Inhaltsverzeichnis

Vorwort zur zweiten Auflage

Das vorliegende Skriptum ist eine möglichst getreue Aufzeichnung des im Kurs 1990/91 vorgetragenen Stoffes mit minimalen Modifikationen und Zusätzen. Eingefügt wurden nur die Gram-Schmidt-Orthogonalisierungen (Abschnitt 2.5), die Erwähnung der diskreten Cosinus-Transformation (in Abschnitt 4.6) und die linearen s-Schritt-Integratoren (Abschnitt 9.4).

Diese Unterlagen sollen zuverlässiger sein als die vom Hörer selbst angefertigte, eilige Mitschrift des Gerippes aus Formeln, Definitionen und Sätzen, wie sie an der Tafel standen.

Diese Unterlagen sollen aber darüber hinaus auch ein bißchen von dem wiedergeben, was *nicht* an der Tafel stand, sondern nur vorgetragen wurde und erfahrungsgemäß in den Hörer-Aufzeichnungen fehlt. Gemeint sind die Einleitungen und ergänzenden Bemerkungen zu den Abschnitten, Überleitungen, Motivierungen, Schlußfolgerungen usw.

Doch wurde sehr darauf geachtet, daß der Charakter eines Vorlesungs-Skriptums erhalten blieb, das den vorgetragenen Stoff zuverlässig katalogisiert. Deshalb ist dieses Skriptum kein Ersatz für die empfohlenen Lehrbücher.

Auffällig ist der sparsame Gebrauch von Zahlenbeispielen, der gewiß nicht auf Geringschätzung oder mangelndem Material beruht. Besser als Tabellen zu lesen ist es, selbst den Computer zu programmieren und die Algorithmen auszuprobieren. Dabei wird schnell klar werden, daß gute Numerik-Software viel mehr ist, als die bloße Programmierung der Schritte eines besprochenen Algorithmus. In der Produktion sollte man deshalb stets die Routinen aus einer guten Programm-Bibliothek verwenden.

Herrn Dieter Bertram, Teo Eris und Mathias Richter möchte ich herzlich danken für ihren unermüdlichen Einsatz beim Setzen dieses Textes.

München, im August 1992 Christian Reinsch

Vorwort zur dritten Auflage

Es wurden einige bekannt gewordene Druckfehler beseitigt. Wieder gilt mein herzlicher Dank Herrn Mathias Richter für die damit verbundene Arbeit.

München, im November 1995 Christian Reinsch

Was ist numerische Mathematik? Ihre Kennzeichen sind:

Konstruktiv, d.h. nicht nur Diskussion von Existenz und Eindeutigkeit einer Problemlösung, sondern befaßt mit der aktuellen Beschaffung dieser Lösung.

Näherungen für die exakte Lösung werden als gleichwertig akzeptiert, wenn deren Fehler durch Steigerung des technischen Aufwands beliebig klein gemacht werden können.

Als **Hilfsmittel** zur Konstruktion werden nur die vier Grundoperationen der Arithmetik $+$, $-$, $\cdot$, $/$ und die sechs arithmetischen Vergleiche zugelassen, wahlweise auch $\sqrt{}$. Damit also reine Zahlenrechnung, keine Formelmanipulation (Eingabe-Daten, alle Zwischenwerte, End-Resultate sind immer reelle Zahlen).

Rundungsfehler: Auch die arithmetischen Grund-Operationen werden nur mit einer festen Stellenzahl ausgeführt, also nicht exakt, sondern nur näherungsweise. Die Fortpflanzung solcher Rundungsfehler und ihr kombinierter Einfluß auf die Endresultate ist zu untersuchen und abzuschätzen.

Anwendungsorientiert: Die numerische Mathematik löst Probleme, die von Anwendern stammen, und die erarbeiteten Methoden nützen diesen Anwendern.

Die Beschränkung auf die Grundoperationen der Arithmetik bzw. auf endlichstellige Näherungen dafür ist selbstverständlich nicht willkürlich, sondern hat ihren Grund. Die ins Auge gefaßten Konstruktionen sollen in erster Linie auf digitalen Rechenautomaten ausgeführt werden, die eben nur dazu in der Lage sind.

Kapitel 1

Grundbegriffe des numerischen Rechnens

In diesem Kapitel findet sich Grundsätzliches, welches aus logischen Gründen an den Anfang gehört. Es empfielt sich, diesen Stoff am Schluß nochmals durchzugehen, wenn man schon etwas konkretes Material vor Augen hat.

1.1 Gleitpunkt- und Maschinenzahlen, Rundung

Das System $\mathbb{R}$ der reellen Zahlen ist unbegrenzt und lückenlos: Zu jeder reellen Zahl gibt es noch größere und noch kleinere Zahlen, und zu jedem Paar von zwei verschiedenen Zahlen gibt es weitere, dazwischenliegende Zahlen.

Das System $\mathbb{M}$ der in einer Maschine *exakt* darstellbaren reellen Zahlen ist immer finit, also beschränkt und diskret. Praktisch alle Hardware- und Software-Implementierungen verwenden ein System, das durch vier Parameter beschrieben wird. Die beiden folgenden Definitionen geben die Details.

Definition 1.1 *Normalisierte t-stellige Gleitpunktzahlen zur Basis B*

$$\mathbb{G}_{B,t} := \{S \cdot B^E : S = 0 \quad \text{oder} \quad B^{t-1} \leq |S| < B^t,\ S \in \mathbb{Z},\ E \in \mathbb{Z}\}.$$

Die festen Parameter sind **Basis (Radix)** $B \in \mathbb{N}\backslash\{1\}$ und **Stellenzahl** $t \in \mathbb{N}$. Die ganzzahligen Variablen S und E heißen **Signifikand** und **Exponent**. Für $g = S \cdot B^E \neq 0$ ist die Zuordnung $g \leftrightarrow S, E$ ein-eindeutig.

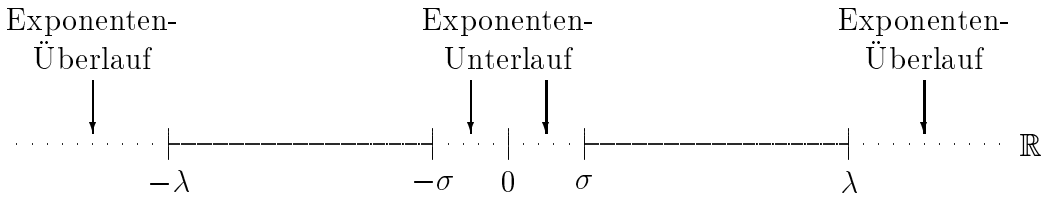
Definition 1.2 *Maschinenzahlen*

$$\mathbb{M}_{B,t,\alpha,\beta} := \{g \in \mathbb{G}_{B,t} : \alpha \leq E \leq \beta\}.$$

Hier ist auch der Exponent begrenzt und das System ist finit. B, t, α und β sind durch die Implementierung festgelegt und werden somit niemals explizit gespeichert. Von einer Maschinenzahl g werden S und E gespeichert und zwar in einer

(Die *un*normalisierten Zahlen haben auch $0 < |S| < B^{t-1}$, doch ist dabei $E > \alpha$.) Sonder-Operanden werden meist mittels Software-Unterstützung implementiert.

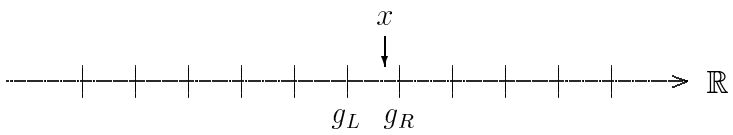
Man spricht von **Bereichs-Überschreitung** oder **Exponenten-Unter/Überlauf**, wenn sich im Lauf einer Rechnung Werte ergeben, die außerhalb des Bereiches $[-\lambda, -\sigma] \cup \{0\} \cup [+\sigma, +\lambda]$ von $\mathbb{M}_{B,t,\alpha,\beta}$ liegen:



Gute Numerik-Software muß Maßnahmen gegen Bereichs-Überschreitung dort vorsehen, wo dies besonders wahrscheinlich ist, zum Beispiel bei der Berechnung von Determinanten. Man würde aber die Verständlichkeit eines Programmes ruinieren, wenn man überall solche Vorkehrungen treffen würde. Stattdessen verläßt man sich darauf, daß jede Bereichs-Überschreitung vom Computer dem Anwender gemeldet wird und nicht stillschweigend mit Ersatz-Resultaten weitergerechnet wird. Diese Einstellung hat zur Konsequenz, daß man in der Numerik von der weiteren Diskussion der Bereichs-Überschreitung absieht, also immer das System $\mathbb{G}_{B,t}$ der Gleitpunktzahlen zur Diskussions-Grundlage nimmt statt des Systems $\mathbb{M}_{B,t,\alpha,\beta}$ der Maschinenzahlen. Auch hier wird es so gehalten und obendrein vereinfachend $\mathbb{G}$ statt $\mathbb{G}_{B,t}$ geschrieben, solange kein Bezug auf die Parameter B und t genommen wird.

Wir kommen zum Runden. Normalerweise assoziiert man diesen Begriff mit den arithmetischen Operationen, wo tatsächlich die wichtigste Anwendung vorliegt. Es ist jedoch zweckmäßiger, das Runden einer beliebigen reellen Zahl x zu betrachten, ohne die einseitige Festlegung der Herkunft von diesem x. Jedes reelle x hat in $\mathbb{G}$ genau einen Nachbarn zur Linken und zur Rechten:

$$g_L := \max\{g \in \mathbb{G} : g \leq x\} \qquad \text{und} \qquad g_R := \min\{g \in \mathbb{G} : g \geq x\}$$



Insbesondere $x \in \mathbb{G} \Rightarrow g_L = g_R = x$, zum Beispiel $x = 0 \Rightarrow g_L = g_R = 0$. Speziell

Beweis: Es genügt, den Fall $x > 0$ anzusehen. Nach obiger konkreten Formel für g_L, g_R ist dann $\varepsilon_x = -\vartheta/(S+\vartheta)$ oder $(1-\vartheta)/(S+\vartheta)$. Der Zähler ist entweder zwischen 0 und ± 1 oder zwischen $-\frac{1}{2}$ und $+\frac{1}{2}$. Der Nenner ist immer $\geq B^{t-1}$. ■

Hinweis: Zum Runden in $\mathbb{G}_{B,t}$ benötigt man nicht eine *vollständige* Kenntnis von x bzw. von ϑ in der Aufspaltung $x = \pm(S+\vartheta)\cdot B^E$. Neben dem Vorzeichen, S und E genügt die Unterscheidung zwischen $\vartheta = 0$ und $\vartheta > 0$ bzw. zwischen $\vartheta \leq \frac{1}{2}$ und $\vartheta \geq \frac{1}{2}$. Das ist entscheidend für die Implementierung der Rundung bei den arithmetischen Operationen, wo das exakte Resultat dieser Operation das zu rundende x ist. Man kann dieses exakte x nicht in allen Fällen bereitstellen, aber immer genau soviel Information wie man sie für die obigen vier Rundungs-Vorschriften benötigt. Die Quadratwurzel ist dafür ein einleuchtendes Beispiel.

1.2 Gleitpunkt-Operationen und ihre Rundungsfehler

Die vier arithmetischen Grundoperationen können im System $\mathbb{G}$ der Gleitpunktzahlen im allgemeinen nicht exakt ausgeführt werden. Der Rechenautomat liefert stattdessen irgendwelche Näherungswerte aus, die mit einem Punkt markiert werden: $a \dot{+} b$ bezeichnet also das Element in $\mathbb{G}$, welches die Maschine liefert, wenn sie die Elemente a und $b \in \mathbb{G}$ addiert (und dabei kein Exponenten-Über/Unterlauf eintritt). Analog für die übrigen Operationen, also $\forall a, b \in \mathbb{G}$ und für $* \in \{+, -, \times, /\}$:

$$\begin{aligned} &\text{exaktes Resultat} && a * b \in \mathbb{R}, \quad \text{i.a. } \notin \mathbb{G}, \\ &\text{berechnete Näherung} && a \dot{*} b \equiv fl(a * b) \in \mathbb{G}. \end{aligned}$$

Die Bezeichnung $fl()$ ist weit verbreitet in der Literatur, basierend auf dem englischen *floating-point* für Gleitpunkt. $fl(a*b)$ ist eine satz-technisch sehr bequeme Schreibweise, aber leider nicht ganz präzise, weil sie suggeriert, daß die berechnete Näherung eine Funktion des exakten Resultats allein ist, während sie bei vielen Maschinen tatsächlich von beiden Operanden und auch von der Operation abhängt, also schwerfälliger mit $fl_*(a,b)$ zu bezeichnen wäre.

Die theoretisch bestmögliche Näherung ist natürlich das im Sinne der Definition 1.3 gerundete exakte Resultat:

$$\forall a, b \in \mathbb{G}, \quad \text{für } * \in \{+, -, \times, /\}: \qquad a \dot{*} b = \mathrm{rd}(a * b)$$
$$\text{(außer bei Exponenten-Über/Unterlauf).}$$

Bei dieser idealen Arithmetik ist übrigens die berechnete Näherung tatsächlich eine Funktion allein des exakten Resultats.

Wichtig: *Dieses Ideal ist kein unerfüllbarer Wunschtraum der Numeriker, sondern kann ohne großen technischen Aufwand realisiert werden. Es erfordert nur ein bißchen Sorgfalt und Aufgeschlossenheit beim Chip-Design.*

In Anbetracht dieser Realisierbarkeit verlangt der schon erwähnte ANSI/IEEE-754

Bemerkungen:

- α, σ, μ, δ sind also die relativen Fehler der berechneten Näherungen.
- Obiger Satz gilt nicht bei Exponenten-Über/Unterlauf.
- Die Formeln in Satz 1.5 sind exakte Gleichungen im mathematischen Sinn und dürfen daher wie üblich in beliebiger Weise umgeformt und miteinander verknüpft werden.

Nicht alle Rechenmaschinen besitzen eine im obigen Sinne ideale Arithmetik. Aber viele Anlagen liefern wenigstens Näherungen, die einen beschränkten relativen Fehler haben. Sie erfüllen dann die

Definition 1.6 *Starke Hypothese für Rundungsfehler der Grundoperationen*

$\exists$ $\varepsilon_{\text{mach}} = O(B^{1-t})$, so daß $\forall a, b \in \mathbb{G}_{B,t}$ gilt

$$\begin{aligned} a \overset{\bullet}{+} b &= (a+b)\cdot(1+\alpha), & a \overset{\bullet}{\times} b &= (a\times b)\cdot(1+\mu),\\ a \overset{\bullet}{-} b &= (a-b)\cdot(1+\sigma), & a \overset{\bullet}{/} b &= (a/b)\cdot(1+\delta), \quad b\neq 0, \end{aligned}$$

mit $|\alpha(a,b)|$, $|\sigma(a,b)|$, $|\mu(a,b)|$, $|\delta(a,b)| \leq \varepsilon_{\text{mach}}$.

Hier sind genau wie in Satz 1.5 α, σ, μ, δ die relativen Fehler bei der Addition, Subtraktion, Multiplikation und Division von a, b. Die Schranke $\varepsilon_{\text{mach}}$ ist für jede solche Rechenanlage gesondert zu ermitteln.

Es gibt Rechenanlagen mit einer so unsauberen Arithmetik, daß der relative Fehler bei der Addition und Subtraktion nicht mehr begrenzt ist. Hier muß man die Hypothese noch weiter abschwächen:

Definition 1.7 *Schwache Hypothese für Rundungsfehler der Grundoperationen*

$\exists$ $\varepsilon_{\text{mach}} = O(B^{1-t})$, so daß $\forall a, b \in \mathbb{G}_{B,t}$ und $* \in \{+, -, \times, /\}$ gilt

$$a \overset{\bullet}{*} b = [a\cdot(1+\varepsilon_1)] * [b\cdot(1+\varepsilon_2)]$$

für geeignete $\varepsilon_1 = \varepsilon_1(a,b)$, $\varepsilon_2 = \varepsilon_2(a,b)$ mit $|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_{\text{mach}}$.

Man beachte, daß ε_1 und ε_2 nicht eindeutig sind. Man gewinnt sie und die Schranke $\varepsilon_{\text{mach}}$ im allgemeinen durch Inspektion der Schritte, die im Rechenwerk einer Anlage ablaufen (Mikroprogramme), wobei man immer den ungünstigsten Fall ins Auge fassen muß.

Meist nimmt man Satz 1.5 bzw. die starke Hypothese 1.6 zum Axiom für die Rundungsfehler-Analyse der Algorithmen. In vielen Fällen ist jedoch die schwache Hypothese 1.7 dafür ausreichend.

große Fehler in den Zwischenresultaten zulassen, sofern diese so korreliert sind, daß sie sich auf das Endresultat insgesamt nur ganz wenig auswirken. Die Intervall-Arithmetik ignoriert diese Korrelation und liefert dann unbrauchbar große Intervalle.

Die Befürworter der Intervall-Arithmetik benützen dieses Instrument deshalb vorwiegend in Spezial-Algorithmen, bei denen Fehler-Korrelationen keine Rolle spielen dürfen. Meistens sind das sogenannte Kontrollrechnungen am Ende eines Standard-Algorithmus, bei denen die Intervall-Arithmetik benutzt wird, um Fehlerschranken für ein Resultat zu gewinnen, das auf konventionelle Weise berechnet wurde.

Was den Aufwand an Speicherplatz und Rechenzeit betrifft, so dürfte die Intervall-Arithmetik in etwa vergleichbar sein mit dem Rechnen in doppelter Genauigkeit.

1.3 Techniken der Rundungsfehler-Analyse

Unter einem **Algorithmus** (Rechenverfahren) versteht man in der Numerik eine endliche Folge von Grundoperationen, deren Reihenfolge beim Ablauf eindeutig festliegt. In Gleitpunkt-Arithmetik ausgeführt, verfälschen Rundungsfehler die Zwischen- und Endresultate.

Zur Aufgabe der Numerik gehört nicht nur die Entwicklung von effizienten Berechnungs-Methoden, sondern auch die Analyse der dabei auftretenden Rundungsfehler. Das soll eine generelle Aussage über die mit dem vorgelegten Algorithmus garantiert erzielbare Genauigkeit sein. Eine solche *a priori* Analyse geht von einer realistischen Hypothese über die Rundungsfehler der Grundoperationen aus und ermittelt daraus Schranken für den Gesamtfehler. Weil dabei immer die Verkettung der ungünstigsten Umstände unterstellt werden muß, ist der tatsächliche Fehler in einer konkreten Anwendung meist viel kleiner.

Naheliegend für solche Untersuchungen wäre eine sogenannte **Vorwärts-Analyse**, welche das bei einer Rechnung in Gleitpunkt-Arithmetik angefallene Resultat vergleicht mit dem exakten Resultat, wie es sich bei der hypothetischen, von Rundungsfehlern freien Rechnung ergeben hätte. Versuche in dieser Richtung hatten bisher nur wenig Erfolg und ein Grund dafür ist bekannt: wird in einem Algorithmus das Zwischenresultat z aus den vorangegangenen Zwischenresultaten x und y gewonnen, so erhält man eine realistische Schranke für den Rundungsfehler in z nicht aus den Schranken für die Rundungsfehler in x und y allein, vielmehr muß man in der Regel auch noch deren Korrelation berücksichtigen. Dadurch wird die Verfolgung von Rundungsfehlern in der Vorwärts-Richtung zu schwierig.

Aus diesem Grund hat sich für die *a priori* Rundungsfehler-Analyse der umgekehrte Weg durchgesetzt, nämlich die sogenannte **Rückwärts-Analyse**, welche das berechnete Resultat interpretiert als das exakte Resultat zu geeignet veränderten Eingabe-Daten. Die zur Interpretation erforderlichen Modifikatoren müssen keineswegs eindeutig sein und im allgemeinen begnügt man sich mit der Angabe von Schranken dafür. Solche Aussagen auf der Eingabe-Seite eines Algorithmus las-

Zwei echte, aus der Praxis stammende Beispiele für die Rückwärts-Analyse finden sich in Abschnitt 3.4 und 7.4. Hier begnügen wir uns mit einer winzigen Demonstration, der Berechnung des Polynomwertes

$$y = c_0 + c_1 x + \ldots + c_n x^n = (\cdots((c_n x + c_{n-1}) \cdot x + c_{n-2}) \cdots + c_1) \cdot x + c_0 .$$

Dieser Algorithmus heißt **Horner-Schema** (engl. *nested multiplications*):

$y := c_n$;
`für` $k := n-1$ `abwärts bis` 0: $y := y \cdot x + c_k$.

Für die in Gleitpunkt-Arithmetik anfallenden Werte gilt nach der Hypothese 1.6

$\tilde{y} := c_n$;
`für` $k := n-1$ `abwärts bis` 0: $\tilde{y} := (\tilde{y} \cdot x \cdot (1+\mu_k) + c_k) \cdot (1+\alpha_k)$.

Zusammen also

$$\begin{array}{rcl} \tilde{y} & = & \tilde{c}_0 + \tilde{c}_1 x + \cdots + \tilde{c}_n x^n \\ \text{mit } \tilde{c}_k & := & c_k \cdot (1+\alpha_k) \cdot (1+\mu_{k-1}) \cdot \ldots \cdot (1+\alpha_0), \quad \alpha_n := 0. \end{array}$$

Man sieht, daß der berechnete Polynomwert interpretiert werden kann als der exakte Wert von einem Polynom mit leicht veränderten Koeffizienten (Eingabe-Werte).

Wie in diesem Beispiel, so treten auch sonst häufig die Modifikatoren $1+\varepsilon$ recht gehäuft auf. Natürlich könnte man linearisieren und n solche Faktoren ersetzen durch $1+n\varepsilon$. Ohne Vernachlässigung höherer Fehlerterme ist das möglich durch geringfügiges Erhöhen der Schranke $\varepsilon_{\text{mach}}$. Es gilt nämlich folgendes

Lemma 1.8 *(W. Sautter)*

$$\text{Sei } \prod_{i=1}^{n} (1+\varepsilon_i)^{\pm 1} =: 1+\varepsilon. \qquad \text{Alle } |\varepsilon_i| \le \varepsilon_{\text{mach}} \implies |\varepsilon| \le \frac{n\varepsilon_{\text{mach}}}{1-n\varepsilon_{\text{mach}}}.$$

Beweis: $(1-\varepsilon_i)(1+\varepsilon_i) \le 1 \Rightarrow 1-\varepsilon_{\text{mach}} \le (1+\varepsilon_i)^{\pm 1} \le (1-\varepsilon_{\text{mach}})^{-1}$. Schluß von $n-1$ auf n gibt $1-n\varepsilon_{\text{mach}} \le (1-\varepsilon_{\text{mach}})^n$ (Bernoullische Ungleichung). ■

Neben der *a priori* Rundungsfehler-Analyse von Algorithmen gibt es noch die Möglichkeit, **a posteriori Fehlerschranken** für die numerisch berechnete Näherung $\tilde{x}$ der exakten Lösung x eines Problems zu ermitteln. Dabei geht es darum, durch eine nachträgliche Kontroll-Rechnung den Fehler $\tilde{x} - x$ abzuschätzen. Meistens ist das eine Einsetz-Probe, ausgeführt in erhöhter Genauigkeit. Man beachte, daß solche *a posteriori* Fehlerschranken immer nur Fallstudien sind, also Aussagen über die erzielte Genauigkeit in dem vorliegenden Zahlenbeispiel machen, niemals dagegen grundsätzliche Aussagen über die mit einem Algorithmus erzielbare Genauigkeit.

Aus der linearen Algebra ist bekannt, daß $(A - \lambda I)x = 0$ genau dann eine Lösung $x \neq 0$ besitzt, wenn $A - \lambda I$ singulär ist. Die Eigenwerte erhält man also als Nullstellen des charakteristischen Polynoms

$$\det(zI - A) =: c_0 + c_1 z + \ldots + c_n z^n.$$

Damit läßt sich **Problem 3** in zwei Teilprobleme zerlegen:

$$\begin{array}{ll} \text{Teil 1:} & A \mapsto c_0, \ldots, c_n \\ \text{Teil 2:} & c_0, \ldots, c_n \mapsto \lambda. \end{array}$$

Eine solche Zerlegung von p in zwei oder mehr Teilprobleme ist typisch. In der abstrakten Form:

$$p = r \circ q, \qquad \text{d.h.} \qquad p(x) = r(q(x)),$$

wobei $q\colon \mathbb{D} \to \mathbb{R}^l$ und $r\colon \mathbb{R}^l \to \mathbb{R}^m$. $z = q(x)$ wird hier **Zwischenresultat** genannt. Eine feinere Zerlegung ist analog formulierbar:

$$p = p^{(k)} \circ \ldots \circ p^{(2)} \circ p^{(1)} \qquad \Longleftrightarrow \qquad p(x) = p^{(k)}\Big(\ldots p^{(2)}(p^{(1)}(x))\ldots\Big).$$

Ein neuer, außerordentlich wichtiger Aspekt tritt auf, wenn die Eingabe-Daten nicht exakt bekannt sind, sondern mit einer Unsicherheit Δx behaftet sind, so daß alle n-Tupel $x + \delta x$ mit $|\delta x| \leq \Delta x$ gleichberechtigt mit x sind. (Beträge und $\leq$ verstehen sich grundsätzlich komponentenweise.) Im Augenblick spielt es kaum eine Rolle, ob Δx eine mittlere oder maximale Fehlerangabe darstellt (**Streuung** oder **Schranke**), auch könnte die Abgrenzung der gleichberechtigten Eingabe-Daten nicht komponentenweise sondern pauschal sein: Definition von $\|\delta x\| \leq \Delta x$ ($\|\cdot\|$ vgl. Abschnitt 2.2). Wichtig ist, daß damit auch das Resultat fehlerbehaftet ist.

Je nach Problem können kleine Änderungen δx in den Eingabe-Daten kleine oder große Änderungen δy im Resultat hervorrufen, und auch die Festlegung von „klein“ und „groß“ muß für jedes Problem passend getroffen werden. In erster Näherung gilt für infinitesimale Änderungen δx

$$\delta y = \frac{\partial p}{\partial x} \cdot \delta x$$

mit $\partial p / \partial x$ die Funktional-Matrix der Abbildung p, welche als differenzierbar vorausgesetzt wird. Für die relativen Fehler $\varrho x_k := \delta x_k / x_k$ und $\varrho y_i := \delta y_i / y_i$ folgt

$$\varrho y_i = \sum_{k=1}^{n} \frac{x_k}{y_i} \frac{\partial p_i}{\partial x_k} \varrho x_k, \qquad i = 1, \ldots, m.$$

Darauf aufbauend wird zum Teil qualitativ vereinbart:

indem man den technischen Aufwand steigert und mit mehr Stellen rechnet („doppelte oder mehrfache Genauigkeit“). Viel schwieriger ist es dagegen, die Fehler Δx der Eingabe-Daten so klein zu machen, daß das Resultat trotz der schlechten Kondition noch signifikant ist. Aber das ist ausschließlich Sache des Anwenders. Der Numeriker als Berater ist hier mehr in der Rolle eines guten Anwalts, der seinem Klienten auch mal von der Führung eines Prozesses abraten sollte.

Wenn das Problem p zerlegt wird in Teilprobleme, dann hat jedes Teilproblem seine eigene Kondition, denn nach der Kettenregel gilt für $p = r \circ q$

$$\frac{\partial p}{\partial x} = \left(\frac{\partial r}{\partial z}\right)_{z=q(x)} \cdot \frac{\partial q}{\partial x}$$

Obige Konditionszahlen sind also nach den Regeln der Matrizenrechnung zu multiplizieren :

$$\begin{aligned} \frac{\partial p_i}{\partial x_k} &= \sum_{j=1}^{l} \frac{\partial r_i}{\partial z_j} \cdot \frac{\partial q_j}{\partial x_k}, \\ \frac{x_k}{y_i}\frac{\partial p_i}{\partial x_k} &= \sum_{j=1}^{l} \left(\frac{z_j}{y_i}\frac{\partial r_i}{\partial z_j}\right) \cdot \left(\frac{x_k}{z_j}\frac{\partial q_j}{\partial x_k}\right), \\ & \qquad i = 1, \ldots, m, \;\; k = 1, \ldots, n. \end{aligned}$$

Möglichst einprägsam formuliert: $K_{r\circ q} = K_r K_q$, wobei K die Matrizen der absoluten oder der relativen Konditionszahlen sind.

Man beachte: Die linke Seite dieser Gleichungen beschreibt die absolute bzw. relative Kondition des *Gesamtproblems* und ist damit unabhängig von der Zerlegung. Also: Die Faktoren hängen von der Zerlegung ab, doch ihr Produkt ist davon unabhängig. Das gilt selbstverständlich auch für feinere Zerlegungen.

Die Kondition der Teilprobleme wird sehr wichtig, wenn man diese nicht exakt lösen kann, sondern dabei Fehler begeht, insbesondere Rundungsfehler. Der Gesamtfehler des berechneten Resultats setzt sich dann zusammen aus den Fehlern in den Eingabe-Daten und den Fehlern, begangen in den Teilschritten. Bei der Zerlegung $p = r \circ q$ hat man:

Eingabe:	$\tilde{x} = x + \delta x$	statt x,
	$\delta x =$ Fehler der Eingabe-Daten,	
Zwischenwert:	$\tilde{z} = q(\tilde{x}) + \delta z$	statt $z = q(x)$,
	$\delta z =$ Fehler des ersten Teils,	
Endresultat:	$\tilde{y} = r(\tilde{z}) + \delta y$	statt $y = r(z)$,
	$\delta y =$ Fehler des zweiten Teils.	

In erster Ordnung gilt dann:

$$\begin{aligned} \tilde{z} &= q(x + \delta x) + \delta z = z + q'\delta x + \delta z, \\ \tilde{y} &= r(z + q'\delta x + \delta z) + \delta y = y + r'q'\delta x + r'\delta z + \delta y. \end{aligned}$$

(a) Profil von $\det(\lambda I - A) = c_0 + c_1\lambda + \cdots + c_{12}\lambda^{12}$ für $[0, 13] \times [-2 \cdot 10^6, +2 \cdot 10^6]$.

(b) Rundung der Koeffizienten c_k auf 24 Bit ändert die Form des Polynoms vollständig und macht Schrott aus seinen Nullstellen (den Eigenwerten).

In Definition 1.11 ist die Herkunft der Näherung $\tilde{y}$ grundsätzlich beliebig, in der Praxis ist sie jedoch das Resultat einer vorangegangenen Berechnung, also numerisch beschafft. In diesem Fall sind es Verfahrens- und Rundungs-Fehler, die die Abweichung vom exakten Resultat bewirken. Zu ersteren gehören u.a. Approximationen von Reihen und Integralen durch Summen, Ableitungen durch Differenzen-Quotienten, Abbrechen einer Iteration nach endlich vielen Schritten. Sie sind zu besprechen bei den einzelnen Problemen und ihren Lösungsmethoden.

Definition 1.12

Ein Algorithmus A zur Lösung von p heißt **numerisch stabil** oder **gutartig** (F.L. Bauer, 1965), wenn er für alle zulässigen Eingabe-Daten x unter dem Einfluß von Verfahrens- und Rundungsfehlern Näherungslösungen $\tilde{y} = p_A(x)$ von $y = p(x)$ produziert, die akzeptabel sind für Eingabe-Fehler in der Größenordnung der Rechengenauigkeit, wenn sie also Definition 1.11 (1.4) genügen mit dem speziellen $U_x = \{x' \in \mathbb{D} : |x' - x| \leq O(\varepsilon_{\text{mach}})|x|\}$.

In Zeichen:

$$\begin{aligned} &\forall x \in \mathbb{D}\ \exists x' \in \mathbb{D}\ \ni \\ &\qquad x' \in U_x \qquad \text{und} \qquad p_A(x) \in U_{p(x)}, \quad d.h. \\ &|x' - x| \leq O(\varepsilon_{\text{mach}})|x| \qquad \text{und} \qquad |p(x') - p_A(x)| \leq O(\varepsilon_{\text{mach}})|p_A(x)|. \end{aligned}$$

(Oder überall $\|\cdot\|$ statt $|\cdot|$.)

Bemerkungen:

- Es ist hier die abgeschwächte Form von „akzeptabel“ verwendet worden, sonst gäbe es zu wenig numerisch stabile Algorithmen.
- Die Definition ist praktisch unabhängig von der Rechengenauigkeit $\varepsilon_{\text{mach}}$, weil die Verfahrens- und Rundungsfehler genau wie die Toleranzen proportional zu $\varepsilon_{\text{mach}}$ sind.
- Es ist aber denkbar, daß ein Algorithmus numerisch stabil ist unter der starken Hypothese 1.6, nicht aber unter der schwachen Hypothese 1.7.
- Der Fehler eines numerisch stabilen Algorithmus muß keineswegs in irgendeinem Sinn klein sein: je schlechter die Kondition eines Problems ist, desto größere Fehler werden zugelassen.
- Die Grundoperationen $+$, $-$, $\times$, $/$ sind in diesem Sinn numerisch stabil, wenn man unterstellt, daß alle Maschinen die schwache Hypothese 1.7 erfüllen.
- Zwei numerisch stabile Algorithmen nacheinander ausgeführt, ergibt keineswegs immer einen numerisch stabilen Algorithmus. (Sonst wären alle Algorithmen numerisch stabil, weil es ja die Grundoperationen sind, aus denen sie sich zusammensetzen.)

Kapitel 2

Vektoren und Matrizen

2.1 Schreibweise und Bezeichnungen

Als bekannt vorausgesetzt wird das Konzept des Vektorraumes (linearen Raumes) über dem Körper der reellen oder komplexen Zahlen, sowie Begriffe wie lineare Abhängigkeit/Unabhängigkeit, Dimension, Untervektorraum (Teilraum), Basis. Ebenfalls bekannt sein sollte die darauf aufbauende lineare Abbildung und Begriffe wie Wertebereich, Rang, Nullraum (= Kern), Inverse, Determinante. Schließlich sei daran erinnert, daß nach Wahl einer Basis jeder n-dimensionale reelle oder komplexe Vektorraum isomorph ist zum $\mathbb{R}^n$ bzw. $\mathbb{C}^n$:

$$x \in \mathbb{R}^n \text{ bzw. } \mathbb{C}^n \qquad \Longleftrightarrow \qquad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \qquad \text{alle} \quad x_i \in \mathbb{R} \text{ bzw. } \mathbb{C}.$$

Aus drucktechnischen Gründen und um Platz zu sparen, wird im folgenden $x = (x_1, \ldots, x_n)^T$ geschrieben: Der Hochindex T steht für **transponiert** und macht aus dem Nacheinander der Komponenten ein Untereinander. Diese Darstellung eines Vektors durch einen Satz von Komponenten (= Koeffizienten bezüglich einer Basis) ist sehr Computer-angepaßt und bedeutet keinerlei Einschränkung der Allgemeinheit. Selbstverständlich kann man x immer auffassen als eine Abbildung der Menge $\{1, \ldots, n\}$ in die reellen bzw. komplexen Zahlen. In der Numerik wird dann gern der Name Gitterfunktion verwendet. Dann leuchtet es auch ein, daß anstelle des einfachen Index i in der Praxis auch ein Index-Paar oder -Tripel treten kann. Das hat dann die äußere Form aber nicht die Aufgabe (Funktion) einer Matrix.

Matrizen dienen zur Beschreibung einer linearen Abbildung des $\mathbb{R}^n$ bzw. $\mathbb{C}^n$ in den $\mathbb{R}^m$ bzw. $\mathbb{C}^m$:

$$A \in \mathbb{R}^{m,n} \text{ bzw. } \mathbb{C}^{m,n} \quad \Longleftrightarrow \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad \text{alle } a_{ij} \in \mathbb{R} \text{ bzw. } \mathbb{C}.$$

Die Regeln der Matrizen-Rechnung werden als bekannt unterstellt. Bei $m = n$

Geht man dabei über zum konjugiert Komplexen, dann erhält man die **hermitisch konjugierte** Matrix $A^H \in \mathbb{C}^{n,m}$, also

$$(A^H)_{ij} := \bar{a}_{ji}, \quad i = 1, \ldots, n \text{ und } j = 1, \ldots, m.$$

Immer gilt $(AB)^T = B^T A^T$ und $(AB)^H = B^H A^H$. Insbesondere ist $(A^T)^{-1} = (A^{-1})^T$, kurz A^{-T}, und $(A^H)^{-1} = (A^{-1})^H$, kurz A^{-H}. $x^H y = \sum_i \bar{x}_i y_i$ ist das **Euklidsche Skalarprodukt**.

A heißt **selbst-adjungiert** oder **hermitisch**, wenn $A = A^H$. Im reellen Fall heißt A dann **symmetrisch**: Also $A = A^T \in \mathbb{R}^{n,n}$. Solche Matrizen sind besonders wichtig (häufig) beim Eigenwertproblem.

Jede selbst-adjungierte Matrix A definiert eine **quadratische Form** $x^H Ax$, die nach obigen Rechenregeln immer reellwertig ist. A heißt

positiv-definit, wenn	$x^H Ax$	$>$	0	$\forall x \neq 0$,
positiv-semidefinit, wenn	$x^H Ax$	$\geq$	0	$\forall x$,
negativ-semidefinit, wenn	$x^H Ax$	$\leq$	0	$\forall x$,
negativ-definit, wenn	$x^H Ax$	$<$	0	$\forall x \neq 0$,
indefinit sonst.				

Positiv-definite Matrizen sind besonders wichtig bei linearen Gleichungssystemen, sie besitzen immer eine Inverse, denn $Ax = 0 \Rightarrow x^H Ax = 0 \Rightarrow x = 0$. Auch A^{-1} ist dann positiv-definit, denn $x^H A^{-1} x = (A^{-1}x)^H A(A^{-1}x) > 0 \quad \forall x \neq 0$.

2.2 Normen für Vektoren und Matrizen

Beträge und $<, \leq, \geq, >$ verstehen wir immer komponentenweise, z.B.

$$\begin{aligned} |x| &:= (|x_1|, \ldots, |x_n|)^T, \\ |A| \leq |B| &\iff |a_{ij}| \leq |b_{ij}| \qquad \forall i, j. \end{aligned}$$

Normen helfen uns, Größenvergleiche und Abschätzungen wie „x nahe an y“ oder „Δx vernachlässigbar klein gegenüber x“ pauschal zu formulieren.

Definition 2.1 *Die für die Praxis wichtigen Vektornormen*

Summennorm:	$\|x\|_1$	$:=$	$\sum_i \|x_i\|$,
Euklidsche Norm (Länge):	$\|x\|_2$	$:=$	$\sqrt{\sum_i \|x_i\|^2}$,
Maximums-Norm:	$\|x\|_\infty$	$:=$	$\max_i \|x_i\|$.

Für welche x werden die Schranken jeweils angenommen?

Tip: $\|x\|_1 = e^T|x| \leq \|e\|_2 \cdot \|x\|_2$ mit $e^T := (1, \ldots, 1)$.

Auch für Matrizen (Abbildungen) braucht man Normen. *Eine* Möglichkeit dafür ist, die Matrix-Elemente als Komponenten eines Vektors zu betrachten. Das wird aber dem Wesen der Matrix als Abbildung nicht gerecht. Viel zweckmäßiger ist das Konzept der

Definition 2.2 *Operatornorm*

Die Operatornorm $\|\cdot\|: \mathbb{C}^{m,n} \to \mathbb{R}$ einer Matrix A wird definiert durch

$$\|A\| := \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

Jede Vektornorm im $\mathbb{C}^n$ und $\mathbb{C}^m$ definiert eine Operator-Norm, insbesondere die obigen drei Normen $\|\cdot\|_p$, $p = 1, 2, \infty$.

Unmittelbar aus der Definition der Operatornorm ergeben sich fünf Eigenschaften:

definit, d.h.	$A \neq 0 \quad \Longrightarrow \quad \|A\| > 0,$	(2.4)
homogen, d.h.	$\|\lambda A\| = \|\lambda\| \cdot \|A\|, \quad \forall \lambda \in \mathbb{C}$	(2.5)
sub-additiv, d.h.	$\|A + B\| \leq \|A\| + \|B\|,$	(2.6)
sub-multiplikativ, d.h.	$\|AB\| \leq \|A\| \cdot \|B\|,$ (A und B müssen nicht quadratisch sein)	(2.7)
Konsistenz von Matrix- und Vektornorm, d.h.	$\|Ax\| \leq \|A\| \cdot \|x\|.$	(2.8)

Beweis: Alles ist klar außer (2.7) falls $B \neq 0$:

$$\begin{aligned}
\max_{x \neq 0}(\|ABx\|/\|x\|) &= \max_{Bx \neq 0}(\|ABx\|/\|x\|) \\
&\quad \text{(denn ein } x \text{ mit } Bx = 0 \text{ hat keine Chancen)} \\
&= \max_{Bx \neq 0}\Big((\|ABx\|/\|Bx\|) \cdot (\|Bx\|/\|x\|)\Big) \\
&\leq \max_{Bx \neq 0}(\|ABx\|/\|Bx\|) \cdot \max_{x \neq 0}(\|Bx\|/\|x\|) \\
&\leq \max_{y \neq 0}(\|Ay\|/\|y\|) \cdot \max_{x \neq 0}(\|Bx\|/\|x\|) \\
&\quad \text{(mehr Konkurrenz).}
\end{aligned}$$

(2.8) ist eine unmittelbare Folge der Definition:

$$x \neq 0: \qquad \frac{\|Ax\|}{\|x\|} \leq \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \|A\|.$$

■

1. $\|A\|_F^2 = \text{Spur}(A^H A) = \lambda_1 + \lambda_2 + \ldots$ (λ_i wie in Satz 2.3).
2. $1 \leq \|A\|_F^2 / \|A\|_2^2 \leq \min(m, n)$.
3. Die Eigenschaften (2.4), (2.5), (2.6) gelten.
4. Eigenschaft (2.7) gilt. Dazu Tip: $B =: (b^1, b^2, \ldots)$, $\|B\|_F^2 = \|b^1\|_2^2 + \|b^2\|_2^2 + \ldots$, $\|AB\|_F^2 = \|Ab^1\|_2^2 + \|Ab^2\|_2^2 + \ldots \leq \|A\|_2^2 \cdot (\|b^1\|_2^2 + \|b^2\|_2^2 + \ldots) \leq \|A\|_F^2 \cdot \|B\|_F^2$.
5. $\|\cdot\|_F$ ist *keine* Operatornorm. Tip: $\|I\|_F = ?$

Übungsaufgabe 2.3: Die Spektral- und die Frobenius-Norm sind unitär invariant: Falls $P \in \mathbb{C}^{m,m}$, $Q \in \mathbb{C}^{n,n}$ unitär, dann

$$\begin{aligned} \|PAQ\|_2 &= \|PA\|_2 = \|AQ\|_2 = \|A\|_2, \\ \|PAQ\|_F &= \|PA\|_F = \|AQ\|_F = \|A\|_F. \end{aligned}$$

Übungsaufgabe 2.4: Schranke für die Eigenwerte einer Matrix: $A \in \mathbb{C}^{n,n}$ und $Ax = \lambda x$ mit $x \neq 0 \quad \Longrightarrow \quad |\lambda| \leq \|A\|$ für jede Operatornorm. Tip: (2.8).

Übungsaufgabe 2.5: Abschätzung für die teuere Spektralnorm: $A \in \mathbb{C}^{m,n} \Rightarrow \|A\|_2^2 \leq \|A^H A\|_\infty \leq \|A^H\|_\infty \cdot \|A\|_\infty = \|A\|_1 \cdot \|A\|_\infty$. Die Spektralnorm ist nie größer als das geometrische Mittel aus Summen- und Maximums-Norm.

Übungsaufgabe 2.6: Für quadratisches A und $\|A\| < 1$ gilt
$\|(I - A)^{-1}\| \leq 1/(1 - \|A\|)$.

Tip: Entweder über die geometrische Reihe und (2.7) oder $y := (I - A)^{-1} x \Leftrightarrow y = x + Ay \Rightarrow \|y\| \leq \|x\| + \|A\| \cdot \|y\|$, also $\|y\|/\|x\| \leq 1/(1 - \|A\|)$.

2.3 Elementare Matrix-Transformationen

In Form einer Liste werden hier die wichtigsten Bausteine zusammengestellt, aus denen Matrix-Algorithmen zusammengesetzt werden. Das Schema ist

- Definition,
- Anwendung auf Vektoren,
- Anwendung auf Matrix von links und von rechts,
- Inverse und Determinante,
- Ähnlichkeits-Transformation (ÄT).

Die elementaren Bausteine werden hier für sich allein eingeführt und nicht eingebunden in einen speziellen Algorithmus und eingesetzt für einen speziellen Zweck. Das soll ihre Vielseitigkeit besser herausstellen und verhindern helfen, daß man die Bausteine zu einseitig mit irgendeinem Beispiel assoziiert. Im nächsten Abschnitt finden sich Skizzen für Anwendungs-Möglichkeiten.

Reihen-Operationen $N_{ij}(\alpha)$

Definition:	$N_{ij}(\alpha)$ hat Extra-Element α in i-j-Position falls $i > j$: $\begin{cases} 1 & 0 \\ \alpha & 1 \end{cases}$ falls $i < j$: $\begin{cases} 1 & \alpha \\ 0 & 1 \end{cases}$
Anwendung:	$N_{ij}(\alpha)x$: α mal Komponente j zu Komponente i addieren: $x_i := x_i + \alpha x_j$. $N_{ij}(\alpha)A$: α mal Zeile j zu Zeile i addieren: $a_{ik} := a_{ik} + \alpha a_{jk}, \quad k = 1, 2, \ldots$. $AN_{ij}(\alpha)$: α mal Spalte i zu Spalte j addieren: $a_{kj} := a_{kj} + \alpha a_{ki}, \quad k = 1, 2, \ldots$.
Inverse:	$N_{ij}(\alpha)^{-1} = N_{ij}(-\alpha), \quad \det(N_{ij}(\alpha)) = 1.$
ÄT:	$A \mapsto N_{ij}(\alpha)AN_{ij}(-\alpha)$. Erst $a_{ik} := a_{ik} + \alpha a_{jk}, \quad k = 1, 2, \ldots,$ dann $a_{kj} := a_{kj} - \alpha a_{ki}, \quad k = 1, 2, \ldots$ (oder umgekehrt).

Ebene Rotation $Q_{ij}(\phi)$

Definition:	$Q_{ij}(\phi)$ hat Extra-Elemente $\begin{cases} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{cases}$
Anwendung:	$(Q_{ij}(\phi)x)_i = x_i\cos\phi + x_j\sin\phi$, $(Q_{ij}(\phi)x)_j = -x_i\sin\phi + x_j\cos\phi$. $Q_{ij}(\phi)A$: Zeile $i :=$ $\cos\phi$ Zeile $i + \sin\phi$ Zeile j, Zeile $j :=$ $-\sin\phi$ Zeile $i + \cos\phi$ Zeile j. $AQ_{ij}(\phi)$: Spalte $i :=$ $\cos\phi$ Spalte $i - \sin\phi$ Spalte j, Spalte $j :=$ $\sin\phi$ Spalte $i + \cos\phi$ Spalte j.
Inverse:	$Q_{ij}(\phi)^{-1} = Q_{ij}(-\phi) = Q_{ij}(\phi)^T$: $Q_{ij}(\phi)$ ist orthogonal. $\det(Q_{ij}(\phi)) = 1$.
ÄT:	$A \mapsto Q_{ij}(\phi)AQ_{ij}(\phi)^T$: Zeilen und Spalten werden in gleicher Weise transformiert!

Im komplexen Fall können sogar zwei Winkel gewählt werden und die Extra-Elemente sind dann

$$\begin{cases} \cos\phi & e^{i\theta}\sin\phi \\ -e^{-i\theta}\sin\phi & \cos\phi \end{cases}$$

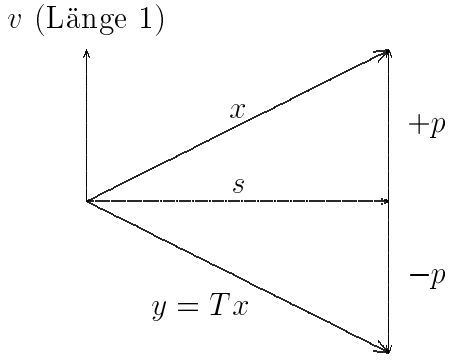
Das damit gebildete Q_{ij} ist unitär: $Q_{ij}^{-1} = Q_{ij}^H$, in der Euklidschen Norm hat es die ideale Konditionszahl 1.

Zum Schluß noch die Erläuterung des Namens.

Dazu zerlegt man x in einen Anteil p parallel zu v und einen Anteil s senkrecht zu v:

$$x = p + s,$$

wobei $p := v(v^H x)$ parallel zu v ist und $s := x - p$ senkrecht zu v ist: $v^H s = v^H x - v^H p = v^H x - v^H v \cdot v^H x = 0$. Dann ist $y = Tx = (I - 2vv^H)(p+s) = p+s-2p = s-p$. T bewirkt also den Übergang $s+p \mapsto s-p$. Dieses Umdrehen des x-Anteils parallel zu v bezeichnet man als Spiegelung von x an der Hyperebene $\perp v$:



Die geometrische Interpretation der Matrix $T := I - 2vv^H$ und die Relation zwischen einfallendem Strahl x, reflektiertem Strahl y und Flächennormale v: Es gilt $x = s + p$, $y = s - p$, wobei p parallel zu v und s senkrecht zu v ist.

Die Spiegelung verknüpft immer drei Richtungen: Flächennormale (heißt hier v), Einfalls-Richtung (hier x), reflektierte Richtung (hier $y = Tx$). Die letzten beiden sind zueinander symmetrisch, wie ausgedrückt durch $T^{-1} = T$. Unter obigem Absatz *Anwendung* ist beschrieben, wie man zu v, x das y bzw. zu v, y das x erhält. Wie bekommt man zu x und y das v? Damit es eine Lösung gibt, müssen natürlich x und y gleich lang und $x^H y = x^H T x$ reell sein, auch sei der uninteressante Fall $x = y$ ausgeschlossen. Obige Formeln geben sofort die Lösung, denn $x - y = (s + p) - (s - p) = 2p$ ist parallel zu v. Dieses ergibt sich somit durch Normierung von $x - y$ auf Länge 1. Oder beim Arbeiten mit dem unnormalisierten u:

$$u = x - y \qquad \text{und} \qquad \kappa = (x-y)^H(x-y)/2 = x^H x - y^H x = u^H x.$$

Das gibt in der Tat

$$Tx = x - u \cdot (u^H x/\kappa) = x - (x - y) \cdot 1 = y.$$

In den Anwendungen hat man es immer mit dem einen Standard-Fall zu tun, nämlich $y \sim e^1$, also dann

$$y = (\pm\|x\|_2 \cdot x_1/|x_1|, 0, \ldots, 0)^T,$$

damit $\|y\|_2 = \|x\|_2$ und $x^H y$ reell ist.

Obige Formeln für u und κ werden besonders robust gegenüber Rundungsfehlern, wenn man $-\|x\|_2 \cdot x_1/|x_1|$ für y_1 wählt (sofern nur $x_1 \neq 0$):

nach Satz 2.5

$$A = N_{2,1}(l_{2,1}) \cdots N_{n,n-1}(l_{n,n-1})R = LR.$$

Man nennt dies die **Dreiecks-** oder die **LR-Zerlegung** der Matrix A. Voraussetzung ist, daß alle Divisionen möglich sind: Zu Beginn des j-ten Hauptschrittes muß $a_{jj} \neq 0$ sein.

Zweites Beispiel: Orthogonale Dreieckszerlegung von $A \in \mathbb{R}^{m,n}$

Man kann im ersten Beispiel die $N_{ij}(-l_{ij})$ ersetzen durch ebene Rotationen Q_{ij}^T. Der Drehwinkel in Q_{ij} wird dabei so eingerichtet, daß wieder die Position i, j annulliert wird

$$0 = -s \cdot a_{jj} + c \cdot a_{ij} \qquad \Longleftrightarrow \qquad \begin{cases} c &= a_{jj}/\sqrt{a_{jj}^2 + a_{ij}^2}\,, \\ s &= a_{ij}/\sqrt{a_{jj}^2 + a_{ij}^2}\,. \end{cases}$$

(Nur wenn $a_{ij} \neq 0$, sonst wähle $c = 1, s = 0$ d.h. $Q_{ij} = I$.) Das führt auf den in-situ-Algorithmus

```
für j := 1 bis n:
für i := j + 1 bis m:
    Berechne c und s in Q_ij wie angegeben;
    A := Q_ij^T A.
```

Wieder stellt die Reihenfolge der Rotationen sicher, daß ein einmal zu 0 reduziertes i-j-Element später nicht wieder von null verschieden werden kann. Die typische Gestalt der Matrix A während des Algorithmus ist wie beim ersten Beispiel, doch ist jetzt die Matrix im allgemeinen nicht mehr quadratisch: meist ist $m \geq n$. Das End-Ergebnis ist eine obere Dreiecksmatrix R, bei $m > n$ sind ihre unteren $m - n$ Zeilen identisch 0

$$\begin{aligned} R &= Q_{m,n}^T \cdots Q_{2,1}^T A, \\ A &= Q_{2,1} \cdots Q_{m,n} R = QR. \end{aligned}$$

Das Produkt Q aller ebenen Rotationen ist selbst eine orthogonale $m \times m$ Matrix. Man nennt diesen Prozeß die orthogonale Dreiecks-Zerlegung oder meistens kurz die **QR-Zerlegung**. Diese Zerlegung ist immer möglich, aber nicht immer eindeutig.

Q braucht man selten, aber fast immer benötigt man $Q^T b = Q_{m,n}^T \cdots Q_{2,1}^T b$ für bestimmte Vektoren b. Man multipliziert die Q_{ij} also niemals aus, sondern wendet sie bei ihrer Entstehung sogleich auf b an: b wird so transformiert wie jede Spalte von A.

Als x in Satz 2.6 wähle man die erste Spalte der Matrix $A \in \mathbb{C}^{m,n}$, so daß

$$T_1 A = \begin{pmatrix} * & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \end{pmatrix}.$$

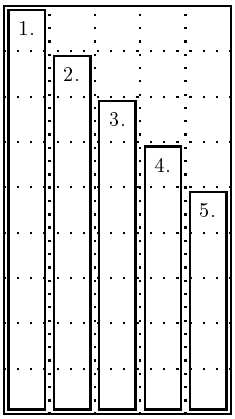
Die erste Spalte hat sich reduziert, alle übrigen Spalten wurden umgerechnet. Dann wird die zweite Spalte reduziert, wobei die erste Zeile ignoriert wird:

$$T_2 T_1 A = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & & * \\ 0 & 0 & * & & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}.$$

Nach n Schritten liegt die QR-Zerlegung vor mit $Q = T_1 \cdots T_n$:

$$T_n \cdots T_1 A = R \qquad \text{bzw.} \qquad A = T_1 \cdots T_n R = QR.$$

Solche Algorithmen beschreibt man am besten allein durch Angabe der **Parkettierung**, hier



Die Parkettierung bei der
Householder-Reduktion
$T_n \cdots T_1 A =: R$
für die Zerlegung $A =: Q \cdot R$
(hier $n = 5$).

Sie gibt die Blöcke der Matrix an, die nacheinander das zu reduzierende x von Satz 2.6 liefern. Der Rest des Algorithmus versteht sich von selbst.

Für rechtsseitige Anwendung von T parkettiert man zeilenweise.

Fünftes Beispiel: Householder-Transformation

Als Übung diskutiere man eine Alternative zum dritten Beispiel, nämlich die Ähnlichkeits-Transformation einer Matrix $A \in \mathbb{C}^{n,n}$ auf obere Hessenberg-Form mittels *Spiegelungen*. Hier sei nur die Parkettierung angegeben:

Klassischer Gram-Schmidt-Prozeß:

für $k := 1$ bis n:
    $x := a^k$;
    für $i := 1$ bis $k-1$: $\{r_{i,k} := q^{iH} a^k;\ x := x - q^i r_{i,k}\}$
    $r_{k,k} := \|x\|_2;\ q^k := x/r_{k,k}$.

Das Skalarprodukt $q^{iH} a^k$ kann hier ersetzt werden durch $q^{iH} x$, weil sich das aktuelle x von a^k nur unterscheidet um Vielfache von $q^1, \ldots, q^{i-1}$, die zum Skalarprodukt nichts beitragen:

Modifizierter Gram-Schmidt-Prozeß:

für $k := 1$ bis n:
    $x := a^k$;
    für $i := 1$ bis $k-1$: $\{r_{i,k} := q^{iH} x;\ x := x - q^i r_{i,k}\}$;
    $r_{k,k} := \|x\|_2;\ q^k := x/r_{k,k}$.

Beide Prozesse verlangen, daß $a^1, \ldots, a^k$ linear unabhängig sind für alle k, denn sonst ergäbe sich $0/0$ für q^k. Das ist in krassem Gegensatz zu den Methoden mit ebenen Rotationen oder Reflexionen. Es kommt noch viel schlimmer: Die beiden Gram-Schmidt-Prozesse sind **numerisch instabil**, d.h. unter dem Einfluß von Rundungsfehlern produzieren sie Vektoren $\tilde{q}^k$ anstelle der exakten q^k, die unter Umständen stark von der Orthogonalität abweichen. In der Tat, die Algorithmen können nur erreichen, daß die Skalarprodukte $q^{iH} x$ bis auf die Rechengenauigkeit zu null gemacht werden. Je kleiner dabei x wird, desto größer wird $\tilde{q}^{iH} \tilde{q}^k$ beim abschließenden Normieren durch die Division mit $\|x\|_2$. Die modifizierte Variante ist hier etwas günstiger als die klassische, doch keine von beiden ist ausreichend genau. Erst Wiederholen der Orthogonalisierung macht die berechneten $\tilde{q}^k$ bis auf die Maschinengenauigkeit orthogonal und den Prozeß numerisch stabil:

Gram-Schmidt-Prozeß mit Re-Orthogonalisierung

für $k := 1$ bis n:
    $x := a^k$; $\rho := \|x\|_2$; für $i := 1$ bis $k-1$: $r_{i,k} := 0$.
    Orthogonalisiere
    falls $\rho = 0$: {Fehlermeldung ‘‘linear abhängige Vektoren’’;
                stop}
    $\rho_{\text{alt}} := \rho$;
    für $i := 1$ bis $k-1$: $\{\sigma := q^{iH} x;\ r_{i,k} := r_{i,k} + \sigma;\ x := x - q^i \sigma\}$;
    $\rho := \|x\|_2$; falls $\rho < 0.1\rho_{\text{alt}}$: zurück zu *Orthogonalisiere*;
    $r_{k,k} := \rho;\ q^k := x/\rho$.
    (Nur äußerst selten wird mehr als einmal zurückgesprungen.)

Kapitel 3

Lineare Gleichungssysteme

Zu gegebenen $A \in \mathbb{R}^{n,n}$ und $b \in \mathbb{R}^n$ ist $x \in \mathbb{R}^n$ gesucht, so daß das **lineare Gleichungssystem der Ordnung** n

$$Ax = b \tag{3.1}$$

erfüllt ist. A ist die **Koeffizienten-Matrix**, b ist die **rechte Seite**, x ist der Satz der **Unbekannten**. Die formale Lösung (bei quadratischem A) ist

$$x = A^{-1}b \tag{3.2}$$

mit $A^{-1} \in \mathbb{R}^{n,n}$ die **Inverse von** A. Bekanntlich ist

$$\det(A) \neq 0 \tag{3.3}$$

die notwendige und hinreichende Bedingung für Existenz und Eindeutigkeit von x und A^{-1}. Selten ist die Berechnung der **Determinante** von A geschlossen möglich, so daß (3.3) keine praktische Bedeutung hat für die Entscheidung der Existenz- und Eindeutigkeits-Frage. Günstiger und meist viel leichter durchführbar ist der Nachweis von

$$Ax = 0 \qquad \text{nur für} \quad x = 0, \tag{3.4}$$

der äquivalent ist zu (3.3). Die Berechnung der Lösung x von (3.1) heißt **Auflösen des Gleichungssystems** und ist mit endlich vielen arithmetischen Grund-Operationen $+, -, \times, /$ möglich, nämlich $O(n^3/3)$ Subtraktionen und Multiplikationen und $O(n^2/2)$ Divisionen. Das gilt auch für komplexe statt reelle Größen: A und $A^{-1} \in \mathbb{C}^{n,n}$, $\det(A) \in \mathbb{C}$.

Die **Cramersche Regel**,

$$x_i = \det(A_{i,b})/\det(A), \qquad i = 1, \ldots, n, \tag{3.5}$$

ist eine formale Angabe der Lösung. Die Matrix $A_{i,b}$ erhält man aus A, indem man dort die i-te Spalte durch b ersetzt. Als numerisches Verfahren ist (3.5) gänzlich ungeeignet (auch bei $n = 2$). Das gilt auch für (3.2), also

Wenn zum Beispiel A und b nicht fehlerfrei vorliegen, sondern mit unbekannten, zufälligen Fehlern der Größenordnung $\|A\|\varepsilon$ und $\|b\|\varepsilon$ zu rechnen ist, dann ist mit einer Unsicherheit $\|x\|\varepsilon\kappa/(1-\varepsilon\kappa)$ in der Lösung x zu rechnen. Wohlgemerkt, es handelt sich hierbei nicht um Rundungsfehler, die beim numerischen Auflösen von (3.1) mit irgendeinem Verfahren passieren, sondern um etwas Prinzipielleres und etwas, was *nichts* mit numerischer Rechnung zu tun hat!

Der Anwender muß die Unsicherheit ε seiner Daten und die Konditionszahl κ seiner Koeffizienten-Matrix feststellen bzw. abschätzen und dann entscheiden, ob $\kappa\varepsilon \ll 1$, d.h. ob es Sinn hat, x zu berechnen. (Wie schon früher festgestellt: der Numeriker ist hier in der Rolle eines guten Rechtsanwaltes, der seinem Klienten auch mal von der Führung eines Gerichtsprozesses abraten sollte, wenn dieser nicht zu gewinnen ist.)

Wenn $\tilde{x}$ irgendeine Näherung für x ist, dann ist $r = b - A\tilde{x}$ der **Residuums-Vektor** oder kurz das **Residuum** („Rest"). Das ist also der Unterschied zwischen linker und rechter Seite in (3.1) falls $\tilde{x} \neq x$. Oft erlebt man, daß beide Seiten bis auf die Rechengenauigkeit übereinstimmen, d.h. $\|r\| = (\|b\| + \|A\tilde{x}\|) \cdot O(\varepsilon_{\text{mach}})$.

> Es ist absolut unzulässig, aus diesem Ausgang der Einsatz-Probe zu folgern, daß auch die Näherung $\tilde{x}$ mit der exakten Lösung x bis auf die Rechengenauigkeit übereinstimmt, d.h. $\|\tilde{x} - x\| = \|x\| \cdot O(\varepsilon_{\text{mach}})$.

Richtig ist $r = b - A\tilde{x} = Ax - A\tilde{x}$, also $x - \tilde{x} = A^{-1}r$

$$\|x - \tilde{x}\| \leq \kappa \cdot \|r\|/\|A\|$$

(Der Faktor $\|A\|$ ist aus Dimensionsgründen notwendig und sinnvoll; man kann sich immer eine Normierung der Koeffizienten-Matrix auf $\|A\| = 1$ durchgeführt denken.)

Wenn zum Beispiel $\kappa = 10^6$ ist, dann kann auch ein 100%ig falsches $\tilde{x}$ beim Einsetzen in $Ax = b$ eine Übereinstimmung auf sechs Dezimalen ergeben!

Ist das Residuum also gar nichts wert und sollte man die Einsetz-Probe strengstens verbieten? Nichts wäre falscher als das, nur der obige, eingerückte Fehlschluß gehört unter Strafe gestellt. Ordnet man nämlich um zu

$$A\tilde{x} = b - r,$$

dann sieht man, daß $\tilde{x}$ die exakte Lösung zu A und $\tilde{b} = b - r$ ist. Wenn das Residuum r also in der Größenordnung der Unsicherheit δb der rechten Seite ist, dann ist $\tilde{x}$ auf jeden Fall akzeptabel, wie schon in Kapitel 1 diskutiert.

Zwei Näherungen, $x^{(1)}$ und $x^{(2)}$, können Fehler gleicher Größenordnung und Residuen sehr ungleicher Größen-Ordnung haben:

$$\|x^{(1)} - x\| \approx \|x^{(2)} - x\| \qquad \text{und} \qquad \|Ax^{(1)} - b\| \ll \|Ax^{(2)} - b\|.$$

von der Zeile i, so daß eine Null in Position $i,1$ entsteht, $i = 2,\ldots,n$. Danach stehen die Koeffizienten des reduzierten Gleichungssystems in Zeilen $2,\ldots,n$. Als Algorithmus lautet dieser erste Eliminations-Schritt

```
für k := 1 bis n:  r_1k := a_1k ;
y_1 := b_1 ;
für i := 2 bis n:
    l_i1 := a_i1 / r_11 ;
    für k := 2 bis n:  a'_ik := a_ik − l_i1 r_1k ;
    b_i := b_i − l_i1 y_1 ;
```

Die Koeffizienten

$$a'_{ik} = a_{ik} - a_{i1}a_{1k}/a_{11}, \qquad i,k \in \{2,\ldots,n\} \tag{3.6}$$

formen die sogenannte **Rest-Matrix**. Die beiseite gestellte erste Gleichung hat die Form

$$\sum_{k=1}^{n} r_{1k}x_k = y_1, \qquad \text{explizit:} \qquad x_1 = \Big(y_1 - \sum_{k=2}^{n} r_{1k}x_k\Big)/r_{11}.$$

Diese erste Zeile mit $r_{11},\ldots,r_{1n}$ und y_1 wird als **Pivotzeile** des ersten Eliminations-Schrittes bezeichnet. Die Umbenennung von a_{1k} in r_{1k} und von b_1 in y_1 dient nur der Logik und ist in einem Computer-Programm überflüssig. Die erzeugten Nullen in Position a_{i1} muß man sich denken, meist speichert man dort die l_{i1} für eine mögliche spätere Wiederverwendung, siehe die in Abschnitt 3.6 besprochene „iterative Nachverbesserung“.

Weil die übrigen Schritte streng analog ablaufen, lautet der Gesamt-Algorithmus der Gauß-Elimination in natürlicher Reihenfolge:

```
für j := 1 bis n:
    für k := j bis n:  r_jk := a_jk ;
    y_j := b_j ;
    für i := j + 1 bis n:
        l_ij := a_ij / r_jj ;
        für k := j + 1 bis n:  a_ik := a_ik − l_ij r_jk ;
        b_i := b_i − l_ij y_j ;
für i := n abwärts bis 1:
    x_i := ( y_i − Σ_{j=i+1}^{n} r_ij x_j ) / r_ii .
```

Es wurde übrigens vorausgesetzt, daß die **Pivot-Elemente** (kurz **Pivots**) r_{jj} alle von 0 verschieden sind. Darauf wird später noch genauer eingegangen.

Das ist genau (3.7), wobei dort immer nach der einen, noch nicht berechneten Größe aufgelöst ist.

Wir haben somit folgendes, sehr wichtige Resultat.

> Die Gauß-Elimination in natürlicher Reihenfolge ist identisch mit der Dreieckszerlegung $A =: L \cdot R$, der Vorwärts-Substitution $Ly = b \Rightarrow y$ und der Rückwärts-Substitution $Rx = y \Rightarrow x$.

Beide Algorithmen unterscheiden sich nur in der Reihenfolge solcher Operationen, die nichts miteinander zu tun haben und parallel ausgeführt werden können. Die Gauß-Elimination formt die $n(n+1)$ Skalarprodukte in (3.8) bzw. (3.9) *simultan* und die Dreiecks-Zerlegung + Vorwärts-Substitution formt sie *nacheinander*.

Dabei werden die gleichen Zwischenresultate gebildet und somit die gleichen Rundungsfehler begangen beim Rechnen mit endlich vielen Stellen.

> Die beiden Algorithmen sind also äquivalent und geben auch unter dem Einfluß der Rundungsfehler identische Resultate. Will man diese Rundungsfehler analysieren (Abschnitt 3.4), so genügt es, nur einen der beiden Prozesse anzusehen.

3.3 Cholesky-Zerlegung bei positiv-definitem A

Besonders günstige Verhältnisse mit Einsparungs-Möglichkeiten gibt es bei einer positiv-definiten Koeffizienten-Matrix

$$A = A^H \text{ ist positiv-definit} \qquad \Longleftrightarrow \qquad x^H A x > 0 \quad \forall x \neq 0.$$

Dieser Spezialfall ist bei der technischen Anwendungen oft gegeben, zum Beispiel kann x der Zustandsvektor eines physikalischen Systems sein und $\frac{1}{2} x^H A x$ die darin gespeicherte Energie, oder es geht um die Minimierung einer Funktion $f(x) = \frac{1}{2} x^T A x - b^T x + c$ im $\mathbb{R}^n$.

Bei Ausgleichsproblemen nach der Methode der kleinsten Quadrate (Abschnitt 3.7) hat die Koeffizienten-Matrix die Form $A^H A$ mit $A \in \mathbb{C}^{m,n}$ vom Rang n, so daß $x^H (A^H A) x = \|Ax\|_2^2 > 0 \Leftrightarrow Ax \neq 0 \Leftrightarrow x \neq 0$.

Die Wahl $x = (1, 0, \ldots, 0)^T$ zeigt, daß $a_{11} > 0$ ist und somit der erste Eliminations-Schritt mit a_{11} als Pivot gewiß durchführbar ist. (3.6) zeigt, daß die Restmatrix selbst-adjungiert bleibt, in der Tat sogar positiv-definit:

$$A =: \begin{pmatrix} a_{11} & s^H \\ s & B \end{pmatrix}, \qquad x =: \begin{pmatrix} \eta \\ y \end{pmatrix}, \qquad Ax = \begin{pmatrix} a_{11}\eta + s^H y \\ s\eta + By \end{pmatrix}$$

`für` $i := 1$ `bis` n: $\qquad y_i := b_i - \sum_{j=1}^{i-1} l_{ij} y_j$

`für` $i := n$ `abwärts bis 1`: $\qquad x_i := \Big(y_i - \sum_{j=i+1}^{n} r_{ij} x_j\Big)/r_{ii}\,.$

Unter dem Einfluß der Rundungsfehler ergeben sich stattdessen die Werte $\tilde{l}_{ik}, \tilde{r}_{ik}, \tilde{y}_i, \tilde{x}_i$ zusammengefaßt in $\tilde{L}, \tilde{R}, \tilde{y}, \tilde{x}$. Betrachten wir dazu die repräsentative Formel

$$l_{ik} = \Big(\ldots((a_{ik} - l_{i1} \cdot r_{1k}) - l_{i2} r_{2k}) - \ldots - l_{i,k-1} r_{k-1,k}\Big)/r_{kk}.$$

Nach dem Rundungsfehler-Axiom gilt für das berechnete $\tilde{l}_{ik}$ dann

$$\begin{aligned}\tilde{l}_{ik} \;&= \Big(\ldots((a_{ik} - \tilde{l}_{i1}\tilde{r}_{1k} \cdot (1+\mu_1))(1+\sigma_1) - \tilde{l}_{i2}\tilde{r}_{2k}(1+\mu_2))(1+\sigma_2)\\ &\quad - \ldots - \tilde{l}_{i,k-1}\tilde{r}_{k-1,k} \cdot (1+\mu_{k-1})\Big)(1+\sigma_{k-1})/\tilde{r}_{kk} \cdot (1+\delta),\end{aligned}$$

wobei die Herkunft der μ_i, σ_i und δ klar ist. Man beachte, daß als Operanden hier die zuvor berechneten und durch Rundungsfehler verfälschten Größen auftreten. Mit $(1+\mu_k)^{-1}$ statt $1+\delta$ und aufgelöst nach a_{ik}:

$$a_{ik} = \sum_{j=1}^{k} \tilde{l}_{ij}\tilde{r}_{jk} \cdot (1+\mu_j) \cdot (1+\sigma_1)^{-1} \cdots (1+\sigma_{j-1})^{-1}.$$

Die μ_j, $\sigma_1, \ldots, \sigma_{j-1}$ hängen noch von i und k ab, so daß tatsächlich

$$(1+\mu_i)/(1+\sigma_1)\cdots(1+\sigma_{j-1}) =: 1+\varepsilon_{ijk}.$$

Nach dem Lemma 1.8 aus Abschnitt 1.3 gilt

$$|\mu_i|, |\sigma_1|, \ldots, |\sigma_{j-1}| \le \varepsilon_{\text{mach}} \qquad \Longrightarrow \qquad |\varepsilon_{ijk}| \le \varepsilon$$

mit

$$\varepsilon := \frac{n \cdot \varepsilon_{\text{mach}}}{1 - n \cdot \varepsilon_{\text{mach}}}.$$

Die drei anderen Formeln lassen sich natürlich genauso behandeln mit $\delta = 0$, wenn die Divison fehlt. Zusammen

$$\begin{aligned} a_{ik} &= \sum_{j=1}^{\min(i,k)} \tilde{l}_{ij}\tilde{r}_{jk} \cdot (1+\varepsilon_{ijk}), \qquad i \text{ und } k = 1, \ldots, n,\\ b_i &= \sum_{j=1}^{i} \tilde{l}_{ij}\tilde{y}_j \cdot (1+\varepsilon'_{ij}), \qquad i = 1, \ldots, n,\\ \tilde{y}_i &= \sum_{j=1}^{n} \tilde{r}_{ij}\tilde{x}_j \cdot (1+\varepsilon''_{ij}), \qquad i = 1, \ldots, n. \end{aligned}$$

gemacht. Von den Eliminations-Faktoren l_{ij} wurde nichts vorausgesetzt, insbesondere nicht $|L| \leq 1$, so daß alle Aussagen für jede noch zu besprechende Pivotstrategie gültig sind.

Man beachte unbedingt, daß die Konditionszahl $\kappa = \|A^{-1}\| \cdot \|A\|$ nirgends aufgetreten ist und daß insbesondere $\kappa\varepsilon_{\text{mach}} \ll 1$ nicht vorausgesetzt worden ist. Alle Aussagen gelten somit für beliebig schlecht konditionierte Gleichungssysteme. Im Gegenteil, ein schlecht konditioniertes A muß nahezu linear abhängige Zeilen haben, so daß bei der Gauß-Elimination meist starke Auslöschung eintritt und die Elemente von $|\tilde{R}|$ dadurch klein werden. Bei der üblichen Pivotstrategie mit $|\tilde{L}| \leq 1$ wird dann auch $|\tilde{L}| \cdot |\tilde{R}|$ klein und obige Schranken besser statt schlechter!

Die Konditionszahl κ kommt erst ins Spiel, wenn man obige Schranke ΔA für $\tilde{A} - A$ umrechnet gemäß Abschnitt 3.1 in Schranken für $\tilde{x} - x$.

3.5 Pivotsuche bei der Gauß-Elimination

Im j-ten Eliminations-Schritt ist das j-j-Element das natürliche Pivot. Es darf nicht null sein, weil damit dividiert wird bei der Berechnung der Eliminationsfaktoren l_{ij}, $i = j + 1, \ldots, n$ und bei der Rückwärts-Substitution.

Ist es jedoch null, dann muß eine andere Gleichung an die Stelle der j-ten treten. Dazu sucht man in der j-ten Spalte für $i = j + 1, \ldots, n$ nach einem geeigneten Pivotelement $a_{ij} \neq 0$. Sind dort ausschließlich Nullen, dann hat die Rest-Matrix eine verschwindende erste Spalte und ihre Determinante ist somit null, also auch die Gesamt-Determinante. Man kann dann abbrechen. Findet man dagegen ein $a_{ij} \neq 0$, dann vertauscht man die i-te mit der j-ten Gleichung, also die beiden kompletten Zeilen einschließlich der rechten Seite.

Diese Vorgehensweise heißt **Spalten-Pivotsuche** (engl. *partial pivoting*). Bei exakter Rechnung ist jedes $a_{ij} \neq 0$ als Pivot gleich gut geeignet. Beim Rechnen mit endlich vielen Stellen wählt man das betragsgrößte. Manchmal sucht man sogar in der gesamten Restmatrix nach dem betragsgrößten Element und bringt es durch Vertauschen von Zeilen und Spalten in die j-j-Position. Das entspricht nicht nur einer Umstellung von Gleichungen sondern auch einer Umnumerierung der Unbekannten. Dies ist aufwendiger, weil man sich diese Vertauschungen merken muß und sie nach der Rückwärts-Substitution wieder rückgängig zu machen hat. Dies nennt man **vollständige Pivotsuche** (engl. *total pivoting*).

Eine möglichst großes Pivot ist auch plausibel nach der Rundungsfehler-Analyse in Abschnitt 3.4. Je größer der Betrag des Pivots, deto kleiner sind die Beträge der Eliminationsfaktoren l_{ij} und desto besser die Chance, daß die Elemente in der nächsten Restmatrix nicht zu stark anwachsen. Auf jeden Fall steht fest, daß ein betragsmäßig sehr kleines Pivotelement ungünstig ist, weil es Elemente in L und R explodieren läßt.

Details zu (1)*:* Wenn $\tilde{x}$ aus der Gauß-Elimination stammt, erfüllt es die Einsetz-Probe normalerweise auf volle Rechengenauigkeit wie diskutiert in Abschnitt 3.4. Schritt 1 in normaler Arithmetik ausgeführt gibt nur Schrott für r auf Grund von Auslöschung. Die **Akkumulation in doppelter Genauigkeit** gibt dagegen ein sehr präzises r. Das ist die Multiplikation von a_{ij} und x_j ohne Rundung und die Summation aller dieser Produkte mit ihrer doppelten Stellenzahl. Erst am Schluß wird gerundet zur normalen Stellenzahl. Mit der richtigen Hardware oder (und) Software geht das so schnell wie die normale Arithmetik.

Details zu (2)*:* Nach der Gauß-Elimination sind L und R schon vorhanden und es genügt die Vorwärts- und Rückwärts-Substitution. Die Theorie von Abschnitt 3.1 und 3.4 sagt, daß Δx einen relativen Fehler von $\kappa\varepsilon_{\text{mach}}$ hat. Je kleiner die Konditionszahl κ ist, desto besser ist die Konvergenz-Geschwindigkeit.

Details zu (3)*:* Der Rundungsfehler in *dieser* Operation begrenzt die Genauigkeit, so daß man bei Konvergenz im allgemeinen ein korrekt gerundetes $A^{-1}b$ hat.

Die Vorteile dieser Ergänzung sind:

- Steigerung der Genauigkeit von $\tilde{x}$ auf volle Stellenzahl,
- billig: jede Iteration nur mit $2n^2$ Subtraktionen und Multiplikationen,
- Konvergenz-Geschwindigkeit gibt Information über Kondition κ.

Übrigens: Während Δx in jeder Iteration um den Faktor $\kappa\varepsilon_{\text{mach}}$ abnimmt, bleibt das Residuum r von Anfang bis Ende in der gleichen Größe oder besser Winzigkeit.

Rang-1-Modifikation

Sei $A = L \cdot R$ bekannt oder leicht zu beschaffen. Wie kann man damit das modifizierte Gleichungssystem

$$(A + uv^H)x = b, \qquad u, v \in \mathbb{C}^n \text{ gegeben}$$

lösen? (uv^H ist eine $n \times n$ Matrix vom Rang 1.) Man merke sich dazu den Trick, $\xi := v^H x \in \mathbb{C}$ als redundante Unbekannte einzuführen, der Rest ergibt sich dann von selbst:

$$Ax = b - u\xi \quad \Longrightarrow \quad x = A^{-1}b - A^{-1}u\xi \quad \Longrightarrow \quad \xi = v^H A^{-1}b - v^H A^{-1}u\xi.$$

`Schritt (1):` `Löse` $Ay = b$ `nach` $y = A^{-1}b$,
`Löse` $Az = u$ `nach` $z = A^{-1}u$,
`Schritt (2):` $\xi := v^H y/(1 + v^H z)$,
`Schritt (3):` $x := y - z\xi$.

Beachte dabei, daß $\det(A + uv^H) = \det(A)\cdot\det(I + zv^H) = \det(A) \cdot (1 + v^H z)$.

b) nach der Dreiecks-Ungleichung stetige Funktionen von A und deshalb numerisch berechenbar.

(Vgl. die Vorlesung *Numerische Mathematik III*.)

Determinanten-Formeln

Aus $A = LR$ folgt $\det(A) = \det(L) \cdot \det(R) = 1 \cdot r_{11} \cdots r_{nn}$. Das gilt auch für alle Zwischenstufen:

$$r_{11} \cdots r_{jj} = \det\left(A_{1\ldots j}^{1\ldots j}\right), \qquad j = 1, \ldots, n,$$

wobei die obere Indexreihe die ausgewählten Zeilen und die untere Indexreihe die ausgewählten Spalten von A angibt:

$$A_{\mu\nu\ldots}^{\kappa\lambda\ldots} = \{a_{ij} : i = \kappa, \lambda, \ldots \text{ und } j = \mu, \nu, \ldots\}.$$

Auswählen der Spalte k anstelle von Spalte j gibt

$$r_{11} \ldots r_{j-1,j-1} \cdot r_{jk} = \det(A_{1\ldots j-1,k}^{1\ldots j-1,j}), \qquad k \geq j$$

und Auswählen der Zeile k anstelle von Zeile j gibt

$$l_{kj} r_{11} \ldots r_{jj} = \det(A_{1\ldots j-1,j}^{1\ldots j-1,k}), \qquad k > j$$

oder aufgelöst:

$$\begin{aligned} l_{kj} &= \det(A_{1\ldots j-1,j}^{1\ldots j-1,k}) / \det(A_{1\ldots j}^{1\ldots j}), \quad k > j, \\ r_{jk} &= \det(A_{1\ldots j-1,k}^{1\ldots j-1,j}) / \det(A_{1\ldots j-1}^{1\ldots j-1}), \quad k \geq j. \end{aligned}$$

Für die Cholesky-Zerlegung gilt dann

$$\hat{r}_{jk} = \det(A_{1\ldots j-1,k}^{1\ldots j-1,j}) / (\det(A_{1\ldots j-1}^{1\ldots j-1}) \cdot \det(A_{1\ldots j}^{1\ldots j}))^{1/2}.$$

Alle Determinaten-Formeln für Dreieckszerlegung einer Matrix sind genauso wie die Cramersche Regel ganz ohne Bedeutung für Rechenverfahren.

Bei der Berechnung von $d := \det(A)$ startet man also mit $d := 1$ und multipliziert sodann mit allen Pivots: $d := d \cdot r_{jj}$, $j = 1, \ldots, n$. Dabei kann es sehr leicht zu Exponenten-Über/Unterlauf kommen, wenn man nicht selbst skaliert, z.B. mit 2^{64}. Bei jeder Vertauschung einer Zeile oder eine Spalte muß man das Vorzeichen von d umdrehen.

Man nennt dies das **lineare Ausgleichsproblem** oder die Ausgleichung der Widersprüche nach der **Methode der kleinsten Quadrate** (eigentlich Quadratsumme). (3.12) ist also äquivalent zur Minimierung von $r(x)^T r(x) = x^T A^T A x - 2x^T A^T b + b^T b$, dessen Ableitung nach x der Vektor $2A^T Ax - 2A^T b$ ist. Also muß $\hat{x}$ die **Normalgleichungen**

$$A^T A\hat{x} = A^T b \qquad \Longleftrightarrow \qquad A^T \hat{r} = 0 \tag{3.13}$$

erfüllen. Diese Gleichungen sind nicht nur notwendig, sondern auch hinreichend, denn es ist $r(x) = \hat{r} + A(\hat{x} - x)$, also

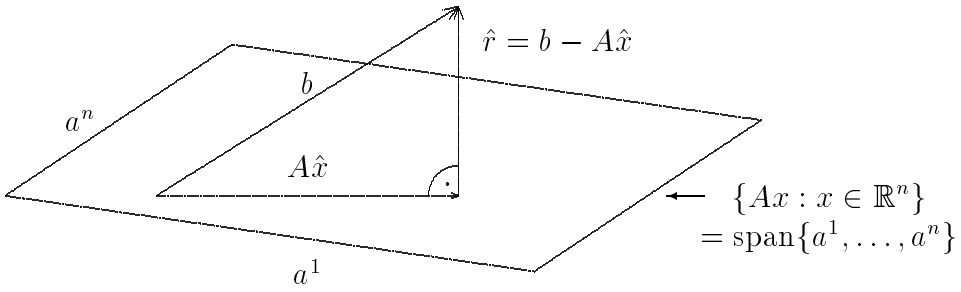
$$r(x)^T r(x) = \hat{r}^T \hat{r} + 0 + (x - \hat{x})^T A^T A(x - \hat{x}) \geq \hat{r}^T \hat{r}$$

und Gleichheit gilt nur für $\|A(x - \hat{x})\|_2 = 0 \Leftrightarrow A(x - \hat{x}) = 0 \Leftrightarrow r(x) = \hat{r}$. Somit gilt der folgende

Satz 3.1 *Existenz und Eindeutigkeit*

> Das Ausgleichsproblem (3.12) hat immer ein eindeutiges kleinstes Residuum $\hat{r}$. Der zugehörige Minimierer $\hat{x}$ ist eindeutig nur wenn die Spalten von A linear unabhängig sind: $\mathrm{Rang}(A) = n$. Notwendig und hinreichend für einen Minimierer sind die Normalgleichungen (3.13).

Nach (3.13) ist das immer eindeutige Residuum $\hat{r}$ senkrecht zu allen Spalten von A bzw. dem davon aufgespannten linearen Untervektorraum.



Zu den Normalgleichungen $A^T(b - A\hat{x}) = A^T\hat{r} = 0$: Das kürzeste Residuum steht senkrecht zu den Spalten von A.

Im folgenden wird angenommen, daß die Spalten von A linear unabhängig sind, also

$$Ax = 0 \qquad \Longrightarrow \qquad x = 0.$$

Damit ist ab jetzt auch der Minimierer $\hat{x}$ eindeutig und die Koeffzienten-Matrix $A^T A$ in den Normalgleichungen positiv-definit, also auch nicht-singulär.

ungenau, weil die Matrix A^TA die Konditionszahl κ^2 hat, so daß Rundungsfehler beim Formen von A^TA und beim Auflösen von (3.13) mit κ^2 verstärkt werden. Seit man den obigen Term mit κ^2 in der Konditions-Abschätzung kennt, urteilt man etwas milder. Im Fall (3.16) mit sehr unterschiedlichen Gewichten $g_1, \ldots, g_n$ darf man diese Methode jedoch nicht verwenden.

Meist wird empfohlen, das System der Koeffizienten (A, b) mit ebenen Rotationen oder Reflexionen von links auf obere Dreiecksform zu bringen, wie ausführlich dargestellt als Beispiel 2 und 4 in Abschnitt 2.4:

$$A \mapsto Q^TA =: R, \qquad b \mapsto Q^Tb =: c, \qquad r(x) \mapsto Q^Tr(x).$$

Wegen $\|r\|_2 = \|Q^Tr\|_2$, kann man dann die Minimierung (3.12) im reduzierten System durchgeführt werden, wo das Resultat klar ist:

$$(Q^Tr)_i = (Q^Tb - Q^TAx)_i = \begin{cases} c_i - \sum_{j=i}^{n} r_{ij}x_j & \text{für } i = 1, \ldots, n, \\ c_i & \text{für } i = n+1, \ldots, m. \end{cases}$$

Die ersten n Komponenten lassen sich bis auf 0 verkleinern, die letzten $m - n$ Komponenten lassen sich überhaupt nicht beeinflussen. Also erhält man $\hat{x}$ aus Rückwärts-Substitution wie bei linearen Gleichungen.

Das gleiche Resultat findet man, wenn man die QR-Zerlegung von A in die Normalgleichungen (3.13) einsetzt:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}} R^TQ^TQR\hat{x} & = & R^TQ^Tb & \iff & R^TR\hat{x} & = & R^Tc & \iff \\ \bar{R}^T\bar{R}\hat{x} & = & \bar{R}^T\bar{c} & \iff & \bar{R}\hat{x} & = & \bar{c} & \end{array}$$

($\bar{R}$ ist R ohne die trivialen letzten $m - n$ Zeilen, $\bar{c}$ sind die ersten n Komponenten von c).

Dieses Vorgehen ist also in strenger Analogie zur Gauß-Elimination. Geändert ist nur, daß man mehr Zeilen als Spalten hat und daß man die elementaren Zeilen-Operationen einschließlich Pivotsuche ersetzt durch ebene Rotationen oder Reflexionen von links.

Es hat verschiedentlich den Vorschlag gegeben, auch bei linearen Gleichungen so vorzugehen, also ebene Rotationen von links an die Stelle der elementaren Zeilen-Operationen einschließlich Vertauschungen zu setzen. Man glaubte dadurch, den Schwierigkeiten mit der Skalierung der Gleichungen aus dem Weg gegangen zu sein, aber das ist natürlich nicht der Fall.

3.8 Iterative Methoden für lineare Gleichungssysteme mit sehr großen, dünn besiedelten Koeffizienten-Matrizen

Eine Matrix heißt sehr groß, wenn die Anzahl ihrer Elemente größer ist als die zur Verfügung stehenden Speicherplätze. Eine Matrix heißt dünn besiedelt (engl.

Hier ist i, j der Index für eine Gleichung und gleichzeitig für eine Variable. Die Komponenten-Relaxation besagt in diesem Fall, daß diese Variable als arithmetisches Mittel der vier Nachbarn zu wählen ist. Die zugehörige Matrix wird weder gebildet noch irgendwie gespeichert. Das demonstriert, wie billig ein solcher Iterationsschritt sein kann.

Nach einem Relaxations-Schritt ist also die i-te Gleichung exakt erfüllt, doch schon im nächsten Schritt wird das im allgemeinen wieder aufgehoben und eine andere Gleichung dafür erfüllt. Die Hoffnung ist, daß bei geeigneter Wahl von Indexpaaren (i, k) der Näherungsvektor x gegen $\bar{x} = A^{-1}b$ konvergiert.

Zu dieser Wahl springt ins Auge, daß jede Gleichung i und jede Unbekannte k im Laufe der Rechnung mindestens einmal an der Reihe sein muß. Am naheliegendsten sind deshalb Iterationszyklen zu je n Schritten, wobei jede Gleichung und jede Unbekannte in starrer Reihenfolge genau einmal „dran kommt“. Etwas anderes ist bei einer automatischen Rechnung kaum denkbar oder zu aufwendig. Durch Umstellen der Gleichungen und Umnumerieren der Unbekannten kommt man dann zur natürlichen Reihenfolge

$$i = k \in \{1, \ldots, n\} \quad \text{zyklisch.}$$

Gauß-Seidel- oder Einzelschritt-Verfahren (GS)

`für` $m := 1, 2, \ldots$:
    `für` $i := 1$ `bis` n: $x_i := x_i + \Big(b_i - \sum_j a_{ij} x_j\Big)/a_{ii}$;
    `Teste auf Abbruch.`

m zählt die Zyklen. Selbstverständlich darf kein Diagonalelement null sein.

Eine Erweiterung ist das Verfahren der

Überrelaxation (ÜR)

`für` $m := 1, 2, \ldots$
    `Wähle Überrelaxationsfaktor` $\omega = \omega(m) \neq 0$;
    `für` $i := 1$ `bis` n: $x_i := x_i + \omega \cdot \Big(b_i - \sum_j a_{ij} x_j\Big)/a_{ii}$;

In den Anwendungen ist meist $\omega > 1$, daher der Name. Besonders geeignet für Parallelrechner ist das

Multiplikation mit $\omega^{-1}D$ von links und Umstellung gibt

$$(\omega^{-1}D + E)x^{(m+1)} + (1 - \omega^{-1})Dx^{(m)} + Fx^{(m)} = b.$$

■

Korollar 3.3

Falls $\{x^{(m)}\}$ konvergiert, dann zur exakten Lösung $\bar{x} = A^{-1}b$.

Beweis: Für $x^{(m)} \to \hat{x}$ folgt aus (3.18), (3.19) $A\hat{x} = b$. ■

Korollar 3.4

Für den Fehler $f^{(m)} := x^{(m)} - A^{-1}b$ gilt

$$f^{(m+1)} = Kf^{(m)} \qquad \text{und} \qquad f^{(m)} = K^m f^{(0)},$$

wobei

$$\begin{aligned} &\text{für (J):} && K = -D^{-1}(E + F), \\ &\text{für (GS):} && K = -(D + E)^{-1}F, \\ &\text{für (ÜR):} && K = (D + \omega E)^{-1}[(1 - \omega)D - \omega F]. \end{aligned}$$

Beweis: Aus (3.19) folgt $A_1 f^{(m+1)} + A_2 f^{(m)} = 0$, also $K = -A_1^{-1}A_2$. ■

Satz 3.5

Die Iteration konvergiert genau dann für alle Startvektoren $x^{(0)}$, wenn die Eigenwerte der Iterations-Matrix K alle betragsmäßig kleiner als 1 sind.

Beweis: In der Vorlesung *Lineare Algebra* wird mit Hilfe der Jordanschen Normalform einer Matrix gezeigt, daß $\lim_{m\to\infty} K^m = 0$ genau dann gilt, wenn alle Eigenwerte von K betragsmäßig kleiner als 1 sind. ■

Aus dem Beweis folgt, daß die Verfahren nur linear konvergieren (vgl. Kapitel 6), sofern sie überhaupt konvergieren. Setzt man den Fehler $f^{(m)}$ als Linearkombination der Eigen- (und Haupt-) Vektoren von K an, so sieht man, daß die zu betragskleineren Eigenwerten gehörenden Beiträge schnell gedämpft und herausgefiltert werden, während die Beiträge von den Eigenwerten nahe an 1 so gut wie überhaupt nicht abnehmen. In der Regel sind dies die langwelligen Anteile.

In der jüngsten Vergangenheit hat man andere Iterationen entwickelt, die gerade den langwelligen Anteil im Fehler $f^{(m)}$ stark abmindern. Zumindest ist das gelungen für die großen dünn besiedelten Gleichungssysteme, welche beim Diskretisieren partieller Differentialgleichungen entstehen. Erst durch abwechselndes Anwenden der beiden Typen von Iterationsschritten kommt man zu einer befriedigenden Iterationsgeschwindigkeit.

Kapitel 4

Interpolation mit Polynomen

Die Polynome vom Grad (höchstens) $n-1$ in *einer* Variablen (reell oder komplex) sind ein n-dimensionaler Vektorraum, aufgespannt zum Beispiel von der **Taylor-Basis**

$$\mathbb{P}_{n-1} := \operatorname{span}\{1, x, \ldots, x^{n-1}\}, \quad n \in \mathbb{N}.$$

An die Stelle von **Grad** $n-1$ tritt heutzutage oft die Bezeichnung **Ordnung** n, weil $\mathbb{P}_{n-1}$ als der Nullraum des Operators D^n angesehen wird:

$$p \in \mathbb{P}_{n-1} \qquad \Longleftrightarrow \qquad D^n p = 0, \quad D \equiv \frac{d}{dx}.$$

Rechenmaschinen fällt der Umgang mit Polynomen leicht, weil deren **Auswertung** (Berechnen des Funktionswertes) in ihr Repertoire fällt, vgl. Kapitel 1. Aus diesem Grunde wurden sie vom Beginn der numerischen Mathematik an für alle Arten der Approximation anderer Funktionen f eingesetzt. *Eine* Art der Approximation ist die **Interpolation** („Zwischenwert-Berechnung"), d.h. man macht die Näherung zwischen Polynom p und Funktion f exakt an n diskreten Stellen, den sogenannten **Stützstellen**:

$$p - f = 0 \quad \text{auf} \quad \{x_1, \ldots, x_n\}.$$

Wenn man sich auf diese einfache Übereinstimmung der Funktionswerte beschränkt, spricht man von **Lagrange-Interpolation**. Fordert man zusätzlich auch die Übereinstimmung der Ableitungen an einigen oder allen Stützstellen,

$$D(p-f) = 0, \qquad D^2(p-f) = 0, \qquad \ldots$$

dann heißt dies **Hermite-Interpolation**. Eine mehr technische Bezeichnung lautet Interpolation mit **doppelten**, **dreifachen**, ... Stützstellen.

In der heutigen Anwendung der Polynom-Interpolation als vielfältiges Hilfsmittel der Approximation und der Gewinnung von Näherungsverfahren für die numerische Integration sind mehrfache Stützstellen so wichtig, daß man ihre Behandlung auch in einem einführenden Text nicht mehr weglassen darf. Das macht die Diskussion nicht schwerer, sondern eher einfacher!

Satz 4.2

Die Interpolations-Aufgabe hat immer genau eine Lösung.

Es sei noch betont, daß dieser Sachverhalt für *jede* Teilmenge der Stützstellen genauso gilt, wobei eine m-fache Stützstelle auch nur *teilweise* selektiert werden darf. Weil deshalb auch jede Teilaufgabe des Interpolations-Problems immer genau eine Lösung besitzt, kann man nach Konstruktionswegen suchen, die die Gesamtlösung aus einer Folge von immer umfassenderen Teillösungen stufenweise aufbauen, vgl. den nächsten Abschnitt.

Mehr für theoretische Zwecke wichtig ist, daß man den Interpolanten auch aus den Lösungen zu den n Einheits-Stützwerten aufbauen kann:

Satz 4.3

$$p(x) = \sum_{k=1}^{n} y_k L_k(x), \tag{4.3}$$

wobei $L_k \in \mathbb{P}_{n-1}$ die Einheits-Stützwerte $\{\delta_{i,k}\}$ interpoliert $(k = 1, \ldots, n)$.

Die $L_1, \ldots, L_n$ bilden wie $1, x, \ldots, x^{n-1}$ eine Basis des $\mathbb{P}_{n-1}$, denn auf Grund ihrer Spezifikation sind sie offensichtlich linear unabhängig.

Für lauter einfache Stützstellen kann man die L_k leicht explizit angeben:

Korollar 4.4 *Lagrange-Polynome*

Für einfache Stützstellen $(i \neq j \Rightarrow x_i \neq x_j)$ ist

$$L_k(x) = \frac{x - x_1}{x_k - x_1} \cdots \frac{x - x_{k-1}}{x_k - x_{k-1}} \cdot \frac{x - x_{k+1}}{x_k - x_{k+1}} \cdots \frac{x - x_n}{x_k - x_n} \tag{4.4}$$

Mit (4.1): $L_k(x) = \omega(x)/((x - x_k)\omega'(x_k))$ für $x \neq x_k$, $k = 1, \ldots, n$.

Die Kombination (4.3), (4.4) heißt **Lagrangesche Interpolationsformel**.

Der andere Extremfall ist eine n-fache Stützstelle: $x_1 = \ldots = x_n = a$. Hier ist offensichtlich

$$L_k(x) = (x - a)^{k-1}/(k-1)! \qquad \text{für} \quad k = 1, \ldots, n \tag{4.5}$$

die Lösung der Einheitsaufgabe $y_i = \delta_{i,k}$.

Damit sind die „Blätter“ des Polynombaums bekannt und die Rekursion kann in Richtung „Wurzel“ bzw. „Stamm“ losgehen. Es gilt

$$\begin{aligned} p_{i,k}(x) &= p_{i,k-1}(x) \cdot \frac{x_{i+k} - x}{x_{i+k} - x_i} + p_{i+1,k-1}(x) \cdot \frac{x - x_i}{x_{i+k} - x_i} \\ &\quad \text{für } i = 1 \text{ bis } n-k \text{ und } x_{i+k} - x_i \neq 0. \end{aligned} \tag{4.6}$$

Beweis: f auf beiden Seiten abziehen, gibt dieselbe Rekursion für die Restglieder $p_{i,k}(x) - f(x)$, denn die beiden Gewichte summieren sich zu 1. Das zeigt dann, daß die rechte Seite $p_{i,k}(x) - f(x)$ die erforderlichen Nullstellen $x_i, \ldots, x_{i+k}$ hat. Außerdem ist klar, daß (4.6) ein Polynom von Grad k ist. Es erfüllt somit alle Bedingungen für das eindeutige $p_{i,k}$. Man beachte, daß diese Argumentation auch für mehrfache Stützstellen korrekt ist. ■

Bei lauter einfachen Stützstellen ist dieses dreieckige **Schema von Aitken und Neville** also

```
Start: für i := 1 bis n: p_{i,0} := y_i;
       für k := 1 bis n − 1:
           für i := 1 bis n − k: (4.6).
```

Bei mehrfachen Stützstellen ist der obere Rand in diesem dreieckigen Schema etwas ausgefranst.

4.3 Dividierte Differenzen und Newtonsche Interpolationsformel

Das Aitken-Neville-Schema ist eine Rekursion für *Polynome*, wenn es auch meist für ein festes Argument x als Rechenschema für die reellen Funktions*werte* benutzt wird. Man könnte in (4.6) für jedes $p_{i,k}$ einen Taylor-Ansatz machen und mittels Koeffizienten-Vergleich die Koeffizienten berechnen.

Wir machen das nur mal für die jeweils höchsten Koeffizienten, also die $a_{i,k}$ in

$$p_{i,k}(x) = a_{i,k} \cdot x^k + b_{i,k} \cdot x^{k-1} + \ldots$$

(4.6) gibt

$$\begin{aligned} a_{i,k} &= \frac{a_{i+1,k-1} - a_{i,k-1}}{x_{i+k} - x_i}, \\ &\quad \text{für } i = 1 \text{ bis } n-k \text{ und } x_{i+k} - x_i \neq 0. \end{aligned} \tag{4.7}$$

Damit steht dem dreieckigen Aitken-Neville-Schema für die *Polynome* ein dreieckiges Schema *reeller Zahlen* gegenüber, nämlich das der jeweils höchsten Koeffizienten. Das folgende Lemma zeigt, daß diese unvollständig anmutende Information in Wahrheit komplett ist, daß man also mit dem Schema der höchsten Koeffizienten gleichzeitig das Schema der Polynome erhält!

$x_i, \ldots, x_{i+k}$ und wird mit $[x_i, \ldots, x_{i+k}]f$ bezeichnet. Es gilt also für $1 \leq i \leq i+k \leq n$

$$\begin{aligned} p_{i,k}(x) &= [x_i, \ldots, x_{i+k}]f \cdot x^k + \ldots \\ &= [x_i, \ldots, x_{i+k}]f \cdot (x - x_i) \cdots (x - x_{i+k-1}) + \ldots \\ &\quad (+\ldots \text{ deutet die Terme in } \mathbb{P}_{k-1} \text{ an}). \end{aligned}$$

Satz 4.7 *Rekursion für die dividierten Differenzen*

Für $x_i = \ldots = x_{i+k}$ gilt $[x_i, \ldots, x_{i+k}]f = D^k f(x_i)/k!$.

Ansonsten gilt die Rekursionsformel (4.7)

$$\begin{aligned} [x_i, \ldots, x_{i+k}]f &= \frac{[x_{i+1}, \ldots, x_{i+k}]f - [x_i, \ldots, x_{i+k-1}]f}{x_{i+k} - x_i} \\ &\quad \text{für } i = 1, \ldots, n-k, \text{ wo } x_{i+k} - x_i \neq 0. \end{aligned}$$

Insbesondere

nullte Ordnung:

$$[x_i]f = f(x_i)$$

erste Ordnung:

$$[x_i, x_{i+1}]f = \begin{cases} f'(x_i) & \text{bei } x_i = x_{i+1} \\ (f(x_{i+1}) - f(x_i))/(x_{i+1} - x_i) & \text{sonst} \end{cases}$$

zweite Ordnung:

$$[x_i, x_{i+1}, x_{i+2}]f = \begin{cases} f''(x_i)/2 & \text{bei } x_i = x_{i+2} \\ ([x_{i+1}, x_{i+2}]f - [x_i, x_{i+1}]f)/(x_{i+2} - x_i) & \text{sonst} \end{cases}$$

usw.

Die dividierten Differenzen der Ordnung k sind das diskrete Analogon zur skalierten k-ten Ableitung $D^k f/k!$, in die sie im konfluenten Grenzfall übergehen.

Diese Formeln erklären auch die Bezeichnung als „dividierte Differenzen“.

Mit der neuen Bezeichnung lauten (4.10), (4.11)

Satz 4.8 *Newtonsche Interpolationsformel*

$$\begin{aligned} p(x) &= [x_1]f + [x_1, x_2]f \cdot (x - x_1) + \cdots \\ &\quad + [x_1, \ldots, x_n]f \cdot (x - x_1) \cdots (x - x_{n-1}), \\ p(x) &= [x_n]f + [x_{n-1}, x_n]f \cdot (x - x_n) + \cdots \\ &\quad + [x_1, \ldots, x_n]f \cdot (x - x_2) \cdots (x - x_n). \end{aligned}$$

Wichtige Eigenschaften der dividierten Differenzen $[x_i, \ldots, x_{i+k}]f$ (o.B.d.A. $i = 1$, $i + k = n$):

(e) Dann ist R der Polynom-Interpolant von $f \cdot g$. Tip: $R - f \cdot g$ ist Vielfaches von ω, weil nach (c) und (d) $R - P \cdot Q$, $P - f$, $Q - g$ Vielfache von ω sind.

(f) Dann ist $[x_1, \ldots, x_n](fg)$ der Koeffizient von x^{n-1} in $R(x)$.

(g) Es gilt für dividierte Differenzen eine Produkt-Regel analog zur Leibnizschen Produktregel für skalierte Ableitungen:

$$[x_1, \ldots, x_n](fg) = \sum_{k=1}^{n} [x_1, \ldots, x_k]f \cdot [x_k, \ldots, x_n]g.$$

Tip: Kombiniere (f), (b), (d). Man notiere diese Aussage speziell für $n = 1$ und für $n = 2$ mit (a) $x_1 = x_2$, (b) $x_1 \neq x_2$. Man kann die Produkt-Regel auch durch Schluß von $n - 1$ auf n beweisen.

Übungsaufgabe 4.3: Die **Hermite-Genocchi-Formel** für dividierte Differenzen:

Man zeige für $f \in C^n(\mathbb{R})$ und

$$\langle x_0 \ , \ldots , \ x_n \rangle f := \int_0^1 d\sigma_1 \int_0^{1-\sigma_1} d\sigma_2 \cdots \int_0^{1-\sigma_1-\cdots\sigma_{n-1}} d\sigma_n \, (D^n f)\Big(x_0 + \sum_{i=1}^{n} \sigma_i (x_i - x_0)\Big)$$

(a) Symmetrie in $x_1, \ldots, x_n$.

(b) Symmetrie in x_0, x_n.

(c) O.B.d.A.: $x_0 \leq \cdots \leq x_n$.

(d) Was ergibt sich dann für $x_0 = x_n$?

(e) $x_0 < x_n$: Was ergibt sich für das innerste Integral?

(f) $x_0 < x_n \Rightarrow \langle x_0, \ldots, x_n \rangle f = (\langle x_1, \ldots, x_n \rangle f - \langle x_0, \ldots, x_{n-1} \rangle f)/(x_n - x_0)$.

(g) $\langle x_0, \ldots, x_n \rangle f = [x_0, \ldots, x_n]f$, dividierte Differenz wie schon definiert.

(h) $\langle x_0, \ldots, x_n \rangle f = \int_0^1 d\tau_1 \int_0^{\tau_1} d\tau_2 \cdots \int_0^{\tau_{n-1}} d\tau_n \, (D^n f)\Big(x_0 + \sum_{i=1}^{n} \tau_i (x_i - x_{i-1})\Big)$.

Tips:

Zu (a): Der Integrationsbereich S ist symmetrisch in allen σ_i:
$S := \{\sigma \geq 0 : \sigma_1 + \cdots \sigma_n \leq 1\}$ (Einheits-Simplex).

Zu (b): Statt σ_n neue Integrations-Variable $\sigma_0 := (1 - \sigma_1 - \cdots - \sigma_{n-1}) - \sigma_n$.

Auf $[x_1, \ldots, x_n, \bar{x}]f$ wendet man Satz 4.9 an mit $i = 1$, $k = n$ und erhält nach Umbenennung $\bar{x} \mapsto x$ den

Satz 4.10 *Restgliedformel*

$$f \in C^n[a,b] \qquad \Longrightarrow \qquad \exists \xi \in [a,b] \ni$$
$$f(x) - p(x) = (x - x_1) \cdots (x - x_n) D^n f(\xi)/n! \,,$$

wobei $a := \min(x_1, \ldots, x_n, x)$ und $b := \max(x_1, \ldots, x_n, x)$, ξ hängt von x ab.

Im konfluenten Grenzfall $x_1 = \ldots = x_n$ erhält man das Restglied der Taylorformel.

4.5 Konditionszahlen für die Interpolation, Eignung von Polynomen für die Interpolation

Als Problem im Sinn vom Abschnitt 1.4 hat die Interpolations-Aufgabe

Eingabe-Daten:	Stützstellen $x_1, \ldots, x_n$,	Stützwerte $y_1 \ldots, y_n$,
	Zwischenstelle x,	
Resultat:	$y := p(x)$.	

Als Konditionszahlen im Sinne von Abschnitt 1.4 treten somit auf:

$\partial y / \partial x = p'(x)$.	Klar.
$\partial y / \partial y_k = L_k(x), \qquad k = 1, \ldots, n.$	Nach (4.3).
$\partial y / \partial x_k = -p'(x_k) \cdot L_k(x), \qquad k = 1, \ldots, n.$	

Beweis: Der Interpolant p ändert sich nicht, wenn man den k-ten Stützpunkt x_k, y_k *auf* p beliebig weit verschiebt, denn dann löst er auch die neue Interpolations-Aufgabe. Für Verschiebung von x_k, y_k nach $x_k + \varepsilon$, $p(x_k + \varepsilon)$ und $\varepsilon \to 0$

$$\frac{dp}{d\varepsilon} = \frac{\partial p}{\partial x_k} + \frac{\partial p}{\partial y_k} \cdot p'(x_k) = 0.$$

■

Am wichtigsten ist der Einfluß von Fehlern δy_k in den Stützwerten auf den interpolierten Wert y, also die Konditionszahlen $\partial y / \partial y_k = L_k(x)$, $k = 1, \ldots, n$. Für lauter einfache Stützstellen gibt (4.4) dafür eine explizite Formel. Zum Beispiel ist für 101 äquidistante Stützpunkte $L_{51} \doteq 10^{26}$ zwischen x_1 und x_2 bzw. x_{100} und x_{101}. Ein Fehler des mittleren Stützwerts wird durch die Polynom-Interpolation in den

$\left\{\sum y_k L_k(x) : \text{alle } |y_k| \leq 1\right\}$ für 13 Stützstellen x_k,

(a) **äquidistant**: $x_k = k$,

(b) **Chebyshev-Verteilung**: $x_k = \cos((13 - k)\pi/12)$.

Beweis: (4.15) in (4.14) gibt

$$\sum_{k=0}^{n-1}\Big(\frac{1}{n}\sum_{l=0}^{n-1} w_l\omega^{-kl}\Big)\omega^{jk} = \frac{1}{n}\sum_{l=0}^{n-1} w_l \sum_{k=0}^{n-1}\omega^{(j-l)k}$$

mit $j-l \in \{1-n,\ldots,-1,0,1,\ldots,n-1\}$ und

$$\xi := \omega^{j-l}\begin{cases} =1 & \text{falls } j-l=0,\\ \neq 1 & \text{sonst,}\end{cases} \qquad \xi^n = \omega^{n(j-l)} = 1 \text{ immer.}$$

Die (innere) Summe über k ist also

$$1+\xi+\ldots+\xi^{n-1} = \begin{cases} n & \text{falls } \xi=1 \Leftrightarrow j=l,\\ (\xi^n-1)/(\xi-1)=0 & \text{falls } \xi\neq 1 \Leftrightarrow j\neq l.\end{cases}$$

Von der (äußeren) Summe über l bleibt somit nur der Term $l=j$ übrig. ■

Insgesamt besteht somit eine weitgehend symmetrische Transformationsregel, bekannt als **diskrete Fourier-Transformation** (**DFT**)

$$c \mapsto w: \quad w_j = \sum_{k=0}^{n-1} c_k e^{+2\pi ijk/n}, \qquad j=0,\ldots,n-1, \tag{4.16}$$

$$w \mapsto c: \quad c_k = \frac{1}{n}\sum_{j=0}^{n-1} w_j e^{-2\pi ijk/n}, \qquad k=0,\ldots,n-1. \tag{4.17}$$

Um die Symmetrie besser zu betonen, kann jede der beiden Formeln mit dem Skalierungsfaktor $1/\sqrt{n}$ geschrieben werden. Dann würde sich (4.16) und (4.17) in Matrix-Notation als

$$\begin{aligned} w &= Ac\\ c &= A^H w\end{aligned}$$

schreiben mit einer unitären Matrix A.

Auch wenn man die w_j und c_k nur für den Indexbereich $\{0,\ldots,n-1\}$ *berechnen* muß, so *definieren* (4.16), (4.17) diese Größen für alle j und $k \in \mathbb{Z}$ und zwar **periodisch**:

$$w_{j+n} = w_j \quad \forall j\in\mathbb{Z} \qquad \text{und} \qquad c_{k+n} = c_k \quad \forall k\in\mathbb{Z}. \tag{4.18}$$

Mit *denselben* Koeffizienten $c_0,\ldots,c_{n-1}$ kann man einen *anderen* trigonometrischen Interpolanten definieren, indem man *andere Basisfunktionen* verwendet, nämlich anstelle von (4.13):

Für ungerades $n =: 2m+1$**:**

$$1, e^{it},\ldots,e^{imt},e^{-imt},\ldots,e^{-it},$$

(also $e^{i(k-n)t}$ anstelle von e^{ikt} für $k=m+1,\ldots,2m$),

$$\hat{T}(t) = \frac{a_0}{2} + \sum_{k=1}^{m}(a_k\cos kt + b_k \sin kt).$$

befreien, indem man $m+1$ beliebige reelle Stützwerte $w_0, \ldots, w_m$ symmetrisch ergänzt mittels $w_{m+k} = w_{m-k}$, $k = 0, \ldots, m-1$. Ein symmetrisches $w \in \mathbb{R}^{2m}$ in (4.19) führt zu

$$a_k = \frac{2}{m}{\sum_{j=0}^{m}}'' w_j \cos \pi jk/m, \qquad b_k = 0,\ k = 0 \ldots, m,$$

$$\tilde{T}(t) := \hat{T}(t/2) = {\sum_{k=0}^{m}}'' a_k \cos kt/2.$$

In $\sum''$ haben nullter und m-ter Term jeweils nur das halbe Gewicht. $t/2$ statt t bewirkt, daß das Intervall $[0, \pi]$ für die äquidistanten Stützwerte $w_0, \ldots, w_m$ aufgespreizt wird zum Standard-Intervall $[0, 2\pi]$.

Bei der trigonometrischen Interpolation auf $[0, 2\pi]$ hat man also drei Möglichkeiten für die Wahl der Basis:

T : Die ersten n Glieder in $1, e^{it}, e^{2it}, \ldots$

$\hat{T}$: Die ersten n Glieder in $1, \cos t, \sin t, \cos 2t, \sin 2t, \ldots$

$\tilde{T}$: Die ersten $m+1$ Glieder in $1, \cos t/2, \cos t, \cos 3t/2, \ldots$

Beispiel: Zu $w_j = n/2 - j$ für $j = 0, \ldots, n-1$ gehört ein $\hat{T}(t)$ mit $a_k = 1$ für $k = 0, 1, 2, \ldots$ und $b_k = \operatorname{cotan}(\pi k/n)$ für $k = 1, 2, 3, \ldots$ Zu $w_j = n/2 - j$ für $j = 0, \ldots, n$ gehört ein $\tilde{T}(t)$ mit $a_k = 0$ für $k = 0, 2, 4, \ldots$ und $a_k = 1/(n \cdot \sin(\pi k/2n)^2)$ für $k = 1, 3, 5, \ldots$ Für hinreichend großes n fallen die Koeffizienten von $\hat{T}(t)$ wie $1/k$ ab und die von $\tilde{T}(t)$ wie $1/k^2$. Der schnellere Abfall ist bei der digitalen Signalverarbeitung sehr wichtig.

4.7 Fourier-Transformation und Abminderungsfaktoren

Bevor wir uns mit den Berechnungsformeln (4.16), (4.17) intensiv beschäftigen und die Ausführung dieser Transformation dramatisch beschleunigen, wird noch ein Abschnitt über die Behandlung der analytischen Fourier-Transformation eingefügt, der den Formeln (4.16), (4.17) weit mehr Bedeutung verleiht, als es die trigonometrische Interpolation allein tun könnte.

Die **Fourier-Koeffizienten** einer Funktion $f\colon [0,1] \to \mathbb{C}$ sind

$$C_k(f) := \int_0^1 f(t) e^{-2\pi ikt} dt, \qquad k \in \mathbb{Z},\ (i := \sqrt{-1}). \tag{4.20}$$

In der *Analysis*-Vorlesung wird gezeigt, wozu sie nützlich sind und für welche umfangreiche Funktionsklasse gilt, daß

$$\sum_{k=-\infty}^{\infty} C_k(f) e^{+2\pi ikt} = f(t). \tag{4.21}$$

gibt es. Für jede Wahl von ϕ ist der Ansatz (4.23) eine Zuordnung $f \mapsto \tilde{f}$, die nur von $f|h\mathbb{Z}$ abhängt, linear und Translations-invariant ist:

$$\begin{array}{lll} \text{linear:} & f \mapsto \tilde{f}, \quad g \mapsto \tilde{g} & \Longrightarrow \quad \alpha f + \beta g \mapsto \alpha \tilde{f} + \beta \tilde{g} \\ \text{Translations-invariant:} & g = f(\cdot \pm h) & \Longrightarrow \quad \tilde{g} = \tilde{f}(\cdot \pm h). \end{array}$$

Umgekehrt folgt aus diesen drei Eigenschaften der Ansatz (4.23). Setzt man nun (4.23) in (4.22) ein, so ergibt eine kurze Rechnung

$$\tilde{C}_k(f) = \tau_k \, \frac{1}{n} \sum_{j=0}^{n-1} w_j \, e^{-2\pi i j k/n}, \tag{4.26}$$

wobei

$$\tau_k := n \int_0^1 \phi(t) e^{-2\pi i k t} \, dt \tag{4.27}$$

der sogenannte **Abminderungsfaktor** ist, der unabhängig von f ist, also nur ein-für-alle-mal ermittelt werden muß. Die gebräuchlichsten sind:

$$\begin{array}{lll} \text{Für (4.24):} & \tau_0 = 1, & \tau_k = (\sin(z)/z)^2, \quad z := \pi k/n. \\ \text{Für (4.25):} & \tau_0 = 1, & \tau_k = (\sin(z)/z)^4 \cdot 3/(1 + 2\cos(z)^2), \quad z := \pi k/n. \end{array}$$

Der zweite Faktor in (4.26) hängt von den Daten ab, ist aber periodisch in k und braucht deshalb nur für $k = 0, \ldots, n-1$ berechnet zu werden. Es ist der Koeffizient c_k des trigonometrischen Interpolanten (4.13), vgl. auch die DFT (4.17). Diese geht sehr schnell mit der FFT wie besprochen im nächsten Abschnitt.

Übungsaufgabe 4.4: Man verifiziere die Formel für die Abminderungsfaktoren zu (4.24) und (4.25).

Übungsaufgabe 4.5: Man diskutiere einen erweiterten Ansatz (4.23) der Form

$$\tilde{f}(t) = \sum_{j=0}^{n-1} (w_j \phi(t - jh) + p_j \psi(t - jh)), \qquad w_j := f(jh), \; p_j := f'(jh).$$

Übungsaufgabe 4.6: Aus $\sum_k |\tau_k| < \infty$ folgt $n\phi(t) = \sum_k \tau_k e^{2\pi i k t}$. Unter dieser Bedingung hat die Zuordnung $\phi \mapsto \{\tau_k\}$ gemäß (4.27) eine Umkehrung.

Übungsaufgabe 4.7: Man zeige, daß die Linearität und Translations-Invarianz der Zuordnung $f \mapsto \tilde{f}$ nicht nur hinreichend für die Existenz der Abminderungsfaktoren ist, sondern auch notwendig. Mit anderen Worten: Aus (4.22) und (4.26) folgt umgekehrt (4.23).

Übungsaufgabe 4.8: $\tau_k = n \int_0^1 \phi(t) e^{-2\pi i k t} dt$ und $c_k = \frac{1}{n} \sum_{j=0}^{n-1} w_j e^{-2\pi i j k/n}$ geben $\sum_{k=-\infty}^{+\infty} \tau_k c_k e^{2\pi i k t} = \sum_{j=0}^{n-1} w_j \phi(t - j/n)$; die Fourier-Synthese (4.21) mit den Abminderungsfaktoren ist also ebenso leicht durchführbar wie die Fourier-Analyse (4.20).

Durchführung von (4.29) bezeichnet wurde, entpuppt sich also als wirklich sehr naiv, ja als geradezu dumm.

In (4.30) tritt allerdings eine geringfügige „Bearbeitungsgebühr“ auf, nämlich je m Additionen, Subtraktionen, Multiplikationen und trigonometrische Koeffizienten zur gewichteten Kombination der beiden Teilsummen.

Die Rekursion

Der Aufspaltungsschritt läßt sich auch auf die beiden Teilsummen in (4.30) anwenden, die die gleiche Struktur besitzen wie die Gesamtsummen in (4.29). Voraussetzung dafür ist, daß auch m gerade ist, also n teilbar durch 4. Dabei reduziert sich der Aufwand nochmals um 50% und eine zweite „Bearbeitungsgebühr“ wird fällig mit wieder m Additionen, Subtraktionen, Multiplikationen, aber nur noch $m/2 = n/4$ trigonometrischen Koeffizienten.

Für $n = 2^p$ lassen sich p solcher Aufspaltungsschritte durchführen, welche n separate 1×1 Transformationen (4.29) erzeugen, die keine Arithmetik erfordern. Der gesamte Aufwand ist nur noch die Summe der „Bearbeitungsgebühren“, also

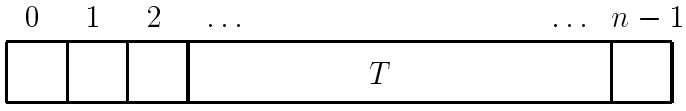
$pn/2$	Additionen, Subtraktionen, Multiplikationen,
$n/2 + n/4 + n/8 + \ldots$	trigonometrische Koeffizienten.

Die organisatorische Durchführung

Zur Gewinnung eines kompakten Algorithmus wird mit T abgekürzt die Transformation, welche w_j nach (4.29) berechnet und damit die c_k überschreibt, wie ausgedrückt durch

$$T\colon \quad \hat{c}_j = \sum_{k=0}^{n-1} c_k e^{2\pi i j k/n}, \qquad j = 0, \ldots, n-1,$$

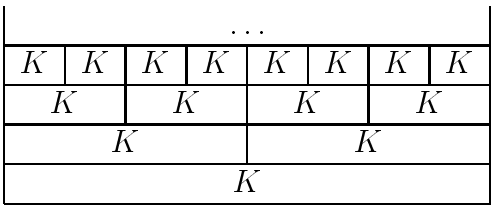
wobei c_j und $\hat{c}_j$ den Speicherplatz im Computer teilen, also nur logisch zu unterscheiden sind. Das folgende Diagramm symbolisiert den Arbeitsspeicher für die Daten und die gewünschte *in-situ*-DFT:



Als erstes sortiert man alle Komponenten nach dem Prinzip **gerade vor ungerade** und bezeichnet diesen Arbeitsschritt mit S.

$$S\colon \quad c_{2k} \mapsto c_k \qquad \text{und} \qquad c_{2k+1} \mapsto c_{k+m}, \quad k = 0, \ldots, m-1.$$

und einen unteren Teil mit lauter Kombinationsschritten zunehmender Blocklänge



In jedem Block werden Partner im Abstand der halben Blocklänge gemäß dem „butterfly diagram“ verknüpft. Zwischen dem oberen und unteren Teil sind die n separaten 1×1 Transformationen, also die Identität beim Arbeiten *in situ*.

Programm für alle Sortierschritte

Alle p Sortierschritte (der letzte davon trivial) lassen sich zu einem einzigen zusammenfassen, bei dem die Komponente in Ausgangs-Position k sofort auf ihren endgültigen Platz k' gebracht wird nach einer merkwürdig anmutenden **Regel**:

k' ist das Spiegelbild von $k \in \{0, \ldots, 2^p - 1\}$, wenn beide als p-stellige Dualzahlen geschrieben werden.

Man überzeuge sich von der Richtigkeit dieser Regel zunächst für $n = 8$ durch persönliche Ausführung der drei Sortierzeilen! Den allgemeinen Beweis erbringt man entweder durch Induktion von n auf $2n$ oder wahlweise durch Betrachten der einzelnen Sortierschritte S in dualer Schreibweise

$$\begin{array}{lcllcl} 2k & \mapsto & k & \text{dual} \quad \alpha\beta\gamma\ldots\delta 0 & \mapsto & 0\alpha\beta\gamma\ldots\delta, \\ 2k+1 & \mapsto & k+m & \text{dual} \quad \alpha\beta\gamma\ldots\delta 1 & \mapsto & 1\alpha\beta\gamma\ldots\delta. \end{array}$$

Hier sind $\alpha, \beta, \gamma, \ldots$ die $p-1$ Bit im Index k. In beiden Fällen ist das neue Bitmuster eine Rotation um einen Platz nach rechts mit Einfädeln links. Im zweiten Sortierschritt werden untere und obere Hälfte getrennt sortiert, also bleibt das oberste Bit dann außer Betrachtung und unverändert, die Rotation erstreckt sich also nur über die $p-1$ rechten Bit. Man sieht jetzt leicht ein, wie im Bitmuster für k die unteren Bit nacheinander nach oben wandern bis zum Schluß das Spiegelbild entstanden ist.

Das Spiegelbild k' von k kann man leicht rekursiv aus dem Spiegelbild $(k-1)'$ von $k-1$ berechnen, wenn k von 0 bis $n-1 = 2^p - 1$ läuft:

$$\begin{array}{lcllcl} k-1 & = & \alpha\beta\gamma\ldots 011\ldots 11 & (k-1)' & = & 11\ldots 110\ldots\gamma\beta\alpha, \\ k & = & \alpha\beta\gamma\ldots 100\ldots 00 & k' & = & 00\ldots 001\ldots\gamma\beta\alpha. \end{array}$$

Man muß also in der Dualzahl $(k-1)'$ von links her alle gesetzten Bit löschen und die erste 0 auf 1 setzen:

`für` $\delta := \beta$ `in Schritten zu` $m2$ `bis` $n-1$.

Die Schleife über α kann ersetzt werden durch

`solange` $m2 < n$.

Dann tritt $p = \log_2(n)$ nirgends auf. Cosinus und Sinus werden für Winkel $\pi\beta/m$ gebraucht in der Reihenfolge

$$\begin{array}{llll} 0 & & & \\ 0 & \pi/2 & & \\ 0 & \pi/4 & 2\pi/4 & 3\pi/4 \\ 0 & \pi/8 & 2\pi/8 & 3\pi/8 \quad \ldots \end{array}$$

Man kann sie deshalb bei Bedarf auch rekursiv berechnen. Damit sich dabei die Akkumulation von Rundungsfehlern in Grenzen hält, muß man dann unbedingt $1-\cos\pi\beta/m$ anstelle von $\cos\pi\beta/m$ im Programm einführen.

FFT von $2m$ reellen Daten

Nach Abschnitt 4.6 ist dann

$$\begin{aligned} a_k &= \frac{1}{m}\sum_{j=0}^{2m-1} y_j \cos \pi jk/m, \qquad k = 0,\ldots,m, \\ b_k &= \frac{1}{m}\sum_{j=0}^{2m-1} y_j \sin \pi jk/m, \qquad k = 1,\ldots,m-1, \\ y_j &= \frac{a_0}{2} + \sum_{k=1}^{m-1}(a_k \cos \pi jk/m + b_k \sin \pi jk/m) + \frac{a_m}{2}(-1)^j, \qquad j = 0,\ldots,2m-1. \end{aligned}$$

Alle y_j, a_k, b_k sind reell. Diesen Umstand gilt es auszunützen. Man sollte also die definierenden Umrechnungsformeln in Abschnitt 4.6 nicht unbesehen anwenden, sondern etwas trickreicher vorgehen, um Speicherplatz und arithmetische Operationen einzusparen. Dazu führt man Hilfsgrößen ein

$$c_k := \frac{1}{m}\sum_{j=0}^{m-1}(y_{2j} + iy_{2j+1})e^{-2\pi ijk/m}, \qquad k = 0,\ldots,m-1 \text{ und } m.$$

Daraus folgt

$$\begin{aligned} \bar{c}_{m-k} &= \frac{1}{m}\sum_{j=0}^{m-1}(y_{2j} - iy_{2j+1})e^{-2\pi ijk/m}, \\ f_k &:= (c_k + \bar{c}_{m-k})/2 = \frac{1}{m}\sum_{j=0}^{m-1} y_{2j}e^{-2\pi ijk/m}, \\ g_k &:= (c_k - \bar{c}_{m-k})ie^{-i\pi k/m}/2 = -\frac{1}{m}\sum_{j=0}^{m-1} y_{2j+1}e^{-\pi i(2j+1)k/m}, \end{aligned}$$

Kapitel 5

Polynom-Splines

Polynomials are wonderful even after they are cut into pieces
(I.J. Schoenberg, Spline-Erfinder).

Aus Abschnitt 4.5 ist bekannt, daß mit wachsendem Grad Polynome rasch ihre vorteilhaften Eigenschaften verlieren: der Rechenaufwand für Konstruktion und Auswertung wird zu teuer und die Kondition zu schlecht. Weil bei der Interpolation Grad $n-1$ und Stützstellenzahl n starr miteinander verknüpft sind, kann man mit Polynomen immer nur eine sehr beschränkte Zahl von Stützpunkten interpolieren bzw. approximieren.

Im Zeitalter der Computer und automatischen Meßeinrichtungen hat man jedoch häufig sehr große Datenmengen zu verarbeiten. Eine sehr primitive Möglichkeit ist das Aufspalten der Daten in viele hinreichend kleine Teilmengen, die für sich mit Polynomen verarbeitet werden, zuständig jeweils für ein Teilintervall der reellen Achse.

Solche **Polynom-Segmente** mit intervallweiser Zuständigkeit definieren gemeinsam eine Funktion, die sich von den Lehrbuch-Funktionen nur dadurch unterscheidet, daß sie eben nicht mit *einer* Formel beschrieben wird, sondern mit Fallunterscheidungen. Zwei geläufige Beispiele dafür sind

die **Sprungfunktion**

$$S(x) := \begin{cases} 0 & \text{falls } x \le 0, \\ 1 & \text{falls } x > 0, \end{cases}$$

das **Polygon** mit Ecken $x_i, y_i, \quad i = 1, \ldots, n$

$$L(x) := y_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i} \qquad \text{für } x_i \le x < x_{i+1}.$$

Ein Polygon ist also die Zusammenstückelung von lauter Geraden-Segmenten (Polynomen ersten Grades) mit der Besonderheit, daß keine Sprünge (Unstetigkeiten) in der Gesamtfunktion auftreten. Dieses Zusammensetzen von Polynom-Segmenten bei **glatten Übergängen** ist der obigen primitiven Zerhackung der

5.1 Grundlagen

Definition 5.1 *Polynom-Splines mit einfachen Knoten*

Sei $x_1 < \ldots < x_n$ und $m \in \mathbb{N}$. $s\colon [x_1, x_n] \to \mathbb{R}$ heißt **Splinefunktion der Ordnung** m (bzw. vom Grad $m-1$), wenn

$$\begin{aligned} s(x) &= p_i(x) \quad \text{für } x_i \le x < x_{i+1} \\ &\quad \text{mit } p_i \in \mathbb{P}_{m-1} \text{ für } i = 1, \ldots, n-1 \text{ und} \qquad (5.1) \\ s &\in C^{m-2}[x_1, x_n], \qquad \text{(leer bei } m = 1). \qquad (5.2) \end{aligned}$$

Die x_i sind ihre **Knoten**. Die Menge aller solchen Splines wird mit $\mathcal{S}_m(x_1, \ldots, x_n)$ bezeichnet. $x_1 = -\infty$ und $x_n = +\infty$ sind zulässig.

Beispiele:

$m = 1$: Treppenfunktionen,
$m = 2$: Polygone,
$m = 3$: Quadratische Splines,
$m = 4$: Kubische Splines, usw.

Alternativ zu Definition 5.1: Eine Splinefunktion der Ordnung m ist das $(m-1)$-fache unbestimmte Integral einer Treppenfunktion.

Die Ableitung einer Splinefunktion der Ordnung m ist also eine Splinefunktion der Ordnung $m-1$ (bei $m = 2$ an den Knoten nur rechtsseitige Ableitung), und ihr unbestimmtes Integral ist eine Spline-Funktion der Ordnung $m+1$.

$\mathcal{S}_m(x_1, \ldots, x_n)$ mit der bei Funktionen üblichen Regel für Addition und Multiplikation mit einem Skalar (reelle Konstante) bildet einen reellen Vektorraum, d.h. die Summe zweier Splinefunktionen und ihr skalares Vielfaches sind selbst Splinefunktionen der gleichen Ordnung und mit den gleichen Knoten. Das ist offensichtlich.

Die Dimension dieses Raumes ist anhand der alternativen Definition sofort erkennbar: Es gibt $n-1$ linear unabhängige Treppenfunktionen über den Intervallen $[x_i, x_{i+1}[,\ i = 1, \ldots, n-1$, so daß

$$\dim \mathcal{S}_1(x_1, \ldots, x_n) = n - 1$$

und jede Integration bringt *eine* Integrationskonstante und damit *einen* weiteren Parameter bzw. Freiheitsgrad, so daß

$$\dim \mathcal{S}_m(x_1, \ldots, x_n) = n + m - 2.$$

Dasselbe ergibt sich auch aus der Definition 5.1, die auch erlaubt, eine für alle theoretischen Zwecke sehr bequeme Basis einzuführen.

Beweis: In der Tat, jedes s mit den Eigenschaften (5.1), (5.2) läßt sich eindeutig schreiben in der Form

$$s(x) = \sum_{k=0}^{m-1} c_k x^k + \sum_{i=2}^{n-1} d_i \cdot (x_i - x)_+^{m-1}.$$

$c_0, \ldots, c_{m-1}$ ergeben sich aus $s|_{[x_{n-1}, x_n]}$ und d_i ergibt sich aus $s|_{[x_i - \varepsilon, x_i + \varepsilon]}$ für $i = n-1, \ldots, 2$. ■

Die angegebene Basis enthält $m+n-2$ Mitglieder und das bestätigt obige Angabe zur Dimension von $\mathcal{S}_m(x_1, \ldots, x_n)$. Die Verwendung von $(x_i - x)_+^{m-1}$ in der Basis 5.3 macht $D^{m-1}s$ stetig von rechts statt von links, entsprechend den halbseitig offen gewählten Zuständigkeitsintervallen $[x_i, x_{i+1}[$ in Definition 5.1.

Man kann in die Basis des Satzes 5.3 weitere Funktionen aufnehmen und dadurch die Definition 5.1 der Polynom-Splines verallgemeinern:

Definition 5.4 *Polynom-Splines mit mehrfachen Knoten*

Wenn die Basis die Funktionen

$$(x_i - x)_+^{m-1}, \ldots, (x_i - x)_+^{m-\mu}, \qquad i \in \{2, \ldots, n-1\}$$

enthält, dann ist x_i ein **μ-facher Knoten**. Die Schreibweise dafür ist, den Knoten x_i in die Knotenliste μ-mal aufzunehmen

$$\ldots x_{i-1} < x_i = \ldots = x_{i+\mu-1} < x_{i+\mu} \ldots$$

Obige Teilmenge von zugehörigen Basisfunktionen wird im folgenden immer mit $(x_i - x)_+^{m-1-}$ angesprochen. Immer ist $x_1 < x_2$ und $x_{n-1} < x_n$.

Bemerkungen:

1. $\mu \leq m$: Splines der Ordnung m können höchstens m-fache Knoten haben.
2. An einem μ-fachen Knoten ist $D^k s$ stetig für $k = 0, \ldots, m-\mu-1$, während $D^{m-\mu}s$ unstetig (von links) ist.
3. Solche Splines haben ein Stetigkeits-Defizit und heißen deshalb **defizient**.
4. $\mu = m$ gibt die vollständige Durchtrennung der Splinefunktion in zwei entkoppelte Zweige und damit einen trivialen Grenzfall.
5. Für $x_i = x_{i+1}$ ist das halbseitig offene Intervall $\{x : x_i \leq x < x_{i+1}\}$ leer und es spielt keine Rolle, ob dafür ein Polynom p_i zuständig ist oder nicht.
6. $\dim \mathcal{S}_m(x_1, \ldots, x_n) = n + m - 2$ bleibt gültig, auch bei mehrfachen Knoten, weil jedem zusätzlichen Mitglied in der Basis ein zusätzliches Mitglied in der Knotenliste gegenübersteht.

gewählt sein, damit in den Interpolations-Bedingungen für $s, \ldots, D^{j-1}s$ keine bloß rechtsseitigen Ableitungen $D^{m-k}s, D^{m-k+1}s, \ldots$ auftreten.

In der Praxis ist die Interpolations-Aufgabe mit ihren Stützpunkten primär vorgegeben und nicht die Splines. Also sind es die Ordnung m und die Knoten $x_1, \ldots, x_n$, die so zu wählen sind, daß die Verschränkungs-Bedingung erfüllt ist.

Die Standard-Wahl ist eine gerade Ordnung m (in der Regel $m = 4$) und Knoten an allen Stützstellen mit Ausnahme der ersten und der letzten $m/2$ Stützstellen. Wenn die Stützstellen mit $x_1, \ldots, x_N$ bezeichnet werden, dann sind die Knoten $x_{1+m/2}, \ldots, x_{N-m/2}$. Dabei wird nicht nur die Lage, sondern gegebenenfalls auch die Multiplizität gleich gewählt. Nach dem oben Gesagten sollte diese gemeinsame Multiplizität dann $m/2$ nicht überschreiten. Ein Stützpunkt/Knoten mit dieser maximalen Multiplizität zerhackt übrigens diese Interpolations-Aufgabe in zwei völlig entkoppelte Teilprobleme.

Die beschriebene Standard-Wahl erfüllt offensichtlich die Verschränkungs-Bedingung.

5.3 Kubische Spline-Interpolation ($m = 4$)

Besonders leicht ist die Situation, wo $x_1, \ldots, x_n$ lauter doppelte Knoten und doppelte Stützstellen sind, denn dann zerfällt das Problem in lauter kubische Teilprobleme für die einzelnen Intervalle (Hermite-Interpolation):

$$\begin{array}{lrcl@{\qquad}rcl} x_i \le x < x_{i+1}: & s(x) & = & p_i(x) \in \mathbb{P}_3, & & & \\ & p_i(x_i) & = & y_i, & p_i(x_{i+1}) & = & y_{i+1}, \\ & p_i'(x_i) & = & y_i', & p_i'(x_{i+1}) & = & y_{i+1}', \end{array} \tag{5.4}$$

wobei y_i und y_i' die vorgeschriebenen Stützwerte für die doppelte Stützstelle x_i sind. (Das sonst verwendete Bezeichnungsschema wäre $x_{2i-1} = x_{2i}$ und y_{2i-1}, y_{2i}, ist aber zu umständlich.)

Die Lösung davon ist

$$\begin{aligned} p_i(x) &= y_i \cdot (1 - 3t^2 + 2t^3) + y_i' h_i \cdot (t - 2t^2 + t^3) \\ &\quad + y_{i+1} \cdot (3t^2 - 2t^3) + y_{i+1}' h_i \cdot (-t^2 + t^3), \\ p_i'(x) &= z_i \cdot (6t - 6t^2) + y_i' \cdot (1 - 4t + 3t^2) + y_{i+1}' \cdot (-2t + 3t^2), \\ p_i''(x) &= \Big(z_i \cdot (6 - 12t) + y_i' \cdot (-4 + 6t) + y_{i+1}' \cdot (-2 + 6t) \Big) / h_i. \end{aligned} \tag{5.5}$$

Dabei ist $t := (x - x_i)/h_i$, $h_i := x_{i+1} - x_i$ und $z_i := (y_{i+1} - y_i)/h_i$ die erste dividierte Differenz.

Setzt man hier $x = x_i$ bzw. x_{i+1} ein, entsprechend $t = 0$ bzw. 1, so sieht man sofort, daß (5.4) erfüllt ist:

$$\begin{aligned} s(x_i - 0) &= p_{i-1}(x_i) = y_i = p_i(x_i) = s(x_i + 0), \\ s'(x_i - 0) &= p_{i-1}'(x_i) = y_i' = p_i'(x_i) = s'(x_i + 0). \end{aligned}$$

Typ 2°: Die sogenannten **natürlichen Randbedingungen** sind der homogene Spezialfall, also

$$s''(x_1) = s''(x_n) = 0.$$

Solche s heißen **natürliche, kubische Splinefunktionen**. „Natürlich“ wird im nächsten Abschnitt erläutert.

Typ 3: Schließlich kann man mit dem Ansatz (5.4), (5.5) auch **periodische Randbedingungen** behandeln,

$$s'(x_1 + 0) = s'(x_n - 0) \qquad \text{und} \qquad s''(x_1 + 0) = s''(x_n - 0).$$

Das erfordert $y'_1 = y'_n$ im Ansatz und $y''_1 = y''_n$ in (5.8), addiert also

$$(2/h_1 + 2/h_{n-1})y'_1 + y'_2/h_1 + y'_{n-1}/h_{n-1} = 3(z_1/h_1 + z_{n-1}/h_{n-1}).$$

Diese Gleichung kann man als (5.7) für $i = 1$ betrachten. Auch dieses Gleichungssystem, jetzt der Ordnung $n - 1$, hat eine positiv-definite Matrix. Sie ist tridiagonal bis auf die Extra-Elemente $1/h_{n-1}$ in der rechten, oberen und der linken, unteren Ecke. Das macht das Auflösen knapp doppelt so teuer. Die Lösung heißt **periodische, kubische Splinefunktion**, sofern man $y_1 = y_n$ wählt, so daß auch $s(x_1) = s(x_n)$.

Die periodische, kubische Spline mit äquidistanten Knoten/Stützstellen war in Abschnitt 4.7 bei den Abminderungsfaktoren erwähnt worden.

Hinweise:

1. Man kann einfache und doppelte Knoten/Stützstellen beliebig mischen. An letzteren ist s' vorgeschrieben und das Interpolations-Problem zerfällt dort in entkoppelte und völlig unabhängige Teilprobleme.

2. Rundungsfehler beim Auflösen des linearen Gleichungssystems (5.7) bzw. (5.7), (5.8) führen zu unkorrekten Steigungen $\tilde{y}'_i$ anstelle der richtigen y'_i und mit Ansatz (5.4) zu verfälschten $\tilde{p}_i$ anstelle der richtigen p_i. Das daraus zusammengesetzte $\tilde{s}$ ist immer noch glatt: $\tilde{s} \in C^1[x_1, x_n]$. Das ist ein sehr wichtiger Vorteil des Ansatzes (5.4).

3. Anstelle von (5.4) kann man einen Ansatz für p_i machen, der mit den zweiten Ableitungen statt der ersten Ableitungen y'_i arbeitet. Die Basisfunktionen sind dann $1-t, 2t-3t^2+t^3, t, -t+t^3$ anstelle von denen in (5.5). Ein solcher Ansatz für jedes Intervall bewirkt, daß die daraus zusammengesetzten s und s'' stetig sind. Die Bedingungen $s'(x_i - 0) = s'(x_i + 0)$ führen dann wieder auf ein positiv-definites, tridiagonales Gleichungssystem. Seine Aufstellung und Auflösung erfordert geringfügig weniger arithmetische Operationen. Aber der unter Punkt 2 soeben diskutierte Vorteil entfällt: Rundungsfehler bewirken hier ein unstetiges s'. Als Übungsaufgabe wird die Durchführung dieses Lösungsweges nachdrücklich empfohlen.

Zerlegung und zweimalige partielle Integration gibt

$$\begin{aligned}\int_{x_1}^{x_n}(f''-s'')\cdot s'' &= \sum_{i=1}^{n-1}\int_{x_i}^{x_{i+1}}(f''-s'')\cdot s'' \\ &= \sum_{i=1}^{n-1}\Big[(f'-s')\cdot s''-(f-s)\cdot s'''\Big]_{x_i+0}^{x_{i+1}-0}.\end{aligned}$$

Auf Grund von (5.9), (5.10) und der Stetigkeit von $f'-s'$ und s'' verschwinden alle diese Summanden oder heben sich gegenseitig weg: das Resultat ist immer 0. Die Stetigkeit von f'' ist übrigens gar nicht benutzt worden, ein stückweise stetiges f'' hätte genügt. Das führt auf

Satz 5.6 *Erste Integral-Relation für $m = 4$*

Sei s der kubische Spline-Interpolant von f mit Randbedingungen vom Typ 1 oder 2° oder 3. Dann gilt

$$\int_{x_1}^{x_n}(f''-s'')\cdot s'' = 0,$$

$$\int_{x_1}^{x_n} f''^2 \geq \int_{x_1}^{x_n} s''^2.$$

Die zweite Aussage folgt durch Integration von

$$f''^2 = (f''-s''+s'')^2 = (f''-s'')^2 + 2(f''-s'')\cdot s'' + s''^2.$$

Man kann f und s als zwei konkurrierende Interpolanten der gemeinsamen Stützpunkte x_i, y_i $(i = 1, \ldots, n)$ betrachten und

$$\int_{x_1}^{x_n} f''^2$$

als ein reziprokes Maß für die Glattheit eines Interpolanten. In diesem Sinne **ist der kubische Spline-Interpolant der glatteste aller Interpolanten** und zwar:

- der natürliche Spline-Interpolant ist glatter als *alle* anderen,
- der komplette Spline-Interpolant ist glatter als alle anderen mit gleicher Steigung bei x_1 und x_n,
- der periodische Spline-Interpolant ist glatter als alle anderen periodischen Interpolanten.

5.5 Die B-Splines

Polynom-Splines sind nicht nur ideal für die Interpolation, sondern ebenso für die Erzeugung sogenannter **Frei-Form-Kurven** im CAD (*Computer Aided Design*). Dabei geht es darum, dem Stylisten („*Designer*“) ein mathematisches Hilfsmittel an die Hand zu geben zur Gestaltung elegant geschwungener und ästhetisch befriedigender Kurven für Autokarosserieteile, Buchstaben in Computer-Schriften usw. Hier hat sich die Einführung von Stützpunkten und deren Interpolation überhaupt nicht bewährt. Somit jetzt wieder weg von der Interpolation und zurück zu der allgemeinen Situation von Abschnitt 5.1: Ordnung $m \in \mathbb{N}$ und bis zu m-fache Knoten, also immer $x_i < x_{i+m}$.

Wie schon in der Bemerkung am Ende von Abschnitt 5.1 erwähnt, ist es bequem, den ersten und letzten Knoten je m-fach anzusetzen, etwa

$$x_1 = \ldots = x_m < x_{m+1} \leq \ldots \leq x_n < x_{n+1} = \ldots = x_{n+m}.$$

x_m und x_{n+1} sind jetzt die beiden Endpunkte, die früher x_1 und x_n hießen; die Anzahl der dazwischen liegenden Knoten war früher $n-2$ und ist jetzt $n-m$. Die Dimension des Spline-Raumes, früher $n-2+m$, ist jetzt n. Schließlich ist die Basis von Satz 5.3 bzw. Definition 5.4 einschließlich der zugehörigen Bemerkung in der neuen Zählung

$$(x_i - x)_+^{m-1-} \qquad \text{für} \quad i = m+1, \ldots, m+n, \tag{5.11}$$

wobei das Minuszeichen hinter dem Exponenten daran erinnert, daß bei einem μ-fachen Knoten die Exponenten $m-1, \ldots, m-\mu$ auftreten. Diese Abkürzung war in Definition 5.4 vereinbart worden.

Die Basis (5.11) ist simpel und für theoretische Zwecke somit ideal. Sie ist leider gänzlich ungeeignet für alle Rechenverfahren. Zum Beispiel würde ein Ansatz mit (5.11) für den Spline-Interpolanten zu einem linearen Gleichungssystem führen, das *nicht* mit $O(n)$ arithmetischen Operationen aufzulösen ist und obendrein eine so schlechte Kondition besitzt, daß die Rundungsfehler für größere n alles unbrauchbar machen.

Der Spline-Erfinder I.J. Schoenberg hat bestimmte gestaffelte Linearkombinationen entdeckt, die eine perfekte Basis für die praktische Arbeit abgeben, weil sie in dem kleinst-möglichen Intervall von Null verschieden sind. Er nannte sie aus diesem Grund die **Minimum-Support-Splines** (der **Support** oder **Träger** von f ist die abgeschlossene Hülle aller Punkte x, wo $f(x) \neq 0$). Recht lakonisch werden sie inzwischen **B-Splines** genannt, weil sie eben *die* Basis schlechthin darstellen. Insbesondere im CAD-Bereich geht ohne sie nichts (mehr).

Die definierende Formel für die B-Splines der Ordnung m ist

$$M_{k,m}(t) := [x_k, \ldots, x_{k+m}](\cdot - t)_+^{m-1} \qquad \text{für} \quad k = 1, \ldots n, \tag{5.12}$$

das sind also die dividierten Differenzen der Ordnung m von $(x-t)_+^{m-1}$ als Funktion in x mit t als einem unbeteiligten Parameter, gestützt auf $x = x_k, \ldots, x_{k+m}$.

1972 ist von drei Autoren unabhängig eine wundervolle Rekursionsformel für die B-Splines entdeckt worden. Sie beruht auf der Produkt-Regel für dividierte Differenzen, angegeben in der Übungsaufgabe 4.2 von Abschnitt 4.3. Dazu schreibt man in (5.12)

$$(\cdot - t)_+^{m-1} = f \cdot g \qquad \text{mit} \quad f := (\cdot - t),\ g := (\cdot - t)_+^{m-2}.$$

f ist linear, seine erste dividierte Differenz ist 1 und alle höheren sind 0, so daß

$$[x_k, \ldots, x_{k+m}]fg = (x_k - t) \cdot [x_k, \ldots, x_{k+m}]g + 1 \cdot [x_{k+1}, \ldots, x_{k+m}]g$$

und

$$[x_k, \ldots, x_{k+m}]g = \frac{[x_{k+1}, \ldots, x_{k+m}]g - [x_k, \ldots, x_{k+m-1}]g}{x_{k+m} - x_k}.$$

Eingesetzt und zusammengefaßt ergibt sich

$$M_{k,m}(t) = \frac{t - x_k}{x_{k+m} - x_k} \cdot M_{k,m-1}(t) + \frac{x_{k+m} - t}{x_{k+m} - x_k} \cdot M_{k+1,m-1}(t). \tag{5.14}$$

Natürlich ist diese Rekursionsformel nur relevant, wo die linke Seite nicht verschwindet, also für $x_k \le t < x_{k+m}$. Dann sind die beiden Gewichtskoeffizienten auf der rechten Seite ≥ 0, so daß alle auftretenden Terme ≥ 0 sind. Dadurch können Rundungsfehler nichts kaputt machen.

Übrigens ist stillschweigend unterstellt worden, daß weder x_k noch x_{k+1} ein m-facher Knoten ist, denn dann wäre $M_{k,m-1}$ bzw. $M_{k+1,m-1}$ undefiniert. Man überzeugt sich anhand obiger expliziter Formel jedoch sofort, daß (5.14) korrekt ist, wenn man den undefinierten Term auf Null setzt. Das ist in bester Übereinstimmung mit dem allgemeinen Fall:

$$\begin{aligned} M_{k,m-1}(t) &= 0 \qquad \text{für} \quad t \notin [t_k, t_{k+m-1}], \\ M_{k+1,m-1}(t) &= 0 \qquad \text{für} \quad t \notin [t_{k+1}, t_{k+m}]. \end{aligned}$$

In einem gut organisierten Computer-Programm erfordern m-fache Knoten deshalb keine Sonder-Maßnahmen.

Nach Einführung dieser Konvention betreffend der mehrfachen Knoten hat man ein dreieckiges Berechnungsschema für $M_{k,m}(t)$:

$$\begin{array}{ccccccc} M_{k,1}(t), & . & . & . & . & ., & M_{k+m-1,1}(t) \\ & . & & & & . & \\ & & . & & . & & \\ & & & . & & & \\ & & & & M_{k,m}(t) & & \end{array}$$

Dieses Rechenschema hat die CAD-Software für Splinefunktionen revolutioniert. Es erfüllt alle vertretbaren Wünsche.

Kapitel 6

Iterative Lösung nicht-linearer Gleichungen

6.1 Nomenklatur

Es sei $f: \mathbb{R}^n \to \mathbb{R}^n$ hinreichend glatt, d.h. die auftretenden Ableitungen seien stetig. Für Vektor-*Komponenten* werden Tief-Indizes verwendet und für Vektor-*Folgen* Hoch-Indizes:

$$f(x) \equiv \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{pmatrix} \quad \text{und} \quad f'(x) \equiv \begin{pmatrix} \partial f_1/\partial x_1 & \ldots & \partial f_1/\partial x_n \\ \vdots & & \vdots \\ \partial f_n/\partial x_1 & \ldots & \partial f_n/\partial x_n \end{pmatrix}.$$

$f(x) = 0$ ist ein System von n im allgemeinen nicht-linearen Gleichungen. Eine Lösung wird mit $\bar{x}$ bezeichnet. Existenz und Eindeutigkeit sind in der Regel offen bzw. werden angenommen (vorausgesetzt).

Elementare Konzepte und Begriffe werden für den n-dimensionalen Fall eingeführt, aber bei der Diskussion einiger Verfahren begnügen wir uns mit dem univariaten Fall, d.h. $n = 1$.

Es wäre falsch, wenn man sich unter der Funktion f immer nur eine einfache Formel vorstellen würde. f kann genausogut eine sehr komplizierte Berechnungsvorschrift sein, die den Computer stundenlang beschäftigt. Man mißt den Aufwand eines Verfahrens zur Lösung von $f(x) = 0$ durch die Anzahl der Funktions-Auswertungen, die es bei gleichen Anfangs- und Abbruchs-Bedingungen erfordert.

Zur Nullstellen-Bestimmung gehört im Prinzip auch die Suche nach Extrema einer Funktion $F: \mathbb{R}^n \to \mathbb{R}$, denn dann muß man die Nullstellen der Funktion $F' \equiv \partial F/\partial x: \mathbb{R}^n \to \mathbb{R}^n$ suchen, kann also f mit F' identifizieren. Dafür ist dann $f' = F''$ immer symmetrisch und in der Umgebung der Extrema sogar positiv/negativ-definit. Dieser Umstand ist so vorteilhaft, daß man lieber Spezial-Algorithmen für die Maximierung/Minimierung einer Funktion F benutzt, die davon guten Gebrauch machen. Insbesondere verzichten diese Algorithmen in der

$p = 1$ heißt auch lineare (geometrische) Konvergenz,
$p = 2$ heißt auch quadratische Konvergenz, usw.

Bei $p = 1$ ist c der **Konvergenz-Faktor**.
$p = 1$ und $\|x^{i+1} - \bar{x}\|/\|x^i - \bar{x}\| \to 0$ heißt **superlineare** Konvergenz.

In der asymptotischen Phase vergrößert sich die Zahl der korrekten Dezimalen mit jedem Iterationsschritt um den Faktor p. Zum Beispiel verdoppelt sich bei quadratischer Konvergenz die Anzahl der korrekten Ziffern mit jedem Iterationsschritt (exakte Rechnung vorausgesetzt).

Bei 1-Schritt-Verfahren ist die Konvergenz-Ordnung oft ganzzahlig und an der Taylor-Entwicklung der Iterationsfunktion ϕ um die Stelle $\bar{x}$ abzulesen:

$$x^{i+1} - \bar{x} = \phi(x^i) - \phi(\bar{x}) = \phi'(\bar{x}) \cdot (x^i - \bar{x}) + \ldots$$

Also $\phi'(\bar{x}) = \ldots = \phi^{(p-1)}(\bar{x}) = 0$ und $\phi^{(p)}(\bar{x}) \neq 0$ bei Konvergenz-Ordnung p.

Bei Mehrschritt-Verfahren ist die Konvergenz-Ordnung im allgemeinen nicht mehr ganzzahlig. Zum Beispiel gilt für manche 2-Schritt-Verfahren ein Fehlergesetz der Form

$$\|x^{i+1} - \bar{x}\| \leq c \cdot \|x^i - \bar{x}\| \cdot \|x^{i-1} - \bar{x}\| \qquad \text{für} \quad i \geq i_0. \tag{6.1}$$

(Das Sekanten-Verfahren im nächsten Abschnitt ist dafür ein Beispiel.) Alle Verfahren mit diesem Fehlergesetz habe die gleiche Konvergenz-Ordnung $p = (1 + \sqrt{5})/2 \doteq 1.618$. Um das zu beweisen, multipliziert man (6.1) mit c durch und logarithmiert. Induktion zeigt dann

$$\log(c \cdot \|x^i - \bar{x}\|) \leq \varepsilon_i \qquad \text{mit} \quad \varepsilon_{i+1} = \varepsilon_i + \varepsilon_{i-1} \text{ für } i \geq i_0.$$

Rechts steht die bekannte **Fibonacci-Rekursion** mit allgemeiner Lösung

$$\varepsilon_i = \alpha\lambda_1^i + \beta\lambda_2^i,$$

wobei $\lambda_{1,2} = (1 \pm \sqrt{5})/2$ die beiden Wurzeln der zugehörigen charakteristischen Gleichung $\lambda^2 = \lambda + 1$ sind (vergleiche die analoge Theorie für homogene, lineare Differentialgleichungen). Also $\varepsilon_i \simeq \alpha\lambda_1^i$ für $i \to \infty$ oder $\varepsilon_{i+1} \simeq \varepsilon_i\lambda_1$, so daß

$$\|x^{i+1} - \bar{x}\| \leq c^{\lambda_1 - 1} \cdot \|x^i - \bar{x}\|^{\lambda_1} \qquad \text{für} \quad i \to \infty.$$

Die Grundlage vieler 1-Schritt-Verfahren ist die **Linearisierung**: Man entwickelt die nicht-lineare Funktion f in eine Taylorreihe um die Stelle x^i:

$$f(x) = f(x^i) + f'(x^i) \cdot (x - x^i) + \ldots$$

Bricht man hier nach den linearen Termen ab, so erhält man eine lineare Näherung für $f(x)$, deren Nullstelle als nächste Näherung x^{i+1} genommen wird:

$$f(x^i) + f'(x^i) \cdot (x^{i+1} - x^i) = 0.$$

$A := f(a)$; $B := f(b)$, so daß $A \cdot B < 0$
solange $b - a >$ *Toleranz*:
        $t := (a + b)/2$; $T := f(t)$;
        falls $A \cdot T > 0$: $\{a := t;\ A := T\}$
        sonst: $\{b := t;\ B := T\}$
$\bar{x} := (a + b)/2$.

Im allgemeinen *keine* Verbesserung bringt die **regula falsi**, die sich von der Bisektion nur unterscheidet durch eine andere Wahl von t: Statt des Mittelpunktes wählt man hier den Nulldurchgang der Sekante von a nach b:

$$A + \frac{B - A}{b - a}(t - a) = 0 \qquad \Longleftrightarrow \qquad t = a + \frac{A}{A - B}(b - a).$$

(Beachte, daß A und $-B$ gleiches Vorzeichen haben.)

Wenn nicht zufällig $f''(\bar{x}) = 0$, dann ist von einem Zeitpunkt an f'' in $[a, b]$ durchwegs positiv (f konvex) oder durchwegs negativ (f konkav). Von da an ist in allen Schritten t nur noch links oder nur noch rechts von $\bar{x}$ und nur noch a oder nur noch b streben gegen $\bar{x}$. Die Intervall-Länge $b - a$ strebt also *nicht* gegen 0. Man hat lineare Konvergenz mit dem Konvergenzfaktor $1 - f'(\bar{x})(b-a)/(B-A)$, wobei a oder $b = \bar{x}$. Dieser Faktor muß durchaus nicht kleiner als $\frac{1}{2}$ sein: Die regula falsi kann langsamer als die Bisektion sein! Für eine genaue Analyse der asymptotischen Phase genügt es, $f(x) = x + cx^2$ im Intervall $[-\varepsilon, +\varepsilon]$ zu betrachten mit $|c| \cdot \varepsilon \ll 1$.

Wirksame Abhilfe schafft eine verblüffend kleine Modifikation, nämlich einen Funktionswert (A oder B) nach jeder Verwendung außer nach der ersten zu halbieren. Dadurch kann t nicht dauernd auf der gleichen Seite der Nullstellen $\bar{x}$ bleiben und sowohl a als auch b werden bewegt und streben gegen $\bar{x}$. Das ist bekannt unter dem Namen **modifizierte regula falsi** oder **Illinois-Algorithmus**. Weil ein früherer Iterationspunkt t als Endpunkt a oder b eine Zeitlang im Spiel bleiben kann, ist dem Verfahren keine genaue Schrittzahl zuzuordnen:

$A := f(a)$; $B := f(b)$, so daß $A \cdot B < 0$
$m := 0$; $t := a + (b - a) \cdot A/(A - B)$;
solange $a < t < b$:
        $T := f(t)$;
        falls $A \cdot T > 0$:
                $a := t$; $A := T$;
                falls $m > 0$: $B := B/2$ sonst $m := 1$;
        sonst
                $b := t$; $B := T$;
                falls $m < 0$: $A := A/2$ sonst $m := -1$;
        $t := a + (b - a) \cdot A/(A - B)$;
$\bar{x} := t$.

Eine vorzügliche Kombination von Einfachheit, Robustheit und Effizienz, die aus jedem Computer die best-mögliche Genauigkeit herausholt!

gegenüber der modifizierten regula falsi und beruht darauf, daß ohne Ausnahme in jedem Iterationsschritt die beiden jüngsten Näherungswerte verwendet werden.

Der Verzicht auf jegliche Sicherheits-Maßnahmen lohnt sich kaum und empfiehlt sich nur bei Rechnungen, die der Anwender persönlich überwacht (Taschenrechner, PC, usw.). In Programm-Bibliotheken findet man meist eine raffinierte Kombination von Bisektion und Sekanten-Methode. Letztere kriegt freien Lauf, solange sie Abstand hält von den Endpunkten eines Einschließungs-Intervalles. Die Details sind ziemlich technisch.

Obiges Fehlergesetz gilt auch für $x^i = x^{i-1}$, also für die Tangente anstelle der Sekante. Das ist das Newton-Verfahren und das Fehlergesetz ist im konfluenten Grenzfall

$$x^{i+1} - \bar{x} = (x^i - \bar{x})^2 f''(\xi)/2f'(x^i).$$

Wie zuvor schon gezeigt, hat also das Newton-Verfahren in der Umgebung einer einfachen Nullstelle die Konvergenz-Ordnung 2. Selbstverständlich wird man einen Newton-Schritt, der die Berechnung von f und f' benötigt, vergleichen mit zwei Schritten der modifizierten regula falsi bzw. des Sekanten-Verfahrens:

$$\begin{array}{rccccccc}
\text{Newton:} & \varepsilon & \mapsto & & & \varepsilon^2 & \mapsto & \dots \\
\text{Modifizierte regula falsi:} & \varepsilon & \mapsto & \varepsilon^{1,442} & \mapsto & \varepsilon^{2,08} & \mapsto & \dots \\
\text{Sekanten-Verfahren:} & \varepsilon & \mapsto & \varepsilon^{1,618} & \mapsto & \varepsilon^{2,618} & \mapsto & \dots
\end{array}$$

Bei diesem Vergleich erscheint das Newton-Verfahren nicht optimal.

6.3 Wurzeln von Polynomen

Wenn f ein Polynom ist, heißt $f(x) = 0$ eine **algebraische Gleichung** und die Nullstellen $\bar{x}$ werden auch **Wurzeln** genannt.

f kann in der Taylorform $c_0 + c_1 x + \ldots + c_n x^n$ gegeben sein oder durch Spezifikation von $n+1$ Stützstellen x_i, y_i $(i = 0, \ldots, n)$ oder als charakteristisches Polynom einer Matrix, $\det(xI - A)$, usw. Im letzten Fall heißen die Nullstellen Eigenwerte und sollten immer mit den dafür entwickelten Verfahren berechnet werden.

Besonderheiten sind:

1. Man möchte *alle* Nullstellen, oder man möchte eine spezielle Nullstelle (z.B. betragskleinste, größter Realteil, ...)
2. Man hat auch komplexe Nullstellen zu finden, selbst wenn das Polynom reell ist.
3. Die Nullstellen sind häufig extrem schlecht konditioniert.

allein $P' \approx 0$. Außerdem ist ein kleiner Nenner hier nicht dieselbe Katastrophe wie beim Newton-Verfahren, denn x^{i+1} wird dadurch zwar sehr groß, aber x^{i+2} fällt wieder mitten unter die n Nullstellen. (Das Newton-Verfahren bleibt dagegen im Punkt ∞ praktisch hängen.)

3. Das Verfahren geht auch bei reellen Polynom-Koeffizienten und reellem Startwert von selbst in die komplexe Zahlenebene, sofern komplexe Nullstellen vorliegen.

4. Falls alle Nullstellen reell sind, konvergiert die Iteration global zur nächsten Nullstelle rechts oder links vom Startwert (auf der projektiven Zahlengeraden mit $+\infty = -\infty$)

Beweisskizze zu 4: Sei dazu

$$P(x) := c \cdot (x - \lambda_1) \cdots (x - \lambda_n), \qquad \text{alle } \lambda_i \text{ reell.}$$

Dann ist

$$\begin{aligned}
S_1 &:= \sum_i (\lambda_i - x)^{-1} = -P'/P, \\
S_2 &:= \sum_i (\lambda_i - x)^{-2} = (-P'/P)' = (P'^2 - PP'')/P^2, \\
(n-1) \cdot (nS_2 - S_1^2) &= W^2/P^2, \\
\phi_\pm(x) - x &= \frac{n}{S_1 \mp \sqrt{(n-1) \cdot (nS_2 - S_1^2)}} \\
&= \frac{S_1 \pm \sqrt{(n-1) \cdot (nS_2 - S_1^2)}}{S_1^2 - (n-1)S_2}.
\end{aligned}$$

Das sind die beiden Nullstellen der Parabel in t

$$Q(t) := (S_1 t - 1)^2 - (n-1) \cdot (S_2 t^2 - 1).$$

Nach der Cauchy-Schwarzschen Ungleichung ist $S_1^2 \le nS_2$, so daß die Quadratwurzel in obigen Formeln immer reell ist: Das Laguerre-Verfahren bleibt im Reellen, wenn der Startwert und alle Nullstellen reell sind.

Für $t = \lambda_k - x$, $(k = 1, \ldots, n)$ ist $Q(t) \le 0$, denn für diese t ist

$$S_1 \cdot t - 1 = \sum_{i \ne k} \frac{\lambda_k - x}{\lambda_i - x}, \qquad S_2 t^2 - 1 = \sum_{i \ne k} \left(\frac{\lambda_k - x}{\lambda_i - x} \right)^2$$

und wiederum nach der Cauchy-Schwarzschen Ungleichung ist

$$(S_1 t - 1)^2 \le (n-1) \cdot (S_2 t - 1).$$

nachzuweisen. Das bekannteste Beispiel dafür ist die Methode von Picard und Lindelöf zum Beweis der Existenz und Eindeutigkeit der Lösung einer Differentialgleichung, vgl. die *Analysis III-Vorlesung*. Es ist klar, daß bei einer nur gedachten Iteration weder Effizienz noch Rundungsfehler eine Rolle spielen, sondern daß es nur darauf ankommt, ob die im Kopf produzierte Folge von Näherungen konvergiert. Es geht also darum, ein Prinzip anzugeben, mit dem man von möglichst vielen Gedanken-Konstruktionen nachweisen kann, daß sie konvergieren.

Solche hypothetischen Iterationen sind natürlich 1-Schritt-Verfahren

$$\{x^i\}_0^\infty \qquad \text{gemäß} \quad x^i = \phi(x^{i-1}), \quad \forall i \in \mathbb{N},$$

weil jedes m-Schritt-Verfahren darauf zurückführbar ist. Sehr wichtig ist dagegen, daß die x^i im allgemeinen keine bloßen Zahlen oder Vektoren mit n Komponenten sind, sondern Punkte in sehr abstrakten Räumen, im allgemeinen ∞-dimensional, nicht mehr linear und normiert, sondern nur noch mit einer Metrik versehen. Dieses Szenario wird in der Vorlesung *Numerische Mathematik III* zugrundegelegt, hier genügt uns der univariate Fall

$$\phi\colon \mathbb{R} \to \mathbb{R} \qquad (\text{mit } |\cdot| \text{ als Metrik bzw. Norm}).$$

Wir behalten aber den Hochindex für die Iterationsnummer bei.

Dieser 1-dimensionale Fall kommt der Anschauung sehr entgegen, denn hier läßt sich der Verlauf einer Iteration anhand der beiden Graphen von $y = \phi(x)$ und $y = x$ leicht überblicken, indem man den treppenartigen Pfeilen folgt.

Die Iterationsfunktion $\phi(x) := (x-1)^3/16 + 1$, $x \geq 0$, mit Fixpunkten bei $x = 1$ (anziehend) und bei $x = 5$ (abstoßend), kontrahierend für $x < 1 + 4/\sqrt{3}$. Die Iteration folgt den Pfeilen. Sie konvergiert für alle $x^0 < 5$ und divergiert für alle $x^0 > 5$.

dann gilt:

Die Folge konvergiert und der Grenzwert ist in $[a,b]$:

$$\lim_{i\to\infty} x^i = \bar{x} \in [a,b]. \tag{6.5}$$

Der Grenzwert ist Fixpunkt: $\phi(\bar{x}) = \bar{x}$. (6.6)

$$|x^i - \bar{x}| \le \kappa \cdot |x^{i-1} - \bar{x}|, \qquad i = 1,2,\ldots \tag{6.7}$$

$$|x^i - \bar{x}| \le \kappa^i \cdot |x^0 - \bar{x}|, \qquad i = 0,1,2,\ldots \tag{6.8}$$

$$|x^i - \bar{x}| \le \frac{\kappa^i}{1-\kappa}|x^1 - x^0|, \qquad i = 0,1,2,\ldots \tag{6.9}$$

$\bar{x}$ ist der einzige Fixpunkt in $[a,b]$. (6.10)

Beweis: Für $i > 0$: $|x^{i+1} - x^i| = |\phi(x^i) - \phi(x^{i-1})| \le \kappa \cdot |x^i - x^{i-1}|$ nach (6.3), (6.4). Induktion liefert $\le \kappa^2 |x^{i-1} - x^{i-2}| \le \ldots \le \kappa^i \cdot |x^1 - x^0|$. Für $j \ge i \ge 0$:

$$\begin{aligned} |x^{j+1} - x^i| &\le |x^{j+1} - x^j| + \ldots + |x^{i+1} - x^i| \le (\kappa^j + \ldots + \kappa^i)|x^1 - x^0| \\ &\le \frac{\kappa^i}{1-\kappa}|x^1 - x^0|. \end{aligned} \tag{6.11}$$

Also ist $|x^{j+1} - x^i|$ beliebig klein für hinreichend große i und j, die x^i bilden eine Cauchy-Folge und konvergieren somit. Weil $[a,b]$ abgeschlossen ist, enthält es mit der Folge x^i auch deren Grenzwert $\bar{x}$. Damit ist (6.5) bewiesen.

Für jedes i ist

$$\begin{aligned} |\phi(\bar{x}) - \bar{x}| &= |\phi(\bar{x}) - \bar{x} + x^i - \phi(x^{i-1})| \le |\phi(\bar{x}) - \phi(x^{i-1})| + |x^i - \bar{x}| \\ &\le \kappa |x^{i-1} - \bar{x}| + |x^i - \bar{x}|, \qquad \text{wegen (6.3), (6.4) und } \bar{x} \in [a,b]. \end{aligned}$$

$\phi(\bar{x}) \ne \bar{x}$ wäre ein Widerspruch dazu. Damit ist (6.6) bewiesen.

$|x^i - \bar{x}| = |\phi(x^{i-1}) - \phi(\bar{x})| \le \kappa \cdot |x^{i-1} - \bar{x}| \le \ldots \le \kappa^i |x^0 - \bar{x}|$. Das beweist (6.7) und (6.8).

$$|x^i - \bar{x}| \le |x^i - x^j| + |x^j - \bar{x}| \le \frac{\kappa^i}{1-\kappa}|x^1 - x^0| + |x^j - \bar{x}| \qquad \text{nach (6.11) für alle } j > i.$$

$j \to \infty$ verlangt (6.9), sonst Widerspruch.

Sei schließlich $\hat{x} \in [a,b]$ und $\phi(\hat{x}) = \hat{x}$. Dann

$$|\hat{x} - \bar{x}| = |\phi(\hat{x}) - \phi(\bar{x})| \le \kappa \cdot |\hat{x} - \bar{x}| \qquad \text{und} \qquad \hat{x} \ne \bar{x} \text{ wäre Widerspruch.}$$

Das zeigt (6.10). ■

Zunächst wird man das als Minimierungsaufgabe im $\mathbb{R}^n$ ansehen und an die entsprechenden Algorithmen für *beliebiges* $Z\colon \mathbb{R}^n \to \mathbb{R}$ zur Lösung denken. Zum Beispiel sind erste Ableitung (Zeilenvektor) und zweite Ableitung (symmetrische $n \times n$ Matrix)

$$\begin{aligned} \partial Z/\partial x_j &= 2\sum_i f_i \partial f_i/\partial x_j, \\ \partial^2 Z/\partial x_j \partial x_k &= 2\sum_i (\partial f_i/\partial x_j \cdot \partial f_i/\partial x_k + f_i \partial^2 f_i/\partial x_j \partial x_k). \end{aligned}$$

Damit könnte man das Newton-Verfahren zum iterativen Lösen von $Z'(\hat{x}) = 0$ versorgen:

$$Z'(x^i) + Z''(x^i)(x^{i+1} - x^i) = 0.$$

Wie beim Newton-Verfahren besprochen, kann man für $Z''(x^i)$ eine Näherung verwenden. Das kann zwar die Iterationsgeschwindigkeit und/oder die Konvergenz-Ordnung herabsetzen, doch die Endgenauigkeit wird dadurch nicht beeinflußt: Wenn die Methoden mit exaktem oder genähertem Z'' konvergieren, dann immer zu einer Stelle, wo Z' verschwindet bzw. wo Z ein Extremum hat.

Sehr bewährt hat sich das Vernachlässigen der Terme $f_i \partial^2 f_i/\partial x_j \partial x_k$ in Z'', beruhend auf der Vorstellung, daß die f_i irgendwie klein sein werden für sinnvolle Ausgleichsprobleme. Damit entfällt die mühsame und teure Berechnung der mn^2 Größen $\partial^2 f_i/\partial x_j \partial x_k$. Außerdem ist dann die Näherung für Z'' immer positiv-definit und die Rechnung verläuft dadurch oft recht stabil. Jeder Schritt der Iteration ist also ein Übergang $x^i \mapsto x^{i+1}$ gemäß

Gauß-Newton-Schritt:

`Schritt` (a): `Berechne` $b := f(x^i) \in \mathbb{R}^m$ `und` $A := f'(x^i) \in \mathbb{R}^{m,n}$`;`
`Schritt` (b): `Löse` $A^T b + A^T A \cdot v = 0$ `nach` $v \in \mathbb{R}^n$ `auf;`
`Schritt` (c): $x^{i+1} := x^i + v$.

Auf dieselbe Iteration kommt man übrigens, wenn man die beim Newton-Verfahren eingeführte Linearisierung auch hier zur Grundlage der Iteration macht:

$$\begin{aligned} f(x) &= f(x^i) + f'(x^i) \cdot (x - x^i) + \ldots = b + A \cdot (x - x^i) + \ldots \\ \|f(x)\|_2 &= \|b + A \cdot (x - x^i) + \ldots\|_2 \doteq \|b + A \cdot (x - x^i)\|_2. \end{aligned}$$

Obiger Teilschritt (b) entpuppt sich also als die Auflösung der Normalgleichungen zu dem linearisierten Ausgleichsproblem. Nur ist dabei vielleicht nicht ganz so leicht zu erkennen, daß die Linearisierung keinen Fehler im Endresultat bewirkt. Auf jedem Fall kann man im Teilschritt (b) jede Lösungsmethode verwenden, die schon im Abschnitt 3.7 diskutiert wurde, insbesondere die QR-Zerlegung mittels

für den Zusammenhang zwischen Einstell-Größen x und Meßgröße y führt. Hier sind $p = (p_1, \ldots, p_n)^T$ unbekannte Parameter. Fehlerbehaftete Messung und Theorie werden nach der Methode der kleinsten Quadrate versöhnt:

$$\text{Finde } p \in \mathbb{R}^n, \text{ so daß } Z(p) := \sum_{i=1}^{m} \left(\frac{y_i - f(x_i, p)}{\delta y_i} \right)^2 \text{ minimal.}$$

Man nennt das die **Anpassung der Parameter** p nach der Methode der kleinsten Quadrate (engl. *least squares fit*). Mit der Umbenennung

$$\frac{y_i - f(x_i, \cdot)}{\delta y_i} \mapsto f_i(\cdot) \qquad \text{und} \qquad p \mapsto x$$

kommt man auf die eingangs gewählte Notation.

Eine gute Anpassung erfordert nicht nur die Berechnung der Parameter p, sondern auch die der Residuen bzw. der angepaßten Meßwerte $f(x_i, p)$, $i = 1, \ldots, m$ und vor allem die Berechnung der Streuungen δp und δf dieser Größen und ihrer Korrelationen.

schon früher einmal erwähnt, heißt das Maximum der Eigenwert-Beträge **Spektralradius**.

Übungsaufgabe 7.1:

$$c_{n-k} = (-1)^k \sum_{1 \le i_1 < \ldots < i_k \le n} \det(A^{i_1 \ldots i_k}_{i_1 \ldots i_k}).$$

Die $k \times k$ Matrix $A^{i_1 \ldots i_k}_{i_1 \ldots i_k}$ entsteht aus A durch Streichen aller Zeilen und Spalten außer $i_1, \ldots, i_k$. Die entsprechenden Determinanten heißen Diagonal-Minoren. Insbesondere ist

$$\begin{aligned} \lambda_1 + \ldots + \lambda_n &= -c_{n-1} = a_{11} + \ldots + a_{nn} = \mathrm{Spur} A, \\ \lambda_1 \ldots \lambda_n &= (-1)^n c_0 = \det(A). \end{aligned}$$

Eine komplexe $n \times n$ Matrix hat also immer n Eigenwerte (mit der obigen Zählweise), aber nicht immer auch n linear unabhängige Eigenvektoren. Es können nämlich zu einem m-fachen Eigenwert eventuell weniger als m linear unabhängige Eigenvektoren gehören, so daß von der Definition 7.1 her a priori nur soviele Eigenvektoren gesichert sind, wie es *verschiedene* Eigenwerte gibt. Die tatsächliche Anzahl von linear unabhängigen Eigenvektoren zu einem Eigenwert λ (also die Dimension des Nullraumes von $A - \lambda I$) ist dessen **geometrische Multiplizität**. Es wird gleich gezeigt werden, daß immer

$$\text{geometrische Multiplizität} \le \text{algebraische Multiplizität}. \tag{7.1}$$

Zuvor jedoch brauchen wir die **Ähnlichkeits-Transformation**

$$\begin{aligned} & A \mapsto TAT^{-1} \\ \text{bzw.} \quad & A \mapsto TAT^H \qquad \text{falls } T \text{ unitär} \end{aligned}$$

als wichtigstes Hilfsmittel für den Umgang mit Eigenwert-Problemen. Wegen

$$Ax = \lambda x \text{ und } x \neq 0 \qquad \Longleftrightarrow \qquad TAT^{-1}(Tx) = \lambda(Tx) \text{ und } Tx \neq 0$$

hat TAT^{-1} die gleichen Eigenwerte wie A und die gleiche Anzahl von Eigenvektoren, die sich standardmäßig transformieren. Es sei daran erinnert, daß die Matrix A eine lineare Abbildung $x \mapsto y = Ax$ vermittelt und daß eine Transformation aller Punkte $x \mapsto x' = Tx$, $y \mapsto y' = Ty$ diese Abbildung darstellt als $x' \mapsto y' = TAT^{-1}x'$.

Unitäre Transformationen lassen alle Längen (Abstände) und alle Winkel invariant, insbesondere die Winkel zwischen den verschiedenen Eigenvektoren. Weil die Kondition der Eigenwerte durch diese Winkel bestimmt wird, werden unitäre Ähnlichkeitstransformationen in der Numerik bevorzugt.

Solche Transformationen werden vorwiegend eingesetzt, um für eine gegebene lineare Abbildung eine möglichst einfache Matrix-Darstellung zu finden, wobei

höchstens m linear unabhängige Lösungen von $(R-\lambda_i I)y = 0$ bzw. $(A-\lambda_i I)x = 0$. Das beweist die Behauptung (7.1).

Das Schursche Lemma zeigt die große Vereinfachung, die in einer Matrix A durch unitäre Ähnlichkeits-Transformation zu erzielen ist: Jede n-dimensionale lineare Abbildung läßt sich in gestaffelte Form bringen. Noch einfacher ist nur noch die Diagonalform, bei der die n-dimensionale lineare Abbildung zerfällt in lauter 1-dimensionale Abbildungen, d.h. wo in der Schurschen Normalform auch die Elemente oberhalb der Diagonale 0 sind:

$$U^H A U = \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n).$$

Notwendig und hinreichend für diese größtmögliche Vereinfachung einer linearen Abbildung durch unitäre Ähnlichkeits-Transformationen ist die Vertauschbarkeit von A und A^H:

Korollar 7.3 *Unitäre Diagonalisierbarkeit*

$$AA^H = A^H A \qquad \Longleftrightarrow \qquad U^H A U = \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \quad \text{für ein unitäres } U. \tag{7.2}$$

Beweis: „$\Rightarrow$“: Sei $U^H A U = R$, also $RR^H = R^H R$. Betrachte das

1-1-Element: $|r_{11}|^2 + \ldots + |r_{1n}|^2 = |r_{11}|^2 \Longrightarrow r_{12} = \ldots = r_{1n} = 0$
2-2-Element: $|r_{22}|^2 + \ldots + |r_{2n}|^2 = |r_{22}|^2 \Longrightarrow r_{23} = \ldots = r_{2n} = 0$
usw.

„$\Leftarrow$“: $U^H A U = \Lambda \Rightarrow U^H A^H U = \Lambda^H$. $\Lambda\Lambda^H = \Lambda^H\Lambda \Rightarrow AA^H = A^H A$. ■

Eine Matrix A heißt **normal**, wenn sie mit A^H vertauschbar ist, bzw. wenn sie durch eine unitäre Ähnlichkeits-Transformation diagonalisiert werden kann. Wichtige Matrizenklassen gehören dazu, z.B.

- Die reell symmetrischen Matrizen ($A^H = A^T = A$).
- Die reell antisymmetrischen Matrizen ($A^H = A^T = -A$).
- Die reell orthogonalen Matrizen ($A^H = A^T = A^{-1}$).
- Die selbst-adjungierten Matrizen ($A^H = A$).
- Die unitären Matrizen ($A^H = A^{-1}$).

Der Zusammenhang mit dem Eigenwertproblem ist offensichtlich: (7.2) besagt

$$AU = U\Lambda, \qquad \text{d.h.} \qquad Au^k = u^k \lambda_k \quad \text{für } k = 1, \ldots, n,$$

wobei $u^1, \ldots, u^n$ die orthonormierten Spalten von U sind: $(u^k)^H u^l = \delta_{k,l}$, $\forall k, l$. Äquivalente Aussagen sind also

Satz 7.4

Jeder Eigenwert von A liegt in mindestens einer Kreisscheibe. Alle Eigenwerte liegen in der Vereinigung $\bigcup_{j=1}^{n} K_j$ aller Kreisscheiben. Kein Eigenwert liegt außerhalb dieser Vereinigung.

Beweis: Nur die erste Aussage muß bewiesen werden, die beiden übrigen sind nur Reformulierungen. Sei λ ein Eigenwert und x ein zugehöriger Eigenvektor mit betragsgrößter Komponente x_j (Definition von j):

Aus $\sum_{k=1}^{n} a_{jk}x_k = \lambda x_j$ folgt

$$\begin{aligned} \lambda - a_{jj} &= \sum_{\substack{k=1 \\ k \neq j}}^{n} a_{jk}x_k/x_j, \\ |\lambda - a_{jj}| &\leq \sum_{k \neq j} |a_{jk}| \cdot |x_k|/|x_j| \leq \sum_{k \neq j} |a_{jk}|, \end{aligned}$$

also $\lambda \in K_j$. ■

Hinweis: Einschließungssätze geben ein Gebiet an, das mindestens einen Eigenwert enthält, zum Beispiel $\{0\}$ für alle singulären Matrizen. Ausschließungssätze wie der von Gershgorin geben ein Gebiet an, das alle Eigenwerte enthält, bzw. das Komplement davon, das dann keine Eigenwerte enthält.

Ohne Beweis sei noch folgende Verschärfung von Satz 7.4 erwähnt:

Korollar 7.5

Sind m Kreisscheiben disjunkt zu den übrigen, dann enthalten sie zusammen genau m Eigenwerte.

Läßt man in Gedanken alle Außerdiagonal-Elemente stetig nach 0 schrumpfen, so tun das auch die Radien aller Kreise. Die Eigenwerte ändern sich stetig und können dehalb aus den disjunkten Teilgebieten nicht herausspringen. Die Stetigkeit von *einfachen* Eigenwerten folgt sofort aus dem Theorem impliziter Funktionen, bei *mehrfachen* Eigenwert kann man den Satz von Rouché benutzen (*Analysis IV*).

Die große Bedeutung des Satzes von Gershgorin liegt in der Anwendung auf Matrizen, die fast diagonal sind. Sei zum Beispiel A eine $n \times n$ Matrix mit Eigenwerten/-vektoren

$$Ax^j = \lambda_j x^j \qquad \text{für} \quad j = 1, \ldots, n, \quad \text{also } AX = X\Lambda.$$

Sei B eine kleine Störung, d.h. $\|B\|_2 \leq \varepsilon$. Die Eigenwerte $\lambda_j(A+B)$ von $A+B$ sind die von $X^{-1}(A+B)X = \Lambda + C$, wobei alle Elemente $c_{jk} = e^{jT}X^{-1}BXe^k = O(\varepsilon)$. Deshalb ist nach Satz 7.4

$$|\lambda_j(A+B) - \lambda_j - c_{jj}| \leq O(\varepsilon).$$

Der richtige Weg zur Berechnung der Eigenwerte ist also eine Folge von unitären Ähnlichkeits-Transformationen, die die Ausgangs-Matrix immer näher an die Diagonalform bringen. Bekanntlich geht das nicht mit endlich vielen arithmetischen Operationen (einschließlich des Ziehens von k-ten Wurzeln), so daß alle diese Verfahren notwendigerweise Iterationen sein müssen, die anhand eines geeigneten Abbruch-Kriteriums terminieren. Der obige Satz von Gershgorin bietet übrigens dafür eine ausgezeichnete Handhabe; stoppt man die Iteration wenn alle Außerdiagonal-Elemente kleiner als ε geworden sind, dann geben die Diagonal-Elemente die Eigenwerte bis auf einen Fehler $\pm(n-1)\varepsilon$. Der folgende Abschnitt demonstriert das an einem sehr einfachen Fall.

7.3 Das Verfahren von Jacobi

Astronomische Nachrichten **22** (1845) 297–306
Crelle's Journal **30** (1846) 59–94

Es ist das simpelste Verfahren zur Berechnung aller n Eigenwerte einer reell-symmetrischen Matrix A. Der komplexe Fall $A = A^H$ geht analog. Wir führen die Quadratsumme aller Außerdiagonal-Elemente als Maß für die Nicht-Diagonalität von A ein

$$\sigma(A) = \sum_{j \neq k} a_{jk}^2$$

und betrachten eine Iteration, bei der jeder Schritt die Ähnlichkeits-Transformation mit einer ebenen Rotation ist (Abschnitt 2.3):

$$A_0 := A \qquad \text{und} \qquad A_i = Q_{pq}(\varphi)^T A_{i-1} Q_{pq}(\varphi), \quad i = 1, 2, 3, \ldots$$

Die drei Parameter p, q und φ sind wählbar für jedes i. Mit

$$c := \cos\varphi, \quad s := \sin\varphi, \quad t := \tan\varphi = s/c,$$

und

$$A_{i-1} = (a_{jk}), \qquad A_i = (a'_{jk})$$

nach Abschnitt 2.3:

Für alle $k \neq p$ oder q:

$$\left.\begin{aligned} a'_{kp} &:= a_{kp}c - a_{kq}s \\ a'_{kq} &:= a_{kq}s + a_{kq}c \end{aligned}\right\} \qquad a'^2_{kp} + a'^2_{kq} = a^2_{kp} + a^2_{kq},$$

$$\left.\begin{aligned} a'_{pk} &:= ca_{pk} - sa_{qk} \\ a'_{qk} &:= sa_{pk} + ca_{qk} \end{aligned}\right\} \qquad a'^2_{pk} + a'^2_{qk} = a^2_{pk} + a^2_{qk},$$

$$\begin{aligned} a'_{pp} &:= (ca_{pp} - sa_{qp})c - (ca_{pq} - sa_{qq})s, \\ a'_{pq} &:= (ca_{pp} - sa_{qp})s + (ca_{pq} - sa_{qq})c, \\ a'_{qq} &:= (sa_{pp} + ca_{qp})s + (sa_{pq} + ca_{qq})c. \end{aligned}$$

eine andere Vorgehensweise eingebürgert. Man gibt sich einen **Schwellenwert** S vor, geht dann in starrer Reihenfolge durch alle Positionen $1 \leq p < q \leq n$ und annulliert nur solche a_{pq}, deren Betrag S überschreitet. Von Zeit zu Zeit wird der Schwellenwert neu festgesetzt entsprechend der Abnahme der Außerdiagonal-Elemente. Die Konvergenz ist dann nicht mehr so leicht zu beweisen. Außerdem hängt die Effizienz des Prozesses stark ab von der richtigen Wahl des dynamischen Schwellenwertes. Man sollte hier unbedingt das Programm aus einer guten Bibliothek benutzen.

7.4 Einzelne Eigenwerte einer reellen Tridiagonal-Matrix

Jede selbst-adjungierte Matrix $A \in \mathbb{C}^{n,n}$ kann mit *endlich vielen* arithmetischen Operationen durch eine unitäre Ähnlichkeits-Transformation auf reelle, symmetrische Tridiagonalform T gebracht werden,

$$T = \begin{pmatrix} a_1 & b_2 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & b_n \\ 0 & & b_n & a_n \end{pmatrix} \qquad \text{alle} \qquad a_i, b_i \in \mathbb{R}.$$

Das geht zum Beispiel mit $(n-1)(n-2)/2$ ebenen Rotationen oder mit $n-2$ Reflexionen wie vorgeführt in den Anwendungs-Beispielen von Abschnitt 2.4. Nach diesem Vorbereitungsschritt kann man sich o.B.d.A. konzentrieren auf die Eigenwert-Berechnung von reellen, symmetrischen Tridiagonalmatrizen.

Im folgenden wird ein Verfahren beschrieben, das sich durch seine besondere Flexibilität und Vielseitigkeit auszeichnet. Es stammt von W. Givens (1953) und beruht auf einer Anwendung des folgenden Satzes:

Satz 7.6 *Sylvesterscher Trägheitssatz*

Zwei selbst-adjungierte Matrizen A und B haben dann und nur dann gleichviele positive/negative/verschwindende Eigenwerte, wenn

$$A = S^H B S \qquad \text{mit nicht-singulärem } S.$$

(Die **Trägheit** ändert sich nicht bei **Kongruenz-Transformationen**.)

Beweis: Es ist leicht zu sehen, daß man den Satz nur zu beweisen braucht für den Fall $A = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$ und $B = \mathrm{diag}(\beta_1, \ldots, \beta_n)$. Falls $\mathrm{sign}(\alpha_i) = \mathrm{sign}(\beta_i)$, $i = 1, 2, \ldots, n$, dann gibt es ein reelles, diagonales S, so daß $\alpha_i = s_i \beta_i s_i$, $i = 1, \ldots, n$. Das zeigt die eine Richtung. Umgekehrt seien genau $\alpha_1 \ldots, \alpha_p > 0$ und genau $\beta_1, \ldots, \beta_q > 0$. Dann ist $p > q$ unmöglich, weil dann

$$x_{p+1} = \ldots = x_n = (Sx)_1 = \ldots = (Sx)_q = 0$$

Nun ist $T - \lambda I = LR = LDL^T$ wobei $D = \mathrm{diag}(d_1, \ldots, d_n)$ und L die normierte untere Dreiecksmatrix mit den Eliminationsfaktoren ist (nur die erste Subdiagonale $\neq 0$). Dieses L ist nicht-singulär, so daß der Satz von Sylvester anwendbar wird:

$$\begin{aligned} & \text{Anzahl } \nu(\lambda) \text{ der negativen Pivots} \\ = \; & \text{Anzahl der negativen Eigenwerte von } T - \lambda I \\ = \; & \text{Anzahl der Eigenwerte von } T \text{ kleiner als } \lambda. \end{aligned}$$

Damit steht uns ein wohlfeiles Mittel für die Lokalisierung der Eigenwerte zur Verfügung, welches viel Flexibilität hat. Zwei mögliche Anwendungen seien herausgegriffen:

a) Die Häufigkeits-Verteilung $m(i)$ der Eigenwerte einer sehr großen Matrix im Bereich $[a, b]$, unterteilt in die Intervalle $[a + (i-1)h,\ a + ih]$:

$h := (b - a)/N$;
$k := \nu(a)$;
`für` $i := 1$ `bis` N:
    { $j := k$;
      $k := \nu(a + ih)$;
      $m(i) := k - j$. }

Das ist viel billiger als die Berechnung aller Eigenwerte und das anschließende Sortieren in Intervalle der Breite h.

b) Die Berechnung des k-ten Eigenwertes einer Matrix auf eine vorgegebene Genauigkeit $\pm\varepsilon$.

Die Nummer zählt dabei von unten: $\lambda_1 \leq \ldots \leq \lambda_k \leq \ldots$

Man startet mit einem Paar a, b, welches λ_k mit Sicherheit enthält, zum Beispiel nach dem Satz von Gershgorin $a = \min(a_i - \rho_i)$, $b = \max(a_i + \rho_i)$ mit $\rho_1 = |b_2|$, $\rho_n = |b_n|$, $\rho_i = |b_i| + |b_{i+1}|$ sonst. Dann

`solange` $b - a > 2\varepsilon$:
        { $\lambda := (a + b)/2$;
          `falls` $k > \nu(\lambda)$: $a := \lambda$ `sonst`: $b := \lambda$; }
$\lambda := (a + b)/2$,

denn $k \leq \nu \Rightarrow \lambda_k \leq \lambda_\nu < \lambda \Rightarrow \lambda_k \in [a, \lambda]$ und $k \geq \nu + 1 \Rightarrow \lambda_k \geq \lambda_{\nu+1} \geq \lambda \Rightarrow \lambda_k \in [\lambda, b]$. Wenn man mehrere Eigenwerte nacheinander so bestimmt, dann kann man die Information $\nu(\lambda)$ benutzen, um die *obere* Grenze der Einschließungs-Intervalle der weiteren Eigenwerte gegebenenfalls zu reduzieren.

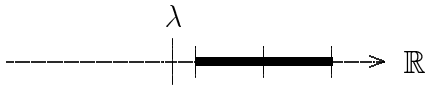
a) $\lambda \notin [\lambda_k - \Delta_k, \lambda_k + \Delta_k]$:

Dann liegen λ_k und $\hat{\lambda}_k$ (der k-te Eigenwert von $\hat{T}$) auf der gleichen Seite von λ und die Bisektion entfernt die *richtige* Intervall-Hälfte, weil die für $\hat{T}$ korrekte Entscheidung auch für T richtig ist.

b) $\lambda \in [\lambda_k - \Delta_k, \lambda_k + \Delta_k]$:

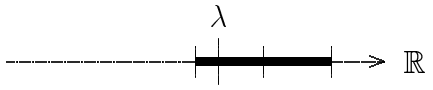
Dann können λ_k und $\hat{\lambda}_k$ auf entgegengesetzten Seiten von λ liegen und die Entscheidung kann falsch sein. Doch welche Hälfte $[a, \lambda]$ oder $[\lambda, b]$ wir auch beibehalten, immer ist λ ein gemeinsamer Punkt von dieser Hälfte und $[\lambda_k - \Delta_k, \lambda_k + \Delta_k]$.

$\hat{\lambda}_k$ liegt irgendwo in dem fett markierten Intervall $[\lambda_k - \Delta_k, \lambda_k + \Delta_k]$:



a) λ außerhalb:

λ_k und $\hat{\lambda}_k$ immer auf der gleichen Seite von λ



b) λ innerhalb:

λ_k und $\hat{\lambda}_k$ möglicherweise auf verschiedenen Seiten von λ

Wir fassen zusammen: Auch unter dem Einfluß von Rundungsfehlern haben das Suchintervall $[a, b]$ und das natürliche Unsicherheits-Intervall $[\lambda_k - \Delta_k, \lambda_k + \Delta_k]$ für den gesuchten Eigenwert λ_k immer einen nicht-leeren Durchschnitt, d.h. in allen Bisektionsstufen gilt:

$$\left|\lambda_k - \frac{a+b}{2}\right| \leq \frac{b-a}{2} + \Delta_k .$$

Ein besseres Resultat ist undenkbar: Die Bisektion ist auch unter dem Einfluß von Rundungsfehlern todsicher und lokalisiert jeden Eigenwert zuverlässig bis auf seine natürliche Unsicherheit.

Es muß noch einmal betont werden, daß dieses günstige Resultat nicht voraussehbar war, sondern nur durch die Rückwärts-Analyse der Rundungsfehler ermöglicht wurde, die übrigens von W. Givens 1953 genau aus diesem Anlaß erfunden worden war.

7.5 Eigenvektoren durch inverse Iteration

Bei einer Tridiagonal-Matrix $T \in \mathbb{R}^{n,n}$ ist folgende Berechnung eines Eigenvektors x zu schon bekanntem Eigenwert λ naheliegend (Methode von Hyman): man wählt als Normierung $x_1 = 1$ und berechnet x_{i+1} aus der i-ten Gleichung von $Tx = \lambda x$:

$$b_i x_{i-1} + (a_i - \lambda) x_i + b_{i+1} x_{i+1} = 0 \qquad \text{gibt} \qquad x_{i+1}, \quad i = 1, \ldots, n-1,$$

Die Theorie der Vektor-Iteration ist für ein orthonormiertes System $\{u^k\}$ von Eigenvektoren (normale Matrizen) sehr simpel:

$$x^0 =: \sum_k u^k \gamma_k \qquad \Longrightarrow \qquad x^i = \sum_k u^k \gamma_k / (\lambda_k - \lambda)^i.$$

Wenn $\lambda \approx \lambda_j$ im Rahmen der Rechengenauigkeit und wenn γ_j nicht zufällig sehr klein ist, dann genügen 1 bis 2 Iterationsschritte, um den Eigenvektor auf Rechengenauigkeit zu erhalten.

Übrigens ist die Hyman-Methode (7.3) interpretierbar als $x^0 = (0, \ldots, 0, 1)^T \mapsto x^1 = x$. Dadurch wird klarer, warum sie in obigem Beispiel versagt: als Vektor-Iteration betrachtet, ist der Startvektor sehr ungünstig.

Die Durchführung von (7.4) erfolgt durch Vorwärts-Rückwärts-Substitution nach vorangegangener (einmaliger) Dreiecks-Zerlegung der Tridiagonal-Matrix $T - \lambda I$ mit Spalten-Pivotsuche:

$$P(T - \lambda I) =: LR.$$

Dabei ist die Permutations-Matrix P das Produkt von $n-1$ möglichen Transpositionen, L hat nur eine nicht-triviale Subdiagonale und R zwei Superdiagonalen. All das ist proportional zu n im Aufwand. Aus (7.4) wird

$$\begin{array}{ll} \text{Vorwärts-Substitution:} & y \text{ aus } Ly = Px^{i-1}, \\ \text{Rückwärts-Substitution:} & x^i \text{ aus } Rx^i = y. \end{array}$$

Sehr bewährt hat sich, die erste Vorwärts-Substitution einzusparen und mit $y = (\pm 1, \ldots, \pm 1)^T$ anzufangen, die Vorzeichen dabei so, daß sich bei der Rückwärts-Substitution möglichst betragsgroße Komponenten ergeben.

Übrigens liefert jede volle Iteration gleichzeitig noch eine Korrektur δ_i für die verwendete Eigenwert-Näherung λ. Diese erhält man durch Minimieren des Residuums von x^i:

$$\|Tx^i - (\lambda + \delta_i) \cdot x^i\|_2 = \|x^{i-1} - x^i \delta_i\|_2 = \min! \qquad \Longleftrightarrow \qquad \delta_i = x^{iT} x^{i-1} / x^{iT} x^i.$$

Dann ist $\lambda + \delta_i = x^{iT} T x^i / x^{iT} x^i$ der Rayleigh-Quotient, der stationär ist für Eigenvektoren und somit nur Fehler zweiter Ordnung besitzt. Das minimierte Residuum $x^{i-1} - x^i \delta_i$ strebt gegen 0 und $\|x^{i-1} - x^i \delta_i\| \leq \varepsilon_{\text{mach}} \|T\| \cdot \|x^i\|$ ist ein bewährtes Abbruch-Kriterium.

Obiges Beispiel in korrekt rundender 3-stelliger Dezimal-Arithmetik liefert für $\lambda = 6.32$

$$\begin{array}{lcl} x^1 \text{ (Halbe Iteration)} & = & (1, 0.318, 0.0524, 0.00637, 0.00100)^T \\ x^2 \text{ (Volle Iteration)} & = & (1, 0.316, 0.0509, 0.00550, 0.000448)^T \\ & & \delta_2 = -0.00244, \text{ also sinnvoll!} \end{array}$$

Seine Analyse gibt Aufschluß über den Anwendungsbereich („performance profile“) des Verfahrens.

Quadraturformeln werden gewonnen durch exakte Integration eines Näherungs-Integranden:

$$\tilde{J}(f) = \int_a^b \tilde{f}(x)\,dx \qquad \text{und} \qquad R(f) = \int_a^b (\tilde{f}(x) - f(x))\,dx$$
$$\text{mit} \qquad \tilde{f}(x) \approx f(x) \quad \forall x \in [a,b].$$

Weil $f(x)$ für $x < a$ und/oder $x > b$ undefiniert sein kann, sollte dort keine Stützstelle sein. Die Gewichte sollten alle positiv sein, damit sich Fehler $\Delta f(x_i)$ in den Funktionswerten möglichst wenig auswirken:

$$|\Delta f(x_i)| \le \varepsilon \qquad \Longrightarrow \qquad \left|\Delta \sum_i g_i f(x_i)\right| \le \varepsilon \sum_i |g_i|.$$

Der kleinstmögliche Wert, den diese Summe annehmen kann, ist

$$\sum |g_i| \ge \sum g_i \approx b - a$$

und dieses Minimum wird nur erreicht, wenn kein Gewicht negativ ist. Also sind unsere Wünsche

$$a \le x_1 < \ldots < x_n \le b \qquad \text{und} \qquad \text{alle } g_i > 0.$$

8.1 Restglieder von elementaren Quadraturformeln

Hier $a = 0$ und $b = h$ geschrieben. Immer $\xi \in [0, h]$.

Rechtecksregel

$$\tilde{J}(f) = hf(h/2).$$

Hier ist $\tilde{f}$ der Polynom-Interpolant vom Grad 0 zu einer Stützstelle in der Intervall-Mitte.

$$R(f) = -\frac{1}{2}\int_0^{h/2} x^2 f''(x)\,dx - \frac{1}{2}\int_{h/2}^{h} (x-h)^2 f''(x)\,dx = -h^3 f''(\xi)/24,$$

also exakt für $\mathbb{P}_1$.

Trapezregel

$$\tilde{J}(f) = \frac{h}{2}(f(0) + f(h)).$$

Trapezsumme

Hier wird das jetzt wieder beliebige Intervall $[a,b]$ zerlegt in n Intervalle der Länge $h=(b-a)/n$. Mit $x_k := a+kh$, $f_k := f(x_k) \quad k=0,\ldots,n$:

$$\tilde{J}(f) = h(\frac{1}{2}f_0 + f_1 + \ldots + f_{n-1} + \frac{1}{2}f_n).$$

$\tilde{f}$ ist das Polygon mit den Ecken x_k, f_k.

$$R(f) = \int_a^b r(x)f''(x)\,dx = nh^3 f''(\xi)/12 = h^2(b-a)f''(\xi)/12.$$

Man skizziere unbedingt die girlandenförmig aus Parabelbögen $x(h-x)/2$ zusammengesetzte Gewichtsfunktion $r(x)$.

Simpson-Summe

Wie zuvor mit geradem n. Die Faßregel wird angewendet auf je zwei benachbarte Teilintervalle:

$$\tilde{J}(f) = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \ldots + 2f_{n-2} + 4f_{n-1} + f_n).$$

Der Interpolant $\tilde{f}$ besteht hier aus Parabelbögen der Weite $2h$.

$$R(f) = \int_a^b r(x)f''''(x)\,dx = nh^5 f''''(\xi)/180 = (b-a)h^4 f''''(\xi)/180.$$

Skizze von $r(x)$. $n=2$ und $h=(b-a)/2$ ist die Faßregel.

Wieder kann man die Simpson-Summe auffassen als Linearkombination der beiden Trapezsummen zur Schrittweite h und $2h$, gemischt im Verhältnis $4:-1$, um den Fehlerterm $O(h^2)$ herauszumitteln.

Genauso kann man zwei Simpson-Summen zur Schrittweite h und $2h$ im Verhältnis $16:-1$ mischen, um den Fehlerterm $O(h^4)$ herauszumitteln.

Im nächsten Abschnitt investieren wir viel Mathematik, um dieses Kombinations-Prinzip systematisch auszubauen und auf die Spitze zu treiben.

8.2 Die Summenformel von Euler-Maclaurin und die asymptotische Entwicklung der Trapezsumme

Notation

Im folgenden ist $f\colon [a,b] \to \mathbb{R}$ eine „hinreichend glatte“ Funktion, d.h. alle auftretenden Ableitungen $f' \equiv Df$, $f'' \equiv D^2 f, \ldots$ sollen stetig sein. Ferner sei vereinbart

$$n \in \{1,2,3,\ldots\} \qquad \text{und} \qquad h := (b-a)/n,$$

Als **Bernoullische Zahlen** bezeichnet man $k! \cdot b_k(0)$ bzw. unter Auslassung der Nullen $(2k)! \cdot b_{2k}(0)$. Ihre *genaue*, rekursive Berechnung ist in Übungsaufgabe 8.4 beschrieben. Mit (8.1) und (8.3) beweist man durch partielle Integration, daß ab $k = 2$

$$\begin{aligned} & \int_0^1 b_{k-1}(t) \cdot D^{k-1} f(t)\, dt = \int_0^1 b_k'(t) \cdot D^{k-1} f(t)\, dt \\ = \;& b_k(0) \cdot \Big(D^{k-1} f(1) - D^{k-1} f(0) \Big) - \int_0^1 b_k(t) \cdot D^k f(t)\, dt. \end{aligned}$$

Der ausintegrierte Bestandteil verschwindet für $k = 3, 5, 7, \ldots$

Mit $k = 1$ beginnend, kann man diese partielle Integration so oft wiederholen als Ableitungen von f existieren; das ergibt die Formel

$$\begin{aligned} & \int_0^1 f(t)\, dt \\ = \;& \frac{1}{2}(f(0) + f(1)) - \sum_{k=1}^{m} b_{2k}(0) \cdot \Big(D^{2k-1} f(1) - D^{2k-1} f(0) \Big) \\ & + \int_0^1 b_{2m}(t) \cdot D^{2m} f(t)\, dt. \end{aligned} \tag{8.4}$$

Die Euler-Maclaurinsche Summenformel

Die Formel (8.4) läßt sich mit der Transformation $t = (x - x_i)/h$ auf alle Intervalle $[x_i, x_{i+1}]$ anwenden (beachte, daß $dx = h \cdot dt$ und $d/dt = h \cdot d/dx$).

Aufsummieren gibt dann die Summenformel von Euler-Maclaurin:

$$\int_a^b f(x)\, dx = h \sum\nolimits'' f_i - R(h) \tag{8.5}$$

mit dem Restglied

$$R(h) = \sum_{k=1}^{m} h^{2k} b_{2k}(0) \cdot \Big(D^{2k-1} f(b) - D^{2k-1} f(a) \Big) - h^{2m} \int_a^b b_{2m}^*(x) \cdot D^{2m} f(x)\, dx. \tag{8.6}$$

Dabei ist

$$b_{2m}^*(x) := b_{2m}\left(\frac{x - x_i}{h} \right) \qquad \text{für} \quad x_i \le x \le x_{i+1}, \quad i = 0, \ldots, n-1.$$

b_{2m}^* ist *kein* Polynom, sondern aus n Polynomstücken girlandenförmig zusammengesetzt. Man beachte, daß sich bei der Summation von (8.4) die Ableitungen $D^{2k-1} f$ an den *inneren* Punkten $x_1, \ldots, x_{n-1}$ wegheben, so daß in (8.6) nur die äußeren Terme für $x_0 = a$ und $x_n = b$ übrigbleiben.

(8.5), (8.6) ist eine in der Praxis öfters verwendete Methode, um *Summen* durch *Integrale* zu approximieren; zum Beispiel bei der Berechnung der **Eulerschen Konstanten** C, dem Grenzwert von $1/1 + 1/2 + 1/3 + \ldots + 1/n - \ln(n)$ für $n \to \infty$.

und den Satz, daß Fourier-Reihen gliedweise integriert werden dürfen, um zu zeigen, daß für $k = 1, 2, 3, \ldots$ und für $0 \leq t \leq 1$ gilt

$$\begin{aligned} b_{2k}(t) &= (-1)^{k+1} 2 \sum_{n=1}^{\infty} \frac{\cos(2\pi n t)}{(2\pi n)^{2k}}, \\ b_{2k+1}(t) &= (-1)^{k+1} 2 \sum_{n=1}^{\infty} \frac{\sin(2\pi n t)}{(2\pi n)^{2k+1}}. \end{aligned}$$

Die rechte Seiten sind periodisch in t, so daß diese Formeln auch für die $b_k^*(t)$ gelten $\forall t \in \mathbb{R}$.

Übungsaufgabe 8.3: Man zeige, daß $ze^{zt}/(e^z - 1)$ die **erzeugende Funktion** der skalierten Bernoulli-Polynome ist, d.h.

$$ze^{zt}/(e^z - 1) = \sum_{k=0}^{\infty} b_k(t) z^k.$$

Diese Potenzreihe konvergiert für alle (komplexen) z mit $|z| < 2\pi$.

Übungsaufgabe 8.4: Man setze $t = 0$ und $z = 2x$ in Übungsaufgabe 8.3 und zeige damit, daß

$$2x/(e^{2x} - 1) = x\cosh x/\sinh x - x = \sum_{k=0}^{\infty} b_k(0) 2^k x^k.$$

Daraus folgere man weiter, daß

$$\cosh x = \frac{\sinh x}{x} \sum_{k=0}^{\infty} b_{2k}(0) \cdot 4^k x^{2k}.$$

Durch Koeffizienten-Vergleich leite man die Rekursion her

$$\begin{aligned} b_0(0) &= 1, \\ b_{2k}(0) + \frac{b_{2k-2}(0)}{4 \cdot 3!} + \frac{b_{2k-4}(0)}{4^2 \cdot 5!} + \ldots + \frac{b_2(0)}{4^{k-1} \cdot (2k-1)!} &= \frac{2k}{4^k \cdot (2k+1)!} \\ &\quad \text{für } k = 1, 2, 3, \ldots \end{aligned}$$

8.3 Konvergenz-Beschleunigung nach Richardson, Romberg-Quadratur

Die Trapezsumme ist die Summe aus exaktem Integral und Restglied. Bezeichnen wir ersteres als τ_0, so gilt also

$$T(h) = \tau_0 + \tau_1 h^2 + \tau_2 h^4 + \ldots + \tau_m(h) \cdot h^{2m}.$$

Konstruiere das Polynom $P(h^2)$, welches die für $h = h_1, \ldots, h_m$ berechneten Trapezsummen interpoliert. Sein Wert an der Stelle $h = 0$ ist die gesuchte Kombination.

Auf Grund dieses Zusammenhangs spricht man auch von der **Extrapolationsmethode** oder von **Extrapolation auf** 0. Insgesamt gilt:

Satz 8.1

Es sei $f \in C^{2m}[a,b]$ und $P \in \mathbb{P}_{m-1}$, so daß $P(h_k^2) = T(h_k)$ für $k = 1, \ldots, m$. Dann ist

$$\left|P(0) - \int_a^b f(x)\,dx\right| = O(h_1^2 \cdots h_m^2).$$

Beweis: Die Polynome h^{2k}, $k = 0, \ldots, m-1$ interpolieren sich selbst, so daß nach der Lagrangeschen Interpolations-Formel gilt

$$\sum_{i=1}^{m} h_i^{2k} L_i(h^2) = h^{2k}, \qquad k = 0, \ldots, m-1.$$

Dabei ist

$$L_i(h^2) = \prod_{\substack{k=1 \\ k \neq i}}^{m} \frac{h^2 - h_k^2}{h_i^2 - h_k^2}, \qquad i = 1, \ldots, m.$$

Deshalb ist nach (8.7)

$$P(0) = \sum_i T(h_i) L_i(0) = \tau_0 + \sum_i \tau_m(h_i) h_i^{2m} L_i(0),$$

so daß

$$|P(0) - \tau_0| \leq M c_m h_1^2 \cdots h_m^2,$$

mit $|\tau_m(h)| \leq M$

$$c_m := \sum_{i=1}^{m} \prod_{\substack{k=1 \\ k \neq i}}^{m} \frac{h_i^2}{|h_i^2 - h_k^2|}.$$

c_m nimmt zu mit wachsendem m und hängt ab von den Verhältnissen $h_i^2 : h_k^2$. Für die klassische Folge $h_k = (b-a)/2^k$ gilt $c_\infty \leq 1.97$ und für die heute bevorzugte Folge von R. Bulirsch (s.u.) gilt $c_\infty \leq 7.76$. ■

Die technische Durchführung der Berechnung von $P(0)$ als Näherung für τ_0 geschieht am bequemsten mit dem Aitken-Neville-Schema wie besprochen in Abschnitt 4.2. Ersetze dort n durch m, x durch 0, x_i durch h_i^2, x_k durch h_k^2 und y_k durch $T(h_k)$:

Begründung: Möglichst hohe Fehler-Ordnung! Möglichst schnelle Konvergenz bei Anwendung auf das Intervall der Länge h statt auf $[-1,+1]$.

Der Grad $2n$ ist gewiß nicht erreichbar, denn für jede solche Methode wäre das Polynom $p(x) = (x-x_1)^2 \cdots (x-x_n)^2$ ein Gegenbeispiel:

Linke Seite von (8.11) wäre > 0, rechte Seite wäre $= 0$.

Also kann man (8.11) konkretisieren:

Stützstellen $x_1, \ldots, x_n$ und Gewichte $g_1, \ldots, g_n$ so, daß

$$\int_{-1}^{+1} p(x)\,dx = \sum_{i=1}^{n} g_i p(x_i) \quad \forall p \in \mathbb{P}_{2n-1}. \tag{8.12}$$

Notwendige Bedingung für die Stützstellen $x_1, \ldots, x_n$**:**

$$\int_{-1}^{+1} q(x) \cdot \underbrace{(x-x_1)\cdots(x-x_n)}_{=:\,\omega_n(x)}\,dx = 0 \quad \forall q \in \mathbb{P}_{n-1}, \tag{8.13}$$

denn alle solchen Integranden sind in (8.12) zugelassen und machen dort die rechte Seite zu 0. *Eine* Lösung $\omega_n(x)$ bzw. *ein* Satz von Stützstellen $x_1, \ldots, x_n$ ist leicht anzugeben, nämlich

$$\omega_n(x) := \frac{n!}{(2n)!} D^n[(x^2-1)^n] = 1 \cdot x^n + \ldots \tag{8.14}$$

$(x^2-1)^n = (x+1)^n(x-1)^n$ hat eine n-fache Nullstelle bei $x=-1$ und $x=+1$, so daß seine k-te Ableitung dort $(n-k)$-fache Nullstellen hat. Deshalb verschwinden beim Einsetzen dieses $\omega_n(x)$ in (8.13) und bei n-facher partieller Integration alle ausintegrierten Bestandteile: (8.13) wird offensichtlich von *diesem* $\omega_n(x)$ erfüllt. Eine *andere* Lösung $\tilde{\omega}_n(x)$ mit *anderen* Nullstellen $\tilde{x}_1, \ldots, \tilde{x}_n$ kann es nicht geben, denn dann wäre $\omega_n(x) - \tilde{\omega}_n(x) \in \mathbb{P}_{n-1}$ und somit

$$\int_{-1}^{+1} (\omega_n - \tilde{\omega}_n)^2\,dx = \int_{-1}^{+1} (\omega_n - \tilde{\omega}_n)\omega_n\,dx - \int_{-1}^{+1} (\omega_n - \tilde{\omega}_n)\tilde{\omega}_n\,dx = 0 - 0 = 0,$$

also $\omega_n(x) - \tilde{\omega}_n(x) = 0 \quad \forall x \in [-1,+1]$.

Damit ist der nicht-lineare und somit schwierigste Teil der Arbeit fast schon erledigt: die Stützstellen sind auf einen einzigen möglichen Satz eingeengt, nämlich die Nullstellen der **Legendre-Polynome vom Grad** n.

Das gefundene ω_n muß im Inneren des Intervalls $[-1,+1]$ mindestens n-mal sein Vorzeichen wechseln, sonst könnte man sofort ein $q \in \mathbb{P}_{n-1}$ angeben, das überall gleiches Vorzeichen wie ω_n hat und damit (8.13) widerlegt. Wir folgern: Die n Nullstellen von (8.14) sind alle einfach und liegen im Inneren von $[-1,+1]$:

$$\text{Stützstellen} \quad -1 < x_1 < \ldots < x_n < +1. \tag{8.15}$$

$x_1, \ldots, x_n$:

$$\begin{aligned}\int_{-1}^{+1} f(x)\,dx - \sum_{i=1}^{n} g_i f(x_i) &= \int_{-1}^{+1} f(x)\,dx - \sum_{i=1}^{n} g_i P(x_i) \\ &= \int_{-1}^{+1} f(x)\,dx - \int_{-1}^{+1} P(x)\,dx \\ &\quad \text{nach (8.12).}\end{aligned}$$

Nun ist nach der Restgliedformel für Interpolation (Satz 4.9)

$$f(x) - P(x) = (x - x_1)^2 \cdots (x - x_n)^2 D^{2n} f(\xi)/(2n)! \begin{cases} \leq \omega_n(x)^2 M/(2n)! \\ \geq \omega_n(x)^2 m/(2n)! \end{cases}$$

mit $M := \max D^{2n} f$ und $m := \min D^{2n} f$. Also

$$m \cdot \int_{-1}^{+1} \omega_n(x)^2\,dx/(2n)! \leq \int_{-1}^{+1} f(x)\,dx - \sum_{i=1}^{n} g_i f(x_i) \leq M \int_{-1}^{+1} \omega_n(x)^2\,dx/(2n)!$$

oder

$$\int_{-1}^{+1} f(x)\,dx - \sum_{i=1}^{n} g_i f(x_i) = D^{2n} f(\xi)/(2n)! \cdot \int_{-1}^{+1} \omega_n(x)^2\,dx.$$

Das letzte Integral hat den Wert $2^{2n+1}(n!)^4/((2n)!(2n+1)!)$.

e) Eine geschlossene Lösung läßt sich nur in den einfachsten Fällen angeben, dagegen fast nie in den praktisch relevanten Fällen.

Auch das folgende Lemma gehört in die Theorie. Es zeigt, wie weit zwei benachbarte Kurven der Lösungsschar auseinanderdriften können.

Lemma 9.1 *Zur Kondition des AWP*

Aus (9.4) folgt für zwei Lösungen y, z von (9.1) und alle x, x_0:

$$\|y(x) - z(x)\| \leq \|y(x_0) - z(x_0)\| \cdot e^{L \cdot |x - x_0|}. \tag{9.5}$$

Beweis: Zunächst sei $x_0 < x$. $D(x) := \|y(x) - z(x)\|$ ist eine stetige, skalare Funktion, $E(x) := e^{-Lx} \int_{x_0}^{x} D(t)\, dt$ ist dann stetig differenzierbar. Also gilt

$$D(x) = \Big(e^{Lx} E(x)\Big)' = L e^{Lx} E(x) + e^{Lx} E'(x). \tag{9.6}$$

Aus (9.3) für $y(x)$ und $z(x)$, aus der Dreiecks-Ungleichung und aus (9.4) folgt

$$D(x) \leq \|y_0 - z_0\| + \int_{x_0}^{x} \Big\| f(t, y(t)) - f(t, z(t)) \Big\|\, dt \leq \|y_0 - z_0\| + \int_{x_0}^{x} L D(t)\, dt.$$

Man setzt links (9.6) ein und benutzt rechts die Definition von $E(x)$:

$$\begin{aligned} L e^{Lx} E(x) + e^{Lx} E'(x) &\leq \|y_0 - z_0\| + L e^{Lx} E(x), \\ E'(x) &\leq \|y_0 - z_0\| e^{-Lx}, \\ E(x) = \int_{x_0}^{x} E'(t)\, dt &\leq \|y_0 - z_0\| (e^{-Lx_0} - e^{-Lx})/L. \end{aligned}$$

(Beachte, daß nach Definition $E(x_0) = 0$.) Diese beiden Abschätzungen rechts in (9.6) eingesetzt, ergeben

$$D(x) \leq \|y_0 - z_0\| \cdot (e^{L(x - x_0)} - 1 + 1).$$

Bei $x < x_0$ verwendet man $-x$ als unabhängige Variable und $-f$ als rechte Seite der zugehörigen DGL. ■

Bemerkung 1: Im Beweis wurde nicht benützt, daß die Lipschitz-Bedingung (9.4) für alle y, z gelten muß, sondern nur für das spezielle Paar $y(x), z(x)$. Beispiel $f(x, y) = y^2$.

Bemerkung 2: Zwei Lösungen können auch nicht einander beliebig nahe kommen wegen

$$\|y(x) - z(x)\| \geq \|y_0 - z_0\| e^{-L|x - x_0|}.$$

(Vertausche x und x_0 in (9.5).)

- **Heun-Verfahren** (E1):

$$\begin{aligned} \Delta_1 &:= hf_{k-1}, \\ \Delta_2 &:= hf(x_k, u_{k-1} + \Delta_1), \\ u_k &:= u_{k-1} + (\Delta_1 + \Delta_2)/2. \end{aligned}$$

- **modifiziertes Euler-Verfahren** (E1):

$$\begin{aligned} \Delta_1 &:= hf_{k-1}, \\ \Delta_2 &:= hf(x_{k-1} + h/2, u_{k-1} + \Delta_1/2), \\ u_k &:= u_{k-1} + \Delta_2. \end{aligned}$$

- **Mittelpunkts-Regel** (E2):

$$u_{k+1} := u_{k-1} + 2hf_k \qquad \text{ab} \quad k = 1.$$

- **Milne-Verfahren** (I2):

$$u_{k+1} := u_{k-1} + \frac{h}{3}(f_{k-1} + 4f_k + f_{k+1}) \qquad \text{ab} \quad k = 1.$$

Nach diesen simplen Beispielen werden noch wichtige Familien von Verfahren vorgestellt.

- Die **backward difference formulas** (BDF), $s = 1, 2, 3, \ldots$ (Typ Is).

  Sei $P(x) := u_{k-s}L_0(x) + \ldots + u_kL_s(x)$ der Polynom-Interpolant vom Grad s zu den $s+1$ Stützpunkten $x_{k-s}, u_{k-s}, \ldots, x_k, u_k$ (Lagrangesche Interpolations-Formel). u_k ergibt sich aus der naheliegenden Forderung

$$P'(x_k) = f(x_k, u_k).$$

- Die **Adams-Moulton-Verfahren**, $s = 0, 1, 2, \ldots$ (Typ Is für $s > 0$).

  Hier wird die analoge Polynom-Näherung für die Ableitung angesetzt: $Q(x) := u'_{k-s}L_0(x) + \ldots + u'_kL_s(x)$ und u_k berechnet aus

$$u_k = u_{k-1} + \int_{x_{k-1}}^{x_k} Q(x)\,dx.$$

- Die **Adams-Bashforth-Verfahren**, $s = 1, 2, 3, \ldots$ (Typ Es).

  Wie zuvor, jedoch ohne die jüngste Stützstelle: $Q(x) = u'_{k-s}L_0(x) + \ldots + u'_{k-1}L_{s-1}(x)$ ist dann nur vom Grad $s-1$.

  Wenn man die Näherung Q über zwei Intervalle integriert, erhält man die Verfahrens-Familien von **Milne** bzw. **Nyström**:

$$u_k = u_{k-2} + \int_{x_{k-2}}^{x_k} Q(x)\,dx.$$

- bei einem **impliziten 1-Schritt-Verfahren** (I1): u_{k-1} und u_k,
- bei einem **impliziten s-Schritt-Verfahren** (Is): $u_{k-s}, \ldots, u_{k-1}$ und u_k.

Die expliziten Verfahren berechnen also u_k nach einer expliziten Formel, während die impliziten Verfahren eine im allgemeinen nicht-lineare Gleichung nach u_k auflösen müssen. Das erfolgt in der Regel mit einer Fixpunkt-Iteration (siehe Kapitel 6):

$$\begin{array}{ll} \text{Start:} & u_k^{(0)} := \text{nach einem expliziten Verfahren,} \\ & (\textbf{Prädiktor-Schritt}), \\ \text{für } i = 1, 2, \ldots: & u_k^{(i)} := u_{k-1} + h\phi_f(h, \ldots, u_k^{(i-1)}), \\ & (\textbf{Korrektor-Schritte}). \end{array}$$

Bei Lipschitz-stetigem ϕ_f und genügend kleiner Schrittweite h ist die zugehörige Iterationsfunktion kontrahierend und die Iteration wird somit konvergieren: $u_k^{(i)} \to u_k$ für $i \to \infty$. Bei einer *festen* Zahl von Korrektor-Schritten (in der Praxis stets 1 oder 2) wird übrigens aus einem impliziten Verfahren ein explizites in obigem Klassifikations-Schema.

Noch eine Anmerkung zu den impliziten Verfahren. Dort kann man das ϕ_f entweder direkt angeben (zum Beispiel im vorangegangenen Abschnitt beim inversen Euler-Verfahren, der Trapezregel oder dem Milne-Verfahren) oder man gibt nur die Idee vor (wie zum Beispiel bei den BDF- oder den Adams-Moulton-Verfahren). Dann bestimmt allein die Form von (9.7) noch nicht eindeutig das zugehörige ϕ_f. Erst die Umsetzung der Idee in auszuführende Formeln produziert das zugrundeliegende ϕ_f. Mißverständnisse können dabei kaum auftreten.

Unter dem **globalen Diskretisierungsfehler** (an der Stelle $x \neq x_0$) versteht man

$$e(h, x) := u_n(h) - y(x), \qquad \text{wobei} \quad nh = x - x_0,$$

also den Unterschied zwischen der diskreten Näherung und dem exakten Wert an einer festen Stelle x. Dies ist das unmittelbare Maß für die Genauigkeit eines Verfahrens, also der Fehler in der natürlichen Anschauung. $e(h, x)$ bzw. realistische Schranken dafür werden den Anwender interessieren. Die Art der h-Abhängigkeit interessiert den Numeriker, insbesondere das asymptotische Verhalten von $e(h, x)$ für $h \to 0$ bei festem x (also gleichzeitig $n \to \infty$). Dabei ist zu beachten:

Wenn man x als *kontinuierliche* Variable betrachtet, dann ist h eine *diskrete* Variable mit Definitionsbereich $(x - x_0)/n$ für $n \in \mathbb{N}$. Dann ist $\partial e/\partial x$ durchaus sinnvoll, nicht dagegen $\partial e/\partial h, \partial^2 e/\partial h^2, \ldots$ oder eine Taylorreihe in h. Jedoch kann $e(h, x)$ sehr wohl eine asymptotische Reihe in h besitzen.

Das Verfahren ϕ_f heißt **konvergent** auf $[a, b]$, wenn

$$h \to 0 \qquad \Longrightarrow \qquad e(h, x) \to 0$$

Das Verfahren hat die **Konsistenz-Ordnung** p, wenn $\sigma(h) = O(h^p)$.

Es ist leicht zu zeigen, daß ein Verfahren konsistent ist genau dann, wenn $\phi_f(h,z) \to f(x,z)$ für $h \to 0$. Konsistenz ist das bare Minimum für ein Verfahren.

9.3 Explizite 1-Schritt-Verfahren

Die expliziten 1-Schritt-Verfahren wie zum Beispiel das klassische, vierstufige Runge-Kutta-Verfahren sind

- robust und zuverlässig,
- schnell zu programmieren,
- besonders geeignet für variable Schrittweite,
- sehr leicht zu analysieren.

Bei den expliziten 1-Schritt-Verfahren gibt es nämlich zwischen dem lokalen und dem globalen Diskretisierungsfehler einen universellen Zusammenhang

Konsistenz	$\Longleftrightarrow$	**Konvergenz**
Konsistenz-Ordnung	$=$	**Konvergenz-Ordnung**

Satz 9.2

Sei $y(\cdot)$ die Lösung des AWP $y'(x) = f(x, y(x))$, $y(x_0) = y_0$ und
$u_0 := y_0$, $u_k := u_{k-1} + h_k \cdot \phi_f(h_k, u_{k-1})$, $h_k := x_k - x_{k-1}$, $k = 1, 2, \ldots$

$$\begin{aligned} \|f(x,y) - f(x,z)\| &\le L \cdot \|y - z\| \quad \text{und} \quad \|\tau(h,\xi,\eta)\| \le \sigma(h) \quad \Longrightarrow \\ \|y(x_n) - u_n\| &\le \sigma(h_{\max}) \cdot \Big(\exp(L \cdot |x_n - x_0|) - 1\Big)/L \quad \text{für alle} \quad n. \end{aligned}$$

Beweis: O.B.d.A. sei $x_0 < x_n$ und alle Schrittweiten h_k positiv. Wir führen die Lösungen $z_k(x)$ der DGL durch die Punkte x_k, u_k ein, so daß also $z_k(x_k) = u_k$, $k = 0, \ldots, n$. Insbesondere ist $z_0(x) = y(x)$. Weiter ist wie zuvor erläutert

$$h_k \tau(h_k, x_{k-1}, u_{k-1}) = z_{k-1}(x_k) - u_k = z_{k-1}(x_k) - z_k(x_k).$$

Daraus

$$\|z_{k-1}(x_k) - z_k(x_k)\| \le h_k \sigma(h_k) \le h_k \sigma(h_{\max}). \tag{9.9}$$

Jetzt benutzen wir Lemma 9.1, welches angibt, wie weit die benachbarten Lösungen z_{k-1} und z_k längs der Strecke $x_n - x_k$ auseinanderdriften können:

$$\|z_{k-1}(x_n) - z_k(x_n)\| \le \|z_{k-1}(x_k) - z_k(x_k)\| \cdot \exp(L \cdot |x_n - x_k|). \tag{9.10}$$

N	Euler-Verf. $h = 2/N$	Heun-Verf. $h = 4/N$	Runge-Kutta-Verf. $h = 8/N$
4	−0.9812	−0.2433 1354	0.1755 6871 99278
8	−0.5676	−0.0792 5875	0.0229 1221 36473
16	−0.3098	−0.0219 2620	0.0021 6896 12996
32	−0.1626	−0.0056 4107	0.0001 7903 63350
64	−0.0834	−0.0014 1657	0.0000 1369 86430
128	−0.0422	−0.0003 5365	0.0000 0100 30984
256	−0.0213	−0.0000 8825	0.0000 0007 15001
512	−0.0107	−0.0000 2203	0.0000 0000 50061
1024	−0.0053	−0.0000 0550	0.0000 0000 03460
2048	−0.0027	−0.0000 0138	0.0000 0000 00237
4096	−0.0013	−0.0000 0034	0.0000 0000 00016
Abnahme:	1/2	1/4	1/16

In jeder Zeile stehen die Ergebnise für vergleichbaren Aufwand, nämlich für N Funktions-Auswertungen von f. In jeder Spalte manifestiert sich die Konvergenz-Ordnung. Gut zum Ausdruck kommt, wie sich der Rechenaufwand verringert mit wachsender Konvergenz-Ordnung: Für $\Delta y = 10^{-6}$ genügen beim Runge-Kutta-Verfahren 128 Funktions-Auswertungen, während das Euler-Verfahren ca. 1 Million benötigen würde.

Zum Schluß noch ein Hinweis zu den impliziten 1-Schritt-Verfahren,

$$(u_k - u_{k-1})/h_k = \phi_f(h_k, u_{k-1}, u_k).$$

Satz 9.2 läßt sich auf diesen Fall erweitern, wenn ϕ_f in u_k Lipschitz-stetig ist mit einer von h unabhängigen Lipschitz-Konstanten.

9.4 s-Schritt-Verfahren

Sie wurden eingeführt in Abschnitt 9.1 und berechnen u_k aus den Näherungen $u_{k-s}, \ldots, u_{k-1}$ an den s vorangegangenen Gitterpunkten. Würde man s starr wählen, dann müßte man in einer sogenannten **Anlaufrechnung** $u_1, \ldots, u_{s-1}$ gesondert ermitteln. Moderne Software hält jedoch das s genauso wie die Schrittweite variabel und paßt beide automatisch an die lokale Situation an. Solche Software startet mit $s = 1$ und benötigt somit keine gesonderte Anlaufrechnung. Die in Abschnitt 9.1 schon eingeführten Verfahrens-Klassen (BDF, Adams und Moulton, Adams und Bashforth, Milne) haben s als Parameter und machen damit Erhöhung oder Erniedrigung von s leicht.

Der größte Vorteil der s-Schritt-Verfahren ist die Steigerung der Konsistenz-Ordnung ohne eine damit einhergehende Vermehrung der Funktions-Auswertungen wie bei den 1-Schritt-Verfahren. Dort lotet man die rechte Seite f im Integrations-Intervall $[x_{k-1}, x_k]$ mehrfach aus, während man bei den s-Schritt-Verfahren billiger zum Ziel zu kommen versucht, indem man sich auf die schon vorhandene

so simpel wie in Satz 9.2. Um die Analyse anzudeuten, schreiben wir die beiden Verfahrens-Gleichungen (9.11) und (9.12) in der einheitlichen Form

$$\sum_{i=0}^{s} \alpha_i u_i = \sum_{i=0}^{s} h\beta_i u'_i \tag{9.13}$$

mit $h = x_s - x_{s-1}$, Normierung $\beta_0 + \ldots + \beta_s = 1$ und $\alpha_s \neq 0$.

Auch alle übrigen im Abschnitt 9.1 kurz erwähnten s-Schritt-Verfahren lassen sich auf diese Form bringen. Alle solchen Verfahren heißen **lineare s-Schritt-Verfahren**, obwohl $u'_k := f(x_k, u_k)$ im allgemeinen kein linearer Zusammenhang ist. Bei $\beta_s = 0$ ist das Verfahren explizit, bei $\beta_s \neq 0$ ist es implizit. Bei äquidistantem Gitter gilt (9.13) auch für den Schritt $u_{k-s}, \ldots, u_{k-1} \mapsto u_k$. Nur diesen äquidistanten Fall kann man analysieren.

Es ist ziemlich leicht einzusehen, daß die homogene Rekursion

$$\sum_{i=0}^{s} \alpha_i \zeta_{i+k} = 0 \qquad \text{für} \quad k = 0, 1, 2, \ldots \tag{9.14}$$

keine anschwellenden Lösungen $\{\zeta_i\}$ besitzen darf, wenn das Verfahren global konvergieren soll beim Grenzübergang $h \to 0$. Die Umkehrung gilt auch, aber der Beweis ist mühsam. Diese grundsätzliche Bedingung läßt sich mit Hilfe der recht einfachen Theorie linearer, homogener Rekursionen (streng analog zu der Theorie linearer DGL'en mit konstanten Koeffizienten) ummünzen zur folgenden

Definition 9.3 *Stabilitäts-Bedingung*

Das charakteristische Polynom $\psi(\zeta) := \alpha_0 + \alpha_1\zeta + \ldots + \alpha_s\zeta^s$ darf nur Nullstellen im Inneren oder auf dem Rand der Einheits-Kreisscheibe haben. Solche auf dem Rand müssen einfach sein:

$$\begin{aligned} \psi(\zeta) = 0 \qquad &\Longrightarrow \qquad |\zeta| \leq 1, \\ \psi(\zeta) = \psi'(\zeta) = 0 \qquad &\Longrightarrow \qquad |\zeta| < 1. \end{aligned}$$

Also: Genau dann wenn die Stabilitätsbedingung gilt, hat die homogene Rekursion (9.14) keine anschwellenden Lösungen und genau dann folgt aus der Konsistenz der Ordnung p die Konvergenz der Ordnung p.

Zum Beispiel ist bei den Adams-Moulton-Verfahren $\psi(\zeta) = \zeta^s - \zeta^{s-1}$, also $\zeta = 1$ (einfach) und $\zeta = 0$ ($(s-1)$-fach): die Stabilitäts-Bedingung ist erfüllt und das Verfahren hat Konvergenz-Ordnung $s + 1$.

Bei den BDF-Verfahren muß man die Nullstellen numerisch berechnen. Es zeigt sich, daß nur für $s = 1$ bis 6 die Stabilitäts-Bedingung erfüllt ist, so daß nur solche s in Betracht kommen.

Ohne Beweis sei noch angegeben

x_1 des Integrations-Intervalles. Bezüglich $[x_0, x_1]$ ist die Extrapolations-Methode ein explizites 1-Schritt-Verfahren, das man übrigens in die Runge-Kutta-Klasse einordnen kann.

Diese Methode hat sich besonders gut bewährt bei Anwendungen, die höchste Genauigkeits-Anforderungen stellen. Allerdings kommt die hohe Konsistenz-Ordnung nur zustande für ein genügend glattes f; eine genügend glatte Lösung y allein genügt nicht.

Zum Beleg betrachten wir nochmal das Beispiel aus Abschnitt 9.3,

$$y'(x) = \sqrt{x^2 + y(x)^2} \qquad \text{auf} \quad [0, 2]$$

mit entweder $y(0) = 1$ oder $y(0) = 0$. In beiden Fällen ist die Lösung beliebig oft differenzierbar. Die folgende Tabelle zeigt den globalen Diskretisierungsfehler am rechten Endpunkt $x = 2$. In der ersten Zeile stehen die Fehler der $\hat{u}_n(h)$, $h = 2/n$. In den Zeilen darunter stehen die Fehler der extrapolierten Werte. Für die AB $y(0) = 1$ entspricht die Fehler-Abnahme genau der Theorie von Abschnitt 8.3. Bei der AB $y(0) = 0$ dagegen gilt das nur für die ersten beiden Zeilen, darunter stagniert der Fehler und geht für $n = 128$ (benötigt 438 Funktions-Auswertungen) kaum unter $3 \cdot 10^{-9}$. Das ist nicht viel besser als das klassische Runge-Kutta-Verfahren, welches mit Schrittweite $h = 1/55$ (440 Funktions-Auswertungen) den Fehler $9 \cdot 10^{-9}$ produziert.

Der globale Diskretisierungsfehler bei der Mittelpunktsregel mit Extrapolation

Beispiel: $y'(x) = \sqrt{x^2 + y(x)^2}$ für $0 < x < 2$.