:

$$P(x_k) = \sum_{i=0}^{n} f_i L_i^{(n)}(x_k) = \sum_{i=0}^{n} f_i \delta_{ik} = f_k, \quad k = 0, 1, \ldots, n.$$

Ferner ist als Linearkombination von Polynomen vom Grad n der Grad von P kleiner oder gleich n.

(Eindeutigkeit:) Die Eindeutigkeit des Interpolationspolynoms ergibt sich wie folgt: Es seien P und Q zwei Polynome jeweils vom Grad höchstens gleich n, $P, Q \in \mathbb{P}_n$ mit

$$P(x_k) = Q(x_k) = f_k \quad (k = 0, 1, \ldots, n). \tag{6.7}$$

Aus dieser Eigenschaft (6.7) folgt, dass $D(x) := P(x) - Q(x)$ ein Polynom vom Grad kleiner oder gleich n definiert mit den $(n+1)$ paarweise verschiedenen Nullstellen $x_0, \ldots, x_n$. Nach dem Fundamentalsatz der Algebra muss nun aber $D(x) \equiv 0$ gelten, also $P(x) = Q(x)$ sein. □

Bemerkung 6.2.6 (Vor- und Nachteile der Lagrange-Darstellung) *Die **Lagrange**-Darstellung (6.6) ist für praktische Zwecke meist zu rechenaufwendig, sie eignet sich jedoch hervorragend für theoretische Fragestellungen. Insbesondere, müssen die Lagrange-Polynome bei Hinzunahme einer weiteren Stützstelle (x_{n+1}, f_{n+1}) komplett neu berechnet werden.*

Im Laufe dieses Abschnitts werden wir eine weitere Basis Darstellung kennenlernen, welche sich besser zur numerischen Berechnung eignet, als die **Lagrange**-Darstellung, dies ist die so genannte **Newton**-Darstelleung. Zunächst aber zur kanonischen Basis Darstellung.

6.2.1.2 Monomiale Basis Darstellung

Eine alternative Basis zur Darstellung des Interpolationspolynoms mit Lagrange-Polynomen bildet die sogenannte **monomiale Basis** $\{1, \ldots, x^n\}$ von $\mathbb{P}_n$, d.h. wir schreiben P in folgender Koeffizientendarstellung

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n.$$

Bemerkung 6.2.7 (Vandermonde-Matrix) *Die Interpolationsbedingungen $P(x_i) = f_i$, $i = 0, \ldots, n$ lassen sich wie bereits bemerkt als folgendes lineares Gleichungssystem auffassen:*

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}.$$

*In Numerik 1 haben wir diese **Vandermonde**-Matrix schon kennengelernt und an dortiger Stelle auch eine zum **Gauß**-Algorithmus alternative Möglichkeit angegeben um das dazugehörige lineare Gleichungssystem zu lösen (Aufwand für dieses spezielle Verfahren $5n^2/2 + \mathcal{O}(n)$). Das Problem der schlechten Kondition hatten wir auch angesprochen.*

Bemerkung 6.2.8 (Nachteil monomiale Basis Darstellung) *Die Darstellung in monomialer Basis ist numerisch instabil und sollte daher für große n im Allgemeinen nicht verwendet werden.*

6.2.2 Hermite-Interpolation

Um die **Hermite**-Interpolation einzuführen, bei der neben den Funktionswerten $f(x_i)$ auch die Ableitungen gegeben sind, benötigen wir noch einige Notationen. Zunächst sei angemerkt, dass generell Ableitungen verschiedener Ordnung an unterschiedlichen Knoten gegeben sein können.

Notation: Wir definieren die Folge von Stützstellen $\triangle := \{x_j\}_{j=0,\dots,n}$ mit

$$a = x_0 \leq x_1 \leq \cdots \leq x_n = b,$$

wobei Stützstellen auch mehrfach auftreten können. Sind an einer Stelle x_i einerseits der Funktionswert $f(x_i)$ und andererseits die Ableitungen $f'(x_i), \dots, f^{(k)}(x_i)$ gegeben, so soll x_i in obiger Folge $(k+1)$-mal auftreten.

Gleiche Knoten nummerieren wir hierbei mit der Vielfachheit

$$d_i := \max\{j \in \mathbb{N} \mid x_i = x_{i-j}\}$$

von links nach rechts durch; z.B.

x_i	$x_0 = x_1$		$< x_2 = x_3 = x_4$			$< x_5$	$< x_6$
d_i	0	1	0	1	2	0	0

Führen wir nun mit diesen Abkürzungen nachfolgende lineare Abbildung

$$\mu_i : C^n[a,b] \to \mathbb{R}, \quad \mu_i(f) := f^{(d_i)}(x_i), \quad i = 0, \dots, n$$

ein, so lautet die **Aufgabe der Hermite-Interpolation:** Gegeben μ_i $(i = 0, \dots, n)$. Finde $P \in \mathbb{P}_n$ mit

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n. \tag{6.8}$$

Die Lösung $P = P(f|x_0, \dots, x_n) \in \mathbb{P}_n$ von (6.8) heißt **Hermite-Interpolierende**.

Satz 6.2.9 (Existenz und Eindeutigkeit der Hermite-Interpolation) *Zu jeder Funktion $f \in C^n[a,b]$ und jeder monotonen Folge*

$$a = x_0 \leq x_1 \leq \ldots \leq x_n = b$$

von (i.Allg. nicht paarweise verschiedenen) Knoten gibt es genau ein Polynom $P \in \mathbb{P}_n$, sodass gilt:

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n.$$

Beweis. Die Abbildung

$$\mu : \mathbb{P}_n \to \mathbb{R}^{n+1}, \; P \mapsto (\mu_0(P), \dots, \mu_n(P))$$

ist offensichtlich eine lineare Abbildung zwischen den $(n+1)$-dimensionalen reellen Vektorräumen $\mathbb{P}_n$ und $\mathbb{R}^{n+1}$, sodass aus der Injektivität der Abbildung bereits die Surjektivität folgen würde und damit der Satz bewiesen wäre. Somit reicht es die Injektivität der linearen Abbildung zu zeigen.

Da $\mu(P) = 0$ gilt, folgt, dass p mindestens $(n+1)$-Nullstellen inklusive Vielfachheiten besitzt, somit aber das Nullpolynom ist. Da ferner $dim\, \mathbb{P}_n = dim\, \mathbb{R}^{n+1} = n+1$, folgt daraus auch wieder die Existenz. □

Bemerkung 6.2.10 (Eigenschaften der Hermite-Interpolation) *Stimmen alle Knoten überein, d.h. $x_0 = x_1 = \cdots = x_n$, dann entspricht das Interpolationspolynom der abgebrochenen Taylor-Reihe um $x = x_0$:*

$$P(f|x_0, \dots, x_n)(x) := \sum_{j=0}^{n} \frac{(x - x_0)^j}{j!} f^{(j)}(x_0).$$

Die **Hermite**-Interpolation und insbesondere deren effiziente Berechnung sind Thema des nun folgenden Abschnitts. Die dort vorgestellten Schemata zur effizienten numerischen Auswertung sind, wie wir an einem Beispiel sehen werden, allerdings natürlich auch im Fall der **Lagrange**-Interpolation anwendbar.

6.2.3 Auswertung von Polynomen

In diesem Abschnitt werden wir uns unter anderem mit der effizienten Auswertung des so gemannten Interpolationspolynoms beschäftigen:

Definition 6.2.11 (Interpolationspolynom) *Das nach Satz 6.2.5 eindeutig bestimmte Polynom $P \in \mathbb{P}_n$ heißt **Interpolationspolynom** von f zu den paarweise verschiedenen Stützstellen $x_0, \ldots, x_n$ und wird mit*

$$P = P(f|x_0, \ldots, x_n)$$

bezeichnet.

Zur Berechnung kommen je nach Fragestellung zwei unterschiedliche Ansätze zum Einsatz:

- Schema von **Aitken**[1] und **Neville**[2],
- Schema der dividierten Differenzen.

Ist man nur an der Auswertung des Interpolationspolynoms P an einer Stelle x (oder ganz wenigen) interessiert und will das Interpolationspolynom selber nicht explizit ausrechnen so verwendet man das Schema von **Aitken** und **Neville**.

Will man hingegen das Interpolationspolynom an mehreren Stellen auswerten, so wendet man das Schema der dividierten Differenzen an. Hierbei berechnet man die Koeffezienten $a_n = [x_0, \ldots, x_n]f$ des so genannten Newtonschen-Interpolationspolynoms (die dividierten Differenzen) und wendet ein dem Horner-Schema ähnliches Schema an.

Das **Newton**sche-Interpolationspolynom ist hierbei eine Darstellung des Interpolationspolynoms bezüglich der so genannten **Newton**-Basis. Aus dem Blickwinkel der Darstellung des Interpolationspolynoms bzgl. der **Newton**-Basis entspricht das Schema von **Aitken** und **Neville** der Berechnung des Wertes des Interpolationspolynoms an einer gesuchten Stelle ohne die Koeffizienten a_n explizit zu berechnen.

6.2.3.1 Schema von Aitken und Neville

Hier betrachen wir den Fall, dass man nur an der Auswertung des Interpolationspolynoms P an einer Stelle x interessiert ist. Dazu muss man nicht erst P bestimmen, sondern kann $P(x)$ durch rekursive Berechnung effektiver (d.h vor allem effektiver bezüglich des Aufwands) bestimmen. Motiviert wird dies im Folgenden durch das Lemma von **Aitken**.

Lemma 6.2.12 (von Aitken) *Für das Interpolationspolynom $P = P(f|x_0, \ldots, x_n)$ gilt die Rekursionsformel*

$$P(f|x_0, \ldots, x_n)(x) = \frac{(x_0 - x)P(f|x_1, \ldots, x_n)(x) - (x_n - x)P(f|x_0, \ldots, x_{n-1})(x)}{x_0 - x_n}. \quad (6.9)$$

Hierbei gilt also insbesondere $P(f|x_k) = f(x_k)$.

Beweis. Sei $\phi(x)$ definiert als der Term auf der rechten Seite von (6.9). Dann ist $\phi \in \mathbb{P}_n$ und es gilt:

$$\phi(x_i) = \frac{(x_0 - x_i)f(x_i) - (x_n - x_i)f(x_i)}{x_0 - x_n} = f(x_i), \quad i = 1, \ldots, n-1.$$

Ebenso leicht folgt $\phi(x_0) = f(x_0)$ sowie $\phi(x_n) = f(x_n)$ und daher obige Behauptung. □

[1] Aitken, Alexander Craig (1895-1967)
[2] Neville, Eric Harold (1889-1961)

Herleitung des Algorithmus: Wir definieren $f_i := f(x_i)$ für $i = 0, \ldots, n$. Man beachte hierbei

$$P(f|x_i) = f_i, \quad i = 0, \ldots, n.$$

Für festes x vereinfachen wir die Notation weiterhin durch

$$P_{i,k} := P(f|x_{i-k}, \ldots, x_i)(x), \quad i \geq k.$$

Die Rekursion (6.9) schreibt sich nun

$$P_{n,n} = \frac{(x_0 - x)P_{n,n-1} - (x_n - x)P_{n-1,n-1}}{x_0 - x_n},$$

oder allgemeiner für $P_{i,k}$, $i \geq k$:

$$\begin{aligned} P_{i,k} &= \frac{\overbrace{(x_{i-k} - x)}^{=(x_{i-k}-x_i+x_i-x)} \quad P_{i,k-1} - (x_i - x)P_{i-1,k-1}}{x_{i-k} - x_i} \\ &= P_{i,k-1} + \frac{x_i - x}{x_{i-k} - x_i}(P_{i,k-1} - P_{i-1,k-1})\,. \end{aligned}$$

Daraus gewinnen wir den folgenden Algorithmus:

Algorithmus 6.2.1 (Aitken-Neville-Verfahren) *Gegeben seien Stützstellen $x_0, x_1, \ldots, x_n$ und Stützwerte $f_0, f_1, \ldots, f_n$.*

1) Setze $P_{i,0} = f_i$, für i=0,..., n.

2) Berechne $P_{i,k} = P_{i,k-1} + \frac{x-x_i}{x_i-x_{i-k}}(P_{i,k-1} - P_{i-1,k-1})$, $i \geq k$.

Zur praktischen Berechnung eignet sich das **Schema von Neville**: Nach diesem lässt sich $P_{n,n} = P(f|x_0, \ldots, x_n)(x)$, d.h. das Interpolationspolynom an einer festen Stelle x, ausgehend von den Daten $(f_0 \ldots, f_n)$ wie folgt berechnen:

$$\begin{array}{lllllllllll}
f_0 & = & P_{0,0} & & & & & & & & \\
 & & & \searrow & & & & & & & \\
f_1 & = & P_{1,0} & \to & P_{1,1} & & & & & & \\
\vdots & & \vdots & & \vdots & & & & & & \\
\vdots & & \vdots & & & & & & & & \\
 & & & & & \searrow & & & & & \\
\vdots & & \vdots & & \vdots & \to & P_{n-2,n-2} & & & & \\
 & & & & & \searrow & & \searrow & & & \\
f_{n-1} & = & P_{n-1,0} & \to & \quad \ldots & \to & P_{n-1,n-2} & \to & P_{n-1,n-1} & & \\
 & & & \searrow & & \searrow & & \searrow & & \searrow & \\
f_n & = & P_{n,0} & \to & P_{n,1} \ldots & \to & P_{n,n-2} & \to & P_{n,n-1} & \to & P_{n,n}
\end{array}$$

MATLAB-Funktion: AitkenNeville.m

```
1 function value = AitkenNeville(x,fx,x0)
2 % evaluate the Interpolation polynomial given
```

```
% by (x,fx) at the point x0
for k = 2:length(fx)
  for j = length(fx):-1:k
    fx(j) = fx(j) + (x0-x(j))/(x(j)-x(j-k+1)) * (fx(j)-fx(j-1));
  end
end
value = fx(end);
```

MATLAB-Beispiel:

Testen wir das **Aitken-Neville**-Verfahren anhand zweier Beispiele. Zum einen werten wir das Interpolationspolynom zu $f(x) = x^2$ und 3 Stützstellen an der Stelle 2 aus.

```
>> x = linspace(0,1,5);
>> AitkenNeville(x,x.^2,2)
ans =
     4
```

Zum anderen werten wir das Interpolationspolynom zu $f(x) = \sin(x)$ und 5 äquidistanten Stützstellen in $[0,1]$ an der Stelle $\pi/3$ aus. Man beachte, dass $\sin(x) = \sqrt{3}/2$ gilt.

```
>> x = linspace(0,1,5);
>> sqrt(3)/2 - AitkenNeville(x,sin(x),pi/3)
ans =
    4.387286117690792e-005
```

Bemerkung 6.2.13 *Im Gegensatz zur Lagrange-Darstellung lässt sich bei der Berechnung des Interpolationspolynoms an einer festen Stelle mit dem Schema von Aitken-Neville zu $(n+1)$-Paaren ohne Probleme eine weiteres Paar (x_{n+1}, f_{n+1}) hinzu nehmen. Und dies somit zu einer Berechnung des Interpolationspolynoms an einer festen Stelle zu $(n+2)$ Paaren ausweiten.*

6.2.3.2 Newton-Darstellung und dividierte Differenzen

Wir hatten bereits zwei Basis-Darstellungen für $\mathbb{P}_n$ kennengelernt. Allerdings eigenen sich sowohl die **Lagrange**-Darstellung als auch die monomiale Basis Darstellung nicht so gut zur numerischen Berechnung des kompletten Polynoms (also nicht nur der Auswertung an wenigen Stellen). Vor diesem Hintergrund führen wir die **Newton**schen Basispolynome ein, eine Darstellung bzgl. dieser ist für numerische Zwecke besser geeignet.

Das Ziel ist eine Aufdatierungsstrategie bezüglich des Polynomgrades zur Berechnung des kompletten Polynoms. Mit einer Aufdatierungsstrategie kann man dann auch ohne Probleme ein weiteres Paar (x_{n+1}, f_{n+1}) hinzunehmen.

Motivation: Zu gegebenen $(n+1)$ Paaren wollen wir das Interpolationspolynom $P(f|x_0,\dots,x_n) \in \mathbb{P}_n$ somit als eine additive Zerlegung

$$\underbrace{P(f|x_0,\dots,x_n)(x)}_{\in\mathbb{P}_n} = \underbrace{P(f|x_0,\dots,x_{n-1})(x)}_{\in\mathbb{P}_{n-1}} + \underbrace{Q_n(x)}_{\in\mathbb{P}_n}$$

darstellen, mit einem Polynom Q_n vom Grad n, welches von den Stützstellen x_i und einem unbekannten Koeffizienten abhängt. Da für

$$Q_n(x_i) = P(f|x_0, \ldots, x_n)(x_i) - P(f|x_0, \ldots, x_{n-1})(x_i) = 0$$

gilt, muss

$$Q_n(x) = a_n(x - x_0) \cdots (x - x_{n-1})$$

gelten. Für x_n setzt man ein und löst nach a_n auf

$$a_n = \frac{f(x_n) - P(f|x_0, \ldots, x_{n-1})(x_n)}{(x_n - x_0) \cdots (x_n - x_{n-1})}$$

Offenbar ist a_n der führende Koeffizient von $P(f|x_0, \ldots, x_n)$, d.h. der Koeffizient vor x^n.

Wir definieren nun die bereits erwähnte Newton-Basis:

Definition 6.2.14 (Newton-Basis) *Es seien $x_0, \ldots x_{n-1} \in \mathbb{R}$ und*

$$\omega_0 := 1, \quad \omega_i(x) := \prod_{j=0}^{i-1}(x - x_j), \quad \omega_i \in \mathbb{P}_i.$$

*Wir bezeichnen $\{\omega_0, \ldots, \omega_n\}$ als **Newton-Basis** des Polynomraums $\mathbb{P}_n$ mit Basiselementen ω_i.*

Bemerkung 6.2.15 *Man beachte, dass bei der Definition der Newton-Basis weder eine Ordnung der Punkte x_k noch „paarweise verschieden“ vorgeschrieben wurde. Je nach Nummerierung, erhält man somit eine andere Newton-Basis. Dies ist bei der Stabilität der folgenden Verfahren zu berücksichtigen.*

Um nun das Verfahren der dividierten Differenzen herzuleiten, mit dem sich sowohl die **Lagrange**- als auch die **Hermite**-Interpolationsaufgabe effizient lösen lässt, verwenden wir die Darstellung des Interpolationspolynoms in der **Newton**-Basis, d.h. das **Newton-Interpolationspolynom**

$$\begin{aligned} P(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n \prod_{j=0}^{n-1}(x - x_j) \\ &= \sum_{i=0}^{n} a_i \omega_i(x), \end{aligned}$$

wobei sich die unbekannten Koeffizienten $c_0, \ldots, c_n$ prinzipiell aus den Interpolationsbedingungen

$$\begin{array}{rcl} P(x_0) &=& c_0 = f(x_0) \\ P(x_1) &=& c_0 + c_1(x - x_0) = f(x_1) \\ P(x_2) &=& c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_1)(x_2 - x_0) = f(x_2) \\ \vdots & & \vdots \end{array}$$

sukzessive berechnen lassen (dieses LGS hat Linksdreiecksgestalt!):

$$\begin{aligned} P(x_0) = c_0 &\quad\Longrightarrow\quad c_0 = f(x_0), \\ P(x_1) = c_0 + c_1(x_1 - x_0) &\quad\Longrightarrow\quad c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

Die Koeffizienten $c_k, k = 0, \ldots, n$ nennt man **dividierte Differenzen**.

Definition 6.2.16 (n-te dividierte Differenz) *Der führende Koeffizient a_n des Interpolationspolynoms*

$$P(f|x_0,\dots,x_n)(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

von f zu den Knoten $x_0 \le x_1 \le \cdots \le x_n$ heißt ***n-te dividierte Differenz*** *von f an $x_0,\dots,x_n$ und wird mit*

$$[x_0,\dots,x_n]f := a_n$$

bezeichnet.

Bemerkung 6.2.17 *Beim Vergleich des* ***Newton****-Interpolationspolynoms mit dem Interpolationspolynom bzgl. der monomialen Basis*

$$P(x) = \sum_{j=0}^{n} c_i \prod_{i=0}^{j-1}(x - x_i) = \sum_{j=0}^{n} a_j x^j$$

sieht man durch Ausmultiplizieren, dass für die Koeffizienten der höchsten $x-$Potenz $a_n = c_n$ gilt. Damit folgt $a_n = c_n = [x_0,\dots,x_n]f$

Satz 6.2.18 (Newton'sche Interpolationsformel) *Für jede Funktion $f \in C^n(\mathbb{R})$ und Knoten $x_0 \le \cdots \le x_n \in \mathbb{R}$ ist*

$$P(x) = \sum_{i=0}^{n} [x_0,\dots,x_i]f \cdot \omega_i(x)$$

das Interpolationspolynom $P(f|x_0,\dots,x_n)$ von f an $x_0,\dots,x_n$ also $P(f|x_0,\dots,x_n) = P(x)$. Gilt darüber hinaus $f \in C^{n+1}(\mathbb{R})$, so folgt:

$$f(x) = P(x) + [x_0,\dots,x_n,x]f \cdot \omega_{n+1}(x)\,. \tag{6.10}$$

Beweis. Wir zeigen die erste Behauptung durch Induktion nach $n \in \mathbb{N}$. Für $n = 0$ ist die Aussage trivialerweise erfüllt. Sei also im Folgenden $n > 0$ und

$$P_{n-1} := P(f|x_0,\dots,x_{n-1}) = \sum_{i=0}^{n-1} [x_0,\dots,x_i]f \cdot \omega_i$$

das Interpolationspolynom von f an den Stützstellen $x_0,\dots,x_{n-1}$. Damit erhalten wir für $P_n = P(f|x_0,\dots,x_n)$, aufgrund der Definition der dividierten Differenz bzw. von P_n, dass

$$\begin{aligned} P_n(x) &= [x_0,\dots,x_n]f \cdot x^n + a_{n-1}x^{n-1} + \dots + a_0 \\ &= [x_0,\dots,x_n]f \cdot \omega_n(x) + Q_{n-1}(x) \end{aligned}$$

mit einem Polynom $Q_{n-1} \in \mathbb{P}_{n-1}$ gilt. Nun erfüllt aber wegen $\omega_n(x_i) = 0, i = 0,\dots,n-1$

$$Q_{n-1} = P_n - [x_0,\dots,x_n]f \cdot \omega_n$$

offensichtlich die Interpolationsaufgabe für $x_0,\dots,x_{n-1}$, sodass wir erhalten:

$$Q_{n-1} = P_{n-1} = \sum_{i=0}^{n-1} [x_0,\dots,x_i]f \cdot \omega_i\,.$$

Dies beweist aber gerade die Aussage des Satzes. Insbesondere folgt nun, dass

$$P_n + [x_0,\dots,x_n,x]f \cdot \omega_{n+1}$$

die Funktion f an den Knoten $x_0,\dots,x_n$ und x interpoliert und damit (6.10). □

Aus den Eigenschaften der **Hermite**-Interpolation lassen sich sofort folgende Aussagen über die dividierten Differenzen zu f ableiten:

Lemma 6.2.19 (Eigenschaften der dividierten Differenzen) *i) **(Rekursive Berechnung der dividierten Differenzen)** Für $x_i \neq x_k$ gilt die Rekursionsformel*

$$[x_0,\dots,x_n]f = \frac{[x_0,\dots,\widehat{x}_i,\dots,x_n]f - [x_0,\dots,\widehat{x}_k,\dots,x_n]f}{x_k - x_i},$$

wobei $\widehat{\ }$ anzeigt, dass die entsprechende Stützstelle weggelassen wird ('seinen Hut nehmen muss')

$ii)$ ***(Darstellung über abgebrochene Taylor-Reihe)****Für zusammenfallende Knoten* $x_0 = \ldots = x_n$ *gilt*

$$[x_0, \ldots, x_n]f = \frac{f^{(n)}(x_0)}{n!}$$

Beweis. Für das **Hermite**-Interpolationspolynom gilt mit $x_i \neq x_k$:

$$P(f|x_0, \ldots, x_n) = \frac{(x_i - x)\overbrace{P(f|x_0, \ldots, \widehat{x}_k, \ldots, x_n)}^{[x_0,\ldots,\widehat{x}_i,\ldots,x_n]f\, x^{n-1}+\mathbb{P}_{n-2}} - (x_k - x)P(f|x_0, \ldots, \widehat{x}_k, \ldots, x_n)}{x_i - x_k}, \tag{6.11}$$

was sich durch Überprüfen der Interpolationseigenschaft zeigen lässt mittels Einsetzen der Definitionen. Aus der Eindeutigkeit des führenden Koeffizienten folgt aus (6.11) unmittelbar Behauptung $i)$. Stimmen dagegen alle Knoten $x_0, \ldots, x_n$ überein, so ist das Interpolationspolynom

$$P(f|x_0, \ldots, x_n)(x) = \sum_{j=0}^{n} \frac{(x - x_0)^j}{j!} f^{(j)}(x_0) =: P(x),$$

wie man durch Einsetzen in μ_i (vgl. (6.8) oben) leicht einsieht. Daraus folgt, dass der führende Koeffizient $f^{(n)}(x_0)/n!$ ist. Somit gilt auch $ii)$. □

Bemerkung 6.2.20 *Aussage $i)$ in Lemma 6.2.19 erklärt die Bezeichnung dividierte Differenzen.*

Die Auswertung der Rekursionsformel (6.11) erfolgt am zweckmäßigsten im **Schema der dividierten Differenzen** unter Verwendung der Startwerte $[x_i]f = f(x_i)$ für paarweise verschiedene Knoten.

$$\begin{array}{llllll}
x_0 & [x_0]f & & & & \\
x_1 & [x_1]f & [x_0, x_1]f & & & \\
x_2 & [x_2]f & [x_1, x_2]f & [x_0, x_1, x_2]f & & \\
x_3 & [x_3]f & [x_2, x_3]f & [x_1, x_2, x_3]f & [x_0, x_1, x_2, x_3]f & \\
x_4 & [x_4]f & [x_3, x_4]f & [x_2, x_3, x_4]f & [x_1, x_2, x_3, x_4]f & [x_0, \ldots, x_4]f
\end{array}$$

Die gesuchten Koeffizienten a_k des **Newton**-Interpolationspolynoms findet man im obigen Schema der dividierten Differenzen in der oberen Diagonalen.

Nachfolgend werden wir das Schema der dividierten Differenzen anhand zweier Beispiele illustrieren. Zum einen für die klassische **Lagrange**-Interpolation bei der Knoten und Funktionswerte gegeben sind und zum anderen für die **Hermite**-Interpolation.

Beispiel 6.2.21 (Lagrange-Interpolation via dem Schema der dividierten Differenzen) Wenn wir das Schema auf gegebene Daten $(x_i, f_i), i = 0, \ldots, 3$ anwenden, so erhalten wir

$$\begin{array}{lcl}
x_0 = 0 & : & f_0 = 1 \\
 & & \searrow \\
x_1 = 3/2 & : & f_1 = 2 \quad \to \quad 2/3 \\
 & & \searrow \qquad\qquad \searrow \\
x_2 = 5/2 & : & f_2 = 2 \quad \to \quad 0 \quad \to \quad -4/15 \\
 & & \searrow \qquad\qquad \searrow \qquad\qquad \searrow \\
x_3 = 9/2 & : & f_3 = 1 \quad \to \quad -1/2 \quad \to \quad -1/6 \quad \to \quad 1/45
\end{array}$$

und das **Newtonsche** Interpolationspolynom lautet demnach

$$P(x) = 1 + \frac{2}{3}(x - 0) - \frac{4}{15}(x - 0)(x - \frac{3}{2}) + \frac{1}{45}(x - 0)(x - \frac{3}{2})(x - \frac{5}{2})$$

MATLAB-Funktionen: NewtonInterpolation.m und EvalNewtonPoly.m

```
function fx = NewtonInterpolation(x,fx)
for k = 2:length(fx)
  for j = length(fx):-1:k
    fx(j) = (fx(j) - fx(j-1))/(x(j)-x(j-k+1));
  end
end
```

```
function value = HornerNewton(a,x0,x)
% evaluate Newton polynom
% p(x0) = a(1) + a(2)*(x0-x(1)) + a(3)*(x0-x(1))*(x0-x(2)) + ...
%       = a(1) + ( (x0-x(1)) * ( a(2) + (x0-x(2)) * ( a(3) ....
value = a(end);
for k=length(a)-1:-1:1
  value = a(k) + value.*(x0-x(k));
end
```

MATLAB-Beispiel:

Testen wir das Differenzen-Verfahren nochmals an den beiden Beispielen aus Bsp. 6.2.3.1. Zum einen werten wir das Interpolationspolynom zu $f(x) = x^2$ und 3 Stützstellen an der Stelle 2 aus.

```
>> x=linspace(0,2,3);
>> a = NewtonInterpolation(x,x.^2)
a =
     0     1     1
>> HornerNewton(a,2,x)
ans =
     4
```

Zum anderen werten das Interpolationspolynom zu $f(x) = \sin(x)$ und 5 äquidistanten Stützstellen in $[0, 1]$ an der Stelle $\pi/3$ aus.

```
>> x=linspace(0,1,5);
>> a = NewtonInterpolation(x,sin(x));
>> sqrt(3)/2-HornerNewton(a,pi/3,x)
ans =
    4.387286117690792e-005
```

Beispiel 6.2.22 (Hermite-Interpolation via dem Schema der dividierten Differenzen)
Betrachten wir nun noch abschließend das Schema für die nichttriviale **Hermite**-Interpolationsaufgabe (d.h. auch Ableitungen werden interpoliert). Unter Beachtung von Lemma 6.2.19, insbesondere (6.11) ergibt sich:

$$
\begin{array}{lllllll}
x_0: & [x_0]f & & & & & \\
 & \searrow & & & & & \\
x_0: & [x_0]f & \to & [x_0,x_0]f = f'(x_0) & & & \\
 & \searrow & & \searrow & & & \\
x_0: & [x_0]f & \to & [x_0,x_0]f = f'(x_0) & \to & [x_0,x_0,x_0]f = \frac{f''}{2}(x_0) & \\
 & \searrow & & \searrow & & \searrow & \\
x_1: & [x_1]f & \to & [x_0,x_1]f & \to & [x_0,x_0,x_1]f & \to \quad [x_0,x_0,x_0,x_1]f
\end{array}
$$

Das resultierende Polynom $P(x)$ mit

$$P(x) = f(x_0) + (x - x_0)\Big(f'(x_0) + (x - x_0)\big(f''(x_0) + (x - x_0)[x_0, x_0, x_0x_1]f\big)\Big)$$

erfüllt dann die Interpolationsaufgaben

$$P(x_0) = f(x_0), P'(x_0) = f'(x_0), P''(x_0) = f''(x_0) \text{ und } P(x_1) = f(x_1).$$

Für den speziellen Fall, dass an allen Knoten $x_0, \ldots, x_n$ sowohl f als auch f' interpoliert werden sollen, erhält man die folgende Darstellung des Interpolationspolynoms $P(x)$.

Satz 6.2.23 (Darstellung des Interpolationspolynoms) *Es sei $\omega(x) = (x - x_0) \cdot \cdots \cdot (x - x_n)$ und $L_k^{(n)}(x)$ seien die **Lagrange**-Polynome zu den Knoten $x_0, \ldots, x_n$. Dann hat*

$$P(x) = \sum_{k=1}^{n} f(x_k)\left(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\right) L_k^{(n)} 2(x) + \sum_{k=1}^{n} f'(x_k)(x - x_k) L_k^2(x)$$

die Interpolationseigenschaften

$$P(x_k) = f(x_k) \text{ und } P'(x_k) = f'(x_k), \quad k = 0, \ldots, n.$$

Beweis. Es sei x_ℓ einer der Knoten $x_0, \ldots, x_n$, dann folgt sofort aus der Interpolationseigenschaft der **Lagrange-Polynome** $P(x_\ell) = f(x_\ell)$, da

$$\omega'(x) = \sum_{i=0}^{n} \prod_{\substack{k=0 \\ k \neq i}}^{n} (x - x_k) \text{ und somit } \omega'(x_\ell) = \prod_{\substack{k=0 \\ k \neq \ell}}^{n} (x_\ell - x_k) \neq 0$$

gilt. Für die Ableitung von P ergibt sich

$$\begin{aligned} P'(x) \quad = \quad & \sum_{k=1}^{n} f(x_k)\Big[\Big(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\Big) 2(L_k^{(n)}(x))' - \frac{\omega''(x_k)}{\omega'(x_k)} L_k^{(n)}(x)\Big] L_k^{(n)}(x) \\ & + \sum_{k=1}^{n} f'(x_k)\Big[(x - x_k) 2(L_k^{(n)}(x))' + L_k^{(n)}(x)\Big] L_k^{(n)}(x), \end{aligned}$$

sodass wir nun Folgendes erhalten:

$$P'(x_\ell) = f(x_\ell)\Big(2(L_\ell^{(n)}(x_\ell))' - \frac{\omega''(x_\ell)}{\omega'(x_\ell)}\Big) + f'(x_\ell).$$

Nutzt man aus, dass $L_\ell^{(n)}(x) = \frac{\omega(x)}{(x - x_\ell)\omega'(x_\ell)}$ gilt (siehe Hausübung), so folgt aus

$$\omega(x) = L_\ell^{(n)}(x)(x - x_\ell)\omega'(x_\ell)$$

nach zweimaligem Differenzieren

$$\omega''(x) = (L_\ell^{(n)}(x))''(x - x_\ell)\omega'(x_\ell) + 2L_\ell'(x)\omega'(x_\ell).$$

Damit gilt an der Stelle x_ℓ

$$\frac{\omega''(x_\ell)}{\omega'(x_\ell)} = 2(L_\ell^{(n)}(x_\ell))''$$

Einsetzen in die Ableitung von P schließt den Beweis ab. □

6.2.4 Interpolationsgüte

Geht man davon aus, dass die Stützwerte f_i als Funktionswerte einer Funktion $f : [a, b] \to \mathbb{R}$ an den Stützstellen $x_i \in [a, b]$ gegeben sind, so stellt sich die Frage nach der Güte der Interpolation, d.h. wie gut approximiert das Interpolationspolynom die Funktion auf $[a, b]$.

6.2.4.1 Interpolationsfehler

Bei der nun folgenden Darstellung des Approximationsfehlers

$$\varepsilon(x) := f(x) - P(x)$$

erweisen sich die im vorherigen Abschnitt hergeleiten Eigenschaften der dividierten Differenzen als hilfreich:

Satz 6.2.24 (Interpolationsfehler) *Es sei $f \in C^n[a, b]$ und $f^{(n+1)}(x)$ existiere für alle $x \in (a, b)$; weiterhin gelte $a \leq x_0 \leq x_1 \leq \cdots \leq x_n \leq b$. Dann gilt:*

$$f(x) - P(f|x_0, \ldots, x_n)(x) = \frac{(x - x_0) \cdot \cdots \cdot (x - x_n)}{(n + 1)!} f^{(n+1)}(\xi), \tag{6.12}$$

wobei $\min\{x, x_0, \ldots, x_n\} < \xi < \max\{x, x_0, \ldots, x_n\}$ ist.

Beweis. Nach Konstruktion von $P(f|x_0,\dots,x_n)$ gilt:

$$P(f|x_0,\dots,x_n)(x_k) = f(x_k), \quad k = 0,1,\dots,n.$$

Es sei nun x fest und dabei ungleich $x_0, x_1, \dots, x_n$. Ferner sei

$$K(x) := \frac{f(x) - P(f|x_0,\dots,x_n)(x)}{(x-x_0)\cdot\dots\cdot(x-x_n)}. \tag{6.13}$$

Nun betrachten wir die Funktion

$$W(t) := f(t) - P(f|x_0,\dots,x_n)(t) - (t-x_0)\cdot\dots\cdot(t-x_n)K(x). \tag{6.14}$$

Die Funktion $W(t)$ verschwindet also an den Stellen $t = x_0, \dots, t = x_n$ und durch (6.13) auch an der Stelle $t = x$. Nach dem verallgemeinerten Satz von **Rolle**[3] verschwindet die Funktion $W^{(n+1)}(t)$ an einer Stelle ξ mit $\min\{x, x_0, \dots, x_n\} < \xi < \max\{x, x_0, \dots, x_n\}$. Das $(n+1)$-fache Differenzieren von (6.14) nach t liefert

$$W^{(n+1)}(t) = f^{(n+1)}(t) - (n+1)!K(x),$$

sodass gilt:

$$0 = W^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (n+1)!K(x).$$

Damit erhalten wir aber unmittelbar

$$K(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi). \tag{6.15}$$

Nach Einsetzen von (6.15) in (6.14) erhalten wir die Behauptung für $t = x$, da x Nullstelle von W ist. □

Bemerkung 6.2.25 (Darstellung des Interpolationsfehlers) *Im Beweis zum Approximationsfehler (vgl. Satz, 6.2.24* (6.12)*) haben wir gezeigt*

$$f(x) - P(f|x_0,\dots,x_n)(x) = \frac{\omega_{n+1}(x)}{(n+1)!} f^{(n+1)}(\xi), \ \min\{x_0,\dots,x_n,x\} < \xi < \max\{x_0,\dots,x_n,x\}.$$

Und mit dem Satz über die ***Newton****-Darstellung gilt:*

$$f(x) - P(f|x_0,\dots,x_n)(x) = [x_0,\dots,x_n,x]f \cdot \omega_{n+1}(x).$$

Somit folgern wir: Für alle Knoten $x_0 \le \dots \le x_n$ existiert ein $\xi \in [x_0, x_n]$, sodass gilt:

$$[x_0,\dots,x_n]f = \frac{f^{(n)}(\xi)}{n!} \tag{6.16}$$

6.2.4.2 Tschebyscheff-Interpolation

Wie das folgende Beispiel zeigen wird, hat die Verteilung der Stützstellen $x_0,\dots,x_n$ über das Interpolationsintervall entscheidenden Einfluss auf die Güte der Approximation. Ein klassisches Beispiel hierfür stammt von **Runge**[4]:

Die Interpolationspolynome $P(f|x_1,\dots,x_n)$ zur Funktion $f(x) = \frac{1}{1+x^2}$ im Intervall $I := [-5,5]$ bei äquidistanten Stützstellen $x_k = -5 + \frac{10}{n}k$ zeigen bei wachsendem n einen zunehmenden Interpolationsfehler.

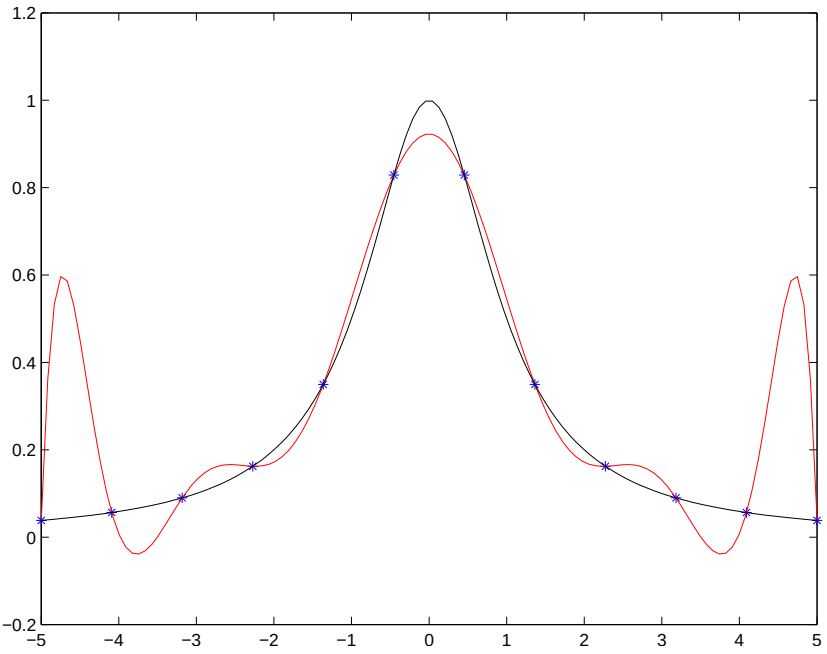
Das Interpolationspolynom zu 12 äquidistanten Stützstellen und Stützwerten zur Funktion

$$f(x) = \frac{1}{1+x^2}$$

ist in Abb. 6.2.4.2 dargestellt.

MATLAB-Beispiel:

```
n = 12;
f = @(x)  1./(x.^2+1);
x = linspace(-5,5,n);
fx = f(x);
s = linspace(x(1),x(end),10*n);
for j=1:length(s)
  ps(j) = AitkenNeville(x,fx,s(j));
end
plot(x,fx,'*',s,ps,'r-',s,f(s),'k')
```



Wie wir im weiteren zeigen werden, kann man bei geschickter, nichtäquidistanter Wahl der Stützstellen dagegen eine Konvergenz erhalten. Genauer gesagt, wählen wir $x_1, \ldots, x_n$ als die Nullstellen der von $[-1, 1]$ auf I transformierten **Tschebyscheff-Polynome** und erhalten dadurch punktweise Konvergenz für $n \to \infty$. Man beachte, dass dieses Phänomen nicht von Rundungsfehlern abhängt.

Bei der Berechnung des Approximations- bzw. des Interpolationsfehlers haben wir gesehen, dass

$$f(x) - P(f|x_0, \ldots, x_n)(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x), \quad x \in [a, b]$$

für ein $\xi = \xi(x) \in (a, b)$ gilt. Wir suchen nun Knoten $x_0, \ldots, x_n \in [a, b]$, die das **Minimax-Problem**

$$\max_{x\in[a,b]} |\omega_{n+1}(x)| = \max_{x\in[a,b]} |(x - x_0) \cdot \ldots \cdot (x - x_n)| = \min$$

lösen. Anders formuliert, es gilt das normierte Polynom $\omega_{n+1} \in \mathbb{P}_{n+1}$ mit den reellen Nullstellen $x_0, \ldots, x_n$ zu bestimmen, für das

$$\max_{x\in[a,b]} |\omega_{n+1}(x)| = \min_{x_0,\ldots,x_n}$$

gilt. Im Folgenden werden wir sehen, dass gerade die **Tschebyscheff**-Polynome T_n diese obige **Minimax**-Aufgabe lösen (sie lösen sie bis auf einen skalaren Faktor und eine affine Transformation). Somit sind die Nullstellen der Tschebyscheff-Polynome (bis auf eine affine Transformation) gerade die gesuchten Stützstellen $x_0, \ldots x_n$.

Zunächst reduzieren wir das Problem auf das Intervall $[-1, 1]$ mit Hilfe der Umkehrabbildung folgender Abbildung

$$y : [a, b] \to [-1, 1], \quad x \mapsto y = y(x) = \frac{2x - a - b}{b - a},$$

[3] Rolle, Michel (1652-1719)
[4] Runge, Carl (1856-1927)

d.h. die Umkehrabbildung lautet:

$$x : [-1,1] \to [a,b], \;\; y \mapsto x = x(y) = \frac{b+a}{2} + \frac{b-a}{2} y.$$

Ist jetzt $P \in \mathbb{P}_n$ mit grad $P = n$ und führendem Koeffizienten $a_n = 1$ die Lösung des **Minimax**-problems

$$\max_{y \in [-1,1]} |P(x)| = \min,$$

so stellt $\widehat{P}(x) := P(y(x))$ die Lösung des ursprünglichen Problems mit führendem Koeffizienten $2^n/(b-a)^n$ dar. Für $y \in [-1,1]$ definieren wir die **Tschebyscheff-Polynome** durch (vgl. hierzu auch Kapitel 7):

$$T_n(y) = \cos(n \arccos(y)), \;\; y \in [-1,1] \tag{6.17}$$

und allgemein für $y \in \mathbb{R}$ durch die **Drei-Term-Rekursion**

$$T_0(y) = 1, \quad T_1(y) = y, \quad T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y), \; k \geq 2. \tag{6.18}$$

Wir benötigen im Folgenden die Eigenschaften der **Tschebyscheff** Polynome, die wir in Kapitel 7 detailliert diskutierten werden und hier der Einfachheit halber bereits schon mal wiedergeben:

Bemerkung 6.2.26 (Eigenschaften der Tschebyscheff-Polynome) (i) *Der führende Koeffizient von T_n ist $a_n = 2^{n-1}$, $n \geq 1$*

(ii) $|T_n(y)| \leq 1$ *für* $y \in [-1,1]$.

(iii) *Die Nullstellen von $T_n(y)$ sind*

$$y_k := \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = 0,1,\ldots,n-1.$$

(iv) $|T_n(y)|$ *nimmt seinen maximalen Wert im Intervall* $[-1,1]$ *an den* $(n+1)$ *Extremalstellen* $\overline{y}_k = \cos(\frac{k\pi}{n})$ *für* $k = 0,\ldots,n$ *an, d.h.*

$$|T_n(y)| = 1 \quad \Leftrightarrow \quad y = \overline{y}_k = \cos\left(\frac{k\pi}{n}\right) \;\; mit \; k = 0,\ldots,n.$$

Satz 6.2.27 (Minimal-Eigenschaft der Tschebyscheff-Polynome) *Jedes Polynom $P \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n \neq 0$ nimmt im Intervall $[-1,1]$ einen Wert vom Betrag $\geq |a_n|/2^{n-1}$ an. Insbesondere sind die **Tschebyscheff**-Polynome $T_n(y)$ minimal bezüglich der Maximumsnorm $\|f\|_\infty = \max_{y\in[-1,1]} |f(y)|$ unter den Polynomen vom Grad n mit führendem Koeffizienten 2^{n-1}.*

Beweis. **(Annahme:)** Sei $P \in \mathbb{P}_n$ ein Polynom mit führendem Koeffizienten $a_n = 2^{n-1}$ und $|P(y)| < 1$ für $y \in [-1,1]$. Dann ist $T_n - P_n$ ein Polynom vom Grad kleiner oder gleich $(n-1)$ (beide besitzen a_n als führenden Koeffizienten). An den **Tschebyscheff**-Abszissen $\overline{y}_k := \cos(\frac{k\pi}{n})$ gilt:

$$T_n(\overline{y}_{2k}) = 1, \;\; P_n(\overline{y}_{2k}) < 1 \;\; \Rightarrow \quad P_n(\overline{y}_{2k}) - T_n(\overline{y}_{2k}) < 0,$$

$$T_n(\overline{y}_{2k+1}) = -1, \;\; P_n(\overline{y}_{2k+1}) > -1 \;\; \Rightarrow \quad P_n(\overline{y}_{2k+1}) - T_n(\overline{y}_{2k+1}) > 0,$$

d.h. die Differenz $T_n - P_n$ ist an den $(n+1)$-**Tschebyscheff**-Abszissen abwechselnd positiv und negativ, damit besitzt die Differenz mindestens n Nullstellen in $[-1,1]$ im Widerspruch zu $0 \neq T_n - P_n \in \mathbb{P}_{n+1}$. Demnach muss es für jedes Polynom $P \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n = 2^{n-1}$ ein $y \in [-1,1]$ geben derart, dass $|P_n(y)| \geq 1$ erfüllt. Für ein beliebiges $P \in \mathbb{P}_n$ mit $a_n \neq 0$ folgt die Behauptung daraus, dass $\widetilde{P}_n := \frac{2^{n-1}}{a_n} P_n$ ein Polynom mit $\widetilde{a}_n = 2^{n-1}$ ist. □

7 Numerische Integration und Differenziation

Die Numerische Integration oder Quadraturtheorie behandelt die Prinzipien und Algorithmen zur numerischen Berechnung von Integralen gegebener Funktionen. Hierbei beschränken wir uns vorerst auf die Berechnung des **Riemann**-Integrals[1]

$$I(f) := \int_a^b f(x)\, dx.$$

Dabei stellt f eine (von Computern ausführbare) Vorschrift dar, die es gestattet zu jedem $x \in [a, b]$ den entsprechenden Funktionswert f zu ermitteln (in den meisten Anwendungen ist f eine stückweise stetige oder stückweise glatte Funktion).

Aufgabe: Gegeben sei $f : [a, b] \to \mathbb{R}$, mit z.B. $f \in C[a, b]$. Approximiere den Ausdruck

$$I(f) := \int_a^b f(x)dx,$$

mit der Vorstellung, dass $I(f)$ nicht, bzw. nicht „leicht“ exakt bestimmt werden kann. Dazu konstruiert man Näherungsformeln („**Quadraturformeln**“)

$$\hat{I}_n : C[a, b] \to \mathbb{R}, \quad f \to \hat{I}_n(f)$$

die das Integral möglichst gut approximieren und dessen Eigenschaften erhalten, so dass der **Quadraturfehler**

$$E_n(f) := I(f) - \hat{I}_n(f)$$

klein wird. Hierzu wird der Integrand durch eine Approximation, welche man z.B. durch Polynominterpolation erhält, ersetzt und diese integriert.

Bevor wir diese Idee der Konstruktion von $\hat{I}_n$ vertiefen, Fragen wir uns zunächst, welche **Kondition** das Problem der Integralberechnung besitzt? Mit

$$\|f\|_1 := \int_a^b |f(t)|dt = I(|f|),$$

der so genannten L^1-Norm gilt

Lemma 7.0.1 (Kondition der Integration) *Die absolute und relative Kondition der Integralberechnung bzgl. $\|\cdot\|_1$ lauten:*

$$\kappa_{abs} = 1, \quad \kappa_{rel} = \frac{I(|f|)}{|I(f)|}. \tag{7.1}$$

Beweis. Sei $\delta f \in L_1([a, b])$ eine Störung, dann gilt

$$\left| \int_a^b (f + \delta f)dx - \int_a^b f\, dx \right| = \left| \int_a^b \delta f\, dx \right| \leq \int_a^b |\delta f|dx = \|\delta f\|_1$$

und hier gilt Gleichheit genau dann, wenn $\delta f \geq 0$. □

[1] Riemann, Georg Friedrich Bernhard (1826-1866)

D.h. in absoluter Betrachtung ist die Integration ein gutartiges Problem. Relativ betrachtet birgt die Integration z.B. von stark oszillierenden Integranten Schwierigkeiten.

7.1 Quadraturformeln

Die **Grundidee** der Quadratur besteht aus dem Folgenden:

- Unterteile $[a, b]$ in m Teilintervalle $[t_k, t_{k+1}], k = 0, 1, \ldots, m-1$, z.B. äquidistant

$$t_k = a + kH,\ k = 0, \ldots, m,\ H = \frac{b-a}{m}.$$

- Approximiere den Integranden f auf jedem Teilintervall $[t_k, t_{k+1}]$ durch eine „einfach“ zu integrierende Funktion g_k (z.B. ein Polynom vom Grad kleiner gleich n bzgl. $(n+1)$ Stützstellen in dem Intervall $[t_k, t_{k+1}]$) und verwende die Approximation

$$I(f) = \sum_{k=0}^{m-1} \int_{t_k}^{t_{k+1}} f(y)\, dy \approx \sum_{k=0}^{m-1} \int_{t_k}^{t_{k+1}} g_k(y)\, dy.$$

  Für g_k verwenden wir zunächst die Polynominterpolation.

Wir beginnen in diesem Abschnitt mit einfachen Beispielen, bei denen wir jeweils zunächst nur ein Teilintervall betrachten auf welchem wir den Integranden mit einer „einfachen“ Funktion approximieren, dieser Ansatz wird dann jeweils für $m \geq 1$ Teilintervalle „zusammengesetzt“. Dadurch erhält man sog. zusammengesetzte oder summierte Regeln.

Insgesamt betrachten wir drei Beispiele; In den ersten zwei Beispielen verwenden wir auf jedem Teilintervall konstante Funktionen zur Approximation, d.h. konstante Interpolationspolynome nullten Grades bzgl. einer Stützstelle und im dritten Beispiel auf jedem Teilintervall lineare Funktionen, d.h. Interpolationspolynome ersten Grades bzgl. zweier Stützstellen.

Im Anschluss konstruieren dann so genannte interpolatorische Quadraturformeln und motivieren damit die allgemeine Definition von Quadraturformeln. Weiterhin werden wichtige Eigenschaften von Quadraturformeln behandelt.

7.1.1 Einfache Beispiele

Beginnen wir mit den sicherlich einfachsten Quadraturformeln. Allgemein ersetzen wir zunächst den Integranten f auf dem Intervall $[a, b]$ durch eine konstante Funktion.

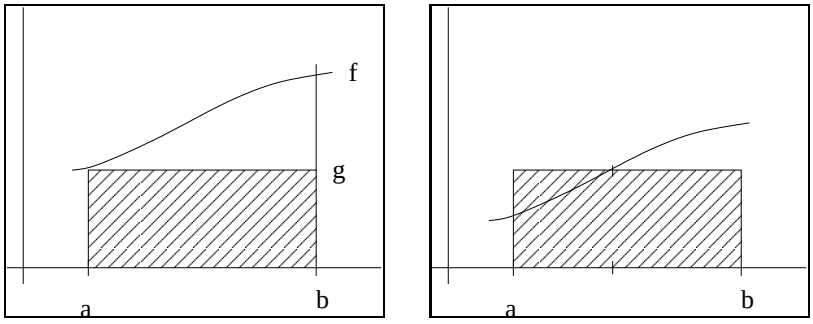


Abb. 7.1: Links: Linke Rechteckregel, rechts: Mittelpunktregel

Beispiel 7.1.1 (Linke Rechteckregel) Zunächst wählen wir als konstante Funktion g den Funktionswert am linken Rand, d.h. $x_0 = a$, $g(y) = f(x_0)$. Zur Veranschaulichung siehe Abbildung 7.1. Die resultierende Quadraturformel

$$R^L(f) := \hat{I}(f) = (b-a)f(a)$$

wird **linke Rechtecksregel** genannt. Natürlich könnte man auch analog den rechten Rand wählen.

Nun zerlegen wir das Integrationsintervall $[a,b]$ in $m \geq 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + kH, k = 0, 1, \ldots, m-1$ und Breite $H = (b-a)/m$. Auf jedem Teilintervall wähle als konstante Funktion g_k den Funktionswert am linken Rand des Teilintervalls $g_k(y) := f(t_k)$, $y \in [t_k, t_{k+1}]$. Wir erhalten die Quadraturformel

$$R_m^L(f) := H \sum_{k=0}^{m-1} f(t_k),$$

diese wird **zusammengesetzte linke Rechtecksregel** genannt.

Nun zur Analyse: Sei f hinreichend glatt. Wegen

$$f(y) = f(t_k) + (y - t_k)f'(\xi)$$

mit einem $\xi \in (t_k, t_{k+1})$ (Taylor-Restglied) folgt

$$\begin{aligned}\int_{t_k}^{t_{k+1}} f(y)\,dy &= Hf(t_k) + \int_{t_k}^{t_{k+1}} f'(\xi)(y-t_k)\,dy \\ &= Hf(t_k) + \frac{f'(\xi)}{2}(t_{k+1}-t_k)^2 = \int_{t_k}^{t_{k+1}} g(y)\,dy + \mathcal{O}(H^2).\end{aligned}$$

Damit gilt

$$\begin{aligned}|I(f) - R_m^L(f)| &= \left| \int_a^b f(x)\,dx - H\sum_{k=0}^{m-1} f(t_k) \right| \\ &\leq \sum_{k=0}^{m-1} \underbrace{\left| \int_{t_k}^{t_{k+1}} f(x)\,dx - Hf(t_k) \right|}_{=\mathcal{O}(H^2)} \\ &\overset{m=\frac{b-a}{H}}{=} \mathcal{O}(H).\end{aligned}$$

Beispiel 7.1.2 (Mittelpunktregel) Wir setzen

$$\mathcal{C}_M^{(r)} := \left\{ f : [a,b] \to \mathbb{R} \;\middle|\; f^{(r)} \text{ stetig und } \sup_{a \leq x \leq b} |f^{(r)}(x)| \leq M \right\}$$

für $r \in \mathbb{N}$ und $M > 0$, und betrachten Grafik 7.2: Sei nun $x_0 \in (a,b)$, dann gilt für $f \in C^1[a,b]$:

$$\int_a^b |f - P(f|x_0)|\,dx \;\leq \sup_{a \leq x \leq b} \frac{|f'(x)|}{1!} \int_a^b |x - x_0|\,dx. \tag{7.2}$$

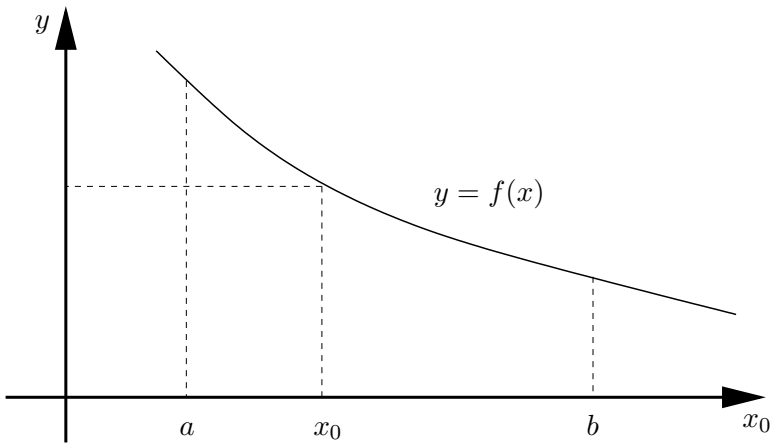


Abb. 7.2: Graph einer Funktion f.

Weiterhin ergibt sich mit $h := b - a$:

$$\begin{aligned} J(x_0) := \int_a^b |x - x_0|\, dx &= -\int_a^{x_0} (x - x_0)\, dx + \int_{x_0}^b (x - x_0)\, dx \\ &= \frac{(x-x_0)^2}{2}\Big|_{x=x_0}^b - \frac{(x-x_0)^2}{2}\Big|_{x=a}^{x_0} \\ &= \frac{(b-x_0)^2}{2} + \frac{(a-x_0)^2}{2} \\ &= x_0^2 - x_0(a+b) + \frac{a^2+b^2}{2}. \end{aligned}$$

Wir wollen die Stützstelle x_0 nun so wählen, dass $J(x_0)$ bzw. der rechte Teil der Ungleichung (7.2) minimal wird. Mit

$$J'(x_0) = 2x_0 - (a+b) \stackrel{!}{=} 0$$

ergibt sich als einziger kritischer Punkt $x_0 = \frac{a+b}{2}$, welcher J minimiert, da $J''(x_0) = 2 > 0$. Wir erhalten die **Mittelpunktregel** (zur Veranschaulichung siehe Abbildung 7.1)

$$M(f) := \hat{I}_0(f) := (b-a) \cdot f\left(\frac{a+b}{2}\right). \tag{7.3}$$

bei der als konstante Funktion $g(y) = f((a+b)/2)$ genommen wird.
Mit obiger Herleitung haben wir auch gezeigt, dass für $M := \sup_{a \le x \le b} |f'(x)|$ gilt:

$$\begin{aligned} \sup_{f \in \mathcal{C}_M^{(1)}} \left| \int_a^b f(x) - \hat{I}_0(f) \right| &\le \frac{M}{1!} \cdot J\Big(\frac{a+b}{2}\Big) \\ &= M \cdot \Big(\frac{(a+b)^2}{4} - \frac{(a+b)^2}{2} + \frac{a^2+b^2}{2}\Big) \\ &= M \cdot \frac{(b-a)^2}{4}, \end{aligned}$$

also die rechte Seite ein Maß für die von (7.3) in $\mathcal{C}_M^{(1)}$ garantierte Genauigkeit ist.

Nun zerlegen wir das Integrationsintervall $[a, b]$ wieder in $m \ge 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + Hk, k = 0, \ldots, m$. Auf jedem Teilintervall setze

$$g_k(y) := f\left(\frac{t_k + t_{k+1}}{2}\right), \quad y \in [t_k, t_{k+1}].$$

Mit $x_k := \frac{1}{2}(t_k + t_{k+1})$ liefert Taylorentwicklung um x_k für $f \in C^3(t_k, t_{k+1})$

$$f(y) = f(x_k) + (y - x_k)f'(x_k) + \frac{1}{2}(y - x_k)^2 f''(x_k) + \mathcal{O}(H^3),$$

und damit

$$\begin{aligned}\int_{t_{k+1}}^{t_k} f(y)\,dy &= \int_{t_k}^{x_k} f(y)\,dy + \int_{x_k}^{t_{k+1}} f(y)\,dy \\ &= \frac{H}{2}f(x_k) + \int_{t_k}^{x_k}(y - x_k)f'(x_k)\,dy + \mathcal{O}(H^3) \\ &\quad + \frac{H}{2}f(x_k) + \int_{x_k}^{t_{k+1}}(y - x_k)f'(x_k)\,dy + \mathcal{O}(H^3) \\ &= Hf(x_k) + \mathcal{O}(H^3),\end{aligned}$$

denn

$$\int_{t_k}^{x_k}(y - x_k)\,dy = \int_{-\frac{H}{2}}^{0} z\,dz = -\int_{0}^{-\frac{H}{2}} z\,dz = -\int_{x_k}^{t_{k+1}}(y - x_k)\,dy.$$

Mit Quadraturknoten $x_k = a + (2k + 1)H/2, k = 0, \ldots, m - 1$ erhalten die **summierte Mittelpunktregel**

$$M_H(f) := \hat{I}_{0,m}(f) := H\sum_{k=0}^{m-1} f\left(\frac{t_k + t_{k+1}}{2}\right) = H\sum_{k=0}^{m-1} f(x_k),$$

mit dem Fehler

$$|I(f) - M_H(f)| = \mathcal{O}(H^2) \quad \text{für } f \in C^3(t_k, t_{k+1}).$$

Nun ersetzten wir den Integranten durch eine lineare Funktion bzw. durch lineare Funktionen auf Teilintervallen.

Beispiel 7.1.3 (Trapezregel) Wählen wir g als lineare Interpolation (Lagrange Interpolationspolynom vom Grad 1) von f bzgl. der Knoten $x_0 = a$ und $x_1 = b$ so erhalten wir die so genannte **Trapezregel**:

$$\hat{I}_1(f) := \frac{b - a}{2}\left(f(a) + f(b)\right).$$

Um die zusammengesetzte Trapezregel zu erhalten, zerlegen wir $[a, b]$ wie im Fall eines Knotens (d.h. $n = 0$) in $m \geq 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + Hk, k = 0, \ldots, m$ der Breite $H = b - a/m$ und wähle g_k als lineare Interpolation von f bzgl. der Stützstellen t_k, t_{k+1} mit Quadraturknoten t_k, d.h.

$$g_k(y) = \frac{t_{k+1} - y}{H}f(t_k) + \frac{y - t_k}{H}f(t_{k+1}), \quad y \in [t_k, t_{k+1}]$$

Das führt zu

$$\int_{t_k}^{t_{k+1}} g_k(y)dy = \frac{H}{2}(f(t_{k+1}) + f(t_k)).$$

Dies setzt man zusammen auf $[a, b] = [t_0, t_m]$:

$$\hat{I}_{1,m}(f) := \frac{H}{2}\sum_{k=0}^{m-1}(f(t_{k+1}) + f(t_k))$$

Da jeder Term bis auf den ersten und letzten doppelt gezählt wird erhalten wir die sogenannte **Trapezsumme**

$$T_H(f) := \hat{I}_{1,m}(f) := H\left\{\frac{1}{2}f(a) + f(t_1) + \ldots + f(t_{m-1}) + \frac{1}{2}f(b)\right\}. \tag{7.4}$$

7.1.2 Konstruktion und Definition von Quadraturformeln

In obigen Beispielen haben wir stets den Integranden f durch eine Approximation $\hat{f}$ ersetzt, d.h. $f \approx \hat{f}$ und dann $\hat{f}$ exakt integriert. Wir haben also als Quadratur $\hat{I}_n$

$$\hat{I}_n(f) = I(\hat{f})$$

gewählt. Um höhere Genauigkeit zu erzielen, reichen konstante bzw. lineare Funktionen, wie bei der Mittelpunkts- bzw. der Trapetzregel, offenbar nicht aus.

Die Kernidee der Mittelpunkts- bzw. der Trapezformel liegt eigentlich darin, die Funktion f durch eine Interpolierende $P(f|x_0,\ldots,x_n)$ bzgl. der Stützstellen x_i (die - wie bei der Mittelpunktsregel - von den t_j verschieden seien können) zu ersetzen, sodass sich für diese die Quadratur einfach ausführen läßt. Es liegt somit nahe, als Approximation $\hat{f}$ das Interpolationspolynom an f bezüglich der Knoten zu wählen, d.h.

$$\hat{f}(x) = P(f|x_0,\ldots,x_n)(x).$$

Wir verwenden dann $I\big(P(f|x_0,\ldots,x_n)\big)$ als Approximation zu $I(f) = \int_a^b f(x)dx$, setzen also:

$$\hat{I}_n(f) = I\big(P(f|x_0,\ldots,x_n)\big).$$

Konstruktion interpolatorischer Quadraturformeln: Die Idee bei der Konstruktion von Quadraturformeln ist es $(n+1)$ Stützstellen zu wählen und, wie bereits erwähnt, das Integral über das Interpolationspolynom zu diesen Knoten als Näherung an das Integral von f zu verwenden.

Da ein besimmtes Integral über ein beliebiges Intervall $[a,b]$ - ohne Beschränkung der Allgemeinheit - mittels Variablentransformation auf das Intervall $[0,1]$ transformiert werden kann, betrachten wir zunächst $[a,b] = [0,1]$.
Die auf $[0,1]$ transformierten Stützstellen nennen wir τ_i.
Die $(n+1)$ Stützstellen $\tau_0,\ldots,\tau_n \in [0,1]$ seien der Einfachheit halber äquidistant, d.h. $\tau_i = ih$ für $i = 0,\ldots,n$ mit $h = 1/n$. Mit der Lagrange-Darstellung des Interpolationspolynoms

$$P(y) = \sum_{i=0}^{n} f(\tau_i) L_i^{(n)}(\tau), \quad L_i^{(n)}(y) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(y-\tau_j)}{(\tau_i-\tau_j)}$$

folgt dann

$$\hat{I}_n(f) = \int_0^1 P(\tau)\,d\tau = \int_0^1 \left(\sum_{i=0}^{n} f(\tau_i) L_i^{(n)}(\tau)\right) d\tau = \sum_{i=0}^{n} f(\tau_i) \underbrace{\left(\int_0^1 L_i^{(n)}(\tau)\,d\tau\right)}_{=:\lambda_i} = \sum_{i=0}^{n} f(\tau_i)\lambda_i$$

Dies nennt man eine **Quadraturformel** mit **Gewichten** λ_i und **Knoten** (**Stützstellen**) τ_i.

Um eine allgemeine Quadraturformel für beliebige Intervalle $[a,b]$ zu bekommen, wird das Intervall $[a,b]$ nun auf $[0,1]$ transformiert. Mit $x = a + \tau(b-a)$ und $dx = d\tau(b-a) = (b-a)d\tau$ erhalten wir

$$I(f) = \int_a^b f(x)\,dx = (b-a)\int_0^1 \underbrace{f(a+\tau(b-a))}_{:=\tilde{f}(\tau)}\,d\tau.$$

Damit erhält man die Quadraturformel

$$\hat{I}_n(f) = (b-a)\sum_{i=0}^{n} f(a+\tau_i(b-a))\cdot\lambda_i$$

mit $\lambda_i := \int_0^1 L_i^{(n)}(\tau)\, d\tau$ und $\tau_i = ih$ für $h = 1/n$. Man beachte, die Gewichte λ_i sind unabhängig von den Integrationsgrenzen a, b und vom aktuellen Integranten und müssen für gegebene Stützstellen also nur **einmal** berechnet werden.

Wir fassen dies zusammen und definieren allgemein das lineare Funktional $\hat{I}_n$ als Quadraturformel:

Definition 7.1.4 (Quadraturformel) *Unter einer **Quadraturformel** $\hat{I}$ zur Approximation des bestimmten Integrals*

$$I(f) = \int_a^b f(x)\, dx$$

versteht man

$$\hat{I}_n(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(x_i) \tag{7.5}$$

*mit den **Knoten** $x_0, \ldots, x_n$ und den **Gewichten** $\lambda_0, \ldots, \lambda_n \in \mathbb{R}$, wobei*

$$\sum_{i=0}^{n} \lambda_i = 1 \tag{7.6}$$

gilt.

Bemerkung 7.1.5

1. *Eine Eigenschaft wie (7.6) wird „Partition der Eins“genannt.*
2. *Sind die Gewichte nicht negativ, dann stellt (7.6) eine Konvexkombination der Funktionswerte $f(x_0), \ldots, f(x_n)$ dar.*
3. *Für die konstante Funktion $f(x) \equiv c = \text{const}$, $x \in [a, b]$ folgt aus (7.6)*

$$\hat{I}_n(f) = (b-a) \underbrace{\sum_{i=0}^{n} \lambda_i}_{=1} c = c(b-a) = I(f),$$

*d.h., konstante Funktionen werden **exakt** integriert. Dies ist der Grund für die Forderung (7.6).*

Definition 7.1.6 (Interpolatorische Quadraturformel) *Sei I ein Integral und I_n eine Quadraturformel dazu. Man nennt die Quadraturformel **interpolatorisch**, wenn gilt*

$$\lambda_i = \frac{1}{(b-a)} I(L_i^{(n)}), \quad i = 0, \ldots, n.$$

Bemerkung 7.1.7 *Mit anderen Worten, im Rahmen der Herleitung der Quadraturformeln haben wir interpolatorische Quadraturformeln verwendet. Im Laufe dieses Kapitels werden wir noch andere Quadraturformeln kennen lernen, dies sind die so genannten **Gauß-Quadraturformeln** (vgl. Abschnitt 7.3).*

7.1.3 Exaktheitsgrad und Quadraturfehler

Aus der Konstruktion der interpolatorischen Quadraturformeln ergeben sich sofort Konsequenzen:

- derartige Quadraturformeln sind für alle $f \in \mathbb{P}_n$ exakt,
- aufgrund der Eindeutigkeit der Polynom-Interpolation ist diese Quadraturformel die **einzige** exakte bzgl. der Knoten $x_0, \ldots, x_n$.

Dies fassen wir in den folgenden Lemma zusammen:

Lemma 7.1.8 (Eindeutige exakte interpolatorische Quadraturformel) *Zu $(n+1)$ paarweise verschiedenen Knoten $x_0, \ldots, x_n$ gibt es genau eine Quadraturformel*

$$\hat{I}_n(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(x_i), \tag{7.7}$$

die für alle $P \in \mathbb{P}_n$ vom Grad kleiner oder gleich n exakt ist.

Beweis. Wie bei der Konstruktion verwenden wird die **Lagrange**-Polynome $L_i^{(n)}$ setzen diese in die Quadraturformel ein und erhalten:

$$I\Big(\sum_{i=0}^{n} f(x_i) L_i^{(n)}(x)\Big) = \sum_{0=1}^{n} f(x_i) I(L_i^{(n)}(x)) = (b-a) \sum_{i=0}^{n} \lambda_i f(x_i). \tag{7.8}$$

Dadurch erhalten wir die Gewichte

$$\lambda_i = \frac{1}{(b-a)} \int_a^b L_i^{(n)}(x)\, dx$$

auf eindeutige Weise zurück. □

Nun untersuchen wir allgemein den Fehler von Quadraturformeln. Dazu definieren wir

Definition 7.1.9 (Quadraturfehler, Exaktheitsgrad, Fehlerordnung) *Sei I ein Integral, $\hat{I}_n$ eine Quadraturformel zu $(n+1)$ Knoten.*

1. *Dann heißt $E_n(f) := I(f) - \hat{I}_n(f)$ **Quadraturfehler**.*
2. *Die größte Zahl $r \in \mathbb{N}$ für die gilt*

   $$E_n(f) = I(f) - \hat{I}_n(f) = 0, \quad \forall f \in \mathbb{P}_r$$

   *wird **Exaktheitsgrad** der Quadraturformel genannt.*
3. *Zu gegebenem Exaktheitsgrad r wird $r+1$ **Ordnung** oder **Fehlerordnung der Quadraturformel** $\hat{I}_n$ genannt.*

Bemerkung 7.1.10 (Exaktheitsgrad interpolatorischer Quadraturformeln) *Nach Lemma 7.1.8 hat jede interpolatorische Quadraturformel, die $(n+1)$ Stützstellen verwendet, mindestens den Exaktheitsgrad n.*

Bemerkung 7.1.11 (Test der Ordnung einer Quadraturformel) *Der Quadraturfehler ist offenbar linear. Daher kann die Ordnung einer Quadraturformel $\hat{I}_n$ leicht über die Monome getestet werden: Gilt $E_n(x^j) = 0, j = 0, \ldots, r-1$ und $E_n(x^r) \neq 0$, so hat $\hat{I}_n$ die Ordnung r.*

Hat der Integrand f „schöne“ Eigenschaften, d.h. ist er z.B. $n+1$-mal stetig differenzierbar, so kann man den Interpolationsfehler mit Satz 6.2.24 abschätzen. Integration über den Interpolationsfehler liefert eine Fehlerschranke für den Integrationsfehler.

Lemma 7.1.12 (Fehlerschranken interpolatorischer Quadraturformeln) *Es gilt für*

$$\hat{I}_n(f) := \int_a^b P(f|x_0,\ldots,x_n)(x)dx$$

die Fehlerabschätzung

$$\begin{aligned} |E_n(f)| &\leq (b-a)\left(\max_{y\in[a,b]}\prod_{j=0}^{n}|y-x_j|\right)\frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \\ &\leq \frac{(b-a)^{n+2}}{(n+1)!}\|f^{(n+1)}\|_\infty \end{aligned} \tag{7.9}$$

für $f \in C^{n+1}[a,b]$.

Beweis. Lagrange–Darstellung für $P(f|x_0,\ldots,x_n)$ einsetzen und Fehlerdarstellung der Polynominterpolation anwenden. □

Bemerkung 7.1.13 (Fehlerschranken) *Die Fehlerschranken aus Lemma 7.1.12 sind i.A.* ***nicht*** *optimal. D.h., die Schranken sind in der Regel zu groß. Schärfere Schranken können von Hand bewiesen werden (mittels Anwendung der Mittelwertsätze aus der Analysis).*

7.1.4 Konvergenz einer Quadraturformel

Das Ziel ist es, mit möglichst geringem Aufwand ein Integral möglichst genau zu approximieren. Als Maß für die Genauigkeit haben wir im letzten Abschnitt den Quadraturfehler eingeführt. Man kann sich zusätzlich noch fragen, ob, bzw. unter welchen Bedingungen Quadraturformeln, für wachsende Anzahl von Stützstellen gegen das exakte Integral konvergieren.

Wir betrachten dazu in diesem Abschnitt nun Folgen von Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ und untersuchen, wann diese gegen das zu approximierende Integral konvergieren. Zunächst definieren wir:

Definition 7.1.14 (Konvergenz einer Quadraturformel) *Sei $(\hat{I}_n)_{n\in\mathbb{N}}$ eine Folge von Quadraturformeln. Gilt*

$$\hat{I}_n(f) \xrightarrow{n\to\infty} \int_a^b f(x)dx \quad \text{für jedes } f \in C[a,b], \tag{7.10}$$

so spricht man von der ***Konvergenz der Quadraturformeln*** *$(\hat{I}_n)_{n\in\mathbb{N}}$.*

Die folgenden zwei Sätze liefern nun Bedingungen unter denen Folgen von Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ konvergieren. Insbesondere sind dies Bedingungen an die Gewichte.

Satz 7.1.15 (Szegó) *Sei $a \leq x_0^{(n)} < \cdots < x_n^{(n)} \leq b$, und seien $\lambda_k^{(n)} \in \mathbb{R}$, $k = 0,\ldots,n$. Dann sind die für $n \in \mathbb{N}$ gemäß*

$$\hat{I}_n : \big(C[a,b], \|\cdot\|_\infty\big) \to \big(\mathbb{R}, |\cdot|\big), \quad f \mapsto \hat{I}_n(f) := (b-a)\sum_{i=0}^{n}\lambda_i^{(n)} f\big(x_i^{(n)}\big)$$

definierten Quadraturformeln $\hat{I}_n$ genau dann konvergent, wenn die beiden folgenden Bedingungen erfüllt sind:

(i)

$$\sup_{n\in\mathbb{N}}\sum_{k=0}^{n}\big|\lambda_k^{(n)}\big| < \infty$$

(ii)

$$I(P) = \lim_{n\to\infty} \hat{I}_n(P), \quad \forall\, P \in \mathbb{P} = \mathbb{P}(a,b).$$

wobei $\mathbb{P}(a,b)$ die Menge der Polynome auf dem Intervall (a,b) bezeichnet.

Beweis. Der Beweis dieses Satzes erfordert grundlegende Kenntnisse der Funktionalanalysis und wird deshalb an dieser Stelle ausgespart. Man findet ihn nichtsdestoweniger beispielsweise in [Heuser, Seite 159]. □

Satz 7.1.16 (Steklov) *Unter den Voraussetzungen des Satzes von **Szegó** gelte*

$$\lambda_k^{(n)} \geq 0, \quad n \in \mathbb{N}, 0 \leq k \leq n \tag{7.11}$$

für die Gewichte $\lambda_k^{(n)}$ der Quadraturformel $\hat{I}_n$. Dann konvergieren die Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ genau dann, wenn sie für alle $P \in \mathbb{P}_n$ konvergieren.

Beweis. Nach dem Satz von **Szegó** ist der Beweis erbracht, wenn dort unter der Voraussetzung von (7.11) (i) aus (ii) folgt. Es gelte also (ii) im Satz von **Szegó**. Für $f \equiv 1$ gilt mit (ii)

$$b - a = I(f) = \lim_{n\to\infty} \hat{I}_n(f) = \lim_{n\to\infty} \sum_{k=0}^{n} \lambda_k^{(n)},$$

was wegen (7.11) die Aussage (i) impliziert. □

7.2 Klassische interpolatorische Quadraturformeln

Wir haben Quadraturformeln über Lagrange-Interpolation zu äquidistanten Knoten konstruiert, d.h. durch Polynominterpolation. Damit haben wir die allgemeine Definition der Quadraturformeln motiviert. Im Fall, dass die Gewichte der Quadraturformel den Integralen über die Lagrange-Polynome entsprechen haben wir von interpolatorischen Quadraturformeln gesprochen.

Im Spezialfall, dass diese Quadraturformeln durch Polynominterpolation an **gleichverteilten** Knoten, d.h. äquidistanter Stützstellen - so wie wir sie auch konstruiert haben - , entstehen, spricht man von **Newton-Cotes[2]-Formeln**. In diesem Abschnitt schauen wir uns diesen Spezialfall noch mal etwas genauer an und fassen einige Eigenschaften zusammen. Wir werden sehen, dass die einführenden Beispiele zu diesen klassischen Quadraturformeln gehören.

Außerdem werden wir sehen, dass die Sätze aus dem vorherigen Abschnitt Grenzen für die numerische Stabilität dieses Ansatzes liefern. Dies führt uns auf die **zusammengesetzten Newton-Cotes-Formeln**.

7.2.1 Newton-Cotes-Formeln

Definition 7.2.1 (Newton-Cotes-Formeln) *Bei äquidistanter Knotenwahl $a \leq x_0 < x_1 \cdots < x_n \leq b$ heißen die resultierenden Integrationsformeln*

$$\hat{I}_n(f) = (b-a) \sum_{i=1}^{n} \lambda_i f(x_i)$$

***Newton-Cotes-Formeln**.*
*Die Newton-Cotes-Formeln heißen **abgeschlossen**, wenn $x_0 = a$ und $x_n = b$, d.h.*

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, \ldots, n.$$

*Die **Newton-Cotes**-Formeln heißen **offen**, wenn $x_0 < a$ und $x_n < b$, wobei meist*

$$x_i = a + \Big(i + \frac{1}{2}\Big)h \quad \text{mit} \quad h = \frac{b-a}{n+1}, \quad i = 0, \ldots, n \tag{7.12}$$

gewählt wird.

[2] Cotes, Roger (1682-1716)

Bemerkung 7.2.2 *Es ist klar, dass die Mittelpunkts- und die Trapezregel, welche wir bereits kennen gelernt haben,* ***Newton-Cotes****-Formeln sind. Man erhält sie mit $n = 0, n = 1$. Hierbei ist die Mittelpunktsregel offen und die Trapezregel abgeschlossen.*

Der Ausdruck für die abgeschlossenen **Newton-Cotes**-Gewichte λ_i vereinfacht sich durch die Substitution $s := \frac{(x-a)}{h}$ zu

$$\lambda_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx = \frac{1}{n}\int_0^n \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\,ds := \frac{1}{n}\alpha_i$$

und für die offenen **Newton-Cotes**-Formeln erhält man unter der Wahl (7.12) der x_i:

$$\lambda_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx = \frac{1}{n+1}\int_{-\frac{1}{2}}^{n+\frac{1}{2}} \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\,ds := \frac{1}{n}\alpha_i$$

Bemerkung 7.2.3 (Praktische Berechnung via a priori Tabellierung der Gewichte) *Die* ***Newton-Cotes****-Formeln besitzen den Vorteil, dass die Gewichte zwar von n und h abhängen nicht aber von a, b. In der Praxis berechnet man daher* ***nicht*** *zunächst $\hat{f} = P(f|\ldots)$ und dann $I_n(\hat{f})$, sondern sucht direkt die Form (7.5) der Quadraturformel, d.h., man berechnet die Gewichte. Diese können* ***a priori*** *tabelliert werden.*

Bemerkung 7.2.4 (Offene Newton-Cotes-Formeln) *Die offenen* ***Newton-Cotes****-Formeln kann man vor allem dann verwenden, wenn die Auswertung des Integranden f an den Integrationsgrenzen schwierig ist, z.B. wenn f am Rand eine Singularität hat. Allerdings haben sie eine deutlich schlechtere Ordnung als die ebenfalls offenen Gauß-Quadraturen und finden daher in der Praxis kaum Anwendung.*

Im folgenden Beispiel tabellieren wir daher die Gewichte der wichtigsten geschlossenen **Newton-Cotes**-Formeln. Da die Stützstellen und Gewichte von dem Polynomgrad n abhängen, den wir wählen, schreiben wir hier, wie auch schon zuvor, $x_j^{(n)}$ und $\lambda_j^{(n)}$.

Beispiel 7.2.5 (Geschlossene Newton-Cotes-Formeln) Wählen wir $(n+1)$ äquidistante Stützstellen auf $[a, b]$, d.h.

$$x_i^{(n)} := a + \tau_i^{(n)}(b-a), \quad i = 0, \ldots, n,$$

wobei $\tau_i^{(n)} = i/n$, die Stützstellen auf $[0, 1]$ sind. Bezeichne $\alpha_i^{(n)} = \lambda_i^{(n)} n$ die Gewichte auf $[0, 1]$. Dann gilt mit $h = (b-a)/n$

$$\hat{I}_n(f) := (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} f(a + \tau_i^{(n)}(b-a)) = h\sum_{i=0}^{n} \alpha_i^{(n)} f(a + \tau_i^{(n)}(b-a)). \tag{7.13}$$

lauten die Eckdaten der wichtigsten geschlossenen **Newton-Cotes**-Formeln

n	$\tau_i^{(n)}$	$\lambda_i^{(n)}$	Restglied	Name
1	$0, 1$	$\frac{1}{2}, \frac{1}{2}$	$\frac{1}{12}h^3 f''(\xi)$	Trapezregel
2	$0, \frac{1}{2}, 1$	$\frac{1}{6}, \frac{2}{3}, \frac{1}{6}$	$\frac{1}{90}2^{-5}h^5 f^{(4)}(\xi)$	Simpson–Regel, Keplersche Fassregel
3	$0, \frac{1}{3}, \frac{2}{3}, 1$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$	$\frac{3}{80}3^{-5}h^5 f^{(4)}(\xi)$	Newtonsche $3/8$-Regel
4	$0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$	$\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{90}$	$\frac{8}{945}4^{-7}h^7 f^{(6)}(\xi)$	Milne–Regel

Als Beispiel betrachte die Simpson-Regel, in obiger Form lautet diese

$$\hat{I}_2(f) = b - a\left(\frac{1}{6}f(a) + \frac{2}{3}f\left(\frac{a+b}{2}\right) + \frac{1}{6}f(b)\right).$$

Bemerkung 7.2.6 *In obigem Zugang entspricht die Anzahl der Stützstellen dem Exaktheitsgrad des Interpolationspolynoms und damit der Quadraturformel. Dies wiederum entspricht der Genauigkeit der Quadraturformel. Vielleicht kann man ja mit weniger Stützstelle dieselbe Genauigkeit erreiche, oder — anders ausgedrückt — mit geschickt gewählten Stützstellen eine höhere Genauigkeit erreichen. Mit dieser Frage wollen wir uns nun beschäftigen.*

Bemerkung 7.2.7 (Grenzen der Newton-Cotes-Formeln) *Bei den **Newton-Cotes**-Formeln treten durchaus negative Gewichte auf, d.h. der Satz 7.1.16 von **Steklov** kann dann nicht angewandt werden. Jedoch hilft auch der Satz 7.1.15 von **Szegó** nicht weiter, da - wie **G. Polya** zeigte - eine Funktion existiert, für die die **Newton-Cotes**-Formeln nicht konvergieren.*

*Für abgeschlossene **Newton-Cotes**-Formeln taucht bei $n = 8$, für offene **Newton-Cotes**-Formeln bei $n = 6$ das erste Mal ein negatives Gewicht auf. Man sollte die jeweiligen **Newton-Cotes**-Formeln nicht für n größer als diese jeweiligen Werte verwenden!*

*Die Gewichte der **Newton-Cotes**-Formeln sind **rational** und können mit **Maple** berechnet werden. Die folgenden Zeilen **Maple**-Code berechnen die Gewichte der abgeschlossenen **Newton-Cotes**-Formeln für $n = 8$ und die der offenen **Newton-Cotes**-Formeln für $n = 6$:*

MAPLE-*Beispiel:*

```
> n:=8:
> zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
> seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),x=0..n)/n,m=0..n);

   989    2944     -464    5248     -454     5248     -464     2944     989
  ----, -----, -----,-----, ----, -----, -----, -----, -----
  28350  14175    14175   14175    2835    14175    14175    14175   28350

> n:=6:
> zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
> seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
        x=-1/2..n+1/2)/(n+1),m=0..n);

   4949     49     6223     -6257     6223     49     4949
  -----, ----, -----, -----, -----, ----, -----
  27648    7680   15360    34560    15360    7680   27648
```

Wie man sieht, enthalten beide Quadratur-Formeln negative Gewichte.

7.2.2 Zusammengesetzte Newton-Cotes-Formeln

Wir haben gesehen, dass bei den geschlossenen **Newton-Cotes**-Formeln ab $n = 7$ und für die offenen **Newton-Cotes**-Formeln ab $n = 5$ negative Gewichte auftreten. I.A. können wir keine

Konvergenz $\hat{I}_n(f) \to I(f)$ für $n \to \infty$ erwarten.

Einen Ausweg bieten die sogenannten zusammengesetzten oder summierten Quadraturfomeln. Als Spezialfall zusammengesetzter **Newton-Cotes**-Formeln haben wir in den einführenden Beispielen bereits die summierte Mittelpunktsregel und die Trapezsumme kennen gelernt.

Allgemein ist die Idee zusammengesetzter Quadratur, dass man das Integrationsintervall $[a, b]$ in m Teilintervalle zerlegt und die Quadraturformel $\hat{I}_n(f)$ auf die Teilintervalle der Länge $H = (b-a)/m$ anwendet.

$$\hat{I}_{n,m}(f) = \sum_{k=0}^{m-1} \hat{I}_n|_{[t_k,t_{k+1}]}(f) = \sum_{k=0}^{m-1} \sum_{i=1}^{n} \lambda_i^{(n,k)} f(x_i^{(n,k)})$$

Sei nun n fest und man betrachte eine Folge von zusammengesetzte **Newton-Cotes**-Formeln $(\hat{I}_m(f))_{m\in\mathbb{N}}$:

Beispiel 7.2.8 (Anwendung des Satzes von Szegó auf die summierte Mittelpunktsregel)
Wir wenden den Satz von **Szegó** auf die summierte Mittelpunktsregel an. Betrachte hierzu eine Folge von Knoten

$$x_k^{(1,m)} := a + \frac{2k+1}{2} \cdot \frac{b-a}{m+1}, \quad k = 0, \ldots, m,$$

mit den Gewichten

$$\lambda_k^{(m)} = \frac{b-a}{m+1}.$$

Es gilt: $\sum_k \left|\lambda_k^{(m)}\right| = b - a$, demnach ist (i) in Satz 7.1.15 erfüllt. Weiterhin gilt (ii) aufgrund der Stetigkeit (damit der **Riemann**-Integrierbarkeit) der Polynome auf $[a, b]$ und der Fehlerabschätzung für die summierte Mittelpunktsregel.

Lemma 7.2.9 (Fehlerschätzung der Trapezsumme) *Es gilt*

$$|E_1(f)| = |I(f) - T_H(f)| = \mathcal{O}(H^2) \text{ für } f \in C^2([a, b]) \tag{7.14}$$

Bemerkung 7.2.10 (Ineffizenz bei hoher gewünschter Genauigkeit) *Angenommen, man möchte mit der Trapezsumme (7.4) die Genauigkeit von 10^{-6} erreichen. Die Fehlerschätzung (7.14) besagt dann*

$$|I(f) - T_H(f)| = \mathcal{O}(H^2) \leq 10^{-6},$$

also muss H^2 in der Größenordung von 10^{-6} sein, demnach muss $H \sim 10^{-3}$ gelten und bei äquidistanter Schrittweite folgt dann für die Anzahl der Knoten

$$m = \left[\frac{b-a}{H}\right] \sim 10^3.$$

Dies wäre viel zu ineffizient, wir brauchen Alternativen!

7.3 Gauß-Quadratur

Wir haben gesehen, dass bei interpolatorische Quadraturen zu $(n+1)$ Stützstellen die Gewichte so gewählt werden können, dass diese mindestens den Exaktheitsgrad n haben. Bisher waren dabei die Stützstellen vorgegeben. Es stellt sich nun die Frage, ob wir Quadraturen mit höcherer Ordnung erreichen, wenn wir sowohl die Gewichte als auch Stützstellen frei wählen.

Wir verfolgen also das **Ziel**, Qudraturformeln zu konstruieren, die exakt sind von möglichst hoher Ordnung, d.h.

$$\hat{I}_n(P) = I(P)$$

für alle $P \in P_N$ mit N maximal (natürlich hängt N von n ab!).

Bevor wir uns allerdings mit der Konstruktion solcher Quadraturformel beschäftigen, d.h. wie mit der Wahl der Stützstellen und Gewichte, überlegen wir uns was N maximal konkret bedeutet. Hierzu halten wir zunächst das folgende fest:

Satz 7.3.1 (Obere Grenze für die Ordnung von Quadraturformeln) *Sind $a \leq x_0^{(n)} < \cdots < x_n^{(n)} \leq b$ und ist $\hat{I}_n$ eine Quadraturformel bzgl. $x_i^{(n)}, i = 0, \ldots, n$, so gilt für ihre Ordnung k:*

$$k \leq 2n + 2.$$

Beweis. Die Anwendung der Quadraturformel in der Form

$$\hat{I}_n(f) = (b-a) \sum_{k=0}^{n} \lambda_k^{(n)} f(x_k^{(n)})$$

auf das Polynom

$$P(x) = \prod_{k=0}^{n} (x - x_k^{(n)})^2, \quad x \in [a, b]$$

vom Grad $2n + 2$ liefert die Aussage

$$\int_a^b \underbrace{P(x)}_{>0}\, dx - (b-a) \sum_{k=0}^{n} \lambda_k^{(n)} \underbrace{P(x_k^{(n)})}_{=0} > 0,$$

also $I(P) > 0$, $\hat{I}_n(P) = 0$, speziell $I(P) \neq \hat{I}(P)$ und damit die Behauptung. □

Der vorstehende Satz besagt, dass für $(n + 1)$ Stützstellen Polynome vom Grad N kleiner gleich $2n + 1$ exakt integriert werden. Man kann sich leicht überlegen, dass dieser **maximale Exaktheitsgrad** auch erreicht wird:

Sei die Anzahl n der Teilintervalle vorgegeben. Die Quadraturformeln haben die Gestalt

$$\hat{I}_n(f) = (b-a) \sum_{i=0}^{n} \lambda_i^{(n)} f(x_i^{(n)})\, dx$$

Versucht man sowohl die $n + 1$ Stützstellen als auch die $n + 1$ Gewichte so zu wählen, dass die Fehlerordnung möglichst groß wird, so hat man $2n + 2$ Unbekannte:

$$\lambda_0^{(n)}, \ldots, \lambda_n^{(n)}, x_0^{(n)}, \ldots, x_n^{(n)},$$

Wenn man zur Bestimmung dieser Unbekannten den folgenden monomialen Ansatz wählt

$$(b-a) \sum_{i=0}^{n} \lambda_i^{(n)} (x_i^{(n)})^j = \int_a^b x^j\, dx = \frac{1}{j+1}(b^{j+1} - a^{j+1}), \quad j = 0, \ldots, 2n+1,$$

so ist dies ein **nichtlineares Gleichungssystem** mit $2n+2$ Gleichungen und $2n+2$ Unbekannten. Hat dieses Gleichungssystem eine Lösung, so integriert die resultierende Quadraturformel Polynome bis zum Grad $2n + 1$ exakt. Mit zunehmendem n wird dieses Gleichungssystem allerdings immer unübersichtlicher. Und insbesondere gehen, wie bereits erwähnt, die Knoten nichtlinear ein.

Die Frage ist nur, wie man die Knoten und Gewichte systematischer wählen kann? Daher verfolgt man einen etwas allgemeineren Ansatz: Hierzu betrachten wir das gewichtete Integral

$$I(f) = \int_a^b f(x)\omega(x)\, dx$$

mit einer integrablen Gewichtsfunktion $\omega : [a, b] \to \mathbb{R}$. Wir nennen ω positiv, falls $\omega(x) > 0$ für alle $x \in (a, b)$.

Bevor wir uns aber mit der numerischen Integration eines solchen Integrals genauer beschäftigen, stellen wir noch einige mathematische Konstrukte zur Verfügung: Dies sind im Wesentlichen orthogonale Polynome und ihre Eigenschaften.

7.3.1 Orthogonale Polynome

In diesem Abschnitt werden orthogonale Polynome bzgl. gewichtetet L^2-Skalarprodukte definiert und ihre Eigenschaften betrachtet. Zunächst wiederholen wir die Definition eines Skalarprodukts:

Definition 7.3.2 (Skalarprodukt) *Es sei V ein reeller Vektorraum. Ein **Skalarprodukt** (oder inneres Produkt) auf V ist eine symmetrische positiv definite Bilinearform $(\cdot, \cdot) \colon V \times V \to \mathbb{R}$, d.h. für x, y, $z \in V$ und α, $\beta \in \mathbb{R}$ gelten die folgenden Bedingungen:*

i.) positive Definitheit

$$(x, x) \geq 0, \quad \text{und } (x, x) = 0 \text{ genau dann, wenn } x = 0,$$

ii.) Symmetrie

$$(x, y) = (y, x)\,,$$

iii.) Bilinearität

$$(\alpha x + \beta y, z) = \alpha(x, z) + \beta(y, z)\,.$$

Man beachte, dass V hier nicht endlich-dimensional zu sein braucht! Für Skalarprodukte gilt die **Cauchy-Schwarz**-Ungleichung:

Satz 7.3.3 (Cauchy-Schwarz-Ungleichung) *Es sei V ein reeller Vektorraum, $(\cdot, \cdot) \colon V \times V \to \mathbb{R}$ ein Skalarprodukt auf V. Dann gilt*

$$(x, y) \leq \sqrt{(x, x)}\sqrt{(y, y)}\,.$$

Beweis. Der Fall $y = 0$ ist trivial. Es bleibt also der Fall $y \neq 0$ und damit $(y, y) \neq 0$. Für jedes $\alpha \in \mathbb{R}$ gilt

$$0 \leq (x - \alpha y, x - \alpha y) = (x, x) - 2\alpha(x, y) + \alpha^2(y, y)\,.$$

Wählt man nun speziell $\alpha := (x, y)/(y, y)$, so ergibt sich

$$0 \leq (x, x) - \frac{(x, y)^2}{(y, y)},$$

also

$$(x, y)^2 \leq (x, x)(y, y)\,.$$

Nun liefert Ziehen der Quadratwurzel die Behauptung. □

Mit Hilfe einer positiven, stetigen Gewichtsfunktion können wir nun das gewichtete L^2-Skalarprodukt einführen und definieren im Anschluss die Orthogonalität bzgl. dieses.

Satz 7.3.4 (Gewichtetes Skalarprodukt) *Es sei $\omega \in C(a, b)$, $\omega(x) > 0$ für $x \in (a, b)$ eine positive Gewichtsfunktion. Dann ist*

$$(f, g) := (f, g)_\omega := \int_a^b \omega(x) f(x) g(x)\, dx$$

für $f, g \in C[a, b]$ ein Skalarprodukt.

Beweis. Die Aussage ergibt sich unmittelbar aus der Verifizierung der bekannten Skalarprodukteigenschaften aus der Linearen Algebra (Additivität, Homogenität, Symmetrie, positive Definitheit) und wird deshalb an dieser Stelle ausgespart. □

Definition 7.3.5 (Orthogonalität) *Wir bezeichnen zwei Funktionen $f, g \in C[a,b]$ bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$ als **orthogonal**, falls gilt:*

$$(f,g)_\omega = 0.$$

Bemerkung 7.3.6 (Norm und Momente zum gewichteten Skalarprodukt) *Im Folgenden setzen wir voraus, dass die durch das Skalarprodukt induzierte Norm*

$$\| Q \| := \| Q \|_\omega := \sqrt{(Q,Q)_\omega}$$

für alle Polynome $Q \in \mathbb{P}_n$, $n \in \mathbb{N}$ wohldefiniert und endlich ist. Somit existieren auch die Momente

$$m_n := \int_a^b \omega(x) x^n \, dx,$$

*da mit der **Cauchy-Schwarz-Ungleichung** folgt:*

$$|m_n| = |(1, x^n)_\omega| \leq \|1\|_\omega \|x^n\|_\omega < \infty.$$

Beispiele 7.3.7 (Gewichtsfunktionen) Einige Beispiele von Gewichtsfunktionen seien hier genannt, die ungewöhnlich genug sind, um numerische Techniken zu erfordern, aber in praktischen Anwendungen verwendet werden.

- $\omega(x) = x^\alpha \log(1/x)$ auf $[0,1]$ mit $\alpha > 0$.

  Die Momente $m_n = (n+\alpha+1)^{-2}$ sind alle endlich und die zugehörigen Orthogonalpolynome werden verwendet, um Quadraturformeln zu konstruieren für Integrale über $[0,1]$, deren Integranden zwei Singularitäten bei Null haben, eine logarithmische und eine algebraische (falls $\alpha \neq 0, 1, 2, \ldots$).

- $\omega(x) = e^{-x}$ und $\omega(x) = e^{-x^2}$ auf $[0,c]$, mit $0 < c < \infty$.

  Dies sind **Laguerre** bzw. **Hermite**-Gewichte auf einem endlichen Intervall. Die Momente m_n lassen sich durch die unvollständige Gamma-Funktion $\gamma(\alpha, x) = \int_0^x t^{\alpha-1} e - t \, dt$ ausdrücken, nämlich $m_n = \gamma(n+1, c)$ bzw. $m_n = \frac{1}{2}\gamma(\frac{1}{2}(n-1), c^2)$. Beide Varianten finden Anwendung bei der Gauss-Quadratur von Integralen in der molekularen Quantenmechanik.

Definition 7.3.8 (Orthogonalpolynome) *Es sei $(P_n)_{n\in\mathbb{N}_0}$ eine Folge paarweise orthogonaler Polynome $P_n \in \mathbb{P}_n$, d.h.*

$$(P_i, P_j)_\omega = \delta_{ij} (P_i, P_i)_\omega \geq 0 \tag{7.15}$$

*und exakt vom Grad n, dann heißen die P_n **Orthogonalpolynome** über $[a,b]$ bzgl. der Gewichtsfunktion ω.*

Aufgabe 7.3.9 Man zeige, dass ein Polynom $P_n(x) = x^n + \ldots \in \mathbb{P}_n$ genau dann ein orthogonales Polynom n-ten Grades ist, wenn

$$\int P_n^2(x)\, \omega(x)\, dx = \min \left\{ \int Q_n^2(x)\, \omega(x)\, dx : Q_n(x) = x^n + \ldots \right\}$$

gilt.

Bemerkung 7.3.10 *1. Da die $P_0, \ldots, P_n$ eine Basis des $\mathbb{P}_n$ darstellen, folgt sofort*

$$(P_n, Q)_\omega = (P_n, \sum_{i=0}^{n-1} \alpha_i P_i)_\omega = 0 \quad \text{für alle } Q \in \mathbb{P}_{n-1}\,. \tag{7.16}$$

2. Die Definition der Orthogonalpolynome über (7.15) ist keineswegs eindeutig. Eine mögliche Standardisierung ist z.B.

$$P_n(x) = x^n + \tilde{a}_{n-1}x^{n-1} + \ldots\,,$$

d.h. der führende Koeffizient ist Eins.

Definition 7.3.11 (Monisches Polynom) *Ist der führende Koeffizient eines Polynoms Eins, so bezeichnet man dieses als* **monisch** *(oder auch normiert).*

Der folgende Satz liefert nun das bzgl. jedes positiven gewichteten L^2-Skalarprodukts eindeutig bestimmte orthogonale momische Polynome existieren. Weiterhin lassen sich diese numerisch durch eine Rekursionsvorschrift berechnen.

Satz 7.3.12 (Existenz und Eindeutigkeit monischer Orthogonalpolynome) *Zu jedem **positiv** gewichteten Skalarprodukt $(\cdot,\cdot)_\omega$ (also $\omega(x) > 0$) gibt es eindeutig bestimmte monische Orthogonalpolynome $P_n \in \mathbb{P}_n$ (d.h. mit führendem Koeffizienten Eins). Gilt zusätzlich $(x\,P_n, P_j)_\omega = (P_n, x\,P_j)_\omega$ für $j, n \geq 0$, dann erfüllen die Orthogonalpolynome die folgende Drei-Term-Rekursion:*

$$P_n(x) = (\alpha_n + x)P_{n-1}(x) + \gamma_n P_{n-2}(x), \quad n = 1, 2, \ldots \tag{7.17}$$

mit $P_{-1} := 0$, $P_0 := 1$ und

$$\alpha_n = -\frac{(xP_{n-1}, P_{n-1})_\omega}{(P_{n-1}, P_{n-1})_\omega}, \quad \gamma_n = -\frac{(P_{n-1}, P_{n-1})_\omega}{(P_{n-2}, P_{n-2})_\omega}. \tag{7.18}$$

Beweis. Man sieht unmittelbar, dass $P_0 \equiv 1 \in \mathbb{P}_0$ gilt, den Rest der Behauptung zeigen wir induktiv. Seien also $P_0, \ldots, P_{n-1}$ bereits bekannte paarweise orthogonale Polynome mit $P_j \in \mathbb{P}_j$ vom Grad j und führendem Koeffizienten gleich Eins. Aus diesen konstruieren wir nun P_n. Soll $P_n \in \mathbb{P}_n$ ebenso normiert sein, dann folgt zwangsläufig, dass

$$P_n(x) - xP_{n-1}(x) \quad \text{ist vom Grade } \leq n-1$$

und $P_0, \ldots, P_{n-1}, xP_{n-1}$ bilden eine Basis von $\mathbb{P}_n$. Da jedoch $P_0, \ldots, P_{n-1}$ eine Orthogonalbasis von $\mathbb{P}_{n-1}$ bzgl. $(\cdot,\cdot)_\omega$ bilden, gilt

$$P_n - xP_{n-1} = \sum_{j=0}^{n-1} \gamma_j P_j \;\text{ mit }\; \gamma_j := \frac{(P_n - xP_{n-1}, P_j)_\omega}{(P_j, P_j)_\omega}.$$

(Man setze die linke Gleichung in $(\cdot, P_\ell)_\omega$ für $\ell = 0, \ldots, n-1$ ein.) Außerdem folgt aus der Orthogonalität von P_n zu $P_0, \ldots, P_{n-1}$, dass nach Voraussetzung

$$\gamma_j = -\frac{(xP_{n-1}, P_j)_\omega}{(P_j, P_j)_\omega} = -\frac{(P_{n-1}, xP_j)_\omega}{(P_j, P_j)_\omega}\,.$$

Da $xP_j \in \mathbb{P}_{j+1}$, gilt somit $\gamma_j = 0$ für $j+1 < n-1$, d.h.

$$\gamma_0 = \cdots = \gamma_{n-3} = 0 \quad \text{bzw. es folgt} \quad P_n - xP_{n-1} = \gamma_{n-2}P_{n-2} + \gamma_{n-1}P_{n-1}\,.$$

Beachtet man $xP_{n-2} = P_{n-1} + \alpha_{n-2}P_{n-2} + \ldots + \alpha_0 P_0$, so folgt

$$\gamma_{n-2} = -\frac{(P_{n-1}, xP_{n-2})_\omega}{(P_{n-2}, P_{n-2})_\omega} = -\frac{(P_{n-1}, P_{n-1})_\omega}{(P_{n-2}, P_{n-2})_\omega}$$

und damit

$$\gamma_n = -\frac{(xP_n, P_n)_\omega}{(P_n, P_n)_\omega}\,.$$

Die Eindeutigkeit folgt induktiv mit Koeffizienten-Vergleich. Sei $Q_n \in \mathbb{P}_n$ eine weitere Folge orthogonaler Polynome. Wegen $Q_0 = P_0 = 1$ ist der Induktionsanfang klar. Weiter gilt

$$Q_n = \sum_{j=0}^{n} \alpha_j P_j \quad \text{mit} \quad \alpha_j = \frac{(Q_n, P_j)_\omega}{(P_j, P_j)_\omega}.$$

Aufgrund der Tatsache, dass der jeweilige führende Koeffizient eins ist, folgt $\alpha_n = 1$. Aus der Induktions-Voraussetzung ergibt sich $\alpha_0 = \cdots = \alpha_{n-1} = 0$, also $Q_n = P_n$. □

Bemerkung 7.3.13 *Die Abbildung*

$$(f,g) := -\int_{-1}^{1}\int_{-1}^{1} f(x)g(y)\log|x-y|\,dx\,dy$$

ist zwar ein Skalarprodukt auf der Menge der stetigen Funktionen, aber die zugehörigen Orthogonalpolynome erfüllen keine Drei-Term-Rekursion. Symmetrieüberlegungen zeigen sofort, dass für $0 < j+k$ *ungerade*

$$(x^j, x^k) = 0$$

gilt. Mit Hilfe von Maple verifiziert man leicht durch mehrfaches Anwenden der Anweisungen

```
j:=0;
k:=1;
-int(int(x^j*y^k*log((x-y)^2)/2,x=-1..1),y=-1..1);
```

zu verschiedenen $j, k \geq 0$*, dass gilt*

$$(x^j, x^k) = \begin{cases} 0 & \text{, falls } 0 \leq k+j \text{ ungerade,} \\ \neq 0 & \text{, falls } 0 \leq k+j \text{ gerade.} \end{cases}$$

Wir definieren $P_0 = 1$, $P_1 = x$ *und normieren weitere* P_k *so, dass der führende Koeffizient 1 ist. Damit das Orthogonalpolynom* $P_2 = x^2 + ax + b \in \mathbb{P}_2$ *orthogonal zu* x *ist, muss* $a = 0$ *gelten und aus* $(P_2, P_0) = 0$ *folgt* $b = -(x^2, 1)/(1, 1)$*. Man beachte, dass hier*

$$16/9 - 4/3\log(2) = (x^2, 1) \neq (x, x) = 1$$

gilt und somit die Formel der Drei-Term-Rekursion (7.17) *nicht gilt.*

Generell sind verschiedene Formen der Normierung der Orthogonalpolynome möglich. Wir führen die **allgemeine Darstellung der Orthogonalpolynome** in der Form

$$P_n(x) = k_n x^n + k'_n x^{n-1} + \ldots, \quad n = 0, 1, 2 \ldots . \tag{7.19}$$

ein. Zusätzlich definieren wir

$$h_n(P_n) := h_n := \int_a^b \omega(x) P_n^2(x) dx = \|P_n\|_\omega^2 . \tag{7.20}$$

Bemerkung 7.3.14 (Normierung der Orthogonalpolynome) *Bezüglich der allgemeinen Darstellung (7.19) und (7.20) sind Polynome* ***orthonormal****, wenn* $h_n(P_n) = 1$ *und* ***monisch****, falls* $k_n = 1$ *gilt .*

Übertragen wir nun die Drei-Term-Rekursion aus Satz 7.3.12 für Orthogonalpolynome mit führendem Koeffizienten Eins auf die allgemeinere Form (7.19), so erhalten wir folgendes Resultat. Hierbei kann man die Koeffizienten der Rekursionsformel aus den Leittermen der monischen Polynomen ablesen.

Satz 7.3.15 (Existenz und Eindeutigkeit allgemeiner orthogonaler Polynome) *Zu jedem Skalarprodukt* $(\cdot,\cdot)_\omega$ *gibt es eindeutig bestimmte Orthogonalpolynome* $P_n \in \mathbb{P}_n$ *mit führendem Koeffizienten* k_n*. Diese Polynome erfüllen die folgende Drei-Term-Rekursion:*

$$P_n(x) = (a_n + b_n x)P_{n-1} + c_n P_{n-2}, \quad n = 1, 2, \ldots, \tag{7.21}$$

mit $P_{-1} = 0$*,* $P_0 = k_0$*. Die Koeffizienten ergeben sich wie folgt:*

$$\begin{aligned} b_n &= \frac{k_n}{k_{n-1}}, \qquad a_n = b_n\left(\frac{k'_n}{k_n} - \frac{k'_{n-1}}{k_{n-1}}\right), \quad n = 1, 2, \ldots, \\ c_1 &= \text{bel.} < \infty, \quad c_n = -\frac{k_n k_{n-2} h_{n-1}}{k_{n-1}^2 h_{n-2}}, \quad n = 2, 3, \ldots . \end{aligned} \tag{7.22}$$

Beweis. Die Behauptung folgt aus Satz 7.3.12 mit der Darstellung (7.19). Sei $P_n(x) = k_n x^n + k'_n x^{n-1} + \ldots$, $n = 0, 1, 2 \ldots$, dann hat $\bar{P}_n(x) := P_n(x)/k_n$ führenden Koeffizienten Eins und Satz 7.3.12 liefert

$$P_n(x) = k_n \, \bar{P}_n(x) = k_n(\alpha_n + x)\bar{P}_{n-1}(x) + k_n\gamma_n\bar{P}_{n-2}(x), \quad n = 1, 2, \ldots \tag{7.23}$$

mit $\bar{P}_{-1} := 0$, $\bar{P}_0 := P_0/k_0 = 1$ und

$$\alpha_n = -\frac{(x\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}$$

sowie

$$\gamma_n = -\frac{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-2}, \bar{P}_{n-2})_\omega} = -\frac{1/k_{n-1}^2\,(P_{n-1}, P_{n-1})_\omega}{1/k_{n-2}^2\,(P_{n-2}, P_{n-2})_\omega} = -\frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\,. \tag{7.24}$$

Man beachte

$$x\bar{P}_{n-1} = x\left(x^{n-1} + \frac{k'_{n-1}}{k_{n-1}}x^{n-2} + \ldots\right) = \bar{P}_n + \left(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n}\right)\bar{P}_{n-1} + Q(x) \quad \text{mit } Q \in \mathbb{P}_{n-2}\,,$$

d.h.

$$\alpha_n = -\frac{(x\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega} = \frac{k'_n}{k_n} - \frac{k'_{n-1}}{k_{n-1}}\,. \tag{7.25}$$

Somit ergibt sich mit (7.24)

$$k_n\gamma_n\bar{P}_{n-2}(x) = -k_n\frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\frac{1}{k_{n-2}}P_{n-2}(x) = -\frac{k_nk_{n-2}}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}P_{n-2}(x)$$

bzw.

$$k_n\, x\, \bar{P}_{n-1}(x) = \frac{k_n}{k_{n-1}}\, x\, P_{n-1}(x)$$

und mit (7.25)

$$k_n\alpha_n\bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}}\left(\frac{k'_n}{k_n} - \frac{k'_{n-1}}{k_{n-1}}\right)P_{n-1}(x)\,.$$

Die letzten drei Gleichungen mit (7.3.1) liefern dann die Behauptung. □

Beispiel 7.3.16 (Historisch wichtige Polynome) In der nachfolgenden Tabelle werden einige, v.a. historisch wichtige Polynomterme P_n, ihre Gewichte und Standardisierungen sowie weitere zentrale Eigenschaften aufgelistet.

Wichtige Beispiele von Orthogonalpolynomen								
$P_n(x)$	Name	a	b	$\omega(x)$	Standard.	h_n	e_n	$g(x)$
$P_n(x)$	**Legendre**	-1	1	1	$P_n(1) = 1$	$\frac{2}{2n+1}$	$(-2)^n n!$	$1-x^2$
$T_n(x)$	**Tscheby.**	-1	1	$1/\sqrt{1-x^2}$	$T_n(1) = 1$	$\binom{\pi/2,\, n\neq 0}{\pi,\, n=0}$	$(-2)^n n!$	$1-x^2$
$L_n(x)$	**Laguerre**[3]	0	∞	e^{-x}	$k_n = (-1)^n/n!$	1	$1/n!$	x
$H_n(x)$	**Hermite**	$-\infty$	∞	e^{-x^2}	$k_n = 2^n$	$\sqrt{\pi}\,2^n\,n!$	$(-1)^n$	1
$P_n^{(\alpha,\beta)}(x)$	**Jacobi**	-1	1	$(1-x)^\alpha\cdot$	$P_n^{(\alpha,\beta)}(1) = \binom{n+\alpha}{n}$		$(-2)^n n!$	$1-x^2$
	$\alpha, \beta > -1$			$\cdot(1+x)^\beta$				

Für die **Jacobi**-Polynome gilt

$$h_n(P_n^{(\alpha,\beta)}) = \frac{2^{\alpha-\beta+1}}{2n+\alpha+\beta+1}\frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{n!\,\Gamma(n+\alpha+\beta+1)}\,.$$

Bemerkung 7.3.17 (Eigenschaften der Jacobi-Polynome) *Weiterhin gelten für die **Jacobi**-Polynome folgende Eigenschaften:*

(i) $P_n^{(0,0)}(x) = P_n(x)$

(ii) $P_n^{(-1/2,-1/2)}(x) = \binom{n-1/2}{n}T_n(x) = \frac{1}{4^n}\binom{2n}{n}T_n(x)$

(iii) $(1-x^2)\left(P_n^{(\alpha,\beta)}\right)'' + \left(\beta - \alpha - (\alpha+\beta+2)x\right)\left(P_n^{(\alpha,\beta)}\right)' + n(n+\alpha+\beta+1)P_n^{(\alpha,\beta)} = 0$

[3] Laguerre, Edmond Nicolas (1834-1886).

Bemerkung 7.3.18 (Spezialfall der Summenformel von Christoffel-Darboux) *Es gilt*

$$\begin{aligned}\sum_{i=0}^{n}\frac{P_i^2(x)}{h_i} &= \frac{k_n}{k_{n+1}h_n}\cdot\lim_{y\to x}\frac{P_{n+1}(x)P_n(y)-P_n(x)P_{n+1}(y)}{x-y}\\ &= \frac{k_n}{k_{n+1}h_n}\cdot\lim_{y\to x}\frac{\big(P_{n+1}(x)-P_{n+1}(y)\big)P_n(y)-P_{n+1}(y)\big(P_n(x)-P_n(y)\big)}{x-y}\\ &= \frac{k_n}{k_{n+1}h_n}\Big(P'_{n+1}(x)P_n(x)-P'_n(x)P_{n+1}(x)\Big).\end{aligned}$$

Damit gilt insbesondere für Nullstellen x^* von P_{n+1}

$$P'_{n+1}(x^*)\,P_n(x^*)=\frac{k_{n+1}h_n}{k_n}\sum_{i=0}^{n}\frac{P_i^2(x^*)}{h_i}\,. \tag{7.26}$$

Wir haben bereits in Abschnitt 6.2.4.2 gesehen, dass der Interpolationsfehler minimierert werden kann, wenn man als Stützstellen die Nullstellen der **Tschebyscheff**-Polynome, d.h. die Nullstellen spezieller Orthogonalpolynome, wählt. Die Nullstellen von Orthogonalpolynomen werden auch der Schlüssel für die **Gauß**-Quadratur sein.

Satz 7.3.19 (Einfachheit der Nullstellen von Orthogonalpolynomen) *Es sei $\omega \in C(a,b)$, $\omega(x)>0$, $x\in(a,b)$ eine positive Gewichtsfunktion, $(\cdot,\cdot)_\omega$ das zugehörige Skalarprodukt. Dann hat ein Orthogonalpolynom $P_k(x)$ von echtem Grad k genau k einfache Nullstellen in (a,b).*

Beweis. Es seien $x_0,\dots,x_\ell$ die $\ell+1\le k$ verschiedenen Nullstellen $a<x_i<b$, an denen P_k sein Vorzeichen wechselt. Das Polynom

$$Q(x):=(x-x_0)\cdot\dots\cdot(x-x_\ell)$$

wechselt dann an den gleichen Stellen sein Vorzeichen, sodass die Funktion $\omega(x)P_k(x)Q(x)$ ihr Vorzeichen in (a,b) nicht ändert ($\omega(x)$ ist eine positive Gewichtsfunktion) und daher gilt

$$(Q,P_k)_\omega=\int_a^b\omega(x)Q(x)P_k(x)\,dx\neq 0.$$

Da jedoch P_k auf allen Polynomen aus $\mathbb{P}_{k-1}$ senkrecht und $Q\in P_\ell$, $\ell<k$, steht, folgt unmittelbar: grad $Q=\ell+1\ge k$ und damit die Behauptung. □

Mit der numerischen Berechnung dieser Nullstellen werden wir uns in Abschnitt 7.3.4 im Rahmen der Gauß-Quadratur genauer beschäftigen.

Zum Abschluss dieses Abschnitts über Orthogonalpolynom betrachten wir nun noch drei Beispiele für Orthogonalpolynom genauer.

7.3.1.1 Tschebyscheff-Polynome

Die **Tschebyscheff**-Polynome T_n, welche wir bereits kennengelernt haben, sind orthogonal bezüglich des Skalarprodukts

$$(f,g)_\omega:=\int_{-1}^{1}\frac{f(y)g(y)}{\sqrt{1-y^2}}dy$$

und werden standardisiert durch $T_n(1)=1$. Durch Einsetzen dieser Eigenschaft in die Formeln von Satz 7.3.12 gewinnt man für $y\in\mathbb{R}$ die **Drei-Term-Rekursion** (6.18), d.h.,

$$T_0(y)=1,\quad T_1(y)=x,\quad T_k(y)=2yT_{k-1}(y)-T_{k-2}(y),\quad k\ge 2.$$

Die **Tschebyscheff**-Polynome besitzen weiterhin die folgenden **Eigenschaften**:

(i) Sie haben stets ganzzahlige Koeffizienten.

(ii) Der höchste Koeffizient von T_n ist $a_n = 2^{n-1}$.

(iii) T_n ist stets eine gerade Funktion, falls n gerade, und eine ungerade, falls n ungerade ist.

(iv) $T_n(1) = 1, T_n(-1) = (-1)^n$.

(v) $|T_n(y)| \leq 1$ für $y \in [-1, 1]$.

(vi) Die Nullstellen von $T_n(y)$ sind

$$y_k := \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = 0, 1, \ldots, n-1.$$

(vii)

$$T_k(y) = \begin{cases} \cos(k \cdot \arccos(y)), & |y| \leq 1 \ , \\ \cosh(k \cdot \operatorname{Arccosh}(y)), & y > 1 \ , \\ (-1)^k \cosh(k \cdot \operatorname{Arccosh}(-y)), & y < -1 \ . \end{cases}$$

$(viii)$ Die **Tschebyscheff**-Polynome besitzen die globale Darstellung

$$T_k(y) = \frac{1}{2}((y + \sqrt{y^2-1})^k + (y - \sqrt{y^2-1})^k), \quad \text{wobei } \ y \in \mathbb{R}.$$

(ix) $|T_n(y)|$ nimmt seinen maximalen Wert im Intervall $[-1, 1]$ an den sogenannten **Tschebyscheff**-Abszissen $\overline{y}_k = -\cos(\frac{k\pi}{n})$ für $k = 0, \ldots, n$ an, d.h.

$$|T_n(y)| = 1 \quad \Leftrightarrow \quad y = \overline{y}_k = -\cos\left(\frac{k\pi}{n}\right) \quad \text{mit } k = 0, \ldots, n. \tag{7.27}$$

Abschließend möchten wir für $n = 0, \ldots, 5$ die T_n explizit angeben und diese, sowie T_6, T_7 und T_8 auch anhand der Grafik 7.3 veranschaulichen:

$$\begin{aligned} T_0(y) &= 1, & T_3(y) &= 4y^3 - 3y, \\ T_1(y) &= y, & T_4(y) &= 8y^4 - 8y^2 + 1, \\ T_2(y) &= 2y^2 - 1, & T_5(y) &= 16y^5 - 20y^3 + 5y. \end{aligned}$$

7.3.1.2 Legendre-Polynome

Die **Legendre**-Polynome $P_n \in \mathbb{P}_n$, $n = 0, 1, 2, \ldots$ sind orthogonal bezüglich des L^2-Skalarprodukts auf $[-1, 1]$, d.h.

$$(f, g)_\omega := \int_{-1}^{1} f(y)g(y)\, dy,$$

und sind durch $P_n(1) = 1$ standardisiert.

Die **Legendre**-Polynome besitzen folgende **Eigenschaften**:

(i) Sie erfüllen die **Drei-Term-Rekursion**

$$P_0(y) = 1, \quad P_1(y) = x, \quad nP_n(y) = (2n-1)yP_{n-1}(y) - (n-1)P_{n-2}(y), \quad n \geq 2$$

(ii) $P_n(1) = 1, P_n(-1) = (-1)^n, \quad n = 0, 1, 2, \ldots$

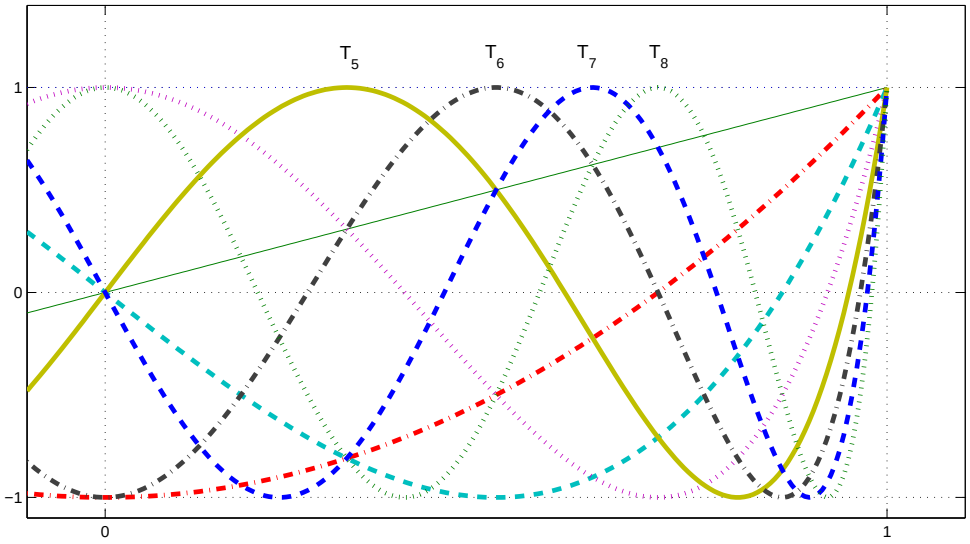


Abb. 7.3: Tschebyscheff-Polynome T_n, $n = 0, \ldots, 8$.

(iii)

$$\int_{-1}^{1} P_n^2(y)\,dy = \frac{2}{2n+1}, \quad n = 0, 1, 2, \ldots \tag{7.28}$$

(iv) Für Ableitung und Stammfunktion gilt

$$P_n'(y) = \frac{n(n+1)}{2n+1}\frac{P_{n+1}(y) - P_{n-1}(y)}{y^2-1}, \quad n \geq 1 \tag{7.29}$$

bzw.

$$\int_{-1}^{y} P_n(\xi)\,d\xi = \frac{1}{2n+1}\Big(P_{n+1}(y) - P_{n-1}(y)\Big), \quad n \geq 1\,. \tag{7.30}$$

Im Folgenden sind $P_0, \ldots, P_5$ explizit angegeben und in Grafik 7.4 zusammen mit P_6, P_7 und P_8 aufgezeichnet:

$$\begin{aligned} P_0(y) &= 1, & P_3(y) &= \frac{1}{2}(5y^3 - 3y), \\ P_1(y) &= y, & P_4(y) &= \frac{1}{8}(35y^4 - 30y^2 + 3), \\ P_2(y) &= \frac{1}{2}(3y^2 - 1), & P_5(y) &= \frac{1}{8}(63y^5 - 70y^3 + 15y). \end{aligned}$$

7.3.1.3 Jacobi-Polynome

Die **Jacobi**-Polynome $P_n^{(\alpha,\beta)}$ sind orthogonal bezüglich des durch die positive Gewichtsfunktion $w(y) = (1-y)^\alpha(1+y)^\beta, \alpha > -1, \beta > -1$ auf dem Intervall $(-1, 1)$ induzierten Skalarprodukts

$$(f, g)_\omega := \int_{-1}^{1} (1-y)^\alpha (1+y)^\beta f(y) g(y)\,dy.$$

Somit sind **Legendre**- und **Tschebyscheff**-Polynome spezielle **Jacobi**-Polynome (vgl. hierzu auch Bemerkung 7.3.17), für die wir im Folgenden eine Aussage über die Verteilung der Nullstellen wiedergeben:

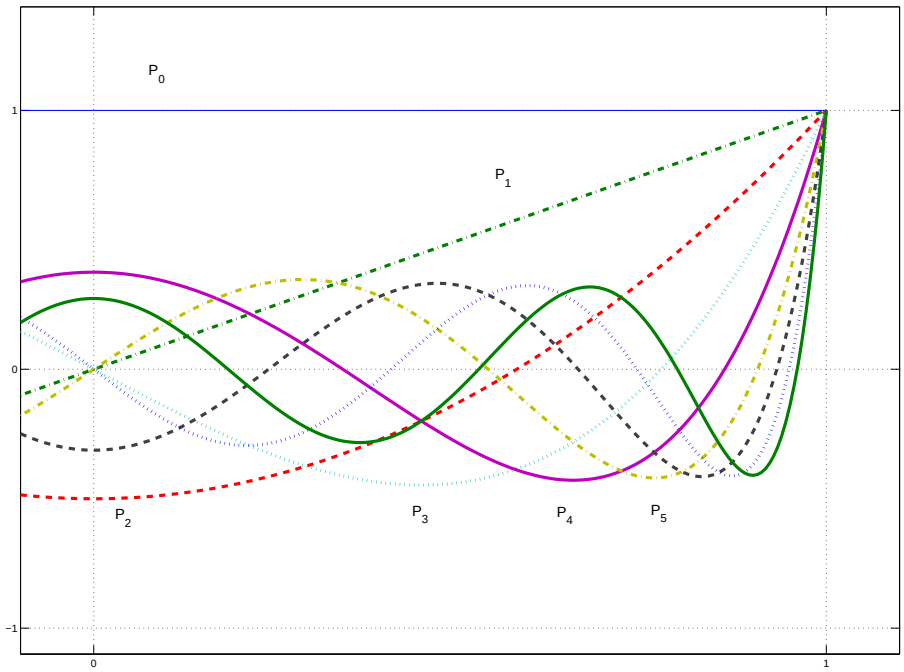


Abb. 7.4: **Legendre**-Polynome P_n, $n = 0, \ldots, 8$.

Satz 7.3.20 ([Sz]) *Seien* $|\alpha|, |\beta| \leq 1/2$ *und* $x_k = \cos\theta_k$ *die Nullstellen von* $P_n^{(\alpha,\beta)}(x)$ *in absteigender Folge, d.h.*

$$1 > x_0 > x_1 > \ldots > x_{n-1} > -1;\ 0 < \theta_0 < \theta_1 < \ldots < \theta_{n-1} < \pi\,.$$

Dann gilt

$$\frac{k+1+(\alpha+\beta-1)/2}{n+(\alpha+\beta+1)/2}\pi < \theta_k < \frac{k+1}{n+(\alpha+\beta+1)/2}\pi,\ \ k = 0,1,2,\ldots$$

und im Falle $\alpha = \beta$ *gilt*

$$\theta_k \geq \frac{k+1+\alpha/2-1/4}{n+\alpha+1/2}\pi,\ \ k = 0,1,2,\ldots,[(n-1)/2],$$

wobei die Gleichheit für $\alpha = \beta = -1/2$ *oder* $\alpha = \beta = 1/2$ *gilt.*

Beweis. Wir verzichten an dieser Stelle auf die Wiedergabe eines Beweises dieser Aussage und verweisen auf [Sz]. □

7.3.2 Konstruktion von Gauß-Quadraturen

Nun zurück zur Gauß-Quadratur, d.h. zur Konstruktion einer Quadraturformel mit maximaler Fehlerordnung $2n+2$ zu $n+1$ Stützstellen. Im Beweis des Satzes 7.3.1 haben wir gesehen, dass das Polynom

$$P(x) = \prod_{i=0}^{n}(x - x_i^{(n)})^2 = \prod_{i=0}^{n}(x - x_i^{(n)})(x - x_i^{(n)}) \in \mathbb{P}_{2n+2}$$

durch eine Quadraturformel $\hat{I}_n$ mit den Knoten $x_i^{(n)}$ **nicht** exakt integriert wird. Die Frage stellt sich, was man über die Knoten sagen kann, wenn man weiß , dass $\hat{I}_n$ maximalen Exaktheitsgrad $2n+1$ hat?

Lemma 7.3.21 (Charakterisierung via Orthogonalität) *Ist $\hat{I}_n$ der Form*

$$\hat{I}_n(f) = (b-a)\sum_{k=0}^{n} \lambda_k^{(n)} f(x_k^{(n)})$$

exakt vom Grad $2n+1$, dann gilt für die Polynome

$$P_{k+1}(x) := \prod_{i=0}^{k}(x - x_i^{(k)}) \in \mathbb{P}_{k+1}, \quad k = 0, \ldots, n \tag{7.31}$$

(die Stützstellen $x_i^{(k)}$ sind also die Nullstellen von P_{k+1}) die Beziehung

$$(P_j, P_{n+1})_\omega = \int_a^b P_j(x)P_{n+1}(x)\omega(x)\,dx = 0 \quad \textit{für alle } j = 0, \ldots, n.$$

Beweis. Wegen $P_k \in \mathbb{P}_k$ gilt $P_j P_{n+1} \in \mathbb{P}_{2n+1}$ für alle $0 \le j \le n$, also

$$\begin{aligned} \int_a^b P_j(x)P_{n+1}(x)\omega(x)dx &= I(P_jP_{n+1}) = \hat{I}_n(P_jP_{n+1}) \\ &= (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} P_j(x_i^{(n)}) \underbrace{P_{n+1}(x_i^{(n)})}_{=0} = 0. \end{aligned}$$

□

Bemerkung 7.3.22 *Nullstellen von orthogonalen Polynomen scheinen eine gute Wahl für die gesuchten optimalen Knoten zu sein!*

Damit sind auch die Gewichte bestimmt: falls $\hat{I}_n$ exakt von der Ordnung n ist, muss dies insbesondere für die **Lagrange**-Polynome $L_i^{(n)}$ bzgl. $x_k^{(n)}$ gelten.

$$\begin{aligned} 0 &= I(L_i^{(n)}) - \hat{I}_n(L_i^{(n)}) \\ &= \int_a^b L_i^{(n)}(x)\omega(x)\,dx - (b-a)\sum_{j=0}^{n} \lambda_j^{(n)} \underbrace{L_i^{(n)}(x_j^{(n)})}_{=\delta_{ij}} \\ &= \int_a^b L_i^{(n)}(x)\omega(x)\,dx - (b-a)\lambda_i^{(n)}, \end{aligned}$$

also

$$\lambda_i^{(n)} = \frac{1}{(b-a)} \int_a^b L_i^{(n)}(x)\omega(x)\,dx. \tag{7.32}$$

Damit gilt (erstaunlicherweise):

Satz 7.3.23 *Seien $x_k^{(n)}$, $k = 0, \ldots, n$ die Nullstellen von P_{n+1}, dann gilt für $\hat{I}_n$ mit den Gewichten (7.32) folgende Aussage: $\hat{I}_n$ ist exakt auf $\mathbb{P}_n$ genau dann, wenn $\hat{I}_n$ exakt auf $\mathbb{P}_{2n+1}$ ist.*

Beweis. „⇐“ ist trivial. Sei also umgekehrt $\hat{I}_n$ exakt auf $\mathbb{P}_n$ und $P \in \mathbb{P}_{2n+1}$. Mit dem Euklidischen Algorithmus folgt, dass Polynome $Q, R \in \mathbb{P}_n$ existieren mit

$$P = QP_{n+1} + R. \tag{7.33}$$

Die orthogonalen Polynome $(P_k)_{k\leq n}$ bilden eine Basis für $\mathbb{P}_n$ (aufgrund der linearen Unabhängigkeit), also folgt

$$(Q, P_{n+1})_\omega = 0$$

für alle $Q \in \mathcal{P}_n$ und damit (wegen $R \in P_n$)

$$I(P) = I(QP_{n+1}) + I(R) = \underbrace{(Q, P_{n+1})_\omega}_{=0} + I(R) = \hat{I}_n(R)$$

und andererseits

$$\begin{aligned} \hat{I}_n(R) &= (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} R(x_i^{(n)}) = (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} \left\{ Q(x_i^{(n)}) \underbrace{P_{n+1}(x_i^{(n)})}_{=0} + R(x_i^{(n)}) \right\} \\ &= \hat{I}_n(P), \end{aligned}$$

also $\hat{I}_n(P) = I(P)$ für alle $P \in \mathbb{P}_{2n+1}$. □

Wir fassen dies Überlegungen in dem folgenden Satz zusammen:

Satz 7.3.24 (Gauß-Christoffel-Quadratur) *Es sei $\omega \in C(a,b)$ eine positive Gewichtsfunktion auf (a,b) und $(P_k)_{k\in\mathbb{N}_0}$ eine Folge von Orthogonalpolynomen bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$. Des Weiteren sei $n \in \mathbb{N}$ und $x_0 < \cdots < x_n$ die Nullstellen von $P_{n+1}(x)$. Dann existieren (von n abhängige) eindeutig bestimmte Gewichte $\lambda_0, \cdots, \lambda_n \in \mathbb{R}_+$, so dass gilt:*

$$\int_a^b \omega(x)P(x)\,dx = \lambda_0\,P(x_0) + \ldots + \lambda_n\,P(x_n) \quad \text{für alle } P \in \mathbb{P}_{2n+1}\,. \tag{7.34}$$

Beweis. Es sei P_{n+1} das bis auf Vielfachheit eindeutig bestimmte Orthogonalpolynom bzgl. $(\cdot,\cdot)_\omega$ vom echten Grad $n+1$. Dann existiert zu jedem $P \in \mathbb{P}_{2n+1}$ die eindeutige Darstellung $P = QP_{n+1} + R$ mit $Q, R \in \mathbb{P}_n$. Das Ausnutzen der Orthogonalität von $(P_{n+1}, x^k)_\omega = 0$ für $k = 0, \ldots, n$ liefert

$$\int_a^b \omega(x)P(x)\,dx = \int_a^b \omega(x)Q(x)P_{n+1}(x)\,dx + \int_a^b \omega(x)R(x)\,dx = \int_a^b \omega(x)R(x)\,dx. \tag{7.35}$$

Seien $L_0^{(n)}(x), \ldots, L_n^{(n)}(x) \in \mathbb{P}_n$ die **Lagrange**-Polynome zu $x_0, \ldots, x_n$. Man beachte die Eigenschaft $L_k(x_j) = \delta_{jk}$, wobei δ_{jk} das Kronecker-Symbol ist. Es gilt dann folgende Darstellung

$$\psi(x) = \sum_{i=0}^{n} \psi(x_i) L_i^{(n)}(x)$$

und somit für (7.35)

$$\int_a^b \omega(x)P(x)\,dx = \int_a^b \omega(x)\left(\sum_{i=0}^{n} R(x_i)L_i^{(n)}(x)\right)\,dx = \sum_{i=0}^{n} R(x_i)\int_a^b \omega(x)L_i^{(n)}(x)\,dx.$$

Dies ist aber genau die Darstellung (7.34) mit $\lambda_i = \int_a^b \omega(x)L_i^{(n)}(x)dx$, da an den Nullstellen von P_{n+1} gilt $P(x_i) = Q(x_i)\,P_n(x_i) + R(x_i) = R(x_i)$.

Beweisen wir nun die **Eindeutigkeit** der λ_i. Dazu seien $\lambda_0, \ldots, \lambda_n$ und $\widetilde{\lambda_0}, \ldots, \widetilde{\lambda_n}$ zwei verschiedene Wahlen der Koeffizienten in (7.34). Durch Differenzbildung erhalten wir also

$$0 = \sum_{0=1}^{n} R(x_i)(\lambda_i - \widetilde{\lambda_i}),$$

setzen wir nacheinander $R(x) = L_k^{(n)}(x)$, $k = 0, \ldots, n$, so folgt sofort $\lambda_k = \widetilde{\lambda_k}$ für alle $k \in \{0, 1 \ldots, n\}$.

Zuletzt müssen wir noch die **Positivität** der λ_i nachweisen. Setzen wir dazu $P(x) = (L_k^{(n)}(x))^2$, dann folgt mit $\lambda(x) > 0$, $x \in (a,b)$ und (7.34)

$$0 < \int_a^b \omega(x)\left(L_k\right)^{(n)}(x)\right)^2\,dx = \sum_{0=1}^{n} \lambda_i\,\left(L_k^{(n)}(x_i)\right)^2 = \lambda_k$$

und damit die Behauptung. □

Bemerkung 7.3.25 (Gauß-Christoffel-Quadratur) *(a) Für $\omega = 1$ spricht man von **Gauß - Quadratur**, allgmein von **Gauß-Christoffel-Quadratur**.*

(b) Nachteile sind einerseits, dass die Berechnung der Gewichte und Knoten vom Integrationsintervall abhängt, und andererseits, dass diese für jedes n neu berechnet werden müssen. Eine nachträgliche Erhöhung der Genauigkeit durch Aufdatierung ist so noch nicht möglich, es muss alles neu berechnet werden!

(c) Die Berechnung der Gewichte und Knoten für beliebige ω wird mit der Drei-Term-Rekursion gemacht. Dies behandeln wir im nächsten Abschnitt genauer.

*(d) Integrale auf mehrdimensionalen Gebieten („**Kubatur**“)*

$$[a_1, b_1] \times \cdots \times [a_d, b_d]$$

*kann man mit **Tensorprodukt-Quadratur-Formeln***

$$\hat{I}_{n_1}(x_1) \cdot \cdots \cdot \hat{I}_{n_d}(x_d)$$

*berechnen. Die Komplexität steigt jedoch stark mit der Raumdimension d („**Fluch der Dimensionen**“). Mögliche Auswege sind Monte-Carlo, Quasi-Monte-Carlo, Dünngitter (siehe Numerical Finance).*

Nun zur Darstellung des Fehlers. Analog zu den interpolatorischen Quadraturformeln kann man vom Exaktheitsgrad auf den Approximationsfehler schließen.

Satz 7.3.26 (Integrationsfehler der Gauß-Quadratur) *Für $f \in C^{2n+2}$ gilt*

$$E_n(f) = I(f) - \hat{I}_n(f) = \frac{f^{2n+2}(\xi)}{(2n+2)!} \|P_{n+1}\|_\omega^2 \tag{7.36}$$

mit einem $\xi \in (a, b)$.

Beweis. Betrachte den Hermite-Interpolanten $P \in \mathbb{P}_{2n+1}$ an f zu den jeweils doppelten Knoten $x_k^{(n)}$, $k = 0, \ldots, n$. Dann folgt

$$f(x) - P(f|x_0^{(n)}, x_0^{(n)}, \ldots, x_n^{(n)}, x_n^{(n)})(x) = \underbrace{(x - x_0^{(n)})^2 \ldots (x - x_n^{(n)})^2}_{=P_{n+1}(x)^2} \frac{f^{2n+2}(\xi)}{(2n+2)!},$$

also

$$I(f) = \underbrace{I(P)}_{=\hat{I}_n(P)} + \frac{f^{2n+2}(\xi)}{(2n+2)!} I(P_{n+1}^2) = \underbrace{(b-a) \sum_{i=0}^{n} \lambda_i^{(n)} \underbrace{P(x_i^{(n)})}_{=f(x_i^{(n)})}}_{=\hat{I}_n(f)} + \frac{f^{2n+2}(\xi)}{(2n+2)!} \underbrace{(P_{n+1}, P_{n+1})_\omega}_{=\|P_{n+1}\|_\omega^2}.$$

□

7.3.3 Gauß-Lobatto-Integration

Mit den n Nulstellen y_k, $k = 0, \ldots, n-1$, der Tschebyscheff-Polynome T_n haben wir eine Quadraturformel vom Exaktheitsgrad $2n+1$ erreicht und haben gesehen, dass dies optimal ist, also der maximal erreichbare Exaktheitsgrad. Allerdings liegen alle $y_k \in (-1, 1)$, also im Inneren des Intervalls. Manchmal ist es aber notwendig (oder zumindest hilfreich), auch die Endpunkte des Intervalls als Knoten zu verwenden. Man kann analog diejenigen Knoten bestimmen, die unter Einbeziehung von ± 1 die maximale Ordnung garantieren – diese maximale Ordnung ist dann $2n-1$, man verliert also genau 2 Ordnungen durch die beiden Randpunkte. Diese optimalen Knoten sind die Nullstellen von T_n', also die Tshcebyscheff-Abszissen aus (7.27). Der so entstehende Interpolant heißt **Gauß-Lobatto-Interpolant** P_n^{GL}, für den die Fehlerabschätzung

$$\|f - P_n^{\mathrm{GL}} f\|_\infty \le C(f)\, n^{-s} \quad \text{falls} \quad f^{(k)} \in L_2(-1,1)\, \forall k = 0, \ldots, s,$$

gilt. Ebenso definiert man die **Gauß-Lobatto-Quadratur** I_n^{GL} mit

$$|I(f) - I_n^{\mathrm{GL}} f| \le C(f)\, n^{-s} \quad \text{falls} \quad f^{(k)} \in L_2(-1,1)\, \forall k = 0, \ldots, s.$$

Tab. 7.1: Momente für einige Standardfälle von Orthogonalpolynomen

Name	a	b	$\omega(x)$	$m_k := \int_a^b \omega(x)x^k\,dx$
Legendre	-1	1	1	$m_k = \begin{cases} 2/(k+1) & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Tscheby.	-1	1	$\frac{1}{\sqrt{1-x^2}}$	$m_k = \begin{cases} \pi \cdot \frac{1\cdot3\cdot5\cdot\ldots\cdot k-1}{2\cdot4\cdot6\cdot\ldots\cdot k} & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Laguerre	0	∞	e^{-x}	$m_k = k!$
Hermite	$-\infty$	∞	e^{-x^2}	$m_k = \begin{cases} \Gamma(\frac{k+1}{2}) & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Jacobi $\alpha = \beta > -1$	-1	1	$(1-x)^\alpha(1+x)^\beta$	$m_k = \begin{cases} \sqrt{\pi}\frac{1\cdot3\cdot5\cdot\ldots\cdot k-1}{2^{k/2}}\frac{\Gamma(\alpha+1)}{\Gamma(\alpha+(k+3)/2)} & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
„log“	0	1	$-\log(x)$	$m_k = 1/(k+1)^2$

7.3.4 Berechnung der Knoten und Gewichte

Aus der Summenformel von **Christoffel-Darboux** ergibt sich die folgende Darstellung der Gewichte λ_k zu den Knoten $x_k, k = 0, 1, \ldots, n$.

Satz 7.3.27 (Darstellung der Gewichte) *Für die Gewichte λ_k, $k = 0, 1, \ldots, n$ in Satz 7.3.24 gelten die Darstellungen*

$$\lambda_k = \frac{k_{n+1}\,h_n}{P'_{n+1}(x_k)\,P_n(x_k)\,k_n} = \left(\sum_{i=0}^{n} \frac{P_i^2(x_k)}{h_i}\right)^{-1}, \quad k = 0, 1, \ldots, n\,,$$

wobei x_k die Nullstellen von P_{n+1} seien.

Beweis. Es sei $x_k, k \in \{0, 1, \ldots, n\}$, eine Nullstelle von P_{n+1}. Dann gilt

$$\lim_{x\to x_k} \frac{P_{n+1}(x)}{x - x_k} = \lim_{x\to x_k} \frac{P_{n+1}(x) - P_{n+1}(x_k)}{x - x_k} = P'_{n+1}(x_k)\,.$$

Wir definieren

$$Q(x) = \begin{cases} P_{n+1}(x)\,P_n(x)/(x - x_k) & \text{falls } x \in \mathbb{R} \setminus \{x_k\} \\ P'_{n+1}(x_k)\,P_n(x_k) & \text{falls } x = x_k \end{cases}.$$

Da $Q \in \mathbb{P}_{2n}$, gilt

$$\int_a^b \omega(x)Q(x)\,dx = \sum_{j=0}^{n} \lambda_j\,Q(x_j) = \lambda_k\,Q(x_k) = \lambda_k\,P'_{n+1}(x_k)\,P_n(x_k)\,. \tag{7.37}$$

Nutzen wir die Darstellung $P_{n+1}(x) = \frac{k_{n+1}}{k_n}(x - x_k)P_n(x) + (x - x_k)R(x)$ mit einem $R \in \mathbb{P}_{n-1}$, so ergibt sich

$$\begin{aligned} \int_a^b \omega(x)Q(x)\,dx &= \int_a^b \omega(x)\left(\frac{k_{n+1}}{k_n}P_n(x) + R(x)\right)P_n(x)\,dx \\ &= \frac{k_{n+1}}{k_n}\int_a^b \omega(x)P_n^2(x)\,dx = \frac{k_{n+1}\,h_n}{k_n}\,. \end{aligned} \tag{7.38}$$

Aus (7.37) und (7.38) erhalten wir die erste und daraus mit (7.26) die zweite Darstellung . □

Bemerkung 7.3.28 *Umfangreiche Tabellen von Stützstellen und Gewichten zu verschiedenen Gewichtsfunktionen findet man z.B. in [AS] und [St].*

Bemerkung 7.3.29 (Gauß-Qadratur in höheren Dimensionen) *In höheren Dimensionen hat man eine solche Theorie der Orthogonalpolynome nicht, aus der man Quadraturformeln gewinnen könnte. Hier bleibt einem häufig nichts anderes übrig, als diese näherungsweise als Lösung nichtlinearer Gleichungen zu bestimmen.*

7.4 Numerische Differenziation

Gegeben sei wiederum eine Funktion $f : [a, b] \to \mathbb{R}$ mit $f \in C^1[a, b]$. Wir suchen eine Näherung der Ableitung $f'(x)$, $x \in (a, b)$. Die einfachste Methode sind **klassische finite Differenzen** unter Ausnutzung der Definition der Ableitung

$$f'(x) = \lim_{h \to 0+} \frac{f(x+h) - f(x)}{h}.$$

Führt man also Stützstellen $x_k = a + kh$, $h := \frac{b-a}{n}$, $k = 0, \dots, n$, ein, so erhält man die Approximation

$$u_i^{\text{FD}} := \frac{1}{h}(f(x_{i+1}) - f(x_i)).$$

Ähnlich wie bei Quadraturformeln zeigt man mit dem Satz von Taylor die Abschätzung

$$|f'(x_i) - u_i^{\text{FD}}| \le \frac{h}{2}\|f''\|_\infty, \quad \text{falls } f \in C^2(a, b)$$

mit der Supremunms-Norm $\|f\|_\infty := \sup_{x \in (a,b)} |f(x)|$. Alternativ kann man den **zentralen Differenzenquotienten**

$$u_i^{\text{ZD}} := \frac{1}{2h}(f(x_{i+1}) - f(x_{i-1}), \quad 1 \le i \le n-1,$$

betrachten. Man zeigt leicht

$$|f'(x_i) - u_i^{\text{ZD}}| \le \frac{h^2}{6}\|f'''\|_\infty, \quad \text{falls } f \in C^3(a, b).$$

Bemerkung 7.4.1 (Merkregel) *Mehr Regularität (Glattheit) kann zu besserer Approximation führen.*

Ähnlich kann man höhere Ableitungen approximieren, z.B.

$$u_i^{\text{ZD2}} := \frac{1}{h^2}\big(f(x_{i+1}) - 2f(x_i) + f(x_{i-1})\big)$$

mit

$$|f''(x_i) - u_i^{\text{ZD2}}| \le \frac{h^2}{12}\|f''''\|_\infty, \quad \text{falls } f \in C^4(a, b).$$

Eine besonders effiziente Ableitungsapproximation beruht auf der Gauß-Lobatto-Interpolation aus §7.3.3. Die so-genannte **pseudo-spektrale Ableitung** ist definiert als

$$\mathcal{D}_n f := (P_n^{\text{GL}} f)' \in \mathbb{P}_{n-1}$$

und es gilt die Abschätzung

$$\|f' - \mathcal{D}_n f\|_\omega \le C(f)\, n^{1-m}, \quad \text{falls } f^{(k)} \in L_2(-1, 1),\ \forall k = 0, \dots, m.$$

Die Approximationsrate ist also **exponentiell** (wenn f genüzend glatt ist), daher der Namensbestandteil "spektral". Außerden erhält man gleichzeitig die Approximation der Ableitung nicht nur an einem Punkt, sondern auf einem Gitter: Mit den Tschebyscheff-Abszissen $\bar{y}_k$ aus (7.27) gilt

$$(\mathcal{D}_n f)(\bar{y}_k) = \sum_{j=0}^{n} f(\bar{y}_j)\, \bar{\ell}'_j(\bar{y}_k), \quad k = 0, \ldots, n,$$

mit den Lagrange-Basispolynomen $\bar{\ell}_j$ zu den Knoten $\bar{y}_k$, $k = 0, \ldots, n$. Da man die Werte $\bar{\ell}'_j(\bar{y}_k)$ einmalig vorab berechnen kann, kann man $(\mathcal{D}_n f)(\bar{y}_k)$ für alle $k = 0, \ldots, n$ nur mit den Werten $f(\bar{y}_j)$ und einer Matrix-Vektor-Multiplikation berechnen. Dies hat vielfältige Anwendungen.

A LANDAU-SYMBOLE

Landau-Symbole beschreiben das asymptotische Verhalten von Funktionen. Genauer vergleichen sie das gesuchte asymptotische Verhalten einer Funktion f mit dem bekannten asymptotischen Verhalten einer Referenzfunktion g.

A.1 Definition

Ist $x_0 \in \mathbb{R}$ oder $x_0 = \infty$ und sind f und g in einer Umgebung von x_0 definierte reellwertige Funktionen, wobei $g(x) > 0$ vorausgesetzt wird, dann haben die Symbole $\mathcal{O}(\cdot)$ und $o(\cdot)$ die folgenden Bedeutungen:

Definition A.1.1 *(a)* $f(x) = \mathcal{O}(g(x))$ *für* $x \to x_0$ *bedeutet, dass* $\frac{f(x)}{g(x)}$ *für* $x \to x_0$ *beschränkt ist.*

(b) $f(x) = o(g(x))$ *für* $x \to x_0$ *bedeutet, dass* $\lim_{x \to x_0} \frac{f(x)}{g(x)} = 0$.

Wenn x_0 aus dem Zusammenhang bekannt ist, schreibt man häufig nur $f(x) = \mathcal{O}(g(x))$ bzw. $f(x) = o(g(x))$ Die gleichen Symbole werden sinngemäß auch für das Wachstum von Folgen für $n \to \infty$ verwendet.

A.2 Beispiele

Beispiel A.2.1 (Landau-Symbole)

1. $f(x) = \mathcal{O}(1)$ bedeutet, dass f beschränkt ist, z.B. $\sin(x) = \mathcal{O}(1)$.
2. $f(x) = \mathcal{O}(x)$ bedeutet, dass f für $x \to \infty$ höchstens linear wächst. Beispiel: $\sqrt{1+x^2} = \mathcal{O}(x)$, $\log(x) = \mathcal{O}(x)$.
3. $f(x) = o(x)$ bedeutet, dass f für $x \to \infty$ langsamer als jede lineare Funktion wächst. Beispiele: $\sqrt{x} = o(x)$, $\log(x) = o(x)$.
4. $\displaystyle\sum_{k=1}^{n} k = \frac{1}{2} n(n+1) = \mathcal{O}(n^2)$.

In der Numerik werden Landau-Symbole häufig verwendet, um den Aufwand eines Algorithmus abzuschätzen.

Beispiel A.2.2 Die Berechnung eines Skalarprodukts zweier Vektoren $x, y \in \mathbb{R}^n$ erfordert n Produkte und $n-1$ Additionen:

$$x^T y = \sum_{i=1}^{n} x_i \, y_i.$$

Zählt man jede Addition und jedes Produkt als gleich aufwändige Gleitpunktoperationen, ergeben sich $2n-1$ Operationen. Diese genaue Zahl ist aber weniger interessant als die Tatsache, dass der Gesamtaufwand linear mit der Dimension steigt: verdoppelt man die Länge der Vektoren, verdoppelt sich (ungefähr!) der Aufwand. Man sagt: das Skalarprodukt besitzt die Komplexität $\mathcal{O}(n)$.

Beispiel A.2.3 Bei der Multiplikation zweier Matrizen $A, B \in \mathbb{R}^{n\times n}$ sind n^2 Elementen des Produkts zu bestimmen. Jedes Element wird mit Hilfe eines Skalarprodukts berechnet. Insgesamt benötigt man also $n^2(2n-1) = \mathcal{O}(n^3)$ Gleitpunktoperationen.

B NORMEN

Normen erfüllen den gleichen Zweck auf Vektorräumen, den Beträge auf der reellen Achse erfüllen. Genauer gesagt, $\mathbb{R}^n$ mit einer Norm auf $\mathbb{R}^n$ liefert einen metrischen Raum. Daraus ergeben sich bekannte Begriffe wie Umgebung, offene Menge, Konvergenz und Stetigkeit für Vektoren und vektorwertige Funktionen.
Wir beginnen mit der Vereinbarung einer Notation.

Notation B.0.1 *Für $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ definieren wir $|x| \in \mathbb{R}^n$ durch*

$$|x| = (|x_1|, \ldots, |x_n|)^T .$$

Für eine Matrix $A \in \mathbb{R}^{m\times n}$ definieren wir die Matrix $|A| \in \mathbb{R}^{m\times n}$ durch

$$|A|_{ij} = |a_{ij}|, \quad 1 \le i \le m,\, 1 \le j \le n.$$

Definition B.0.2 (Vektornorm) *Eine Vektornorm auf $\mathbb{R}^n$ ist eine Funktion $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ mit den Eigenschaften*

$$\begin{aligned} &\|x\| \ge 0 && x \in \mathbb{R}^n, && \|x\| = 0 \Leftrightarrow x = 0, \\ &\|x + y\| \le \|x\| + \|y\| && x, y \in \mathbb{R}^n, \\ &\|\alpha x\| = |\alpha|\,\|x\| && \alpha \in \mathbb{R},\, x \in \mathbb{R}^n. \end{aligned} \tag{B.1}$$

Eine nützliche Klasse von Vektornormen sind die **p-Normen** ($p \in \mathbb{N}$), definiert durch

$$\|x\|_p = \left(\sum_{k=0}^{n} |x_k|^p \right)^{\frac{1}{p}} . \tag{B.2}$$

Von diesen sind besonders die 1, 2 und ∞ interessant:

$$\begin{aligned} \|x\|_1 &= |x_1| + \ldots + |x_n|, \\ \|x\|_2 &= (|x_1|^2 + \ldots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}}, \\ \|x\|_\infty &= \max_{1\le j\le n} |x_j|. \end{aligned} \tag{B.3}$$

Ein **Einheitsvektor** bzgl. der Norm $\|\cdot\|$ ist ein Vektor x der Lïage 1, d.h. mit $\|x\| = 1$.

B.1 Vektornorm-Eigenschaften

Ein klassisches Ergebnis bzgl. p-Normen ist die **Hölder-Ungleichung**

$$|x^T y| \le \|x\|_p \, \|y\|_q \qquad \text{mit } \frac{1}{p} + \frac{1}{q} = 1. \tag{B.4}$$

Spezialfall der Hölder-Ungleichung ist die **Cauchy-Schwarz-Ungleichung** (CSU)

$$|x^T y| \le \|x\|_2 \, \|y\|_2. \tag{B.5}$$

Alle Normen auf $\mathbb{R}^n$ sind äquivalent, d.h. es seien $\|\cdot\|_\alpha$ und $\|\cdot\|_\beta$ Normen auf $\mathbb{R}^n$, dann existieren positive Konstanten c_1, c_2, so dass

$$c_1 \, \|x\|_\alpha \le \|x\|_\beta \le c_2 \, \|x\|_\alpha \qquad (x \in \mathbb{R}^n). \tag{B.6}$$

Zum Beispiel sei $x \in \mathbb{R}^n$, dann gilt

$$\begin{aligned} \|x\|_2 &\leq \|x\|_1 \leq \sqrt{n}\,\|x\|_2, \\ \|x\|_\infty &\leq \|x\|_2 \leq \sqrt{n}\,\|x\|_\infty, \\ \|x\|_\infty &\leq \|x\|_1 \leq n\,\|x\|_\infty. \end{aligned} \tag{B.7}$$

Aufgabe B.1.1 Man beweise die Aussagen (B.4)–(B.7).

B.2 Matrixnormen

Da $\mathbb{R}^{m\times n}$ isomorph (d.h. es existiert eine bijektive Abb. mit $\varphi(\lambda A + \mu B) = \lambda\varphi(A) + \mu\varphi(B)$) zu $\mathbb{R}^{m\cdot n}$ ist, sollte die Definition einer Matrixnorm äquivalent sein zur Definition einer Vektornorm.

Definition B.2.1 (Matrixnorm) *Es sei $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$. Dann ist $\|\cdot\|$ eine Matrixnorm, wenn folgende Eigenschaften gelten:*

$$\begin{aligned} &\|A\| \geq 0 && A \in \mathbb{R}^{m\times n}, \qquad \|A\| = 0 \Leftrightarrow A = 0) \\ &\|A + B\| \leq \|A\| + \|B\| && A, B \in \mathbb{R}^{m\times n} \\ &\|\alpha A\| = |\alpha|\,\|A\| && \alpha \in \mathbb{R}, A \in \mathbb{R}^{m\times n}. \end{aligned} \tag{B.8}$$

Bemerkung B.2.2 *Die mit am häufigsten verwendeten Normen in der Numerischen Linearen Algebra sind die* ***Frobenius–Norm***

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2} \tag{B.9}$$

*und die p–**Normen***

$$\|A\|_p := \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}\,. \tag{B.10}$$

Man beachte folgende äquivalente Definition

$$\|A\|_p = \sup_{x\neq 0} \left\| A\,\frac{x}{\|x\|} \right\|_p = \sup_{\|x\|_p=1} \|Ax\|_p\,. \tag{B.11}$$

Bemerkung B.2.3 (Submultiplikativität) *Die Frobenius-Norm und die p-Normen erfüllen zusätzlich auch noch die Eigenschaft der* **Submultiplikativität**, *d.h.*

$$\|A \cdot B\| \leq \|A\|\,\|B\|. \tag{B.12}$$

Bemerkung B.2.4 *Es sei y ein Vektor mit $\|y\|_p = 1$, dann gilt*

$$\begin{aligned} \|A\cdot B\|_p &= \max_{x\neq 0} \frac{\|ABx\|_p}{\|x\|_p} = \max_{x\neq 0} \frac{\|ABx\|_p\,\|Bx\|_p}{\|Bx\|_p\,\|x\|_p} \\ &\leq \max_{x\neq 0} \frac{\|ABx\|_p}{\|Bx\|_p} \cdot \max_{x\neq 0} \frac{\|Bx\|_p}{\|x\|_p} \\ &= \max_{y\neq 0} \frac{\|Ay\|_p}{\|y\|_p} \cdot \max_{x\neq 0} \frac{\|Bx\|_p}{\|x\|_p} = \|A\|_p\,\|B\|_p\,. \end{aligned}$$

Bemerkungen B.2.5 *i) Nicht alle Matrix-Normen erfüllen diese Eigenschaft, z.B. gilt für* $\|A\|_A := \max |a_{ij}|$ *und* $A = B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ *die Ungleichung*

$$\|A \cdot B\|_A = 2 > \|A\|_A \, \|B\|_A = 1 \, .$$

ii) Für die p*-Normen haben wir die wichtige Eigenschaft, dass für alle* $A \in \mathbb{R}^{m \times n}$ *und* $x \in \mathbb{R}^n$

$$\|Ax\|_p \leq \|A\|_p \, \|x\|_p. \tag{B.13}$$

Die Frobenius und p-Normen (speziell $p = 1, 2, \infty$) erfüllen gewisse Ungleichungen, welche in der Analysis von Matrixberechungen verwendet werden. Es sei $A \in \mathbb{R}^{m \times n}$, dann gilt

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n} \, \|A\|_2, \tag{B.14}$$

$$\max_{i,j} |a_{ij}| \leq \|A\|_2 \leq \sqrt{m \, n} \max_{i,j} |a_{ij}|, \tag{B.15}$$

$$\frac{1}{\sqrt{n}} \, \|A\|_\infty \leq \|A\|_2 \leq \sqrt{m} \, \|A\|_\infty, \tag{B.16}$$

$$\frac{1}{\sqrt{m}} \, \|A\|_1 \leq \|A\|_2 \leq \sqrt{n} \, \|A\|_1. \tag{B.17}$$

Des Weiteren gilt für $A \in \mathbb{R}^{m \times n}$

$$\begin{aligned} \|A\|_\infty &= \max_{\|x\|_\infty = 1} \|Ax\|_\infty = \max_{\|x\|_\infty = 1} \left\{ \max_{1 \leq j \leq m} \left| \sum_{k=1}^{n} a_{jk} x_k \right| \right\} \\ &= \max_{1 \leq j \leq m} \left\{ \max_{\|x\|_\infty = 1} \left| \sum_{k=1}^{n} a_{jk} x_k \right| \right\} = \max_{1 \leq j \leq m} \sum_{k=1}^{n} |a_{jk}|. \end{aligned}$$

Aufgrund dieser Gleichung bezeichnet man $\|\cdot\|_\infty$ auch als **Zeilensummennorm**. Diese ist mittels der letzten Gleichung auch leicht zu bestimmen, d.h.

$$\|A\|_\infty = \max_{1 \leq j \leq m} \sum_{k=1}^{n} |a_{jk}| \, . \tag{B.18}$$

Analog gilt für die 1-Norm:

$$\begin{aligned} \|A\|_1 &= \max_{\|x\|_1 = 1} \|Ax\|_1 = \max_{\|x\|_1 = 1} \sum_{j=1}^{m} \left| \sum_{k=1}^{n} a_{jk} x_k \right| = \max_{\|x\|_1 = 1} \sum_{j=1}^{m} \sum_{k=1}^{n} |a_{jk}| \, |x_k| \, \operatorname{sign}(a_{jk} x_k) \\ &= \max_{\|x\|_1 = 1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}| \, \operatorname{sign}(a_{jk} x_k) = \max_{\|x\|_1 = 1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}| = \max_{1 \leq k \leq m} \sum_{j=1}^{n} |a_{jk}| \, . \end{aligned}$$

Also gilt:

$$\|A\|_1 = \max_{1 \leq k \leq n} \sum_{j=1}^{m} |a_{jk}| \, . \tag{B.19}$$

Diese Norm bezeichnet man auch als **Spaltensummennorm**.

Bemerkung B.2.6 *Einfache Eselbrücke:* $\boxed{1}$*-Spaltensummen,* $\boxed{\infty}$*-Zeilensummen.*

Auch für die 2-Norm (die sogenannte **Euklidische Norm**)lässt sich eine äquivalente Formulierung finden.

Satz B.2.7 *Es sei $A \in \mathbb{R}^{m\times n}$. Dann existiert ein Vektor $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$, so dass*

$$A^T Az = \mu^2 z\,, \quad \text{wobei } \mu = \|A\|_2\,.$$

Bemerkung B.2.8 *Der Satz impliziert, dass $\|A\|_2^2$ eine Nullstelle des Polynoms $p(z) = \det(A^T A - \lambda I)$ ist. Genauer betrachtet, ist die 2-Norm von A die Wurzel des größten Eigenwerts von $A^T A$.*

Beweis. Es sei $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$ und $\|Az\|_2 = \|A\|_2$. Da z die Funktion

$$g(x) = \frac{1}{2}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2}\frac{x^T A^T Ax}{x^T x} \tag{B.20}$$

maximiert, folgt daraus, dass er $\nabla g(z) = 0$ erfüllt, wobei ∇g der Gradient von g ist $\left(\nabla := \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}\right)\right)^T$. Die partiellen Ableitungen von g lauten für $i = 1, \ldots, n$

$$\frac{\partial g(z)}{\partial x_i} = \left[x^T x \sum_{j=1}^{n} (A^T A)_{ij} x_j - (x^T A^T Ax) x_i\right] / (x^T x)^2. \tag{B.21}$$

In Vektornotation bedeutet dies für z, da $\nabla g(z) = 0$ und $\|z\|_2 = 1$, dass $A^T Az = (z^T A^T Az)z$. Setzt man nun $\mu = \|A\|_2$ so ergibt sich daraus die Behauptung. □

Lemma B.2.9 *Es sei $A \in \mathbb{R}^{m\times n}$. Dann gilt*

$$\|A\|_2 \le \sqrt{\|A\|_1 \|A\|_\infty}\,. \tag{B.22}$$

Beweis. Es sei $z \in \mathbb{R}^n$ mit $A^T Az = \mu^2 z$ und $\mu = \|A\|_2$, d.h. es sei $z \neq 0$ ein Vektor, für den das Maximum von $\dfrac{\|Ax\|_2}{\|x\|_2}$ angenommen wird. Dann gilt

$$\mu^2 \|z\|_1 = \|A^T Az\|_1 \le \|A^T\|_1 \|Az\|_1 \le \|A\|_\infty \|A\|_1 \|z\|_1\,. \tag{B.23}$$

Kondensation von $\|z\|_1$ und Wurzelziehen liefert das gewünschte Ergebnis. □

Definition B.2.10 (verträgliche Vektornorm) *Eine Matrixnorm $\|A\|$ heißt* ***kompatibel*** *oder* ***verträglich*** *mit der Vektornorm $\|x\|$, falls folgende Ungleichung erfüllt ist*

$$\|Ax\| \le \|A\|\,\|x\|\,, \qquad x \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}\,. \tag{B.24}$$

Kombinationen von verträglichen Normen sind etwa

$\|A\|_G, \|A\|_\infty$ sind verträglich mit $\|x\|_\infty$;

$\|A\|_G, \|A\|_1$ sind verträglich mit $\|x\|_1$;

$\|A\|_G, \|A\|_F, \|A\|_2$ sind verträglich mit $\|x\|_2$, wobei

$$\|A\|_G := n \max_{i,j} |a_{ij}| \qquad (A \in \mathbb{R}^{m\times n})\,.$$

Aufgabe B.2.11 Man verifiziere selbständig an einigen Beispielen die Verträglichkeit von o.g. Normenpaaren.

Abschließend erhalten wir am Ende des Kapitels.

Satz B.2.12 *Die Matrix-p-Normen sind unter allen mit der Vektornorm $\|x\|_p$ verträglichen Matrixnormen die kleinsten.*

C EINFÜHRUNG IN MATLAB

C.1 Grundlegende MATLAB-Befehle

Ruft man das Programm MATLAB mit dem Befehl `matlab` auf, so erscheinen auf dem Monitor einige Fenster. Auf den Linux-Rechnern des KIZ müssen Sie vorher die notwendigen Pfade ergänzen. Geben Sie dazu in einem Konsole-Fenster den Befehl option matlab ein.
Von diesen ist das Befehl-Fenster der primäre Ort um Befehle einzugeben und Fehler oder Ergebnisse abzulesen! Das Prompt-Zeichen $\gg$ ist im Befehl-Fenster dargestellt und dort findet man üblicherweise einen blinkenden Cursor. Der blinkende Cursor und der MATLAB-Prompt zeigen einem, dass MATLAB eine Eingabe erwartet.

C.1.1 Einfache mathematische Operationen

Genauso wie mit einem simplen Taschenrechner kann man auch mit MATLAB einfache mathematische Operationen ausführen, z.B. ergibt die Eingabe

```
≫ 3 + 4
```

die Ausgabe

```
ans =
     7
```

Man beachte, dass MATLAB im Allgemeinen keine Zwischenräume benötigt, um Befehle eindeutig zu verstehen. Alternativ zu dem obigen Beispiel können in MATLAB auch Variablen verwendet werden.

```
≫ a = 3
a =
     3
≫ b = 4
b =
     4
≫ c = a + b
c =
     7
```

MATLAB besitzt folgende einfache arithmetische Operationen

Operation	**Symbol**	**Beispiel**
Addition, $a + b$	+	5 + 3
Subtraktion, $a - b$	−	23−12
Multiplikation, $a \cdot b$	*	13.3 * 63.13
Division, $a \div b$	/ or \	$17/4 = 4\backslash 17$
Potenz, a^b	^	3^4

Die Reihenfolge, in der eine Folge von Operationen abgearbeitet wird, lässt sich wie folgt beschreiben. Ausdrücke werden von links nach rechts ausgeführt, wobei die Potenzierung die höchste Priorität besitzt gefolgt von Punktoperation, sprich Multiplikation und Division. Die geringste Priorität haben Addition und Subtraktion. Mit Hilfe von Klammern kann diese Vorgehensweise geändert werden, wobei innere Klammern vor äußeren Klammern berechnet werden.

C.1.2 Variablen

Wie in anderen Programmiersprachen hat auch MATLAB Regeln für Variablennamen. Eine Variable repräsentiert ein Datenelement, dessen Wert während der Programmausführung – gegebenenfalls mehrfach – geändert werden kann. Variablen werden anhand ihrer „Namen“ identifiziert. Namen bestehen aus ein bis neunzehn Buchstaben, Ziffern oder Unterstrichen, wobei das erste Zeichen ein Buchstabe sein muss. Man beachte, dass MATLAB Groß- und Kleinschreibung unterscheidet. (Windows ist im Gegensatz zu Linux nicht so restriktiv, da Sie jedoch Programme austauschen wollen, sollten Windows-Benutzer besondere Aufmerksamkeit walten lassen.)

Einer Variablen ist Speicherplatz zugeordnet. Wenn man eine Variable verwendet, dann meint man damit entweder den zugeordneten Speicherplatz oder den Wert, der dort augenblicklich abgespeichert ist. Einen Überblick über alle Variablen erhält man mit dem Befehl `who` oder `whos`, wobei letzterer die Angabe des benutzten Speicherplatzes beinhaltet.
Zusätzlich zu selbstdefinierten Variablen gibt es in MATLAB verschiedene spezielle Variablen. Diese lauten

spezielle Variablen	**Wert**
ans	standard Variablenname benutzt für Ergebnisse
pi	$3.1415\ldots$
eps	Maschinengenauigkeit
flops	Zähler für die Anzahl der Fließkommaoperationen
inf	steht für Unendlich (eng. infinity). z.B. $1/0$
NaN	eng. Not a Number, z.B. $0/0$
i (und) j	$i = j = \sqrt{-1}$

In MATLAB kann der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
≫ clear a b c
```

C.1.3 Kommentare und Punktion

Der Text, der nach einem Prozentzeichen `%` folgt, wird in MATLAB als Kommentar verstanden

```
≫ dummy = 4 % Wert von dummy
dummy =
     4
```

Mehrere Befehle können in eine Zeile geschrieben werden, wenn sie durch Kommata oder Semikola getrennt werden. Kommata veranlassen MATLAB, die Ergebnisse anzuzeigen. Bei einem Semikolon wird die Ausgabe unterdrückt. Durch eine Sequenz von drei Punkten kann man einen Befehl in der folgenden Zeile fortsetzen.

C.1.4 Spezielle Funktionen

Eine unvollständige Liste von Funktionen, die MATLAB bereitstellt, ist im folgenden dargestellt. Die meisten Funktionen sind so definiert, wie man sie üblicherweise benutzt.

```
≫ y = cos(pi)
y =
     -1
```

MATLAB bezieht sich im Zusammenhang mit Winkelfunktionen auf das Bogenmaß.

Funktion	Bedeutung
abs(x)	Absolutbetrag
cos(x)	Kosinus
exp(x)	Exponentialfunktion: e^x
fix(x)	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
floor(x)	rundet auf die nächste, kleinere ganze Zahl
gcd(x,y)	größter gemeinsamer Teiler von x und y
lcm(x,y)	kleinstes gemeinsames Vielfaches von x und y
log(x)	natürlicher Logarithmus
rem(x,y)	Modulo (eng. remainder of division), z.B. rem(5,2)=1
sign(x)	Signum Funktion, z.B. sign(2.3) = 1, sign(0) = 0, sign(-.3) = -1
sin(x)	Sinus
sqrt(x)	Quadratwurzel
tan(x)	Tangens

C.1.5 Skript-Dateien

Für einfache Probleme ist es schnell und effizient, die Befehle am MATLAB-Prompt einzugeben. Für größere und umfangreichere Aufgabenstellungen bietet MATLAB die Möglichkeit, sogenannte Skript-Dateien zu verwenden, in denen die Befehle in Form einer Textdatei aufgeschrieben sind und die man am Prompt übergibt. MATLAB öffnet dann diese Dateien und führt die Befehle so aus, als hätte man sie am Prompt eingegeben. Die Datei nennt man Skript-Datei oder M-Datei, wobei der Ausdruck M-Datei daher rührt, dass diese Dateien das Suffix `.m` haben, z.B. `newton.m`. Um eine M-Datei zu erstellen, ruft man einen Editor auf und speichert die Datei in dem Verzeichnis, von dem aus man MATLAB gestartet hat oder starten wird. Die Datei, z.B. `newton.m`, wird in MATLAB dann durch Eingabe von `newton` am Prompt aufgerufen.
Für die Benutzung von Skript-Dateien hat MATLAB unter anderem folgende hilfreichen Befehle

M-Datei-Funktionen	
disp(ans)	zeigt den Wert der Variablen `ans`, ohne ihren Namen auszugeben
input	erwartet vom Benutzer eine Eingabe
keyboard	übergibt zeitweise die Kontrolle an die Tastatur
pause	hält das Programm an, bis eine Taste betätigt wird

Die folgende Skript-Datei `beispiel1.m`

```
% beispiel1.m
% Beispiel fuer eine Skript-Datei
tmp = input(' Geben Sie bitte eine Zahl an >' );
3 * tmp;
```

führt zu der Ausgabe

```
≫beispiel1
Geben Sie bitte eine Zahl an > 6
ans =
     18
```

C.1.6 Dateiverwaltung

MATLAB unterstützt eine Vielzahl von Dateiverwaltungsbefehlen, welche es einem ermöglichen Dateien zu listen, Skript-Dateien anzusehen oder zu löschen und Verzeichnisse zu wechseln.

Datei-Managment-Funktionen	
cd path	wechselt in das Verzeichnis `path`
delete beispiel	löscht die Datei `beispiel.m`
ls	zeigt alle Dateien im aktuellen Verzeichnis an
pwd	zeigt den aktuellen Verzeichnispfad an
type beispiel	zeigt den Inhalt der Datei `beispiel.m` im Befehl-Fenster
what	zeigt alle M-Dateien und MAT-Dateien im aktuellen Verzeichnis an

C.1.7 Hilfe

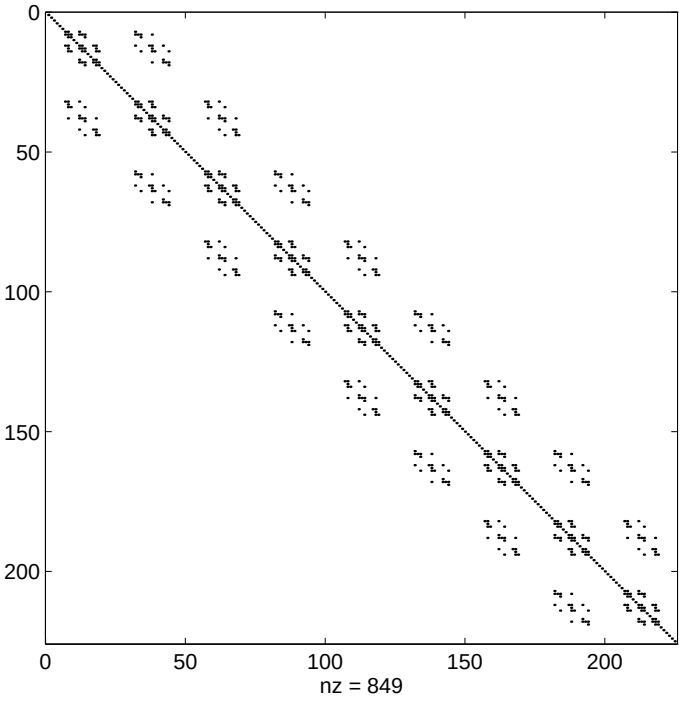
Online-Hilfe: Da sich nicht jeder Benutzer alle MATLAB-Befehle merken kann oder auch von einigen auch nur die Syntax unklar ist, bietet MATLAB die Möglichkeit der Online-Hilfe. Dabei gibt es prinzipiell mehrere Möglichkeiten. Ist einem ein Befehl bekannt und man sucht Informationen über die Syntax, so gibt es den Befehl `help`.

```
≫ help sqrt

  SQRT   Square root.
         SQRT(X) is the square root of the elements of X. Complex
         results are produced if X is not positive.

         See also SQRTM.
```

Als Beispiel haben wir uns hier die Hilfe zu dem Befehl `sqrt` ausgeben lassen.
Die andere Möglichkeit der von MATLAB gelieferten Hilfe ist durch den Befehl `lookfor` gegeben. Hier durchsucht das Programm alle ersten Zeilen der MATLAB Hilfe-Kennwörter und Skript-Dateien die im MATLAB Suchpfad zu finden sind. Das Bemerkenswerte dabei ist, dass dieser Begriff kein Befehl zu sein braucht.

```
≫ lookfor cholesky
 CHOL    Cholesky factorization
≫  CHOL   Cholesky factorization.
 CHOL(X) uses only the diagonal and upper triangle of X.
 The lower triangular is assumed to be the (complex conjugate)
 transpose of the upper.  If X is positive definite, then
 R = CHOL(X) produces an upper triangular R so that R'*R =X.
 If X is not positive definite, an error message is printed.

 With two output arguments, [R,p] = CHOL(X) never produces an
 error message.  If X is positive definite, then p is 0 and R
 is the same as above.   But if X is not positive definite, then
 p is a positive integer and R is an upper triangular matrix of
 order q = p-1 so that R'*R = X(1:q,1:q).
≫ lookfor  factorization
 CHOL    Cholesky factorization.
 QRDELETE Delete a column from the QR factorization.
 QRINSERT Insert a column in the QR factorization.
 SYMBFACT Symbolic factorization analysis.
```

Eine weitere Möglichkeit, sich Hilfe zu verschaffen, besteht darin, das Helpdesk aufzurufen. Wenn Sie helpdesk am Prompt eingeben, öffnet sich die folgende Hilfsumgebung



C.2 Mathematik mit Matrizen

C.2.1 Matrixkonstruktion und Adressierung

einfache Matrix Konstruktionen	
x=[1 4 2*pi 4]	erstelle einen Zeilenvektor x mit genannten Einträgen
x=$anfang$:$ende$	erstelle einen Zeilenvektor x beginnend mit $anfang$, Inkrement 1 und endend mit $ende$
x=$anfang$:$inkrement$:$ende$	Ähnliches wie oben mit dem Inkrement $inkrement$
x=linspace($anfang$,$ende$,n)	erzeugt einen Zeilenvektor der Dimension n mit $x(i) = \frac{(n-i)\cdot anfang+(i-1)\cdot ende}{n-1}$

Im Folgenden sind einige charakteristische Beispiele aufgeführt.

```
≫ B = [1 2 3 4; 5 6 7 8]
B =
     1 2 3 4
     5 6 7 8
```

Der Operator ′ liefert für reelle Matrizen die Transponierte.

```
≫ C = B'
C =
     1 5
     2 6
     3 7
     4 8
```

Der Doppelpunkt `:` in der zweiten Komponente spricht alle vorhandenen Spalten an, d.h. er ist ein zu `1:4` äquivalenter Ausdruck.

```
≫C = B(1,:)
C =
     1 2 3 4
≫C = B(:,3)'
C =
     3 7
```

Es lassen sich auch einzelne Komponenten neu definieren.

```
≫A = [1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9
≫A(1,3) = 9
A =
     1 2 9
     4 5 6
     7 8 9
```

Ist ein Eintrag noch nicht definiert, so verwendet MATLAB die minimale Erweiterung dieser Matrix und setzt undefinierte Einträge zu Null.

```
≫A(2,5) = 4
A =
     1 2 9 0 0
     4 5 6 0 4
     7 8 9 0 0
```

Im Folgenden werden die Vektoren `(3,2,1)` und `(2,1,3,1,5,2,4)` dazu verwendet, die Matrix `C` zu indizieren, d.h. `C` hat die Struktur

```
A(3,2) A(3,1) A(3,3) A(3,1) A(3,5) A(3,2) A(3,4)
A(2,2) A(2,1) A(2,3) A(2,1) A(2,5) A(2,2) A(2,4)
A(1,2) A(1,1) A(1,3) A(1,1) A(1,5) A(1,2) A(1,4)
```

In MATLAB erhält man nun

```
≫C=A(3:-1:1,[2 1 3 1 5 2 4])
C =
     8 7 9 7 0 8 0
     5 4 6 4 4 5 0
     2 1 9 1 0 2 0
```

Ein weiteres Beispiel für Indizierung ist

```
≫C=C(1:2,2:3)
C =
     7 9
     4 6
```

Im nächsten Beispiel wird ein Spaltenvektor dadurch konstruiert, dass alle Elemente aus der Matrix `C` hintereinander gehängt werden. Dabei wird spaltenweise vorgegangen.

```
≫ b=C(:)'
b =
     7  4  9  6
```

Das Löschen einer ganzen Zeile oder Spalte kann durch das Umdefinieren in eine 0×0-Matrix geschehen, z.B.

```
≫ C(2,:)=[ ]
C =
     7  9
```

C.2.2 Skalar-Matrix-Operationen

In MATLAB sind Skalar-Matrix-Operationen in dem Sinne definiert, dass Addition, Subtraktion, Division und Multiplikation mit einem Skalar elementweise durchgeführt werden. Es folgen zwei erklärende Beispiele.

```
≫ B - 1
ans =
     0  1  2  3
     4  5  6  7
≫ 9 + 3 * B
ans =
     12  15  18  21
     24  27  30  33
```

C.2.3 Matrix-Matrix-Operationen

Die Operationen zwischen Matrizen sind nicht so kanonisch zu definieren wie die zwischen Skalar und Matrix, insbesondere sind Operationen zwischen Matrizen unterschiedlicher Dimension schwer zu definieren. Des Weiteren sind die Operationen $*$ und $.*$, bzw. $/$ und $./$ sowie $\backslash$ und $.\backslash$ zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben.

komponentenweise Matrixoperationen	
Beispieldaten	$a = [a_1, a_2, \ldots, a_n]$, $b = [b_1, b_2, \ldots, b_n]$, c ein Skalar
komp. Addition	$a + c = [a_1 + c\ a_2 + c \ldots a_n = c]$
komp. Multiplikation	$a * c = [a_1 \cdot c\ a_2 \cdot c\ \ldots\ a_n \cdot c]$
Matrix–Addition	$a + b = [a_1 + b_1\ a_2 + b_2 \ldots a_n + b_n]$
komp. Matrix-Multiplikationen	$a. * b = [a_1 \cdot b_1\ a_2 \cdot b_2 \ldots a_n \cdot b_n]$
komp. Matrix-Div. von rechts	$a./b = [a_1/b_1\ a_2/b_2 \ldots a_n/b_n]$
komp. Matrix-Div. von links	$a.\backslash b = [b_1/a_1\ b_2/a_2 \ldots b_n/a_n]$
komp. Matrix-Potenz	$a.\hat{}\,c = [a_1^c\ a_2^c \ldots a_n^c]$
	$c.\hat{}\,a = [c^{a_1}\ c^{a_2} \ldots c^{a_n}]$
	$a.\hat{}\,b = [a_1^{b_1}\ a_2^{b_2} \ldots a_n^{b_n}]$

Es folgen nun einige Beispiele zu Matrixoperationen

```
≫g=[1 2 3; 4 5 6]; % zwei neue Matrizen
≫h=[2 2 2; 3 3 3];
≫g+h   % addiere g und h komponentenweise
ans =
     3  4  5
     7  8  9
≫ans-g   % subtrahiere g von der vorherigen Antwort
ans =
     2  2  2
     3  3  3
≫h.*g   % multipliziere g mit h komponentenweise
ans =
      2   4   6
     12  15  18
≫g*h'   % multipliziere g mit h'
ans =
     12  18
     30  45
```

C.2.4 Matrix-Operationen und -Funktionen

Matrixfunktionen	
reshape(A,m,n)	erzeugt aus den Eintägen der Matrix A eine $m \times n$-Matrix, wobei die Einträge spaltenweise aus A gelesen werden.
diag(A)	ergibt die Diagonale von A als Spaltenvektor
diag(v)	erzeugt eine Diagonalmatrix mit dem Vektor v in der Diagonalen
tril(A)	extrahiert den unteren Dreiecksanteil der Matrix A
triu(A)	extrahiert den oberen Dreiecksanteil der Matrix A

Es folgen einige Beispiele

```
≫g=linspace(1,9,9)   % ein neuer Zeilenvektor
g =
      1    2    3    4    5    6    7    8    9
≫B=reshape(g,3,3)   % macht aus g eine 3 x 3 Matrix
B =
      1    4    7
      2    5    8
      3    6    9
≫tril(B)
ans =
      1    0    0
      2    5    0
      3    6    9
```

Funktion	Bedeutung
R=chol(A)	Choleskyzerlegung
cond(A)	Konditionszahl der Matrix A
d=eig(A)	Eigenwerte und -vektoren
[V,d]=eig(A)	
det(A)	Determinante
hess(A)	Hessenbergform
inv(A)	Inverse
[L,U]=lu(A)	Zerlegung gegeben durch Gauss-Algorithmus
norm(A)	euklidische-Norm
rank(A)	Rang der Matrix A

Bemerkung: Der \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. `A^(-1)x=A\x` . 

C.2.5 Spezielle Matrizen

	spezielle Matrizen
eye(n)	erzeugt eine Einheitsmatrix der Dimension n
ones(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 1
zeros(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 0

C.2.6 Spezielle Funktionen für schwachbesetzte Matrizen

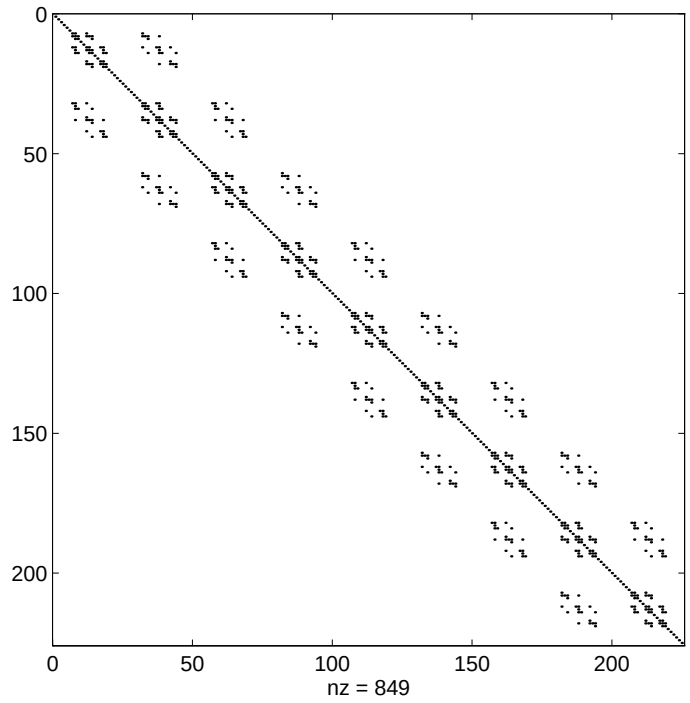
Bei vielen numerischen Anwendungen treten schwachbesetzte Matrizen auf. MATLAB hat für solche Matrizen besondere Sparse-Funktionen, die dieser Eigenschaft Rechnung tragen. 

Funktion	Bedeutung
find(A)	findet Indizes von Nichtnulleinträgen
nnz(A)	Anzahl an Nichtnulleinträgen
spdiags(v)	erzeugt eine Sparse-Diagonalmatrix mit dem Vektor v als Diagonale
speye(n)	erzeugt eine Sparse-Einheitsmatrix
spy(A)	visualisiert die Struktur der Matrix A

Kurze Illustration der Funktionsweise obiger Befehle anhand einiger Beispiele.

```
≫ E=eye(100);    % vollbesetzte 100 x 100 Einheitsmatrix
≫ Es=sparse(E);  % Sparse-Version von E
≫ whos
      Name      Size    Elements  Bytes  Density  Comple
         E  100 by 100    10000  80000     Full     No
        Es  100 by 100      100   1600   0.0100     No
 Grand total is 10100 elements using 81600 bytes
≫ A=spdiags([7*ones(4,1),ones(4,1),2*ones(4,1)],[-1,0,1],4,4);
≫ nnz(A)
ans =
     10
≫ full(A)
ans =
      1   2   0   0
      7   1   2   0
      0   7   1   2
      0   0   7   1
```

Als Beispiel für `spy` sei hier die Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix gezeigt.



C.3 Datenverwaltung

Für die meisten Anwendungen genügt es, Datenfelder in einem Format abzuspeichern und wieder laden zu können. Die Befehle `load` und `save` setzen voraus, dass die Daten in einem System unabhängigen, binären Format in einer Datei mit dem Suffix `.mat` gespeichert sind oder in einem einfachen ASCII-Format vorliegen.

C.3.1 Daten speichern

Im Folgenden wird eine 3×5 -Matrix im binär-Format in der Datei `A.mat` gespeichert. Diese Daten sind sehr kompakt gespeichert.

```
≫ A=zeros(3,5);
≫ save A
```

Gibt man sich aber den Inhalt dieser Datei auf dem Bilschirm aus, so gibt er wenig Sinn. Möchte man sich also z.B. einen Lösungsvektor sichern um ihn später „per Hand zu analysieren“ so speichere man die Daten als ASCII-Datei, dabei kann man wählen zwischen einer 8-stelligen oder 16-stelligen Abspeicherung.

```
≫ save mat1.dat A -ascii          %  8-stellige Speicherung
≫ save mat2.dat A -ascii -double  % 16-stellige Speicherung
```

Hier wurde die Matrix mit 8-stelligem Format in der Datei `mat1.dat` gespeichert, bzw. 16-stellig in der Datei `mat2.dat`.

C.3.2 Daten laden

Mit dem Befehl `load A` versucht MATLAB, die in `A.mat` gespeicherten Daten in einem Datenfeld `A` zu speichern. Auch ASCII-Dateien kann MATLAB lesen. Da es hier jedoch keine Standardendung gibt, ist die Datei inklusive Endung anzugeben. Es ist darauf zu achten, dass ein rechteckiges Feld an Daten vorliegt, d.h. dass m Zeilen mit jeweils n numerischen Werten vorliegen. MATLAB erstellt dann eine $m \times n$-Matrix mit dem Namen der Datei ohne Suffix.

```
≫ load  mat1.dat
≫ whos
    Name        Size    Elements  Bytes  Density  Complex
    mat1       3 by 5         15    120     Full    No
 Grand total is 15 elements using 120 bytes
```

C.4 Ausgabe von Text

Mit dem Befehl `fprintf` lassen sich Strings, d.h. Zeichenfolgen, auf dem Bildschirm ausgeben.

```
≫ fprintf('\n Hello world %12.3e\n',4);
 Hello world    4.000e+00
```

Man sieht, dass der auszugebende Text zusätzliche Zeichen enthält, die nicht mit ausgedruckt werden. Diese Zeichen nennt man Escape-Sequenzen. In obigem Beispiel ist die Sequenz `\n` eingebaut. `\n` steht für „newline“ und sorgt dafür, dass bei der Textausgabe an diesen Stellen eine neue Zeile begonnen wird. Der Ausdruck `%12.3e` dient als Platzhalter für einen reellen Wert, der durch Komma getrennt hinter der Zeichenkette folgt. Dabei sei die Zahl in Exponentialdarstellung auszugeben, wofür 12 Stellen mit 3 Nachkommastellen bereitgestellt. Im Beispiel ist dies der Wert 4. Anstatt eines expliziten Wertes können auch Variablen oder Ausdrücke, z.B. `3 * 4` stehen. Ein Platzhalter kann mehrfach in einem `printf`-Befehl vorkommen. In diesem Fall müssen hinter der Zeichenkette genau so viele Werte folgen, wie Platzhalter angegeben sind. Die Reihenfolge der Werte muss mit der Reihenfolge der Platzhalter übereinstimmen, da die Ausdrücke von links nach rechts bewertet werden. Die Escape-Sequenzen dürfen im Text an beliebiger Stelle stehen.

Ausgabeformate

Befehl	Ausgabe
fprintf('%.0e\n',1.234567)	`1e00+`
fprintf('%.2e\n',1.234567)	`1.23e00+`
fprintf('%.5e\n',1.234567)	`1.23456e00+`
fprintf('%10.0e\n',1.234567)	`1e00+`
fprintf('%10.2e\n',1.234567)	`1.23e00+`
fprintf('%10.5e\n',1.234567)	`1.2346e00+`
fprintf('%10.2f\n',1.234567)	`1.23`
fprintf('%10.5f\n',1.234567)	`1.23457`

MATLAB rundet numerische Werte bei der Ausgabe, wenn nötig!

C.5 Kontrollbefehle

C.5.1 For-Schleifen

1. Mit jeglicher gültigen Matrix-Darstellung lässt sich eine FOR-Schleife definieren, z.B.

```
≫ data = [1 7 3 2; 5 4 7 2]
data =
     1 7 3 2
     5 4 7 2
≫ for n=data
   x=n(1)-n(2)
  end
x =
     -4
x =
      3
x =
     -4
x =
     0
```

2. FOR-Schleifen können nach Belieben geschachtelt werden.

```
≫ for k=3:5
   for l=4:-1:2
    A(k,l)=k^2-l;
   end
  end
≫ A
A =
      0  0  0  0
      0  0  0  0
      0  7  6  5
      0 14 13 12
      0 23 22 21
```

△ **3. FOR-Schleifen sollten vermieden werden, wann immer sie durch eine äquivalente Matrix-Darstellung ersetzt werden können**. Der folgende Ausdruck ist darunten in optimierter Version aufgeführt.

```
≫ for n=1:10
   x(n)=sin(n*pi/10);
  end
≫x
x =
     Columns 1 through 7
       0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
     Columns 8 through 10
       0.5878  0.3090  0.0000

≫ n=1:10;
≫ x=sin(n*pi/10);
≫ x
x =
     Columns 1 through 7
       0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
     Columns 8 through 10
       0.5878  0.3090  0.0000
```

4. Um die Ausführungsgeschwindigkeit zu maximieren, sollte benötigter Speicherplatz vor Ausführung der FOR-Schleife alloziert werden. △

```
≫ x=zeros(1,10);
≫ x(1)=0.5;
≫ for n=2:10
   x(n)=x(n-1)*sin(n*pi/10);
  end
≫ x
x =
     Columns 1 through 7
       0.5000  0.2939  0.2378  0.2261  0.2261  0.2151  0.1740
     Columns 8 through 10
       0.1023  0.0316  0.0000
```

C.5.2 WHILE-Schleifen

WHILE-Schleifen sind wie folgt aufgebaut.

```
while Aussage
     Anweisungen
end
```

Die Anweisungen zwischen `while` **und** `end` **werden so lange ausgeführt, wie** `Aussage` **wahr ist**, z.B.

```
≫ num=0; EPS=1;
≫ while (1+EPS) > 1
   EPS=EPS/2;
   num=num+1;
  end
≫ num
num =
     53
```

C.5.3 IF-ELSE-END Konstrukte

Eine IF-ELSE-END-Schleife enthält nach dem IF eine Aussage, die daraufhin überprüft wird, ob sie wahr oder falsch ist. Ist sie wahr, so werden die in den folgenden Zeilen stehenden Anweisungen ausgeführt und die Schleife beendet. Ist sie falsch, so erfolgen die Anweisungen, die dem ELSE folgen (ELSE ist optional). Solch ein Konstrukt kann erweitert werden um beliebig viele ELSIF Befehle, die dieselbe Funktion haben wie der am Anfang stehende IF Befehl, aber nur beachtet werden, falls alle vorher überprüften Aussagen falsch sind.

```
if Aussage1
     Anweisungen, wenn Aussage1 wahr
elseif Aussage2
     Anweisungen, wenn Aussage1 falsch und Aussage2 wahr
else
     Anweisungen, wenn Aussage1 und Aussage2 falsch
end
```

C.5.4 Relationen und logische Operatoren

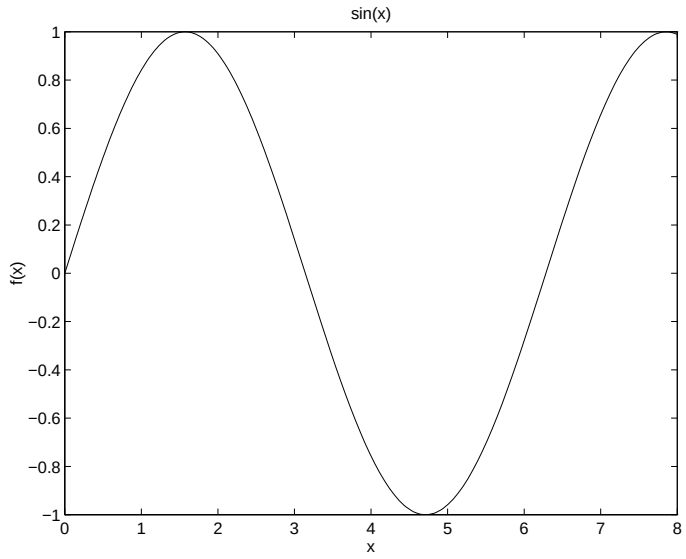
Relationen	
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
==	gleich
~=	ungleich

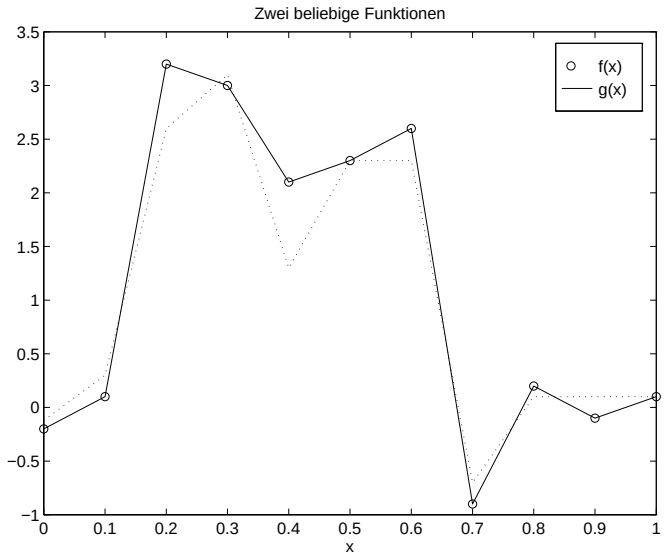
logische Operatoren	
&	UND
|	ODER
~	NICHT

C.6 Graphische Darstellung

C.6.1 Zweidimensionale Graphiken

```
≫ f='sin(x)';
≫ fplot(f,[0 8]);
≫ title(f),xlabel('x'),ylabel('f(x)');

≫ x=0:0.1:1;
≫ y=[-0.2 0.1  3.2  3  2.1  2.3  2.6 -0.9 0.2 -.1 .1];
≫ z=[-0.12 0.3  2.6  3.1  1.3  2.3  2.3 -0.7 0.1 .1 .1];
≫ plot(x,y,'o',x,y,x,z,':');
≫ title('Zwei beliebige Funktionen'),xlabel('x');
≫ legend('f(x)','g(x)')
```





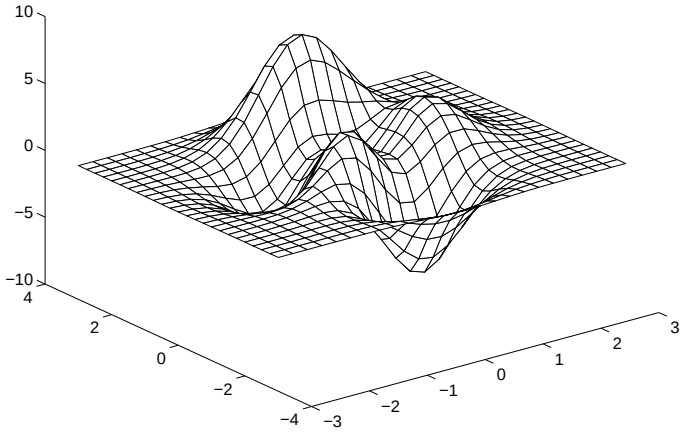
Linientypen und Farben

Symbol	Farbe	Symbol	Linientyp
y	gelb	$\cdot$	Punkt
m	magenta	$\circ$	Kreis
c	cyan	x	x-Markierung
r	rot	$+$	$+$-Markierung
g	grün	$*$	Sternchen
b	blau	$-$	durchgezogene Linie
w	weiß	:	gepunktete Linie
k	schwarz	$-.$	Strichpunkt-Linie
		$--$	gestrichelte Linie

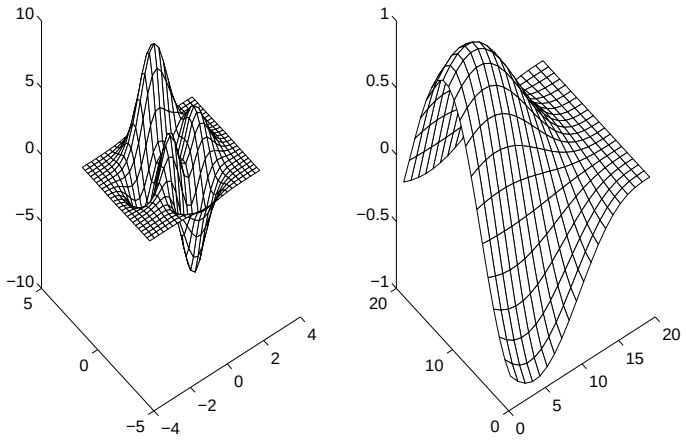
2-D Graphikanweisung	
axis	modifiziert die Axen-Proportionen
clf	löscht die Graphik im Graphik-Fenster
close	schließt das Graphik-Fenster
grid	erzeugt ein achsenparalleles Gitter
hold	ermöglicht das Überlagern von Graphiken
subplot	erstellt mehrere Teilgraphiken in einem Fenster
text	gibt Text an vorgegebener Stelle aus
title	zeigt einen Titel an
xlabel	beschriftet die x-Achse
ylabel	beschriftet die y-Achse
colormap(white)	wechselt die Farbtabelle, für S/W-Monitore

C.6.2 Dreidimensionale Graphiken

```
≫ [X,Y,Z] = peaks(25)
≫ mesh(X,Y,Z)
```




```
≫ subplot(1,2,1);
≫ [X,Y,Z] = peaks(25);
≫ mesh(X,Y,Z);
≫ subplot(1,2,2);
≫ X=1:20;
≫ Y=1:20;
≫ Z(X,Y)=(-(cos(X/4)))'*(sin((20-Y)/10).^3);
≫ mesh(X,Y,Z);
```



C.6.3 Graphiken drucken

Graphik-Druckbefehl		
print [-dAusgabetyp] [-Optionen] [Dateiname]		
Ausgabetyp	-dps	Postscript für Schwarzweißdrucker
	-dpsc	Postscript für Farbdrucker
	-deps	Encapsulated Postcript
	-depsc	Encapsulated Color Postcript
Optionen	-P<Drucker>	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
≫ print fig4 -deps
```

erzeugt die Datei `fig4.eps`.

C.7 Fortgeschrittenes

C.7.1 MATLAB-Skripte

Wie schon im Abschnitt C.1.5 erwähnt, lässt sich eine Abfolge von MATLAB-Befehlen auch in einer Datei speichern. Diese kann man dann am Befehl-Fenster aufrufen und die gespeicherte Folge von Befehlen wird ausgeführt. Solche Dateien werden als MATLAB-Skripte oder M-Files bezeichnet. Alle o.g. Befehle lassen sich in einer Datei z.B. mit dem Namen `abc.m` zusammenstellen. Für die Wahl des Dateinamens gelten dabei die folgenden Regeln:

- das erste Zeichen ist ein Buchstabe und
- die Datei hat die Endung **.m** .

Um ein solches M-File zu erstellen, ruft man den Editor auf, der es erlaubt Text einzugeben und diesen als Datei zu speichern. Wenn man den Editor gestartet hat, gebe man die folgenden Befehle ein und speichere die Datei unter dem Namen `abc.m`

```
a = 1;
b = 3;
c = -5;
x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Um nun die Befehle aus der Datei `abc.m` auszuführen, gibt man im MATLAB-Befehlsfenster

```
≫ abc
```

ein. MATLAB sucht im aktuellen Pfad nach der Datei `abc.m` und führt die darin enthaltenen Befehle aus. Das aktuelle Verzeichnis wird angezeigt, wenn man

```
≫ pwd
```

eingibt (pwd, engl. print working directory).

C.7.2 Erstellen eigener Funktionen

Es wird sicherlich etwas komfortabler sein, eine eigene Funktion zu haben, der man *a,b* und *c* als Argument übergibt un die beiden Wurzeln als Eregebnis erhält, als jedesmal erneut ein eigenes M-File anzufertigen. Ein solches Programm könnte z.B. die folgende Datei `root.m` sein.

```
%-----------------
modified "abc.m"
%-----------------

a = input('Enter a: ');
b = input('Enter b: ');
c = input('Enter c: ');

x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Die Prozentzeichen in den ersten Zeilen dienen der Dokumentation. Alles was einem solchen Zeichen folgt, wird von MATLAB nicht ausgewertet, sprich interpretiert. Die Argumente werden in dieser Funktion jedoch nur bedingt übergeben und ausgegeben. Eine Funktion, die diese Aufgabe erfüllt, ist die folgende.

```
%-----------------
modified "abc.m"
%-----------------

function [x1, x2] = quadroot(a,b,c)

radical = sqrt(b^2 - 4*a*c);

x1 = (-b + radical)/(2*a)
x2 = (-b - radical)/(2*a)
```

Wenn eine Funktion keine Rückgabewerte hat, können die eckigen Klammern mit den Variablen und das Gleichheitszeichen fehlen. Fehlen die Eingabeparameter, so kann auch der Klammerausdruck nach `quadroot` fehlen. Auf die in der Funktion verwendeten Variablen, hier `radical`,

kann man vom Befehlsfenster nicht zugreifen. Ist die Funktion in der Datei `quadroot.m` gespeichert, so erhält man die Wurzeln, indem man

```
[x1, x2] = quadroot(1,3,-5);
```

eingibt. Es kann vom Befehlsfenster oder einer anderen Datei immer nur die erste Funktion in einer Datei aufgerufen werden. Dies bedeutet, in einer Datei können mehrere Funktionen stehen, aber nur die erste Funktion kann extern aufgerufen werden. Alle weiteren Funktionen können nur von Funktionen in der gleichen Datei aufgerufen werden. Für den Anfang ist es einfacher, wenn jede Datei nur eine Funktion enthält. Entscheidend für den Funktionsnamen ist der Name der Datei.

Literaturverzeichnis

[AS] M. ABRAMOWITZ, I.A. STEGUN Pocketbook of Mathematical Functions with Formulas, Verlag Harri Deutsch, Frankfurt/Main (1984).

[A] R.A. ADAMS, "Sobolev Spaces", Pure Appl. Math. 65, Academic Press, New York, 1975.

[Arendt/Urban] W. ARENDT, K. URBAN, Partielle Differenzialgleichungen: Eine Einführung in analytische und numerische Methoden, Spektrum Akademischer Verlag 2010.

[Calvetti] D. CALVETTI, G.H. GOLUB, W.B. GRAGG UND L. REICHEL, Computation of Gauss-Kronrod rules. Math. Comp. 69, 1035–1052 (2000).

[Cull] P. CULL, M. FLAHIVE UND R. ROBSON, Difference equations, Undergraduate Texts in Mathematics. Springer, New York (2005).

[Cuyt/Wuytack] A. CUYT, L. WUYTACK, Nonlinear Methods in Numerical Analysis, North-Holland (Amsterdam).

[Deuflhard/Hohmann] P. DEUFLHARD, A. HOHMANN, Numerische Mathematik 1, de Gruyter-Verlag, Berlin, 1993.

[Elayadi] S. ELAYADI An introduction to difference equations, Undergraduate Texts in Mathematics. Springer, New York (2005).

[Fischer] G. FISCHER, Linare Algebra, Vieweg-Verlag, Wiesbaden, 1989.

[Analysis I] S. FUNKEN, Skript zur Vorlesung „Analysis I", gehalten im Wintersemester 07/08 an der Universität Ulm.

[Analysis II] S. FUNKEN, Skript zur Vorlesung „Analysis II", gehalten im Sommersemester 2008 an der Universität Ulm.

[Discroll/Maki] T. A. DISCROLL, K. L. MAKI: Searching for Rare Growth Factors Using Multicanonical Monte Carlo Methods. SIAM Review. Vol. 49, No. 4, pp. 673-692. 2008.

[Golub/Loan] G. H. GOLUB, C. F. VAN LOAN, Matrix Computations, 3. ed., Hopkins Univ. Press, 1996.

[Hackbusch] W. HACKBUSCH, Iterative Lösung großer schwachbesetzter Gleichungssysteme, 2. Auflage, Teubner-Verlag, Stuttgart, 1993.

[Hämmerlin/Hoffmann] G. HÄMMERLIN, K.-H. HOFFMANN, Numerische Mathematik, 4. Auflage, Springer-Verlag, Berlin, 1994.

[Hanke] M. HANKE-BOURGEOIS, Grundlagen der Numerischen Mathematik und des wissenschaftlichen Rechnens , 1. Auflage, Teubner-Verlag, Stuttgart, 2002.

[Heuser] H. Heuser: Funktionalanalysis. Teubner, 3. Auflage, 1992.

[Higham] N. J. HIGHAM, How accurate is Gaussian elimination? Numerical Analysis 1989, Proceedings of the 13th Dundee Conference, Vol. 228 of Pitman Research Notes in Mathematics, 137–154.

[Knuth] D.E. KNUTH The Art of Computer Programming, Band 2, Addison-Wesley, 3. Auflage, 1998.

[Niethammer] W. NIETHAMMER, Relaxation bei nichtsymmetrischen Matrizen. Math. Zeitschr. **85** 319-327, 1964.

[Kiefer] J. KIEFER, Optimum sequential search and approximation methods under minimum regularity assumptions. J. Soc. Ind. Appl. Math. 5, 105-136 (1957).

[Kronrod] A.S. KRONROD, Nodes and weights of quadrature formulas. Sixteen-place tables. New York: Consultants Bureau. Authorized translation from the Russian (1965).

[Laurie] D.P. LAURIE, Calculation of Gauss-Kronrod quadratur rules. Math. Comp. 1133-1145 (66) 1997.

[Lebed] G. K. LEBED, Quadrature formulas with minimum error for certain classes of functions, Mathematical Notes 3, 368-373 (1968).

[Marsden] M. J. MARSDEN, An identity for spline functions with applications to variation-dimishing spline approximation. J. Approx. Theory 3 (1970), 7-49.

[Meyberg/Vachenauer] K. MEYBERG, P. VACHENAUER, Höhere Mathematik 1. Springer, Berlin (1999)

[Plato] R. PLATO, Numerische Mathematik kompakt, Vieweg-Verlag.

[Quarteroni et. al.] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik, Band 1 & 2, Springer-Verlag, Berlin, 2002.

[QSS1] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik 1, Springer 2002.

[QSS2] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik 2, Springer 2002.

[Rivlin] T.J. RIVLIN, An Introduction to the Approximation of Functions, Blaisdell Publ., Waltham, MA, 1969.

[Schönhage] A. SCHÖNHAGE, Approximationstheorie, de Gruyter, Berlin, 1971.

[Schwarz] H. R. SCHWARZ, Numerische Mathematik, 4. Auflage, Teubner-Verlag, Stuttgart, 1997.

[Stör/Bulirsch] J. STÖR, R. BULIRSCH, Numerische Mathematik, Band 1 & 2, Springer-Verlag, Berlin, 1994.

[SW] K. STREHMEL, R. WEINER, Numerik gewöhnlicher Differentialgleichungen, Teubner-Verlag, Stuttgart, 1995.

[St] A. H STROUD, Gaussian quadrature formulas, Prentice-Hall, Englewood Cliff (1966).

[Sz] G. SZEGÖ, Orthogonal Polynomials, AMS 3. Auflage, 1967.

[Törnig/Spellucci] W. TÖRNIG, P. SPELLUCCI, Numerische Mathematik für Ingenieure und Physiker, Band 1 & 2, Springer-Verlag, Berlin, 1988.

[W] W. WALTER, Gewöhnliche Differentialgleichungen, 6. Auflage, Springer-Verlag, Berlin u.a., 1996.

[Wilkinson65] J. H. WILKINSON, The Algebraic Eigenvalue Problem, Oxford University Press, 1965.

[Wilkinson69] J. H. WILKINSON, Rundungsfehler, Springer-Verlag, Berlin, Heidelberg, New York, 1969.

[Wille] D. WILLE, Repetitorium der Linearen Algebra, Teil 1, 1. Auflage, Feldmann-Verlag, Springer, 1989.

Index

absoluter Fehler, 24
absoluter/relativer Fehler, 15
Aquivalenzoperationen, 35
Ausgleichsgerade, 55
Auslöschung, 18

b-adisch, 7
b-adischer Bruch, 7
Bandbreite, 49
Bandgleichungen, 48
Bruch, b-adischer, 7

Cauchy-Schwarz-Ungleichung, 141
Cholesky, 45
    -Verfahren, 47, 49
    -Zerlegung, 48
    -Zerlegung, rationale, 46
Cramersche Regel, 28
CSU, 141

Darstellung, Gleitpunkt-, 10
Darstellung, Leibnizsche, 29
Daumenregel zur Genauigkeit, 53
diagonaldominant, 42
Diagonalmatrix, 32
Dreiecksmatrix, 32
Dreieckszerlegung, Gaussche, 37

Eliminationsmethode, Gaußsche, 35
Eliminationsschritt, 36

Fehler, 24
    absoluter, 24
    absoluter/relativer, 15
    relativer, 24
Fehlerarten, 5
Festpunktzahl], 9
Frobenius–Norm, 142

Gaußsches-Eliminations-Verfahren, 37
Gaus-Elimination mit Spaltenpivotstrategie, 40
Gaussche Dreieckszerlegung, 37
Gauß-Lobatto, 134, 136
Gaußschen Normalengleichung, 56
Gaußsche Eliminationsmethode, 35
Genauigkeit, Daumenregel zur, 53
Gestaffelte Systeme, 31
Givens-Rotationen, 59
Gleitkommaarithmetik, 17
Gleitkommaoperation, 29
Gleitkommazahl, 5
Gleitpunkt-Darstellung, 10
gross-oh, 139
Guard Digit, 19

Holder-Ungleichung, 141
Householder–Spiegelungen, 62
Householder-Transformation, 62
Householder-Vektor, 62

kaufmännische Rundung, 15
klein-oh, 139
Kondition, 5, 21
Konditionszahl, 51
Konditionszahlen, 22
Konsistenz, 23

Laplacescher Entwicklungssatz, 29
Leibnizsche Darstellung, 29
LR-Zerlegung, 37

Maschinengenauigkeit, 16
MATLAB, 145
Matrix
    Diagonalmatrix, 32
    Dreiecksmatrix, 32
    positiv definite, 45
    Rotationsmatrizen, 58
    unipotente, 37
Matrixnorm, 142
Methode der kleinsten Quadrate, 55

Nachiteration, 45
Neumann-Reihe, 52
Normalengleichung, Gaußschen, 56
Normalisierung, 7

Operationen
    Aquivalenzoperationen, 35
    Gleitkommaoperation, 29
    Rechenoperationen, 29
orthogonal, 57

p-adisch, 7
Permutation, 39

Pivot-Strategien, 38
Pivotelement, 36
positiv definite Matrix, 45

QR-Zerlegung, 57

Rückwärtssubstitution, 33
rationale Cholesky-Zerlegung, 46
Rechenoperationen, 29
Regel, Cramersche, 28
Reihe
    Neumann-, 52
relativer Fehler, 24
Residuum, 45, 50
Rotationsmatrizen, 58
Rundung, 15
    kaufmännische, 15
    Standardrundung, 15

Satz
    Laplacescher Entwicklungssatz, 29
Schutzziffer, 19
Spaltenpivotisierung, 42
Spaltenpivotstrategie, 40
Spaltenpivotstrategie, Gaus-Elimination mit, 40
Spaltensummennorm, 143
Stabilität, 5, 23
Standardfehler, 24
Standardmatrix, 28
Strategien, Pivot-, 38
Submultiplikativitat, 142
Substitution, Vorwärts- Rückwärts-, 33
Systeme, Gestaffelte, 31

Ungleichung, Cauchy-Schwarz-, 141
Ungleichung, Holder-, 141
unipotente Matrix, 37

Vektornorm, 141
Vektornorm, vertragliche, 144
Verfahren
    Cholesky, 47
    Cholesky-, 49
    Gaußsches-Eliminations-, 37
vertragliche Vektornorm, 144
Vorwärtssubstitution, 33

Zahlendarstellung, 5
Zahlenformat, 12
Zeilenpivotisierung, 42
Zeilensummennorm, 143
Zerlegung, 42
Zerlegung, Cholesky-, 48
Zerlegung, Cholesky-, rationale, 46