# Angewandte Mathematik (Lehramt)

**Wintersemester 2015/2016**

Universität Bayreuth

Michael Stoll

Inhaltsverzeichnis



Druckversion vom 21. April 2017, 13:33 Uhr.

# 1. Einführung

In der LPO I, die in Bayern die Ausbildung der Lehrer an staatlichen Schulen regelt, sind für das Lehramt an Gymnasien im Fach Mathematik 8 Leistungspunkte „Angewandte Mathematik“ vorgesehen:

---

**LPO I, §73**
**Mathematik**

(1) Fachliche Zulassungsvoraussetzungen

Nachweis von

1. mindestens 23 Leistungspunkten aus dem Gebiet Analysis (Differential- und Integralrechnung im $\mathbb{R}^n$, Gewöhnliche Differentialgleichungen, Funktionentheorie),
2. mindestens 23 Leistungspunkten aus dem Gebiet Lineare Algebra, Algebra und Elemente der Zahlentheorie,
3. mindestens 8 Leistungspunkten aus dem Gebiet Stochastik,
4. mindestens 8 Leistungspunkten aus dem Gebiet Geometrie,
5. mindestens 8 Leistungspunkten aus einem Gebiet der Angewandten Mathematik (z.B. Computeralgebra, Algorithmische Geometrie, Diskrete Mathematik, Optimierung, Numerik),
6. mindestens 8 Leistungspunkten aus der Fachdidaktik.

---

Um die nötigen Kenntnisse bereitzustellen, die für die erfolgreiche Teilnahme an den Staatsexamensklausuren in Analysis und Algebra gebraucht werden, reichen die genannten 23 Leistungspunkte nicht aus. Tatsächlich sind es in Bayreuth 36 Leistungspunkte in Analysis (Analysis I und II, Einführung in die gewöhnlichen Differentialgleichungen, Einführung in die und Vertiefung der Funktionentheorie) und 34 Leistungspunkte in Algebra (Lineare Algebra I und II, Einführung in die Zahlentheorie und algebraische Strukturen, Einführung in die Algebra). Mit den 24 Leistungspunkten aus Stochastik, Geometrie und Angewandter Mathematik sind die eigentlich vorgesehenen 92 Leistungspunkte aus dem fachwissenschaftlichen Bereich (LPO I, §22 Abs. 3) schon knapp überschritten. Das bedeutet, dass für mehr „Angewandte Mathematik“ kein Platz ist. Das ist bedauerlich, denn so ist das Lehramtstudium sehr „Reine-Mathematik-lastig“; außerdem bekommen Sie zwar eine solide Grundausbildung in vielen Bereichen der (hauptsächlich reinen) Mathematik, haben aber im Gegensatz zu Ihren Kommilitoninnen und Kommilitonen in den mathematischen Fachstudiengängen nicht die Gelegenheit, in Vertiefungs- und Spezialvorlesungen zu erleben, wo und wie diese Grundlagen dann Verwendung finden. Das hinterlässt dann gerne das unbefriedigende Gefühl, dass Sie nicht so recht wissen, wofür diese ganze Mathematik denn nun eigentlich gut ist (für den Schulunterricht in den meisten Fällen offenbar nicht). Das kann natürlich eine einzelne Vorlesung im „Wert“ von 8 Leistungspunkten nicht reparieren. Wir versuchen aber, das Beste daraus zu machen, indem wir Ihnen einen Einblick in drei verschiedene der in der LPO genannten Bereiche der Angewandten Mathematik geben, nämlich Numerik, Optimierung und Computeralgebra. Dabei wird nicht an der Tiefe gespart, wohl aber (notwendigerweise) an der Breite: Es wird jeweils eine typische Fragestellung im Detail behandelt.

Mathematik anzuwenden bedeutet heutzutage in den meisten Fällen, dass man sie in Verfahren umsetzt, die von Computern abgearbeitet werden können. Angewandte Mathematik hat also eine starke algorithmische Komponente und beinhaltet das Schreiben von Computerprogrammen. Bevor man also ernsthaft (z.B.) Numerik, Optimierung oder Computeralgebra betreiben kann, sind einige Grundkenntnisse im Programmieren erforderlich. Wenn Sie solche Grundkenntnisse bereits besitzen und sich für eines der drei genannten Gebiete besonders interessieren, dann haben Sie die Möglichkeit, statt dieser Vorlesung wahlweise auch die entsprechende Einführungsvorlesung zu hören. Ansonsten werden wir uns bemühen, Ihnen die nötigen Grundlagen zu Beginn dieser Vorlesung beizubringen.

Dieses Skript stützt sich auf ein Skript von Sascha Kurz [Ku] zur selben Vorlesung aus den Wintersemestern 2013/14 und 2014/15, welches wiederum teilweise auf dem Skripten „Numerische Mathematik I“ von Lars Grüne [Gr], „Mathematische Grundlagen für Wirtschaftswissenschaftler“ von Sascha Kurz und Jörg Rambau (und einem älteren Skript zur Computeralgebra von mir) sowie einem früheren Skript für diese Vorlesung von Karl Worthmann aufbaut.

## 2. Grundlegendes

In diesem Abschnitt geht es darum, wie man Mathematik so umsetzt, dass sie für einen Computer verdaulich ist. Das geschieht zunächst auf einer abstrakten Ebene, indem man einen *Algorithmus* formuliert, der das jeweilige Problem löst. In einem zweiten Schritt wird dieser Algorithmus dann in einer konkreten Programmiersprache *implementiert* und damit auf dem Computer zum Laufen gebracht. In der Vorlesung werden wir uns hauptsächlich auf den ersten Schritt konzentrieren, aber Sie müssen auch lernen, den zweiten Schritt zu machen, damit Sie die Algorithmen dann auch ausprobieren können und ein Gefühl dafür bekommen, was funktioniert und was eher nicht und warum.

Was ist nun ein Algorithmus? Wir bleiben etwas informell und sagen, ein *Algorithmus* ist ein „Rezept“, das eine Abfolge von Rechenoperationen vorschreibt, die zu gegebenen Eingabedaten in endlich vielen Schritten die gewünschten Ausgabedaten bestimmt.

David Hilbert (1862–1943)

Diese Beschreibung ist normalerweise gut genug, um ein gegebenes Verfahren als Algorithmus zu erkennen. Eine formale Definition ist dann notwendig, wenn man zeigen will, dass ein gewisses Problem *nicht* algorithmisch lösbar ist. Solche Probleme gibt es durchaus: Zum Beispiel gibt es keinen Algorithmus, der entscheiden kann, ob ein gegebenes Polynom $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ (für beliebiges $n$) eine Nullstelle in $\mathbb{Z}^n$ hat. Die Aufforderung, so ein Verfahren zu finden, war Nummer 10 in der berühmten Liste von 23 Problemen, die David Hilbert auf dem Internationalen Mathematikerkongress in Paris im Jahr 1900 vorstellte. Es dauerte eine Weile, bis die Möglichkeit der Unmöglichkeit in Betracht gezogen wurde; die Lösung des Problems erforderte die Formalisierung des Algorithmus-Begriffs (in den 1930er Jahren) und wurde 1970 durch Matiyasevich abgeschlossen.

Einen solchen Algorithmus kennen Sie aus der „Einführung in die Zahlentheorie und algebraische Strukturen“, nämlich den Euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teilers von zwei natürlichen Zahlen. Er lässt sich zum Beispiel so formulieren:

**ALGO**
ggT

---

**input:** $a, b \in \mathbb{Z}_{\geq 0}$
**output:** $\mathrm{ggT}(a, b)$
**while** $b \neq 0$ **do**
  bestimme $q, r \in \mathbb{Z}$ mit $a = qb + r$ und $0 \leq r < b$
  $(a, b) := (b, r)$
**end while**
**return** $a$

---

Was Sie hier sehen, ist sogenannter *Pseudocode*. Er dient zur Formulierung eines Algorithmus' auf einer mittleren Abstraktionsebene, die formal genug ist, um deutlich zu machen, dass es sich um einen Algorithmus handelt, aber von den Eigenheiten konkreter Programmiersprachen unabhängig ist. Er kann auch Anweisungen enthalten, die sich nicht unmittelbar in eine Programmiersprache umsetzen lassen, wie zum Beispiel die Anweisung „bestimme $q, r \in \mathbb{Z} \ldots$“. (In den meisten Programmiersprachen gilt das auch für die folgende Anweisung „$(a, b) := (b, r)$“, die man meistens in zwei Zuweisungen „$a := b$; $b := r$“ zerlegen muss. Dabei ist die Reihenfolge wichtig — was passiert, wenn man statt dessen „$b := r$; $a := b$“ verwendet?)

Der Beispiel-Algorithmus zeigt auch schon einige wichtige Strukturelemente des Pseudocodes:

(1) Die Eingabedaten werden in sogenannten *Variablen* gespeichert, die durch Namen (hier $a$ und $b$) bezeichnet werden. Es können auch weitere Variablen verwendet werden (wie $q$ und $r$). Der Wert, der einer Variablen zugewiesen ist, kann durch eine (erneute) *Zuweisung* geändert werden. Zuweisungen werden oft in der Form $a := w$ notiert (in vielen Programmiersprachen auch einfach „=“), wo $a$ der Name der Variablen und $w$ der zugewiesene Wert ist. Im Beispiel ist „$(a, b) := (b, r)$“ eine parallele Zuweisung an zwei Variablen.

(2) Der größte Teil des Algorithmus' besteht aus einer **while**-*Schleife*. Das bedeutet, dass die Anweisungen zwischen „**do**“ und „**end while**“ wiederholt ausgeführt werden, solange die *Bedingung* zwischen „**while**“ und „**do**“ erfüllt ist. Ist die Bedingung schon am Anfang nicht erfüllt, werden die Anweisungen der Schleife übersprungen und es geht nach dem „**end while**“ weiter.

(3) Schließlich wird durch die „**return**“-Anweisung ein *Ergebnis* zurückgegeben.

Pseudocode ist nicht normiert; es gibt viele Varianten, die sich in Details unterscheiden, sich aber in den Grundstrukturen sehr ähneln. Weitere dieser Strukturen sind die *Fallunterscheidung*, die wir als

**if** Bedingung **then** Anweisungen **end if**

oder

**if** Bedingung **then** Anweisungen **else** Anweisungen **end if**

schreiben werden und die **for**-*Schleife*

**for** $i = 1$ **to** $n$ **do** Anweisungen **end for**.

In ihr werden die Anweisungen wiederholt ausgeführt, wobei die *Schleifenvariable* $i$ (die natürlich auch anders heißen kann) nacheinander die Werte $1, 2, \ldots, n$ erhält. Der erste und der letzte Wert können natürlich auch anders lauten; wenn der Endwert kleiner als der Anfangswert ist, dann werden die Anweisungen gar nicht ausgeführt. Ein Stück Pseudocode, das die ersten hundert positiven ganzen Zahlen addiert, könnte zum Beispiel so aussehen:

**ALGO**
Summe

```
s := 0
for i = 1 to 100 do
   s := s + i
end for
```

Danach ist der Wert von $s$ die Summe $1 + 2 + \ldots + 100$.

Die Einrückung der Befehle innerhalb der Schleife (und in analogen Fällen) dient lediglich der besseren Lesbarkeit und hat keine eigene Bedeutung.

Ein weiteres Beispiel aus der Lehrpraxis ist die Berechnung der Klausurnote aus der erreichten Punktzahl. Dabei ist eine Richtpunktzahl als „100%“ vorgegeben. Die Klausur ist nicht bestanden, wenn weniger als 50% der Punkte erreicht wurden. Bei mindestens 50% verbessert sich die Note in 5%-Schritten von 4,0 (50 bis unter 55%) auf 1,0 (95% oder mehr). Der Befehl „round$(x, n)$“ soll dabei den Wert von $x$ auf $n$ Nachkommastellen runden und $\lceil x \rceil = \min\{n \in \mathbb{Z} \mid n \geq x\}$ rundet nach oben auf eine ganze Zahl.

**ALGO**
Noten

**input:** Richtpunktzahl $r$, erreichte Punkte $p$
**output:** Klausurnote

**if** $p < 0{,}5\,r$ **then**
  **return** 5,0
**else**
  **if** $p \geq r$ **then**
    **return** 1,0
  **else**
    $a := \lceil 22 - 20\,p/r \rceil$
    **return** round$(a/3,\ 1)$
  **end if**
**end if**

Die Aussage, dass ein Algorithmus das tut, was er soll, ist eine mathematische Aussage und bedarf eines Beweises. Genauer heißt das, es ist nachzuweisen, dass das Verfahren nach endlich vielen Schritten beendet wird (der Algorithmus *terminiert*; das ist nur dann ein Problem, wenn **while**-Schleifen vorkommen — man muss dann zeigen, dass man nicht in einer „Endlosschleife“ landen kann) und das zurückgegebene Ergebnis stimmt (der Algorithmus ist *korrekt*). Das wird allerdings nicht so sehr im Fokus dieser Vorlesung stehen.

Wesentliche Gesichtspunkte, die je nach Kontext unterschiedlich stark zum Tragen kommen, können sein:

(1) Gibt es überhaupt einen Algorithmus, der das Gewünschte leistet? Wie kann man ggfs. einen solchen konstruieren?

(2) Wie *effizient* ist ein gegebener Algorithmus im Hinblick auf Rechenzeit und/oder Speicherplatzverbrauch? Gibt es schnellere oder sparsamere Möglichkeiten, dasselbe Ergebnis zu bekommen?

(3) Ist ein gegebener Algorithmus *numerisch stabil* im Hinblick auf die Fortpflanzung von Ungenauigkeiten in den Eingabedaten und Rundungsfehlern während der Berechnung? Kann man die Stabilität verbessern?

Die Frage nach der *Berechenbarkeit* (Punkt (1)) wird uns hier nicht beschäftigen. Die Effizienz hingegen spielt in vielen Anwendungsbereichen eine wichtige Rolle: Man möchte mit den begrenzten Ressourcen an Zeit und Hardware (insbesondere schnell zugreifbarer Speicherplatz) möglichst viele oder möglichst umfangreiche Probleme lösen können. Zum Beispiel werden wir uns im dritten Teil der Vorlesung darüber Gedanken machen, wie man Zahlen (oder Polynome) schneller miteinander multiplizieren kann als wir das in der Schule gelernt haben. Numerische Stabilität spielt dagegen in der Computeralgebra keine Rolle, da man dort mit exakten Daten arbeitet. Dafür ist sie ein wichtiger Gesichtspunkt in der Numerik, denn dort wird mit näherungsweise dargestellten Zahlen gearbeitet und auch die Eingabedaten sind oft Messwerte und daher mit Ungenauigkeiten behaftet.

# Teil I: Numerik

Im Bereich der Numerik werden wir uns exemplarisch mit der numerischen Lösung von linearen und nichtlinearen Gleichungen und Gleichungssystemen befassen und dabei auch etwas in die Tiefe gehen. Auf diese Weise bekommen Sie einen Eindruck von den spezifischen Fragestellungen und Methoden der numerischen Mathematik. Damit sollten Sie in der Lage sein, sich anhand von geeigneten Lehrbüchern auch in andere Bereiche der Numerik (wie zum Beispiel das numerische Lösen von Differentialgleichungen) einzuarbeiten.

## 3. Gleitkommazahlen

In der Numerik geht es unter Anderem darum, Rechnungen mit konkreten Zahlen möglichst effizient durchzuführen. In unserer Vorstellung rechnen wir dabei mit reellen Zahlen. Allerdings gibt es, wie Sie aus der Analysis I wissen, überabzählbar viele reelle Zahlen, sodass es schon prinzipiell unmöglich ist (anders als etwa bei ganzen oder auch rationalen Zahlen), jede beliebige einzelne reelle Zahl zu bezeichnen, selbst wenn man beliebig viel Platz zur Verfügung hätte. Dazu kommt natürlich noch die Einschränkung, dass ein Computer nur endlich viel (Speicher-) Platz hat. Daher arbeitet man mit einer näherungsweisen Darstellung von Zahlen. Man könnte zum Beispiel mit einer festen Anzahl von Nachkommastellen rechnen, also mit einer fixierten *absoluten Genauigkeit*. Das ist in gewissen Bereichen sinnvoll (zum Beispiel, wenn es um Geldbeträge geht, die normalerweise ganzzahlige Vielfache von einem Cent sind), in vielen anderen aber nicht, weil dort Zahlen in vielen verschiedenen Größenordnungen auftreten. Es ist dann viel sinnvoller, mit einer festen *relativen Genauigkeit* zu arbeiten. Sie kennen das vielleicht von der „wissenschaftlichen Zahlschreibweise“, in der etwa die Lichtgeschwindigkeit als $c = 2{,}99792458 \cdot 10^8 \,\frac{\mathrm{m}}{\mathrm{s}}$ angegeben wird. Auch Ihr Taschenrechner arbeitet mit diesem Zahlformat, wenn er es mit sehr großen oder sehr kleinen Zahlen zu tun hat. Eine so geschriebene Zahl besteht aus zwei Teilen, der *Mantisse*, im Beispiel 2,99792458, die eine Vor- und eine feste Anzahl von Nachkommastellen hat, und dem *Exponenten*, im Beispiel 8. Dann gibt es noch die *Basis*, die bei der Kommunikation mit oder zwischen Menschen üblicherweise 10 ist, bei der Darstellung im Computer aber 2 oder eine Potenz davon. Mantisse und Exponent haben ein Vorzeichen, das im Beispiel jeweils positiv ist. Auch der Exponent hat im Computer eine beschränkte Länge. Das führt zu folgender Definition:

**DEF** Gleitkommazahl

3.1. **Definition.** Eine *Gleitkommazahl* $x$ zur Basis $B \in \mathbb{Z}_{\geq 2}$ der Länge $l \in \mathbb{Z}_{>0}$ und der Exponentenlänge $n \in \mathbb{Z}_{>0}$ ist gegeben durch

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

mit der *Mantisse*

$$m = v_1 \cdot \sum_{i=0}^{l-1} m_i B^{-i}$$

und dem *Exponenten*

$$e = v_2 \sum_{j=1}^{n} e_j B^{n-j} \,.$$

Dabei sind die Vorzeichen $v_1, v_2 \in \{-1, 1\}$ und die Ziffern $m_i$ der Mantisse und $e_j$ des Exponenten in $\{0, 1, \ldots, B-1\}$.

Wir verwenden die Schreibweise

$$x = \pm m_0{,}m_1m_2\ldots m_{l-1} \cdot B^{\pm e_1e_2\ldots e_n}\,;$$

das Vorzeichen '+' wird auch weggelassen. ♢

Auf englisch heißt Gleitkommazahl „floating point number".

Um eine eindeutige Darstellung zu haben, verlangt man im Fall $x \neq 0$ noch, dass $m_0 > 0$ ist. Für $x = 0$ soll der Exponent $e = 0$ sein. Allgemein soll das Vorzeichen $v_1$ bzw. $v_2$ positiv sein, wenn Mantisse (das kommt nur für $x = 0$ vor) bzw. Exponent null sind. Wenn man nicht die Länge des Exponenten betonen will, lässt man führende Nullen in der Darstellung von $e$ auch weg.

**BSP** Gleitkommazahlen

3.2. **Beispiele.** Der Zahlenwert der Lichtgeschwindigkeit in Metern pro Sekunde oben ist ein Beispiel für eine Gleitkommazahl zur Basis 10 der Länge 9 (und der Exponentenlänge mindestens 1). Wenn wir mit Gleitkommazahlen kürzerer Länge arbeiten wollen (oder müssen) und mit der Lichtgeschwindigkeit rechnen wollen, dann müssen wir die Zahl entsprechend *runden*. Um den Fehler klein zu halten, runden wir kaufmännisch, also jeweils zur nächstgelegenen darstellbaren Zahl. Das führt zu $2{,}99792 \cdot 10^8$ bei einer Länge von 6, zu $2{,}998 \cdot 10^8$ bei einer Länge von 4 und zu $3{,}00 \cdot 10^8$ bei einer Länge von 3. Wenn wir umgekehrt mit Gleitkommazahlen größerer Länge arbeiten, dann fügen wir Nullen an, um die Mantisse auf die gewünschte Länge zu bringen, also z.B. $2{,}99792458000 \cdot 10^8$ bei einer Länge von 12 Stellen.

Wollen wir die Lichtgeschwindigkeit auf dem Computer darstellen mit $B = 2$ und einer Länge von 16, dann müssen wir die Zahl 299792458 als Binärzahl schreiben:

$$299792458 = [10001110111100111100001001010]_2$$

(diese Darstellung erhält man von rechts nach links durch wiederholte Division durch $B = 2$ und Aufschreiben des jeweiligen Restes), dann feststellen, zwischen welchen Potenzen von 2 sie liegt, um den Exponenten zu bestimmen, und schließlich die Mantisse $1{,}000111\ldots$ auf 16 Stellen runden. Der Exponent ist $28 = [11100]_2$, sodass wir die Darstellung

$$1{,}000111011110100 \cdot 2^{11100}$$

erhalten. Wir brauchen also eine Exponentenlänge von mindestens 5.

Der Standard „IEEE-754" legt für das „single"-Format eine Länge von 23 Bit für die Mantisse und 8 Bit einschließlich Vorzeichen für den Exponenten fest. Mit einem weiteren Bit für das Vorzeichen der Mantisse passt das gerade in 32 Bit. Für das „double"-Format, das auf 64 Bit ausgelegt ist, sind die entsprechenden Längen 52 Bit für die Mantisse und 11 Bit für den Exponenten. ♣

Da mit endlich vielen Ziffern (für Mantisse und Exponent) nur endlich viele Zahlen dargestellt werden können, ist klar, dass zum Beispiel betragsmäßig sehr große oder sehr kleine Zahlen nicht dargestellt werden können. Beim „double"-Format liegt die Grenze zum Beispiel bei etwa $1{,}8 \cdot 10^{308}$ für große und bei etwa $2{,}2 \cdot 10^{-308}$ für kleine Zahlen.

Wie rechnet man (bzw. der Computer) nun mit diesen Gleitkommazahlen? Um das präzise formulieren zu können, legen wir zunächst die Basis und die Länge der Mantisse und des Exponenten in der Gleitkommadarstellung fest. Dann sei $\mathbb{G} \subset \mathbb{Q}$ die (endliche) Menge der in diesem Format exakt darstellbaren Zahlen. Wir definieren die Funktion $G\colon \mathbb{R} \to \mathbb{G}$, die jeder reellen Zahl $x$ die nächstgelegene

Gleitkommazahl zuordnet. Für reelle Zahlen $x$, die exakt in der Mitte zwischen zwei Gleitkommazahlen liegen, legen wir eine geeignete Rundungsvorschrift fest (bei $B = 2$ verwendet man üblicherweise „round to even“, also zu der Zahl, deren Mantisse auf 0 endet). Die Addition, Subtraktion, Multiplikation und Division von Gleitkommazahlen $x, y \in \mathbb{G}$ sind dann definiert als

$$x \oplus y = G(x + y)\,, \quad x \ominus y = G(x - y)\,, \quad x \odot y = G(xy)\,, \quad x \oslash y = G(x/y)\,;$$

Letzteres natürlich nur für $y \neq 0$. Jeder Rechenschritt beinhaltet also einen Rundungsvorgang, der einen Fehler verursacht. Ein Thema in der Numerik ist die Kontrolle dieser Rundungsfehler — man möchte nach Möglichkeit vermeiden, dass sich die Fehler im Lauf der Rechnung so verstärken, dass das Resultat völlig ungenau wird.

**BSP**
Auslöschung signifikanter Stellen

3.3. **Beispiel.** Ein typisches Problem ist der Verlust an (relativer) Genauigkeit bei der Subtraktion zweier etwa gleich großer Zahlen. Betrachten wir zum Beispiel $x = \sqrt{150} - \sqrt{149}$. In Gleitkomma-Arithmetik (mit $B = 10$ und $l = 5$) würde das so berechnet (wie man $G(\sqrt{150})$ usw. berechnet, werden wir gegen Ende dieses ersten Teils noch besprechen):

$$\begin{aligned}\hat{x} &= G(\sqrt{150}) \ominus G(\sqrt{149}) = 1{,}2247 \cdot 10^1 \ominus 1{,}2207 \cdot 10^1 \\ &= G(0{,}0040 \cdot 10^1) = 4{,}0000 \cdot 10^{-2}\end{aligned}$$

(Wir schreiben $\hat{x}$ für den in Gleitkomma-Arithmetik berechneten Näherungswert für $x$.) Auf der anderen Seite ist

$$G(x) = G(\sqrt{150} - \sqrt{149}) = 4{,}0893 \cdot 10^{-2}\,;$$

der berechnete Wert ist also nicht einmal auf zwei Stellen genau!

Man wird also versuchen, diese „Auslöschung signifikanter Stellen“ möglichst zu vermeiden. Das ist nicht immer so einfach, denn man muss erst einmal die Stellen im jeweiligen Verfahren identifizieren, an denen ein solches Problem auftreten kann. Im Beispiel ist eine mögliche Lösung, den Ausdruck erst einmal umzuformen und dann zu berechnen. Es gilt nämlich

$$x = \sqrt{150} - \sqrt{149} = \frac{1}{\sqrt{150} + \sqrt{149}}\,,$$

und damit

$$\begin{aligned}\hat{x} &= 1 \oslash \big(G(\sqrt{150}) \oplus G(\sqrt{149})\big) = 1 \oslash (1{,}2247 \cdot 10^1 \oplus 1{,}2207 \cdot 10^1) \\ &= 1 \oslash (2{,}4454 \cdot 10^1) = G(1/24{,}454) = 4{,}0893 \cdot 10^{-2}\,,\end{aligned}$$

was genau $G(x)$ ist. ♣

Das im obigen Beispiel angesprochene Problem führt dazu, dass in $(\mathbb{G}, \oplus, \ominus, \odot, \oslash)$ viele der gewohnten Rechenregeln nicht immer gelten, zum Beispiel das Assoziativgesetz der Addition.

Es gibt aber noch ein anderes Problem als das, die Akkumulation von Rundungsfehlern im Griff zu behalten.

**BSP** Empfindlichkeit gegenüber Störung

3.4. **Beispiel.** Wir betrachten das lineare Gleichungssystem $A\boldsymbol{x} = \boldsymbol{b}$ mit

$$A = \begin{pmatrix} 1{,}2969 & 0{,}8648 \\ 0{,}2161 & 0{,}1441 \end{pmatrix} \qquad \text{und} \qquad \boldsymbol{b} = \begin{pmatrix} 0{,}8642 \\ 0{,}1440 \end{pmatrix}.$$

Dieses hat die eindeutige und exakte Lösung $\boldsymbol{x} = (2, -2)^\top$.

Wenn wir nun die rechte Seite $\boldsymbol{b}$ ein wenig „stören“ und durch

$$\boldsymbol{b}' = \boldsymbol{b} + \begin{pmatrix} -10^{-8} \\ 10^{-8} \end{pmatrix} = \begin{pmatrix} 0{,}86419999 \\ 0{,}14400001 \end{pmatrix}$$

ersetzen, dann ist die eindeutige (und exakte) Lösung $\boldsymbol{x}' = (0{,}9911, -0{,}4870)^\top$ und damit sehr weit weg von $\boldsymbol{x}$! ♣

Das Problem ist hier kein numerisches, das sich durch geschicktes Rechnen umgehen ließe, sondern kommt aus der Fragestellung an sich. Im Beispiel kann man das daran sehen, dass die beiden Zeilen von $A$ „fast“ linear abhängig sind: Die Determinante von $A$ ist $10^{-8}$ (exakt) und damit nahe null. Wenn man sich die Lösung des linearen Gleichungssystems grafisch vorstellt als Bestimmung des Schnittpunkts zweier Geraden, dann sind die beiden Geraden im Beispiel fast parallel, sodass sich der Schnittpunkt ziemlich weit verschieben kann, wenn man die Geraden nur leicht bewegt. Wir werden bald ein Maß dafür kennenlernen, wie gut oder schlecht sich ein lineares Gleichungssystem in dieser Hinsicht verhält.

# 4. Lineare Gleichungssysteme

Viele relevante Konzepte der Numerik tauchen beim Studium der numerischen Lösung von linearen Gleichungssystemen auf. Algorithmen zur Lösung linearer Gleichungssysteme spielen auch in vielen anderen Bereichen der Numerik als Grundlage für weitere Algorithmen eine Rolle. Daher werden wir hier (wie das auch in vielen Numerik-Vorlesungen geschieht) die numerische Lösung linearer Gleichungssysteme als Einstieg in die Numerik verwenden.

Wir betrachten also ein lineares Gleichungssystem

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \end{array}$$

oder in Matrixschreibweise

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

mit reellen Koeffizienten $a_{ij}$ und rechten Seiten $b_i$, das in $\mathbb{R}^n$ zu lösen ist. Wir wissen aus der Linearen Algebra, dass die Lösungsmenge eines solchen Systems ein affiner Unterraum von $\mathbb{R}^n$ ist, und wir haben dort auch schon ein Verfahren kennengelernt, mit dem man entscheiden kann, ob es eine Lösung gibt, und gegebenenfalls eine Lösung und eine Basis des zugehörigen linearen Unterraums finden kann. Um die Diskussion einfach zu halten, nehmen wir im Folgenden stets an, dass das System eine eindeutige Lösung hat, die wir berechnen wollen; es gilt also $m = n$ und die Determinante der Koeffizientenmatrix $A$ verschwindet nicht.

**BSP** Ausgleichsrechnung

4.1. **Beispiel.** Eine typische Fragestellung, die auf ein lineares Gleichungssystem führt, ist die möglichst gute Approximation einer Reihe von Datenpunkten $(x_i, y_i)$ (zum Beispiel Messwerte) mit $1 \le i \le n$ durch eine Funktion $y = f(x)$ einer gegebenen Gestalt („Ausgleichsrechnung“). Ist diese Gestalt eine Linearkombination fest gegebener Funktionen, wie zum Beispiel eine Gerade $y = ax + b$, dann führt das auf ein lineares Gleichungssystem. Sei allgemeiner $f$ der Form

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \dots + a_m f_m(x)$$

mit vorgegebenen Funktionen $f_1, f_2, \dots, f_m$ gesucht. Wir schreiben $f_j(\boldsymbol{x})$ für den Vektor $\big(f_j(x_1), \dots, f_j(x_n)\big)^\top$ und $\boldsymbol{y} = (y_1, \dots, y_n)^\top$. Wir messen die Güte der Approximation durch die summierte quadratische Abweichung von den Daten (daher auch der Name „Methode der kleinsten Quadrate“; das geht schon auf Gauß zurück — gegenüber anderen Maßen für die Abweichung wie der Summe über $|f(x_i) - y_i|$ hat diese Funktion den Vorteil, glatt zu sein):

C.F. Gauß
1777–1855

$$\begin{aligned} \varphi(a_1, a_2, \dots, a_m) &:= \sum_{i=1}^{n} (a_1 f_1(x_i) + \dots + a_m f_m(x_i) - y_i)^2 \\ &= \|a_1 f_1(\boldsymbol{x}) + \dots + a_m f_m(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 \,. \end{aligned}$$

Anders interpretiert, wollen wir den euklidischen Abstand des Vektors $f(\boldsymbol{x})$ zu $\boldsymbol{y}$ minimieren, wobei $f(\boldsymbol{x})$ den von $f_1(\boldsymbol{x}), \dots, f_n(\boldsymbol{x})$ aufgespannten Untervektorraum von $\mathbb{R}^n$ durchläuft. Dazu können wir die partiellen Ableitungen von $\varphi$ null setzen, was ein lineares Gleichungssystem für die Koeffizienten $a_j$ ergibt. Konkret erhalten

wir

$$\begin{aligned}\frac{\partial\varphi}{\partial a_j} &= \sum_{i=1}^{n} 2(a_1 f_1(x_i) + \ldots + a_m f_m(x_i) - y_i) f_j(x_i) \\ &= 2\Big(\sum_{i=1}^{n} f_1(x_i) f_j(x_i) \cdot a_1 + \ldots + \sum_{i=1}^{n} f_m(x_i) f_j(x_i) \cdot a_m - \sum_{i=1}^{n} f_j(x_i) y_i\Big)\,,\end{aligned}$$

was auf das Gleichungssystem

$$\begin{array}{ccccccccc} f_{11}a_1 & + & f_{12}a_2 & + & \ldots & + & f_{1m}a_m & = & b_1 \\ f_{21}a_1 & + & f_{22}a_2 & + & \ldots & + & f_{2m}a_m & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ f_{m1}a_1 & + & f_{m2}a_2 & + & \ldots & + & f_{mm}a_m & = & b_m \end{array}$$

führt, wobei wir

$$f_{jk} = \sum_{i=1}^{n} f_j(x_i) f_k(x_i) = \langle f_j(\boldsymbol{x}), f_k(\boldsymbol{x})\rangle \qquad \text{und} \qquad b_j = \sum_{i=1}^{n} f_j(x_i) y_i = \langle f_j(\boldsymbol{x}), \boldsymbol{y}\rangle$$

gesetzt haben. Sind die Vektoren $f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$ linear unabhängig, dann ist die symmetrische Koeffizientenmatrix positiv definit, also insbesondere invertierbar, und das System hat eine eindeutige Lösung. Da $\varphi(a_1, \ldots, a_m)$ für betragsmäßig große Werte der Parameter $a_j$ ebenfalls groß wird, muss der dadurch gegebene einzige kritische Punkt von $\varphi$ das gesuchte Minimum sein. ♣

Wie löst man nun ein lineares Gleichungssystem? Ein Ansatz dafür ist, sich zu überlegen, dass das recht einfach ist, wenn das System Dreiecksgestalt hat:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & = & b_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & a_{nn}x_n & = & b_n \end{array}$$

(mit $a_{11}, \ldots, a_{nn} \neq 0$) lässt sich einfach durch Auflösen von unten her („Rückwärtseinsetzen“) lösen. Wir formulieren das in Pseudocode:

**ALGO** Rückwärtseinsetzen

---

**input:** obere Dreiecksmatrix $A = (a_{ij})_{1\le i,j\le n}$, Vektor $\boldsymbol{b} = (b_i)_{1\le i\le n}$
**output:** eindeutige Lösung $\boldsymbol{x}$ des Systems $A\boldsymbol{x} = \boldsymbol{b}$

**for** $i = n$ **to** $1$ **by** $-1$ **do**
  $s := b_i$
  **for** $j = i + 1$ **to** $n$ **do**
    $s := s - a_{ij}x_j$
  **end for**
  $x_i := s/a_{ii}$
**end for**
**return** $\boldsymbol{x} = (x_1, \ldots, x_n)$

---

(Das „**by** $-1$“ in der ersten **for**-Schleife bedeutet, dass $i$ von $n$ bis 1 heruntergezählt wird.) Dieser Algorithmus setzt einfach die Beziehungen

$$\begin{aligned} x_n &= b_n/a_{nn} \\ x_{n-1} &= (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1} \\ &\vdots \qquad\qquad \vdots \\ x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n)/a_{11} \end{aligned}$$

um.

Leider sind die wenigsten Gleichungssysteme in Dreiecksgestalt gegeben. Damit wir diesen Lösungsalgorithmus verwenden können, müssen wir das gegebene System also erst einmal so umformen, dass es Dreiecksgestalt bekommt. Das leistet die „Gauß-Elimination“. Dabei wird von den Gleichungen jeweils ein Vielfaches einer anderen Gleichung subtrahiert, was die Lösungsmenge nicht ändert, und zwar so, dass sukzessive Variablen eliminiert werden. Wir schauen uns das an einem Beispiel an.

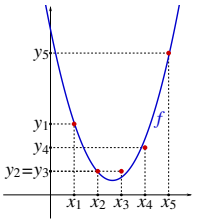
**BSP** Gauß-Verfahren

4.2. **Beispiel.** Gegeben seien die Datenpunkte $(1,3)$, $(2,1)$, $(3,1)$, $(4,2)$, $(5,6)$; wir wollen die beste Approximation durch eine quadratische Funktion

$$f(x) = a + bx + cx^2$$

finden. Das führt auf das Gleichungssystem

$$\begin{pmatrix} 5 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 13 \\ 46 \\ 198 \end{pmatrix} .$$




Wir ziehen nun zunächst geeignete Vielfache der ersten Gleichung von den übrigen ab, um aus diesen die Variable $a$ zu eliminieren. Konkret heißt das, dass wir das Dreifache der ersten Gleichung von der zweiten und das Elffache der ersten Gleichung von der dritten abziehen. Das ergibt das neue, äquivalente System

$$\begin{pmatrix} 5 & 15 & 55 \\ 0 & 10 & 60 \\ 0 & 60 & 374 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 13 \\ 7 \\ 55 \end{pmatrix} .$$

Dann eliminieren wir $b$ aus der dritten Gleichung, indem wir das Sechsfache der zweiten davon abziehen. Damit haben wir schließlich ein System in Dreiecksgestalt:

$$\begin{pmatrix} 5 & 15 & 55 \\ 0 & 10 & 60 \\ 0 & 0 & 14 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 13 \\ 7 \\ 13 \end{pmatrix} .$$

Wir erhalten als Lösung

$$c = 13/14\,, \qquad b = (7 - 60c)/10 = -341/70\,, \qquad a = (13 - 15b - 55c)/5 = 7\,. \ \clubsuit$$

Wenn man das für allgemeine (eindeutig lösbare) Systeme formuliert, dann sieht das z.B. so aus:

**ALGO** Gauß-Elimination

---

**input:** invertierbare Matrix $A = (a_{ij})_{1\le i,j\le n}$, Vektor $\boldsymbol{b} = (b_i)_{1\le i\le n}$

**output:** obere Dreiecksmatrix $A' = (a'_{ij})$, Vektor $\boldsymbol{b}' = (b'_i)$ mit $A'\boldsymbol{x} = \boldsymbol{b}' \iff A\boldsymbol{x} = \boldsymbol{b}$

```
for i = 1 to n do
  for j = i + 1 to n do
    f := a_ji/a_ii          | ziehe a_ji/a_ii-mal i-te Zeile von j-ter Zeile ab
    for k = i to n do
      a_jk := a_jk − f a_ik
    end for
    b_j := b_j − f b_i      | einschließlich rechte Seite
  end for
end for
return A, b                 | Matrix A, Vektor b wurden durch A′, b′ ersetzt
```

Hier gibt es ein Problem: Es kann vorkommen, dass $a_{ii} = 0$ ist; dann liefert die Zuweisung $f := a_{ji}/a_{ii}$ einen Fehler. Das lässt sich beheben, indem man in diesem Fall eine Zeile unterhalb der $i$-ten Zeile sucht, in der der Eintrag in der $i$-ten Spalte nicht null ist. (Eine solche Zeile muss es geben; anderenfalls wäre die Matrix nicht invertierbar.) Dann vertauscht man die $i$-te mit dieser Zeile. Die Wahl eines solchen Elements nennt man „Pivotierung“; den Eintrag, den man in der $i$-ten Spalte auswählt, *Pivotelement*. Das ergibt dann:

**DEF** Pivotelement

**ALGO** Gauß-Elimination mit Pivotierung

**input:** invertierbare Matrix $A = (a_{ij})_{1\le i,j\le n}$, Vektor $\boldsymbol{b} = (b_i)_{1\le i\le n}$
**output:** obere Dreiecksmatrix $A' = (a'_{ij})$, Vektor $\boldsymbol{b}' = (b'_i)$ mit $A'\boldsymbol{x} = \boldsymbol{b}' \iff A\boldsymbol{x} = \boldsymbol{b}$

```
for i = 1 to n do
  if a_ii = 0 then
    j := i + 1
    while a_ji = 0 do           | suche Pivot-Zeile
      j := j + 1
    end while
    for k = i to n do           | Zeilen tauschen
      (a_ik, a_jk) := (a_jk, a_ik)
    end for
    (b_i, b_j) := (b_j, b_i)    | einschließlich rechte Seite
  end if
  for j = i + 1 to n do
    f := a_ji/a_ii              | ziehe a_ji/a_ii-mal i-te Zeile von j-ter Zeile ab
    for k = i to n do
      a_jk := a_jk − f a_ik
    end for
    b_j := b_j − f b_i          | einschließlich rechte Seite
  end for
end for
return A, b                     | A, b wurden durch A′, b′ ersetzt
```

Wir werden das später noch variieren und (unter Umständen auch dann, wenn $a_{ii} \neq 0$ ist) nicht einfach das erste, sondern gezielt ein Pivotelement auswählen, um eine bessere numerische Stabilität zu erhalten.

## 5. Fehlerabschätzung und Kondition

Wir hatten schon gesehen, dass für konkrete Berechnungen mit dem Computer Zahlen im Allgemeinen gerundet werden müssen, da nur mit den Zahlen der Menge $\mathbb{G}$ der in einem gegebenen Format darstellbaren Gleitkommazahlen gerechnet werden kann. Das bedeutet, dass häufig schon die Eingabedaten, mit denen ein Algorithmus arbeitet, mit Ungenauigkeiten behaftet sind. Andere Fehlerquellen sind Messungenauigkeiten oder auch die Tatsache, dass die Eingaben für einen Algorithmus durch die Ausgaben eines anderen gegeben sind, die durch Rundungsfehler im Lauf der Rechnung ungenau sind. Wir wollen jetzt am Beispiel von linearen Gleichungssystemen untersuchen, wie „sensibel“ die Lösung auf eine „Störung“ in der Eingabe reagiert. Dabei blenden wir erst einmal die Gleitkomma-bedingten Rundungsfehler aus und betrachten den idealisierten Fall exakter Rechnung. Sei also

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

unser lineares Gleichungssystem. Wir stören es, indem wir die rechte Seite $\boldsymbol{b}$ durch $\tilde{\boldsymbol{b}} = \boldsymbol{b} + \Delta\boldsymbol{b}$ ersetzen (wobei wir uns $\Delta\boldsymbol{b}$ als „klein“ vorstellen). Das bewirkt eine Änderung von der ursprünglichen Lösung $\boldsymbol{x}$ zu $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$; dabei gilt

$$A\,\Delta\boldsymbol{x} = \Delta\boldsymbol{b}\,.$$

Man kann auch umgekehrt von einer Näherungslösung $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$ ausgehen und fragen, welche Störung des ursprünglichen Systems zu dieser Lösung gehört. In diesem Zusammenhang heißt $\Delta\boldsymbol{b} = A\,\Delta\boldsymbol{x}$ auch das *Residuum* oder der *Defekt* von $\tilde{\boldsymbol{x}}$.

**DEF** Residuum Defekt

Die spannende Frage ist jetzt, wie groß die Änderung $\Delta\boldsymbol{x}$ in der Lösung werden kann bei einer kleinen Störung $\Delta\boldsymbol{b}$. Dazu müssen wir die Größe von Vektoren in geeigneter Weise messen. Dies wird durch eine Norm auf dem jeweiligen Vektorraum geleistet. Zur Erinnerung:

5.1. **Definition.** Sei $V$ ein reeller Vektorraum. Eine *Norm* auf $V$ ist eine Abbildung

**DEF** Norm

$$V \longrightarrow \mathbb{R}_{\geq 0}\,, \qquad \boldsymbol{x} \longmapsto \|\boldsymbol{x}\|$$

mit den folgenden Eigenschaften:

(1) $\forall \boldsymbol{x} \in V\colon \|\boldsymbol{x}\| = 0 \iff \boldsymbol{x} = \boldsymbol{0}$;

(2) $\forall \lambda \in \mathbb{R}, \boldsymbol{x} \in V\colon \|\lambda\boldsymbol{x}\| = |\lambda| \cdot \|\boldsymbol{x}\|$;

(3) $\forall \boldsymbol{x}, \boldsymbol{y} \in V\colon \|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$. ♢

In der Analysis wurde bewiesen, dass alle Normen auf einem *endlich-dimensionalen* reellen Vektorraum äquivalent sind in dem Sinne, dass jede Norm durch ein Vielfaches jeder anderen Norm abgeschätzt werden kann.

5.2. **Beispiele.** Wichtige Beispiele von Normen auf dem $\mathbb{R}^n$ sind (jeweils mit $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top$)

**BSP** Normen

(1) die *euklidische Norm* $\|\boldsymbol{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$;

(2) die *Maximumsnorm* $\|\boldsymbol{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$

(3) und die *Summennorm* oder 1-*Norm* $\|\boldsymbol{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n|$.

Es gilt

$$\|\boldsymbol{x}\|_\infty \le \|\boldsymbol{x}\|_2 \le \sqrt{n}\|\boldsymbol{x}\|_\infty\,,$$
$$\|\boldsymbol{x}\|_2 \le \|\boldsymbol{x}\|_1 \le \sqrt{n}\|\boldsymbol{x}\|_2\,,$$
$$\|\boldsymbol{x}\|_\infty \le \|\boldsymbol{x}\|_1 \le n\|\boldsymbol{x}\|_\infty$$

und

$$|\langle \boldsymbol{x},\boldsymbol{y}\rangle| \le \|\boldsymbol{x}\|_1 \cdot \|\boldsymbol{y}\|_\infty\,, \qquad |\langle \boldsymbol{x},\boldsymbol{y}\rangle| \le \|\boldsymbol{x}\|_2 \cdot \|\boldsymbol{y}\|_2\,. \qquad \clubsuit$$

Für eine gegebene Matrix $A \in \mathrm{Mat}(n,\mathbb{R})$ und eine vorgegebene Norm auf $\mathbb{R}^n$ kann man dann fragen, um welchen Faktor sich die Norm eines Vektors durch Multiplikation mit $A$ schlimmstenfalls vergrößern kann. Das führt auf den Begriff der (durch eine Norm induzierten) Matrixnorm.

**DEF** Matrixnorm

5.3. **Definition.** Sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$. Die von $\|\cdot\|$ auf $\mathrm{Mat}(n,\mathbb{R})$ induzierte *Matrixnorm* ist dann definiert durch

$$\|A\| := \max\{\|A\boldsymbol{x}\| \mid \boldsymbol{x}\in\mathbb{R}^n, \|\boldsymbol{x}\|=1\} = \max\Big\{\frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \;\Big|\; \boldsymbol{x}\in\mathbb{R}^n\setminus\{\mathbf{0}\}\Big\}\,. \qquad \diamondsuit$$

Die Einheitssphäre $\{\boldsymbol{x}\in\mathbb{R}^n \mid \|\boldsymbol{x}\|=1\}$ ist nach einem Satz der Analysis kompakt und $\boldsymbol{x}\mapsto\|A\boldsymbol{x}\|$ ist stetig, daher wird das erste Maximum tatsächlich angenommen. Die zweite Gleichung folgt für $\boldsymbol{x} = \|\boldsymbol{x}\|\cdot\mathbf{e}$ mit $\|\mathbf{e}\|=1$ aus

$$\frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \frac{\big\|A(\|\boldsymbol{x}\|\mathbf{e})\big\|}{\|\boldsymbol{x}\|} = \frac{\big\|\|\boldsymbol{x}\|\cdot A\mathbf{e}\big\|}{\|\boldsymbol{x}\|} = \frac{\|\boldsymbol{x}\|\cdot\|A\mathbf{e}\|}{\|\boldsymbol{x}\|} = \|A\mathbf{e}\|\,.$$

Diese Matrixnorm definiert dann wirklich eine Norm auf dem reellen Vektorraum $\mathrm{Mat}(n,\mathbb{R})$ (Übung).

Wir bezeichnen eine Norm und die von ihr induzierte Matrixnorm mit demselben Symbol.

Für die drei Normen aus Beispiel 5.2 erhalten wir:

**SATZ** Matrixnormen

5.4. **Satz.** *Die Normen $\|\cdot\|_2$, $\|\cdot\|_\infty$ und $\|\cdot\|_1$ auf $\mathbb{R}^n$ induzieren die folgenden Matrixnormen. Dabei sei $A=(a_{ij})\in\mathrm{Mat}(n,\mathbb{R})$.*

$$\|A\|_2 = \sqrt{\rho(A^\top A)} \qquad \textit{(Spektralnorm)}$$
$$\|A\|_\infty = \max\{|a_{i1}|+|a_{i2}|+\ldots+|a_{in}| \mid 1\le i\le n\} \qquad \textit{(Zeilensummennorm)}$$
$$\|A\|_1 = \max\{|a_{1j}|+|a_{2j}|+\ldots+|a_{nj}| \mid 1\le j\le n\} \qquad \textit{(Spaltensummennorm)}$$

*Hier ist $\rho(A^\top A)$ der größte Eigenwert der positiv semidefiniten symmetrischen Matrix $A^\top A$.*

Man sieht, dass sich die Normen $\|A\|_\infty$ und $\|A\|_1$ im Vergleich zu $\|A\|_2$ recht einfach berechnen lassen.

*Beweis.* $\|\cdot\|_2$: Es ist klar, dass $A^\top A$ symmetrisch ist (denn $(A^\top A)^\top = A^\top(A^\top)^\top = A^\top A$); wegen $\boldsymbol{x}^\top(A^\top A)\boldsymbol{x} = (A\boldsymbol{x})^\top(A\boldsymbol{x}) = \|A\boldsymbol{x}\|_2^2 \ge 0$ ist die Matrix positiv semidefinit. Nach einem Satz aus der Linearen Algebra ist jede symmetrische Matrix orthogonal diagonalisierbar, d.h., es gibt eine orthogonale Matrix $P$, sodass $P^{-1}A^\top AP = P^\top A^\top AP$ eine Diagonalmatrix $D$ ist. Da $A^\top A$ positiv semidefinit ist, sind die Einträge von $D$ (also die Eigenwerte von $A^\top A$) nichtnegativ und der größte

Eintrag ist definitionsgemäß $\rho(A^\top A)$. Da $P$ orthogonal ist, gilt $\|P^{-1}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$. Es folgt mit $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ und $P^{-1}\boldsymbol{x} = (y_1, \ldots, y_n)^\top$:

$$\begin{aligned}\|A\boldsymbol{x}\|_2^2 &= \boldsymbol{x}^\top(A^\top A)\boldsymbol{x} = (P^{-1}\boldsymbol{x})^\top P^\top A^\top A P(P^{-1}\boldsymbol{x}) = (P^{-1}\boldsymbol{x})^\top D(P^{-1}\boldsymbol{x})\\ &= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \ldots + \lambda_n y_n^2\\ &\le \rho(A^\top A)(y_1^2 + y_2^2 + \ldots + y_n^2) = \rho(A^\top A)\|P^{-1}\boldsymbol{x}\|_2^2 = \rho(A^\top A)\|\boldsymbol{x}\|_2^2\end{aligned}$$

und damit $\|A\|_2 \le \sqrt{\rho(A^\top A)}$. Wenn (ohne Einschränkung) $\lambda_1 = \rho(A^\top A)$ ist, dann haben wir oben Gleichheit für $\boldsymbol{x} = P\mathbf{e}_1$ (also $y_1 = 1$ und $y_2 = \ldots = y_n = 0$), also gilt sogar $\|A\|_2 = \sqrt{\rho(A^\top A)}$.

$\|\cdot\|_\infty$: Mit $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top$ haben wir

$$\begin{aligned}\|A\boldsymbol{x}\|_\infty &= \max\bigl\{|a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n|, \ldots, |a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n|\bigr\}\\ &\le \max\bigl\{(|a_{11}| + \ldots + |a_{1n}|)\|\boldsymbol{x}\|_\infty, \ldots, (|a_{n1}| + \ldots + |a_{nn}|)\|\boldsymbol{x}\|_\infty\bigr\}\\ &= \max\bigl\{|a_{11}| + |a_{12}| + \ldots + |a_{1n}|, \ldots, |a_{n1}| + |a_{n2}| + \ldots + |a_{nn}|\bigr\} \cdot \|\boldsymbol{x}\|_\infty\,;\end{aligned}$$

das zeigt $\|A\|_\infty \le \max\bigl\{|a_{11}|+|a_{12}|+\ldots+|a_{1n}|, \ldots, |a_{n1}|+|a_{n2}|+\ldots+|a_{nn}|\bigr\}$. Wenn die $i$te Zeile von $A$ die (oder eine) mit der größten Summe der Beträge der Einträge ist, dann haben wir Gleichheit, wenn wir $\boldsymbol{x} = \bigl(\operatorname{sign}(a_{i1}), \operatorname{sign}(a_{i2}), \ldots, \operatorname{sign}(a_{in})\bigr)^\top$ wählen. Also gilt die behauptete Formel für $\|A\|_\infty$.

$\|\cdot\|_1$: Hier gilt

$$\begin{aligned}\|A\boldsymbol{x}\|_1 &= |a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n| + \ldots + |a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n|\\ &\le |a_{11}||x_1| + |a_{12}||x_2| + \ldots + |a_{1n}||x_n| + \ldots\\ &\qquad + |a_{n1}||x_1| + |a_{n2}||x_2| + \ldots + |a_{nn}||x_n|\\ &= \bigl(|a_{11}| + \ldots + |a_{n1}|\bigr)|x_1| + \ldots + \bigl(|a_{1n}| + \ldots + |a_{nn}|\bigr)|x_n|\\ &\le \max\bigl\{|a_{11}| + \ldots + |a_{n1}|, \ldots, |a_{1n}| + \ldots + |a_{nn}|\bigr\} \cdot \|\boldsymbol{x}\|_1\end{aligned}$$

mit Gleichheit für $\boldsymbol{x} = \mathbf{e}_j$, wenn die $j$-te Zeile die maximale Summennorm hat. Das zeigt die Behauptung für $\|A\|_1$. ❑

Aus der Definition der von einer Norm auf $\mathbb{R}^n$ induzierten Matrixnorm folgt unmittelbar:

**LEMMA** Schranke für absoluten Fehler

5.5. **Lemma.** *Seien $A \in \operatorname{Mat}(n, \mathbb{R})$ invertierbar, $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$ mit $A\boldsymbol{x} = \boldsymbol{b}$. Sei weiter eine Norm auf $\mathbb{R}^n$ gegeben; wir bezeichnen sie und die von ihr induzierte Matrixnorm mit $\|\cdot\|$. Wenn $\Delta\boldsymbol{x}$ die Änderung von $\boldsymbol{x}$ bei einer Änderung von $\boldsymbol{b}$ um $\Delta\boldsymbol{b}$ ist, dann gilt*

$$\|\Delta\boldsymbol{x}\| \le \|A^{-1}\| \cdot \|\Delta\boldsymbol{b}\|\,.$$

*Beweis.* Nach Voraussetzung ist $A(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b}$, woraus mit $A\boldsymbol{x} = \boldsymbol{b}$ folgt, dass $A \cdot \Delta\boldsymbol{x} = \Delta\boldsymbol{b}$ ist. Da $A$ invertierbar ist, ist das äquivalent zu $\Delta\boldsymbol{x} = A^{-1} \cdot \Delta\boldsymbol{b}$. Aus der Definition der Matrixnorm folgt dann $\|\Delta\boldsymbol{x}\| \le \|A^{-1}\| \cdot \|\Delta\boldsymbol{b}\|$. ❑

Nun ist man aber in vielen Fällen nicht so sehr am *absoluten* Fehler interessiert als am *relativen* Fehler, also hier $\|\Delta\boldsymbol{x}\|/\|\boldsymbol{x}\|$ im Vergleich zu $\|\Delta\boldsymbol{b}\|/\|\boldsymbol{b}\|$. Um diesen abzuschätzen, können wir die Ungleichung aus dem obigen Lemma kombinieren mit einer Abschätzung von $1/\|x\|$: Aus $A\boldsymbol{x} = \boldsymbol{b}$ folgt $\|\boldsymbol{b}\| \le \|A\| \cdot \|\boldsymbol{x}\|$ und damit

(für $\boldsymbol{b}, \boldsymbol{x} \neq \mathbf{0}$) $1/\|\boldsymbol{x}\| \leq \|A\|/\|\boldsymbol{b}\|$. Durch Multiplikation der beiden Ungleichungen bekommen wir

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \|A\| \cdot \|A^{-1}\| \frac{\|\Delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|} .$$

Daher führen wir folgenden Begriff ein:

**DEF** Kondition

5.6. **Definition.** Sei $\|\cdot\|$ eine Matrixnorm. Die *Kondition* einer invertierbaren Matrix $A \in \mathrm{Mat}(n, \mathbb{R})$ (bezüglich $\|\cdot\|$) ist

$$\mathrm{cond}_{\|\cdot\|}(A) = \|A\| \cdot \|A^{-1}\| .$$

Ist die Norm $\|\cdot\|_2$, $\|\cdot\|_\infty$ oder $\|\cdot\|_1$, dann schreiben wir $\mathrm{cond}_2$, $\mathrm{cond}_\infty$, $\mathrm{cond}_1$ für die zugehörige Kondition. ♢

Obige Rechnung beweist dann:

**LEMMA** Schranke für relativen Fehler

5.7. **Lemma.** *Seien $A \in \mathrm{Mat}(n, \mathbb{R})$ invertierbar, $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$ mit $A\boldsymbol{x} = \boldsymbol{b}$. Sei weiter eine Norm auf $\mathbb{R}^n$ gegeben; wir bezeichnen sie und die von ihr induzierte Matrixnorm mit $\|\cdot\|$. Wenn $\Delta\boldsymbol{x}$ die Änderung von $\boldsymbol{x}$ bei einer Änderung von $\boldsymbol{b} \neq \mathbf{0}$ um $\Delta\boldsymbol{b}$ ist, dann gilt*

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \mathrm{cond}_{\|\cdot\|}(A) \frac{\|\Delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|} .$$

**BSP** Kondition

5.8. **Beispiel.** Die Kondition der Matrix

$$A = \begin{pmatrix} 1{,}2969 & 0{,}8648 \\ 0{,}2161 & 0{,}1441 \end{pmatrix}$$

in Beispiel 3.4 bezüglich unserer drei Normen ergibt sich wie folgt. Wir müssen die Normen von $A$ und $A^{-1}$ berechnen; dazu beachten wir, dass

$$A^{-1} = 10^8 \begin{pmatrix} 0{,}1441 & -0{,}8648 \\ -0{,}2161 & 1{,}2969 \end{pmatrix}$$

ist ($10^8 = 1/\det(A)$). Damit ist

$$\begin{aligned}
\|A\|_\infty &= \max\{|1{,}2969| + |0{,}8648|, |0{,}2161| + |0{,}1441|\} = 2{,}1617 \\
\|A^{-1}\|_\infty &= 10^8 \max\{|0{,}1441| + |-0{,}8648|, |-0{,}2161| + |1{,}2969|\} = 1{,}5130 \cdot 10^8 \\
\|A\|_1 &= 1{,}5130 \\
\|A^{-1}\|_1 &= 2{,}1617 \cdot 10^8 \\
\|A\|_2 &= \sqrt{\rho(A^\top A)} = 1{,}5803 \quad \text{(gerundet)} \\
\|A^{-1}\|_2 &= \sqrt{\rho(A^{-\top} A^{-1})} = 1{,}5803 \cdot 10^8 \quad \text{(gerundet)},
\end{aligned}$$

woraus sich

$$\mathrm{cond}_2(A) = 2{,}4973 \cdot 10^8 \qquad \text{und} \qquad \mathrm{cond}_\infty(A) = \mathrm{cond}_1(A) = 3{,}2707 \cdot 10^8$$

ergibt (jeweils auf vier Nachkommastellen gerundet).

In Beispiel 3.4 war

$$\Delta\boldsymbol{b} = \begin{pmatrix} -10^{-8} \\ 10^{-8} \end{pmatrix} \qquad \text{und} \qquad \Delta\boldsymbol{x} = \begin{pmatrix} -1{,}0089 \\ 1{,}5130 \end{pmatrix} ,$$

also

$$\frac{\|\Delta \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} = 3{,}9831 \cdot 10^7 \frac{\|\Delta \boldsymbol{b}\|_2}{\|\boldsymbol{b}\|_2}$$
$$\frac{\|\Delta \boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} = 6{,}5377 \cdot 10^7 \frac{\|\Delta \boldsymbol{b}\|_\infty}{\|\boldsymbol{b}\|_\infty}$$
$$\frac{\|\Delta \boldsymbol{x}\|_1}{\|\boldsymbol{x}\|_1} = 3{,}1782 \cdot 10^7 \frac{\|\Delta \boldsymbol{b}\|_1}{\|\boldsymbol{b}\|_1}\,.$$

Wir sehen also, dass der Vergrößerungsfaktor im tatsächlichen relativen Fehler relativ nahe an die Kondition herankommt. ♣

Man kann mit Lemmas 5.5 und 5.7 eine nachträgliche („a posteriori“) Abschätzung der Genauigkeit der berechneten Lösung $\tilde{\boldsymbol{x}}$ erhalten. Dazu bilden wir das Residuum $\Delta \boldsymbol{b} = A\tilde{\boldsymbol{x}} - \boldsymbol{b}$. Für die Abweichung $\Delta \boldsymbol{x} = \tilde{\boldsymbol{x}} - \boldsymbol{x}$ (wobei $\boldsymbol{x}$ die wahre Lösung ist) gilt dann $A \cdot \Delta \boldsymbol{x} = \Delta \boldsymbol{b}$, also

$$\|\Delta \boldsymbol{x}\| \le \|A^{-1}\| \cdot \|A\tilde{\boldsymbol{x}} - \boldsymbol{b}\| \qquad \text{und} \qquad \frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \operatorname{cond}_{\|\cdot\|}(A) \frac{\|A\tilde{\boldsymbol{x}} - \boldsymbol{b}\|}{\|\boldsymbol{b}\|}\,.$$

Dazu muss man aber natürlich (obere Schranken für) die Norm $\|A^{-1}\|$ bzw. die Kondition $\operatorname{cond}_{\|\cdot\|}(A)$ berechnet haben. Das kann sich allerdings lohnen, wenn man viele lineare Gleichungssysteme mit derselben Koeffizientenmatrix $A$ lösen muss.

Rundungsfehler während der Abarbeitung eines Lösungsverfahrens können ähnliche Auswirkungen haben wie Fehler in der rechten Seite des Gleichungssystems und daher bei „schlecht konditionierten“ Matrizen (also Matrizen mit großer Kondition) möglicherweise zu großen (relativen) Fehlern im Ergebnis führen. Daher versucht man, Algorithmen so zu entwerfen, dass sie auch im Fall von schlecht konditionierten Matrizen noch möglichst gut funktionieren. Dazu muss man die Schritte identifizieren, bei denen große relative Fehler auftreten können; das sind im Wesentlichen Subtraktionen oder Additionen von betragsmäßig fast gleich großen Zahlen mit deutlich kleinerem Ergebnis („Auslöschung signifikanter Stellen“, vergleiche Beispiel 3.3).

Beim Gauß-Verfahren werden hauptsächlich Differenzen

$$a_{jk} - \frac{a_{ji}}{a_{ii}} a_{ik} \qquad \text{bzw.} \qquad b_j - \frac{a_{ji}}{a_{ii}} b_i$$

(für $j > i$) berechnet. Eine Möglichkeit der Einflussnahme besteht dabei in der Wahl des Pivotelements: Wir können statt mit Zeile $i$ auch mit einer beliebigen anderen Zeile mit Index $p \ge i$ arbeiten, um die $i$-te Spalte „auszuräumen“. Dann berechnen wir statt der obigen Differenzen

$$a_{jk} - \frac{a_{ji}}{a_{pi}} a_{pk} \qquad \text{bzw.} \qquad b_j - \frac{a_{ji}}{a_{pi}} b_p$$

(für $j \ge i$, $j \ne p$). Wir haben das bereits benutzt, um sicherzustellen, dass die Division durch $a_{ii}$ ausführbar ist. Jetzt versuchen wir diese Freiheit dazu zu nutzen, die Rundungsfehler möglichst klein zu halten. Die Idee dabei ist, die Terme $a_{ji}a_{pk}/a_{pi}$ und $a_{ji}b_p/a_{pi}$ möglichst klein zu machen und damit das Risiko der Auslöschung zu minimieren. Eine einfache Variante dieses Ansatzes wählt einfach das betragsmäßig größte Element $a_{pi}$ in der $i$-ten Spalte. Etwas besser (aber dafür auch aufwändiger, also langsamer) ist es, alle Quotienten $a_{pk}/a_{pi}$ ($i < k \le n$) und $b_p/a_{pi}$ zu berücksichtigen. (Beachte, dass die Differenz mit $k = i$ immer exakt null ergibt; dieser Term muss also bei der Wahl nicht berücksichtigt werden.) Nach der

Wahl des Pivotelements vertauschen wir die $i$-te mit der $p$-ten Zeile, falls nötig. Das führt zu folgender Variante der Gauß-Elimination.

**ALGO** Gauß-Elimination mit Pivotsuche

**input:** invertierbare Matrix $A = (a_{ij})_{1\le i,j\le n}$, Vektor $\boldsymbol{b} = (b_i)_{1\le i\le n}$
**output:** obere Dreiecksmatrix $A' = (a'_{ij})$, Vektor $\boldsymbol{b}' = (b'_i)$ mit $A'\boldsymbol{x} = \boldsymbol{b}' \iff A\boldsymbol{x} = \boldsymbol{b}$

**for** $i = 1$ **to** $n$ **do**
  $p :=$ Zeilenindex $\ge i$ mit $a_{pi} \neq 0$ und $\max\left\{\frac{|a_{p,i+1}|}{|a_{pi}|}, \dots, \frac{|a_{pn}|}{|a_{pi}|}, \frac{|b_p|}{|a_{pi}|}\right\}$ minimal | suche Pivot-Zeile
  **if** $p \neq i$ **then**
    vertausche Zeilen $i$ und $p$ in $A$
    vertausche $b_i$ und $b_p$
  **end if**
  **for** $j = i + 1$ **to** $n$ **do**
    $f := a_{ji}/a_{ii}$
    **for** $k = i$ **to** $n$ **do**
      $a_{jk} := a_{jk} - f a_{ik}$
    **end for**
    $b_j := b_j - f b_i$
  **end for**
**end for**
**return** $A, \boldsymbol{b}$ | $A$, $\boldsymbol{b}$ wurden durch $A'$, $\boldsymbol{b}'$ ersetzt

Die hier verwendete Form der Pivotsuche wird *Spaltenpivotsuche* genannt, da man das Pivotelement jeweils in der aktuell bearbeiteten Spalte sucht. Man kann auch eine *vollständige Pivotsuche* verwenden; dabei werden alle Elemente rechts und unterhalb von $a_{ii}$ in Betracht gezogen (und gegebenenfalls außer der Zeilen- auch eine Spaltenvertauschung vorgenommen. Letztere bewirkt eine entsprechende Vertauschung der Komponenten des Lösungsvektors, die am Ende wieder rückgängig gemacht werden muss.) Solche Pivotierungsmethoden verbessern die Robustheit des Verfahrens gegenüber dem Einfluss von Rundungsfehlern, bieten aber keine Garantie, dass die Fehler bei schlechter Kondition klein bleiben.

## 6. Abschätzung des Rechenaufwands

Ein numerisches Rechenverfahren soll nicht nur möglichst genaue Ergebnisse liefern, sondern diese auch möglichst schnell berechnen. Daher ist es wichtig, verschiedene Algorithmen hinsichtlich der von ihnen für verschiedene Größen der Eingabe benötigten Rechenzeit miteinander zu vergleichen. Da die real gebrauchte Zeit von vielen Einflüssen abhängt (insbesondere von der Hardware des verwendeten Computers), braucht man ein davon unabhängiges Maß. Dafür zählt man die Anzahl an Operationen, die ein Algorithmus bei gegebener Eingabe ausführt. Dabei nimmt man meistens noch weitere Vereinfachungen vor, zum Beispiel indem man nur elementare Rechenoperationen (Addition, Subtraktion, Vergleich, Multiplikation und Division von Gleitkommazahlen) berücksichtigt und etwa die Operationen, die den Programmablauf steuern (wie die „Verwaltung“ von **for**-Schleifen, wobei die Schleifenvariable hochgezählt und mit dem Endwert verglichen werden muss), vernachlässigt. Das ist gerechtfertigt, da für jede Verwaltungsoperation in der Regel mehrere Rechenoperationen durchgeführt werden, sodass die ersteren gegenüber den letzteren kaum ins Gewicht fallen.

Wir werden das jetzt beispielhaft für die Lösung eines linearen Gleichungssystems mit dem Gauß-Verfahren ohne Pivotsuche und anschließendem Rückwärtseinsetzen durchführen. Dabei machen wir zunächst die vereinfachende Annahme, dass stets $a_{ii} \neq 0$ ist, sodass keine Zeilen vertauscht werden müssen.

Zur späteren Verwendung erinnern wir uns an einige Summenformeln:

6.1. **Lemma.** **LEMMA** Summenformeln

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}, \qquad \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}.$$

Wir beginnen mit dem Gauß-Verfahren. In der innersten ($k$-)Schleife wird jeweils die Zuweisung $a_{jk} := a_{jk} - f a_{ik}$ durchgeführt. Das zerlegt sich in die Multiplikation $f a_{ik}$ und eine anschließende Subtraktion, erfordert also zwei Rechenoperationen. Durchlaufen wird die entsprechende Zeile für alle Tripel $(i, j, k)$ mit $1 \leq i < j \leq n$ und $i \leq k \leq n$, also für festes $i$ jeweils $(n-i)(n+1-i)$ mal. Das ergibt

$$\begin{aligned}\sum_{i=1}^{n}(n-i)(n+1-i) &\overset{k=n+1-i}{=} \sum_{k=1}^{n}(k-1)k = \sum_{k=0}^{n} k^2 - \sum_{k=0}^{n} k \\ &= \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{1}{3}n^3 - \frac{1}{3}n\end{aligned}$$

Durchläufe und damit $\frac{2}{3}n^3 - \frac{2}{3}n$ Rechenoperationen.

Nur innerhalb der $j$-Schleife haben wir noch die Zuweisungen $f := a_{ji}/a_{ii}$ und $b_j := b_j - f b_i$, die zusammen drei Rechenoperationen erfordern. Sie werden

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

mal ausgeführt; das ergibt weitere $\frac{3}{2}n^2 - \frac{3}{2}n$ Rechenoperationen.

Für das Rückwärtseinsetzen haben wir entsprechend $n^2 - n$ Rechenoperationen in der inneren Schleife (je zwei pro Zuweisung) und dann noch $n$ Divisionen $s/a_{ii}$. Insgesamt ergeben sich

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

Rechenoperationen für die Lösung des linearen Gleichungssystems.

Ganz so genau will man es meistens gar nicht wissen (auch deshalb, weil in dieser Zählweise ja schon vereinfachende Annahmen stecken). Um die Größenordnung einer Funktion von $n$ knapp notieren zu können, verwendet man gerne folgende Notation.

**DEF**
$O$, $\Omega$, $\Theta$

6.2. **Definition.** Sind $f, g\colon \mathbb{Z}_{\geq 0} \to \mathbb{R}$, dann schreiben wir

(1) $f(n) \in O(g(n))$, falls es $n_0 \in \mathbb{Z}_{\geq 0}$ und $c > 0$ gibt, sodass $|f(n)| \leq cg(n)$ für alle $n \geq n_0$ gilt;

(2) $f(n) \in \Omega(g(n))$, falls es $n_0 \in \mathbb{Z}_{\geq 0}$ und $c > 0$ gibt, sodass $|f(n)| \geq cg(n)$ für alle $n \geq n_0$ gilt;

(3) $f(n) \in \Theta(g(n))$, falls $f(n) \in O(g(n))$ und $f(n) \in \Omega(g(n))$. ♢

Für die Funktion $f$, die die Rechenoperationen im Lösungsverfahren für lineare Gleichungssysteme zählt, gilt dann zum Beispiel

$$f(n) \in O(n^3)\,,\ f(n) \in O(n^4)\,,\ f(n) \in \Omega(n^3)\,,\ f(n) \in \Omega(n^2) \text{ und } f(n) \in \Theta(n^3)\,.$$

Man sieht auch häufig die Schreibweise „$f(n) = O(g(n))$“ etc. Wenn man den führenden Term im Wachstum von $f$ genau angeben will, schreibt man auch z.B. $f(n) = \frac{2}{3}n^3 + O(n^2)$.

Wenn wir den Algorithmus mit der Pivotsuche verwenden, dann müssen wir zusätzlich für alle $i$ und alle $p \geq i$ die Quotienten $|a_{p,k}|/|a_{pi}|$ für $i{+}1 \leq k \leq n$ und $|b_p|/|a_{pi}|$ berechnen und vergleichen und außerdem die Maxima wieder untereinander vergleichen. Das sind

$$\sum_{i=1}^{n}(n+1-i)(2n+1-2i) + \sum_{i=1}^{n}(n-i) = 2\sum_{k=0}^{n}k^2 - n = \frac{2}{3}n^3 + n^2 - \frac{2}{3}n$$

Rechenoperationen und damit annähernd so viele, wie der restliche Algorithmus braucht, womit sich die Rechenzeit etwa verdoppelt. Die einfache Pivotsuche, bei der man nur nach dem Maximum von $|a_{pi}|$ sucht, braucht dem gegenüber nur $O(n^2)$ zusätzliche Operationen und beeinflusst die Rechenzeit damit nur unwesentlich (genauer: der Anteil der für die Suche verwendeten Zeit ist in $O(1/n)$ und wird daher klein für großes $n$). Man sieht, dass man hier eine Balance zwischen Robustheit und Effizienz finden muss.

Die oben eingeführte „$O$-Notation“ wird uns später, zum Beispiel in der Computeralgebra, in einem ähnlichen Zusammenhang wieder begegnen.

## 7. LR- und QR-Zerlegung

Was im Gauß-Verfahren „hinter den Kulissen“ wirklich passiert, ist eine Produktzerlegung $A = LR$, wobei $R$ die obere Dreiecksmatrix ist, die dabei herauskommt; $L$ ist eine *untere* Dreiecksmatrix, die sich aus den Zeilenumformungen ergibt, die im Algorithmus vorgenommen werden. (Genauer: Multiplikation einer Matrix $M$ von links mit $L$ ist äquivalent zur Durchführung der Zeilenumformungen an $M$.) Wenn man die Faktoren $f = a_{ji}/a_{ii}$, die im Algorithmus auftreten, in geeigneter Form abspeichert, erhält man die Matrix $L$; man kann sie also ohne zusätzlichen Aufwand mitberechnen. (Das setzt voraus, dass $a_{ii}$ stets $\neq 0$ ist. Im Allgemeinen muss man die Zeilenvertauschungen noch in einer Permutationsmatrix $P$ codieren und bekommt eine Zerlegung $PA = LR$.)

Hat man so eine Zerlegung $A = LR$ bestimmt, dann kann man das lineare Gleichungssystem $A\boldsymbol{x} = \boldsymbol{b}$ recht effizient in zwei Schritten lösen:

(1) Zuerst löst man $L\boldsymbol{y} = \boldsymbol{b}$; das geht durch *Vorwärtseinsetzen* von oben nach unten analog zum Rückwärtseinsetzen.

(2) Dann löst man $R\boldsymbol{x} = \boldsymbol{y}$ durch Rückwärtseinsetzen wie gehabt.

Es folgt $A\boldsymbol{x} = LR\boldsymbol{x} = L\boldsymbol{y} = \boldsymbol{b}$, und der Rechenaufwand ist $\Theta(n^2)$. Das ist vor allem dann nützlich, wenn man viele Gleichungssysteme mit derselben Koeffizientenmatrix $A$, aber verschiedenen rechten Seiten $\boldsymbol{b}$ lösen möchte. Dann hat man nur einmal den Aufwand von $\Theta(n^3)$ Rechenoperationen, um die LR-Zerlegung zu bestimmen.

Allerdings hat dieses Verfahren einen Nachteil: Die Kondition einer der Matrizen $L$ und $R$ kann deutlich schlechter sein als die von $A$, sodass man sich Probleme mit der numerischen Genauigkeit einhandelt. Deswegen wollen wir jetzt eine in dieser Hinsicht bessere Zerlegung finden. Dazu erinnern wir uns an die folgende Definition aus der Linearen Algebra.

**DEF** orthogonale Matrix

7.1. **Definition.** Eine Matrix $Q \in \mathrm{Mat}(n, \mathbb{R})$ heißt *orthogonal*, wenn $QQ^\top = I_n$ gilt. ♢

Eine orthogonale Matrix $Q$ ist invertierbar mit $Q^{-1} = Q^\top$. Orthogonale Matrizen repräsentieren *Isometrien* des $\mathbb{R}^n$, d.h., es gilt $\|Q\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$ für alle $\boldsymbol{x} \in \mathbb{R}^n$. Insbesondere ist $\|Q\|_2 = 1$.

**SATZ** QR-Zerlegung

7.2. **Satz.** *Sei $A \in \mathrm{Mat}(n, \mathbb{R})$ invertierbar. Dann gibt es eindeutig bestimmte Matrizen $Q, R \in \mathrm{Mat}(n, \mathbb{R})$ mit $A = QR$, sodass $Q$ orthogonal und $R$ eine obere Dreiecksmatrix mit positiven Diagonaleinträgen ist.*

*Außerdem ist* $\mathrm{cond}_2(R) = \mathrm{cond}_2(A)$.

*Beweis.* Wenn wir das Gram-Schmidtsche Orthonormalisierungsverfahren auf die Spalten von $A$ anwenden, dann erhalten wir als Resultat eine orthogonale Matrix $Q$. Das Verfahren nimmt Spaltenoperationen an $A$ vor, was der Multiplikation von $A$ von rechts mit einer Matrix $M$ entspricht. Da die $j$-te Spalte durch eine Linearkombination der Spalten 1 bis $j$ ersetzt wird, ist $M$ eine obere Dreiecksmatrix, und da der Skalierungsfaktor der $j$-ten Spalte positiv ist, hat $M$ positive Diagonaleinträge. Mit $R = M^{-1}$ folgt die Existenz der QR-Zerlegung (mit $M$ ist auch $R$ eine obere Dreiecksmatrix mit positiven Diagonaleinträgen).

Für die Eindeutigkeit sei $A = Q'R'$ eine weitere solche Zerlegung. Dann gilt $Q'^{-1}Q = R'R^{-1}$ und diese Matrix ist sowohl orthogonal als auch eine obere Dreiecksmatrix mit positiven Diagonaleinträgen. Man überlegt sich leicht, dass die Einheitsmatrix die einzige solche Matrix ist; es folgt $Q' = Q$ und $R' = R$.

Wir hatten uns schon überlegt, dass $\|Q\|_2 = \|Q^{-1}\|_2 = 1$ ist. Da (wie man sich leicht überlegt) $\|MM'\| \leq \|M\| \cdot \|M'\|$ für jede (induzierte) Matrixnorm gilt, folgt

$$\|R\|_2 = \|Q^{-1}A\|_2 \leq \|A\|_2 \qquad \text{und} \qquad \|R^{-1}\|_2 = \|A^{-1}Q\|_2 \leq \|A^{-1}\|_2$$

und damit $\operatorname{cond}_2(R) \leq \operatorname{cond}_2(A)$. Genauso zeigt man $\operatorname{cond}_2(A) \leq \operatorname{cond}_2(R)$. ❑

Hat man so eine Zerlegung $A = QR$ gefunden, dann kann man das lineare Gleichungssystem $A\boldsymbol{x} = \boldsymbol{b}$ wieder effizient in zwei Schritten lösen:

(1) Zuerst berechnet man $\boldsymbol{y} = Q^\top \boldsymbol{b}$. Eine Multiplikation „Matrix mal Vektor“ benötigt $\Theta(n^2)$ Rechenoperationen.

(2) Dann löst man wieder $R\boldsymbol{x} = \boldsymbol{y}$ durch Rückwärtseinsetzen.

Es ist dann $A\boldsymbol{x} = QR\boldsymbol{x} = Q\boldsymbol{y} = \boldsymbol{b}$.

Die Frage ist nun, wie man die Matrizen $Q$ (oder besser $Q^{-1} = Q^\top$, die man zur Lösung benötigt) und $R$ numerisch stabil und effizient berechnen kann. Das Gram-Schmidt-Orthonormalisierungsverfahren, das wir oben im Beweis verwendet haben, ist weder numerisch besonders stabil noch besonders effizient (in einer die Stabilität etwas verbessernden Form braucht es etwa $2n^3$ Rechenoperationen). Ein alternativer Ansatz ist, statt wie bei Gram-Schmidt aus den Spalten von $A$ sukzessive die Spalten von $Q$ zu erzeugen, die Matrix $A$ durch sukzessive Multiplikationen mit orthogonalen Matrizen auf Dreiecksgestalt zu bringen. Verwendet man dabei Spiegelungen, dann führt das auf das *Householder-Verfahren*.

Eine Matrix, die eine Spiegelung an der Hyperebene $\{\boldsymbol{x} \in \mathbb{R}^n \mid \langle \boldsymbol{x}, \boldsymbol{v}\rangle = 0\}$ beschreibt, hat die Form ($\boldsymbol{v}$ als Spaltenvektor)

$$H_{\boldsymbol{v}} = I_n - \frac{2}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle} \boldsymbol{v}\boldsymbol{v}^\top, \quad \text{also} \quad H_{\boldsymbol{v}}\boldsymbol{x} = \boldsymbol{x} - \frac{2}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle} \boldsymbol{v}\boldsymbol{v}^\top \boldsymbol{x} = \boldsymbol{x} - 2\frac{\langle \boldsymbol{v}, \boldsymbol{x}\rangle}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle}\boldsymbol{v}\,.$$

Denn ist $\langle \boldsymbol{x}, \boldsymbol{v}\rangle = 0$, dann ist $H_{\boldsymbol{v}}\boldsymbol{x} = \boldsymbol{x}$, und außerdem gilt $H_{\boldsymbol{v}}\boldsymbol{v} = -\boldsymbol{v}$.

Wir nehmen jetzt einmal an, dass die ersten $j-1$ Spalten von $A$ schon der oberen Dreiecksgestalt entsprechen. Wir suchen dann nach einer Spiegelung $H_{\boldsymbol{v}}$, die diese ersten $j-1$ Spalten unverändert lässt und die $j$-te Spalte auf einen Vektor abbildet, in dem höchstens die ersten $j$ Einträge von null verschieden sind. Dazu muss $\boldsymbol{v}$ so gewählt werden, dass $\langle \mathbf{e}_i, \boldsymbol{v}\rangle = 0$ ist für $1 \leq i < j$ (hier ist $\mathbf{e}_i$ der $i$-te Standard-Basisvektor). Wenn $\boldsymbol{a}_j$ die $j$-te Spalte von $A$ ist, dann muss zusätzlich

$$H_{\boldsymbol{v}}\boldsymbol{a}_j = \boldsymbol{a}_j - 2\frac{\langle \boldsymbol{a}_j, \boldsymbol{v}\rangle}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle}\boldsymbol{v}$$

eine Linearkombination der Vektoren $\mathbf{e}_1, \ldots, \mathbf{e}_j$ sein. Aus der ersten Bedingung folgt, dass $\boldsymbol{v} = (0, \ldots, 0, v_j, v_{j+1}, \ldots, v_n)^\top$ sein muss und aus der zweiten Bedingung ergibt sich dann

$$a_{ij} = 2\frac{\langle \boldsymbol{a}_j, \boldsymbol{v}\rangle}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle} v_i \qquad \text{für } j < i \leq n.$$

Wir können also zum Beispiel $v_i = a_{ij}$ setzen für $j+1 \le i \le n$; dann müssen wir $v_j$ noch so wählen, dass $\langle \boldsymbol{v}, \boldsymbol{v}\rangle = 2\langle \boldsymbol{a}_j, \boldsymbol{v}\rangle$ wird. Das führt auf die Gleichung

$$v_j^2 + \sum_{i=j+1}^{n} a_{ij}^2 = 2a_{jj}v_j + 2\sum_{i=j+1}^{n} a_{ij}^2 \iff (v_j - a_{jj})^2 = \sum_{i=j}^{n} a_{ij}^2 \,.$$

Wir wählen also $v_j = a_{jj} + c$ mit $c = \pm\sqrt{a_{jj}^2 + a_{j+1,j}^2 + \ldots + a_{nj}^2}$. Da in der Rechnung durch $\langle \boldsymbol{v}, \boldsymbol{v}\rangle$ geteilt wird, ist es numerisch vorteilhaft, das Vorzeichen so zu wählen, dass $|v_j|$ möglichst groß ist. Das erreicht man, indem man das Vorzeichen von $c$ gleich dem von $a_{jj}$ wählt. Damit haben wir folgenden Algorithmus.

**ALGO**
QR-Zerlegung nach Householder

---

**input:** $A \in \mathrm{Mat}(n, \mathbb{R})$ invertierbar
**output:** $R \in \mathrm{Mat}(n, \mathbb{R})$ obere Dreiecksmatrix; Vektoren $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1} \in \mathbb{R}^n$ mit $H_{\boldsymbol{v}_{n-1}} \cdots H_{\boldsymbol{v}_1} A = R$

**for** $j = 1$ **to** $n-1$ **do**
  $c := \sqrt{a_{jj}^2 + a_{j+1,j}^2 + \ldots + a_{nj}^2}$
  **if** $a_{jj} < 0$ **then** $c := -c$ **end if**
  $\boldsymbol{v}_j := (0, \ldots, 0, a_{jj} + c, a_{j+1,j}, \ldots, a_{nj})^\top$
  $d := c \cdot (\boldsymbol{v}_j)_j$
  $a_{jj} := -c$
  **for** $i = j+1$ **to** $n$ **do** $a_{ij} := 0$ **end for**
  **for** $k = j+1$ **to** $n$ **do**
    $f := (\sum_{i=j}^{n} a_{ik} \cdot (\boldsymbol{v}_j)_i)/d$
    **for** $i = j$ **to** $n$ **do** $a_{ik} := a_{ik} - f \cdot (\boldsymbol{v}_j)_i$ **end for**
  **end for**
**end for**
**return** $A; \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{n-1}$

---

**SATZ**
Berechnung der QR-Zerlegung

7.3. **Satz.** *Der obige Algorithmus ist korrekt; die Matrix* $Q = (H_{\boldsymbol{v}_{n-1}} \cdots H_{\boldsymbol{v}_1})^\top$ *ist orthogonal mit* $A = QR$.

*Beweis.* Wir bemerken zunächst, dass

$$\begin{aligned}\langle \boldsymbol{v}_j, \boldsymbol{v}_j\rangle &= (a_{jj} + c)^2 + a_{j+1,j}^2 + \ldots + a_{nj}^2 \\ &= (a_{jj}^2 + a_{j+1,j}^2 + \ldots + a_{nj}^2) + 2a_{jj}c + c^2 \\ &= c^2 + 2a_{jj}c + c^2 = 2c(a_{jj} + c)\end{aligned}$$

ist; damit ist $2/\langle \boldsymbol{v}_j, \boldsymbol{v}_j\rangle = 1/d$ und die Schleife über $k$ ersetzt die $k$-te Spalte $\boldsymbol{a}_k$ von $A$ durch $H_{\boldsymbol{v}_j}\boldsymbol{a}_k$. Die ersten $j-1$ Spalten werden durch die Multiplikation mit $H_{\boldsymbol{v}_j}$ nicht verändert und in der $j$-te Spalte wird alles unterhalb der Diagonalen durch Nullen ersetzt. Es bleibt zu zeigen, dass der $j$-te Eintrag in $H_{\boldsymbol{v}_j}\boldsymbol{a}_j$ gerade $-c$ ist. Dieser Eintrag ist aber

$$a_{jj} - 2\frac{\langle \boldsymbol{a}_j, \boldsymbol{v}\rangle}{\langle \boldsymbol{v}, \boldsymbol{v}\rangle}(\boldsymbol{v}_j)_j = a_{jj} - (a_{jj} + c) = -c\,.$$

(Wir hatten $c$ ja so gewählt, dass der Faktor vor $(\boldsymbol{v}_j)_j$ gleich 1 wird.) Es folgt durch Induktion, dass $R = H_{\boldsymbol{v}_{n-1}} \cdots H_{\boldsymbol{v}_1} A$ ist; $R$ ist offensichtlich eine obere Dreiecksmatrix. Da jede Spiegelungsmatrix orthogonal ist, ist das Produkt der $H_{\boldsymbol{v}_j}$

ebenfalls orthogonal. Damit ist $Q$ orthogonal; die Relation $A = QR$ folgt aus der Definition von $Q$. ❑

In der Praxis wird man sich das Füllen der halben Matrix mit Nullen sparen und nur den oberen Dreiecksteil weiterverarbeiten. Man kann die untere Hälfte dann zum (teilweisen) Speichern der Vektoren $\boldsymbol{v}_j$ verwenden. Dazu braucht die $j$-te Spalte nicht geändert zu werden; man muss lediglich den Wert $a_{jj} + c$ separat speichern, um den $j$-ten Eintrag von $\boldsymbol{v}_j$ rekonstruieren zu können.

Man kann sich überlegen, dass dieser Algorithmus $\frac{4}{3}n^3 + O(n^2)$ Rechenoperationen benötigt. Damit ist er halb so schnell wie die einfacheren Varianten der LR-Zerlegung (Gauß-Elimination) und vergleichbar schnell wie das Gauß-Verfahren mit (Spalten-)Pivotsuche, vermeidet aber die dort auftretenden Probleme mit der Verschlechterung der Kondition. In jedem Fall ist er schneller als Gram-Schmidt.

Der Algorithmus kann auch auf Matrizen $A \in \mathrm{Mat}(m, n, \mathbb{R})$ mit $m \geq n$ und beliebigem Rang angewendet werden. Dazu muss man nur zwei Änderungen vornehmen:

(1) Falls $c = 0$ ist (dann enthält die $j$-te Spalte von $A$ auf und unterhalb der Diagonalen nur Nullen), dann ist $\boldsymbol{v}_j = \mathbf{0}$ und die restlichen Befehle innerhalb der $j$-Schleife werden übersprungen. (Das kann auftreten, wenn $A$ nicht vollen Rang hat.)

(2) Die $j$-Schleife ist im Fall $m > n$ bis $j = n$ zu durchlaufen, da auch in der $n$-ten Spalte unter der Diagonalen Nullen zu erzeugen sind.

Damit liefert der Algorithmus eine orthogonale Matrix $Q \in \mathrm{Mat}(n, \mathbb{R})$, sodass

$$A = QR \qquad \text{mit} \qquad R = \begin{pmatrix} R_1 \\ \mathbf{0} \end{pmatrix}$$

gilt, wobei $R_1 \in \mathrm{Mat}(n, \mathbb{R})$ eine obere Dreiecksmatrix ist.

Dies kann man verwenden, um das Ausgleichsproblem aus Beispiel 4.1 zu lösen. In seiner abstrakten Formulierung ist das Problem Folgendes: Gegeben ist eine Matrix $A \in \mathrm{Mat}(n, m, \mathbb{R})$ und ein Vektor $\boldsymbol{y} \in \mathbb{R}^n$. Gesucht ist ein Vektor $\boldsymbol{x} \in \mathbb{R}^m$, der $\|A\boldsymbol{x} - \boldsymbol{y}\|_2$ minimiert. (In Beispiel 4.1 ist $\boldsymbol{x} = (a_1, \ldots, a_m)^\top$ und die Spalten der Matrix $A$ sind die Vektoren $(f_j(x_1), \ldots, f_j(x_n))^\top$.) Wir hatten gesehen, dass das äquivalent zum Lösen des linearen Gleichungssystems

$$A^\top A\boldsymbol{x} = A^\top \boldsymbol{y}$$

ist. Die Gleichungen dieses Systems heißen auch *Normalgleichungen*, weil sie ausdrücken, dass $A\boldsymbol{x}$ die orthogonale Projektion von $\boldsymbol{y}$ auf den Bildraum von $A$ ist („normal“ bedeutet auch „senkrecht“): $A^\top \boldsymbol{v} = \mathbf{0}$ ist äquivalent dazu, dass $\boldsymbol{v}$ auf allen Spalten von $A$ (und damit auf dem Spaltenraum = Bildraum von $A$) senkrecht steht; das Gleichungssystem ist äquivalent zu $A^\top(A\boldsymbol{x} - \boldsymbol{y}) = \mathbf{0}$. Wir nehmen im Folgenden an, dass $A$ vollen Rang $m$ hat (also insbesondere $m \leq n$); dann ist $A^\top A$ invertierbar und es gibt eine eindeutige Lösung.

Die in den Normalgleichungen auftretende Koeffizientenmatrix $A^\top A$ ist häufig schlecht konditioniert, sodass die Lösung des Ausgleichsproblems durch Umwandlung in das äquivalente lineare Gleichungssystem und anschließender Lösung dieses Systems nicht unbedingt die beste Methode ist. Die QR-Zerlegung erlaubt eine alternative und in dieser Hinsicht bessere Lösung. Sei dazu $A = QR$ wie oben. Dann ist

$$\|A\boldsymbol{x} - \boldsymbol{y}\|_2 = \|Q^\top(A\boldsymbol{x} - \boldsymbol{y})\|_2 = \|R\boldsymbol{x} - Q^\top \boldsymbol{y}\|_2\,,$$

denn $Q^\top = Q^{-1}$ ist orthogonal, ändert also die euklidische Norm nicht. Wir haben

$$R\boldsymbol{x} = \begin{pmatrix} R_1\boldsymbol{x} \\ \mathbf{0} \end{pmatrix}. \qquad \text{Wenn wir} \quad Q^\top \boldsymbol{y} = \begin{pmatrix} \boldsymbol{v}^{(1)} \\ \boldsymbol{v}^{(2)} \end{pmatrix}$$

setzen, wobei $\boldsymbol{v}^{(1)} \in \mathbb{R}^m$ sei, dann ist

$$\|R\boldsymbol{x} - Q^\top \boldsymbol{y}\|_2^2 = \|R_1\boldsymbol{x} - \boldsymbol{v}^{(1)}\|_2^2 + \|\boldsymbol{v}^{(2)}\|_2^2 .$$

Da $A$ vollen Rang hat, ist $R_1$ invertierbar, und $\|R\boldsymbol{x} - Q^\top \boldsymbol{y}\|_2^2$ wird minimal für $\boldsymbol{x} = R_1^{-1}\boldsymbol{v}^{(1)}$. Das führt auf folgenden Algorithmus:

**ALGO**
Ausgleichsrechnung mit QR-Zerlegung

---

**input:** $A \in \mathrm{Mat}(n, m, \mathbb{R})$ mit Rang $m$; $\boldsymbol{y} \in \mathbb{R}^n$

**output:** $\boldsymbol{x} \in \mathbb{R}^m$ mit $\|A\boldsymbol{x} - \boldsymbol{y}\|_2$ minimal

Bestimme $Q \in \mathrm{Mat}(n, \mathbb{R})$ orthogonal und obere Dreiecksmatrix $R_1 \in \mathrm{Mat}(m, \mathbb{R})$ mit $A = Q \begin{pmatrix} R_1 \\ \mathbf{0} \end{pmatrix}$

$Q_1 = (\text{obere } m \text{ Zeilen von } Q^\top) \in \mathrm{Mat}(m, n, \mathbb{R})$

$\boldsymbol{v} = Q_1 \boldsymbol{y} \in \mathbb{R}^m$

Löse $R_1\boldsymbol{x} = \boldsymbol{v}$ durch Rückwärtseinsetzen

**return** $\boldsymbol{x}$

---

## 8. Nichtlineare Gleichungssysteme

Viele in der Praxis auftretenden Probleme sind *nichtlinear:* Statt eines linearen Gleichungssystems $A\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ muss eine allgemeinere Gleichung $f(\boldsymbol{x}) = \boldsymbol{0}$ gelöst werden, wobei $f\colon U \to \mathbb{R}^n$ eine auf einer Teilmenge $U \subset \mathbb{R}^n$ definierte stetige Funktion ist. Ein einfaches Beispiel für $n = 1$ ist die Bestimmung der Quadratwurzel $\sqrt{a}$ einer positiven reellen Zahl $a$, denn das ist äquivalent zur Lösung von $f(x) := x^2 - a = 0$ mit $U = \mathbb{R}_{>0}$.

Da es im Allgemeinen (anders als bei linearen Gleichungssystemen) keine exakte Lösungsformel gibt, die man einfach numerisch „ausrechnen“ kann, braucht man ein Verfahren, das immer bessere Näherungen an die gewünschte Lösung liefert. Ein einfacher Ansatz, um eine Nullstelle einer stetigen reellen Funktion $f\colon [a,b] \to \mathbb{R}$ zu finden, ist das *Bisektionsverfahren.* Dabei setzen wir voraus, dass im Intervall $[a,b]$ ein Vorzeichenwechsel stattfindet, also $f(a)f(b) < 0$.

**ALGO**
Bisektions-
verfahren

---

**input:** $a < b$, $f\colon [a,b] \to \mathbb{R}$ stetig mit $f(a)f(b) < 0$; $\varepsilon > 0$
**output:** $x \in [a,b]$ mit $|x - x^*| \le \varepsilon$ für ein $x^*$ mit $f(x^*) = 0$

$(l, r) := (f(a), f(b))$
**while** $b - a > 2\varepsilon$ **do**
  $m := (a + b)/2$
  $w := f(m)$
  **if** $w = 0$ **then return** $m$ **end if**
  **if** $lw < 0$ **then**
    $(b, r) := (m, w)$
  **else**
    $(a, l) := (m, w)$
  **end if**
**end while**
**return** $(a + b)/2$

---

Man sieht leicht durch Induktion, dass nach Durchlauf der Befehle in der Schleife im (dann neuen) Intervall wieder ein Vorzeichenwechsel von $f$ stattfindet. Die Länge des Intervalls hat sich halbiert, da einer der Endpunkte durch den Mittelpunkt ersetzt wurde. Nach endlich vielen Schritten ist dann die Intervall-Länge $\le 2\varepsilon$. Nach dem Zwischenwertsatz muss $f$ eine Nullstelle $x^*$ in $[a,b]$ haben, die vom Mittelpunkt $x = (a + b)/2$ höchstens den Abstand $\varepsilon$ haben kann.

Das Bisektionsverfahren ist sehr zuverlässig, denn es liefert stets eine Approximation vorgegebener Genauigkeit an eine Nullstelle, wenn die Voraussetzung des Vorzeichenwechsels erfüllt ist. Andererseits ist es relativ langsam: Um eine Genauigkeit von $d$ Nachkommastellen zu erreichen, sind

$$\frac{\log(b-a)}{\log 2} + d\frac{\log 10}{\log 2} \approx \frac{\log(b-a)}{\log 2} + 3{,}322\,d$$

Halbierungsschritte erforderlich.

Ein anderer sehr allgemein (insbesondere auch im Höherdimensionalen) anwendbarer Ansatz für ein Approximationsverfahren nimmt eine gegebene Näherung als Eingabe und produziert eine neue (und hoffentlich bessere) Näherung als Ausgabe.

Formal haben wir eine stetige Funktion $F\colon U \to U$, und wir bilden, ausgehend von einem *Startwert* $\boldsymbol{x}_0 \in U$ die Iterationsfolge

$$\boldsymbol{x}_1 = F(\boldsymbol{x}_0)\,, \quad \boldsymbol{x}_2 = F(\boldsymbol{x}_1)\,, \quad \dots\,, \quad \boldsymbol{x}_{n+1} = F(\boldsymbol{x}_n)\,, \quad \dots$$

und hoffen, dass die Folge $(\boldsymbol{x}_n)$ gegen einen Grenzwert $\boldsymbol{x}^*$ konvergiert, der unser Problem löst. Aus der Stetigkeit von $F$ folgt dann

$$F(\boldsymbol{x}^*) = \boldsymbol{x}^*\,;$$

$\boldsymbol{x}^*$ ist also ein *Fixpunkt* von $F$. Daher nennt man dieses Verfahren auch *Fixpunktiteration*. Damit das funktioniert, sind zwei Dinge zu tun:

(1) Das ursprüngliche Problem $f(\boldsymbol{x}) = \mathbf{0}$ ist äquivalent in ein Fixpunktproblem $F(\boldsymbol{x}) = \boldsymbol{x}$ umzuformen.

(2) Es ist sicherzustellen, dass die Iterationsfolge $(\boldsymbol{x}_n)$ bei geeigneter Wahl des Startwerts $\boldsymbol{x}_0$ konvergiert.

Für Teil (1) gibt es im Allgemeinen viele Möglichkeiten. Zum Beispiel kann man einfach $\boldsymbol{x}$ auf beiden Seiten der Gleichung $f(\boldsymbol{x}) = \mathbf{0}$ addieren; das liefert

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{x}\,.$$

Dass das nicht immer funktioniert, zeigt das folgende Beispiel.

8.1. **Beispiel.** Wir formen $x^2 - 2 = 0$ um zu $F(x) := x^2 + x - 2 = x$. Mit dem Startwert $x_0 = 1$ erhalten wir die Folge **BSP** $\sqrt{2}$ 1. Versuch

$$1\,, \quad F(1) = 0\,, \quad F(0) = -2\,, \quad F(-2) = 0\,, \quad F(0) = -2\,, \quad \dots\,,$$

die offensichtlich nicht konvergiert. Mit dem Startwert $x_0 = 2$ erhalten wir

$$2\,, \quad 4\,, \quad 18\,, \quad 340\,, \quad 115938\,, \quad \dots\,,$$

was noch schlimmer aussieht. Dass hier ein grundsätzliches Problem vorliegt, sehen wir, wenn wir $x_0$ nahe bei $\sqrt{2}$ wählen, also $x_0 = \sqrt{2} + \varepsilon$ mit $\varepsilon$ klein. Dann ist

$$\begin{aligned} F(x_0) &= (\sqrt{2} + \varepsilon)^2 + (\sqrt{2} + \varepsilon) - 2 \\ &= 2 + 2\sqrt{2}\varepsilon + \varepsilon^2 + \sqrt{2} + \varepsilon - 2 \\ &= \sqrt{2} + (2\sqrt{2} + 1 + \varepsilon)\varepsilon\,. \end{aligned}$$

Wenn $|\varepsilon|$ klein ist, dann ist $2\sqrt{2}+1+\varepsilon \approx 2\sqrt{2}+1 \approx 3{,}8284$, also wird der „Fehler“ $|x - \sqrt{2}|$ größer und nicht kleiner! ♣

Bevor wir uns überlegen, wie man diese Schwierigkeit umgehen kann, wollen wir ein wichtiges Kriterium für Teil (2) beweisen.

8.2. **Definition.** Sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$ und sei $F\colon U \to \mathbb{R}^n$ mit $U \subset \mathbb{R}^n$. Dann heißt $F$ eine *Kontraktion* (mit Kontraktionsfaktor $\gamma$), wenn es $\gamma < 1$ gibt, sodass **DEF** Kontraktion

$$\|F(\boldsymbol{x}) - F(\boldsymbol{y})\| \le \gamma \|\boldsymbol{x} - \boldsymbol{y}\|$$

gilt für alle $\boldsymbol{x}, \boldsymbol{y} \in U$. ♢

**SATZ** Banachscher Fixpunktsatz

8.3. **Satz.** *Sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^N$. Seien weiter $A \subset \mathbb{R}^N$ abgeschlossen und $F\colon A \to A$ eine Kontraktion mit Kontraktionsfaktor $\gamma$. Seien $\boldsymbol{x}_0 \in A$ beliebig und $\boldsymbol{x}_{n+1} = F(\boldsymbol{x}_n)$ für $n \geq 0$. Dann gilt:*

(1) *$F$ hat einen eindeutig bestimmten Fixpunkt $\boldsymbol{x}^* \in A$.*

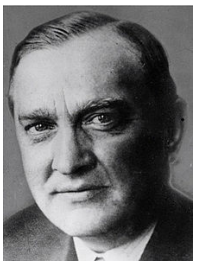
(2) *Die Folge $(\boldsymbol{x}_n)$ konvergiert gegen $\boldsymbol{x}^*$.*

(3) *(A-priori-Fehlerabschätzung)*

$$\|\boldsymbol{x}_n - \boldsymbol{x}^*\| \leq \frac{\gamma^n}{1-\gamma}\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|\,.$$

(4) *(A-posteriori-Fehlerabschätzung)*

$$\|\boldsymbol{x}_n - \boldsymbol{x}^*\| \leq \frac{1}{1-\gamma}\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\|\,.$$



Stefan Banach
1892–1945

*Beweis.* Wir zeigen zuerst die Eindeutigkeit des Fixpunkts. Ist $\boldsymbol{x}$ ein Fixpunkt, dann gilt

$$\|\boldsymbol{x} - \boldsymbol{x}^*\| = \|F(\boldsymbol{x}) - F(\boldsymbol{x}^*)\| \leq \gamma\|\boldsymbol{x} - \boldsymbol{x}^*\|\,,$$

was wegen $\gamma < 1$ nur für $\|\boldsymbol{x} - \boldsymbol{x}^*\| = 0$, also $\boldsymbol{x} = \boldsymbol{x}^*$ möglich ist. Als Nächstes zeigen wir, dass $(\boldsymbol{x}_n)$ konvergiert. Es gilt

$$\|\boldsymbol{x}_{n+2} - \boldsymbol{x}_{n+1}\| = \|F(\boldsymbol{x}_{n+1}) - F(\boldsymbol{x}_n)\| \leq \gamma\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\|$$

und daraus durch Induktion $\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\| \leq \gamma^n\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|$. Damit gilt für $m \geq 0$

$$\begin{aligned}\|\boldsymbol{x}_{n+m} - \boldsymbol{x}_n\| &\leq \|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\| + \|\boldsymbol{x}_{n+2} - \boldsymbol{x}_{n+1}\| + \ldots + \|\boldsymbol{x}_{n+m} - \boldsymbol{x}_{n+m-1}\| \\ &\leq (\gamma^n + \gamma^{n+1} + \ldots + \gamma^{n+m-1})\|\boldsymbol{x}_1 - \boldsymbol{x}_0\| \leq \frac{\gamma^n}{1-\gamma}\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|\,.\end{aligned}$$

Da $\gamma^n \to 0$ für $n \to \infty$, ist $(\boldsymbol{x}_n)$ eine Cauchy-Folge in $\mathbb{R}^N$ und hat daher einen Grenzwert $\boldsymbol{x}^* \in \mathbb{R}^N$. Da alle $\boldsymbol{x}_n$ in $A$ liegen und $A$ abgeschlossen ist, gilt auch $\boldsymbol{x}^* \in A$. Aus der Abschätzung oben erhalten wir für $m \to \infty$ auch schon die Aussage (3). Wir müssen noch zeigen, dass $\boldsymbol{x}^*$ ein Fixpunkt von $F$ ist. Es gilt

$$\|F(\boldsymbol{x}^*) - \boldsymbol{x}^*\| \leq \|F(\boldsymbol{x}^*) - F(\boldsymbol{x}_n)\| + \|\boldsymbol{x}_{n+1} - \boldsymbol{x}^*\| \leq \gamma\|\boldsymbol{x}^* - \boldsymbol{x}_n\| + \|\boldsymbol{x}_{n+1} - \boldsymbol{x}^*\| \to 0$$

für $n \to \infty$, also muss $F(\boldsymbol{x}^*) = \boldsymbol{x}^*$ sein. Schließlich ist Aussage (4) der Fall $n = 0$ von Aussage (3), wenn wir $\boldsymbol{x}_n$ als Startwert nehmen. ❑

Dieser Satz garantiert uns also, dass unsere Fixpunktiteration erfolgreich ist: Unabhängig vom Startwert konvergiert die Iterationsfolge gegen den eindeutigen Fixpunkt. Außerdem haben wir eine Abschätzung für die Geschwindigkeit der Konvergenz: Aus $\gamma$, dem Startwert $\boldsymbol{x}_0$ und $\boldsymbol{x}_1 = F(\boldsymbol{x}_0)$ bekommen wir mit (3) eine Abschätzung für $\|\boldsymbol{x}_n - \boldsymbol{x}^*\|$. Wenn wir $\boldsymbol{x}^*$ bis auf eine (absolute) Genauigkeit von $\varepsilon = 10^{-d}$ approximieren wollen, dann brauchen wir höchstens $N$ Iterationsschritte mit

$$\frac{\gamma^N}{1-\gamma}\|\boldsymbol{x}_1 - \boldsymbol{x}_0\| \leq 10^{-d} \iff \gamma^{-N} \geq 10^d\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|}{1-\gamma}\,,$$

also

$$N \geq \left\lceil \frac{\log\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|}{1-\gamma}}{\log\gamma^{-1}} + d\frac{\log 10}{\log\gamma^{-1}} \right\rceil.$$

Für $\gamma = 1/2$ entspricht die Konvergenzgeschwindigkeit der des Bisektionsverfahrens. Allgemein ist die Konvergenz umso schneller, je kleiner $\gamma$ ist; für $\gamma$ nahe 1 wird die Konvergenz sehr langsam.

Aussage (4) liefert uns eine Abbruchbedingung: Wir iterieren solange, bis

$$\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\| \leq (1-\gamma)\varepsilon$$

ist. Das ist im Allgemeinen für kleineres $n$ der Fall, als die obere Schranke aus Aussage (3) liefert.

Man kann den Satz allgemeiner formulieren für vollständige metrische Räume statt $A \subset \mathbb{R}^n$. Ein metrischer Raum $(X, d)$ ($d\colon X \times X \to \mathbb{R}_{\geq 0}$ ist die Metrik: $d(x, y)$ ist der „Abstand“ von $x$ und $y$) ist *vollständig*, wenn jede Cauchy-Folge in $X$ konvergiert. Eine Abbildung $F\colon X \to X$ ist *kontrahierend* mit Kontraktionsfaktor $\gamma < 1$, wenn $d(F(x), F(y)) \leq \gamma d(x, y)$ gilt für alle $x, y \in X$. Im Wesentlichen derselbe Beweis zeigt, dass $F$ dann einen eindeutigen Fixpunkt in $X$ hat und dass jede Iterationsfolge mit einem beliebigen Startwert in $X$ gegen den Fixpunkt konvergiert; auch die Abschätzungen (3) und (4) gelten in der Form $d(x_n, x^*) \leq \gamma^n/(1-\gamma) \cdot d(x_1, x_0)$ und analog für (4).

Der kritische Punkt dabei ist natürlich, dass wir die Gültigkeit der Voraussetzungen nachweisen müssen, nämlich einerseits $F(A) \subset A$ und vor allem andererseits, dass $F$ auf $A$ kontrahierend ist. Für die Fehlerabschätzung brauchen wir auch (eine obere Schranke für) $\gamma$. Wenn $F$ stetig differenzierbar ist, dann können wir dafür die Ableitung von $F$ heranziehen.

**LEMMA**
Kontraktion wenn $\|DF\| < 1$

8.4. **Lemma.** *Sei $U \subset \mathbb{R}^n$ konvex und sei $F\colon U \to \mathbb{R}^n$ stetig differenzierbar. Sei weiter $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$ mit zugehöriger induzierter Matrixnorm.*

(1) *Ist $\|DF(\boldsymbol{x})\| \leq \gamma < 1$ für alle $\boldsymbol{x} \in U$, dann ist $F$ kontrahierend mit Kontraktionsfaktor $\leq \gamma$.*

(2) *Hat $F$ zusätzlich einen Fixpunkt $\boldsymbol{x}^* \in U$, dann bildet $F$ jede ganz in $U$ gelegene Kugel mit Mittelpunkt $\boldsymbol{x}^*$ in sich ab.*

(3) *Hat $F$ einen Fixpunkt $\boldsymbol{x}^* \in U$ und gilt $\|DF(\boldsymbol{x}^*)\| < 1$, dann gibt es $\varepsilon > 0$, sodass jede Iterationsfolge mit Startwert $\boldsymbol{x}_0$, der $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \leq \varepsilon$ erfüllt, gegen $\boldsymbol{x}^*$ konvergiert.*

*Beweis.*

(1) Für $\boldsymbol{x}, \boldsymbol{y} \in U$ gilt nach dem Hauptsatz der Differential- und Integralrechnung sowie der mehrdimensionalen Kettenregel

$$\begin{aligned}
\|F(\boldsymbol{y}) - F(\boldsymbol{x})\| &= \Big\| \int_0^1 \Big( \frac{d}{dt} F(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) \Big)\, dt \Big\| \\
&= \Big\| \int_0^1 DF(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) \cdot (\boldsymbol{y} - \boldsymbol{x})\, dt \Big\| \\
&\leq \int_0^1 \big\| DF(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) \cdot (\boldsymbol{y} - \boldsymbol{x}) \big\|\, dt \\
&\leq \int_0^1 \|DF(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x}))\| \cdot \|\boldsymbol{y} - \boldsymbol{x}\|\, dt \leq \gamma \|\boldsymbol{y} - \boldsymbol{x}\| \,.
\end{aligned}$$

(2) Aus $\|\boldsymbol{x} - \boldsymbol{x}^*\| \le \delta$ folgt mit (1) $\|F(\boldsymbol{x}) - \boldsymbol{x}^*\| = \|F(\boldsymbol{x}) - F(\boldsymbol{x}^*)\| \le \gamma\delta < \delta$.

(3) Da $DF$ stetig ist, gibt es $\varepsilon > 0$, sodass $\|DF(\boldsymbol{x})\| \le \gamma < 1$ ist für $\|\boldsymbol{x} - \boldsymbol{x}^*\| \le \varepsilon$. Die Aussage folgt dann aus (1) und (2) (mit der $\varepsilon$-Kugel um $\boldsymbol{x}^*$ als Menge $U$) sowie dem Banachschen Fixpunktsatz 8.3. ❑

In Beispiel 8.1 ist $|F'(\sqrt{2})| = |2\sqrt{2} + 1| > 1$ und das Kriterium ist nicht erfüllt. Allgemeiner sehen wir für $F(x) = x + f(x)$, dass wir $F'(x^*) = 1 + f'(x^*)$ betrachten müssen. Das Kriterium ist in diesem Fall genau dann erfüllt, wenn $-2 < f'(x^*) < 0$ ist. Wie können wir die Situation verbessern, wenn das nicht gilt?

Eine Möglichkeit besteht darin, statt $F(x) = x + f(x)$ folgende Iterationsfunktion zu verwenden:

$$F(x) = x + \lambda f(x) \qquad \text{oder} \qquad F(x) = x + \lambda(x) f(x)\,,$$

wobei im ersten Fall $\lambda \neq 0$ eine Konstante ist und im zweiten Fall eine Funktion mit $\lambda(x) \neq 0$ für alle betrachteten $x$. Im ersten Fall ist dann $F'(x^*) = 1 + \lambda f'(x^*)$. Wenn wir das Vorzeichen von $f'(x^*)$ kennen und $\lambda$ genügend klein und mit umgekehrtem Vorzeichen wählen, dann können wir $|F'(x^*)| < 1$ erreichen.

**BSP** $\sqrt{2}$ 2. Versuch

8.5. **Beispiel.** Wir können $2 \le f'(\sqrt{2}) = 2\sqrt{2} \le 4$ abschätzen. Mit $\lambda = -1/4$ und $F(x) = x + \lambda(x^2 - 2)$ ist dann $|F'(\sqrt{2})| < 1$. Mit dem Startwert $x_0 = 2$ erhalten wir die Iterationsfolge (auf zehn Nachkommastellen)

$$2\,,\quad \underline{1},5\,,\quad \underline{1},\underline{4}375\,,\quad \underline{1},\underline{4}208984375\,,\quad \underline{1},\underline{41}61603451\,,\quad \underline{1},\underline{4147}828143\,,$$
$$\underline{1},\underline{4143}802114\,,\quad \underline{1},\underline{4142}623658\,,\quad \underline{1},\underline{4142}278560\,,\quad \underline{1},\underline{41421}77488\,,\quad \underline{1},\underline{41421}47886$$
$$\underline{1},\underline{41421}39215\,,\quad \underline{1},\underline{41421}36676\,,\quad \underline{1},\underline{4142135}932\,,\quad \underline{1},\underline{4142135}714\,,\quad \ldots\,,$$

die tatsächlich gegen $\sqrt{2} \approx 1{,}4142135624$ konvergiert. ♣

Bevor wir uns Gedanken über weitere Verbesserungen machen, wollen wir ein Maß für die Konvergenzgeschwindigkeit einführen.

**DEF** Konvergenzordnung

8.6. **Definition.** Sei $p \ge 1$. Ein Verfahren, das eine Folge von Approximationen $\boldsymbol{x}_n$ an die exakte Lösung $\boldsymbol{x}^*$ eines Problems liefert, heißt *konvergent von Ordnung* $p$, wenn es eine Konstante $C > 0$ gibt, sodass

$$\|\boldsymbol{x}_{n+1} - \boldsymbol{x}^*\| \le C\|\boldsymbol{x}_n - \boldsymbol{x}^*\|^p \qquad \text{für alle } n$$

gilt. Im Fall $p = 1$ wird zusätzlich $C < 1$ verlangt; dann heißt das Verfahren auch *linear konvergent*. Im Fall $p = 2$ heißt das Verfahren auch *quadratisch konvergent*. ♢

Im linear konvergenten Fall erhält man durch Induktion die Abschätzung

$$\|\boldsymbol{x}_n - \boldsymbol{x}^*\| \le C^n \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|\,.$$

Man kann das so interpretieren, dass man in jedem Iterationsschritt etwa gleich viele (nämlich $\log C^{-1}/\log 10$) korrekte Dezimalstellen hinzubekommt. Man kann das in Beispiel 8.5 ganz gut sehen (die korrekten Stellen sind unterstrichen); dort ist (im Limes) $C = 1 - \sqrt{2}/2$ und $\log C^{-1}/\log 10 \approx 0{,}5333$, d.h., man braucht etwa zwei Schritte für eine zusätzliche Dezimalstelle.

Auf das Bisektionsverfahren passt die Definition nicht unmittelbar. Wenn $[a_n, b_n]$ das Intervall nach dem $n$-ten Halbierungsschritt ist, dann gilt

$$|b_{n+1} - a_{n+1}| = \frac{1}{2}|b_n - a_n|\,;$$

das kann man als lineare Konvergenz mit $C = 1/2$ interpretieren.

Im quadratisch konvergenten Fall gilt dagegen $C\|\boldsymbol{x}_{n+1} - \boldsymbol{x}^*\| \le \big(C\|\boldsymbol{x}_n - \boldsymbol{x}^*\|\big)^2$ und daraus mit Induktion

$$\|\boldsymbol{x}_n - \boldsymbol{x}^*\| \le \frac{1}{C}\big(C\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|\big)^{2^n}.$$

Wenn $C\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < 1$ ist (anderenfalls ist die Abschätzung wertlos), dann bedeutet das, dass sich die Anzahl hinzukommender korrekter Dezimalstellen bei jedem Iterationsschritt etwa *verdoppelt:* Man kommt mit wesentlich weniger Iterationsschritten aus als bei einem linear konvergenten Verfahren (vor allem dann, wenn man sehr große Genauigkeit braucht).

Können wir in unserer Fixpunktiteration mit $F(x) = x + \lambda f(x)$ den Parameter $\lambda$ so wählen, dass wir quadratische Konvergenz erhalten? Wir beweisen erst einmal einen allgemeinen Satz.

**SATZ**
Konvergenzordnung der Fixpunktiteration

8.7. **Satz.** *Sei $U \subset \mathbb{R}^n$ und $F\colon U \to U$ kontrahierend mit Kontraktionsfaktor $\gamma$. Dann ist die Fixpunktiteration wie in Satz 8.3 linear konvergent mit $C = \gamma$.*

*Ist $F$ zweimal stetig differenzierbar und gilt $DF(\boldsymbol{x}^*) = \mathbf{0}$, dann ist die Fixpunktiteration in einer hinreichend kleinen Umgebung von $\boldsymbol{x}^*$ quadratisch konvergent.*

*Beweis.* Die erste Aussage folgt unmittelbar aus der Definition von „kontrahierend“. Für die zweite Aussage betrachten wir zunächst eine Komponente $F_j$ von $F$ (d.h., $F_j\colon U \to \mathbb{R}$). Der Satz über die mehrdimensionale Taylor-Entwicklung liefert unter Beachtung von $DF_j(\boldsymbol{x}^*) = \mathbf{0}$ für $\boldsymbol{x}$ nahe bei $\boldsymbol{x}^*$ die Beziehung

$$F_j(\boldsymbol{x}) - x_j^* = F_j(\boldsymbol{x}) - F_j(\boldsymbol{x}^*) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^\top D^2F_j(\boldsymbol{\xi})(\boldsymbol{x} - \boldsymbol{x}^*)\,,$$

wobei $D^2F_j = \big(\frac{\partial^2 F_j}{\partial x_i \partial x_k}\big)_{1\le i,k\le n}$ die Matrix der zweiten Ableitungen von $F_j$ und $\boldsymbol{\xi}$ ein Punkt auf der Verbindungsstrecke von $\boldsymbol{x}^*$ und $\boldsymbol{x}$ ist. Es folgt

$$|F_j(\boldsymbol{x}) - x_j^*| \le \frac{1}{2}\|D^2F_j(\boldsymbol{\xi})\|_2\|\boldsymbol{x} - \boldsymbol{x}^*\|_2^2\,.$$

Da die zweite Ableitung von $F$ nach Voraussetzung stetig ist, gibt es $\varepsilon > 0$, sodass $\|D^2F_j(\boldsymbol{y})\|_2 \le C$ ist für ein $C > 0$ und alle $\boldsymbol{y}$ mit $\|\boldsymbol{y} - \boldsymbol{x}\|_2 \le \varepsilon$ sowie alle $1 \le j \le n$. Quadrieren der obigen Ungleichung und Summieren über $j$ (und anschließendes Ziehen der Quadratwurzel) liefert dann

$$\|F(\boldsymbol{x}) - \boldsymbol{x}^*\|_2 \le \frac{C\sqrt{n}}{2}\|\boldsymbol{x} - \boldsymbol{x}^*\|_2^2 \qquad \text{für } \boldsymbol{x} \text{ mit } \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 \le \varepsilon.$$

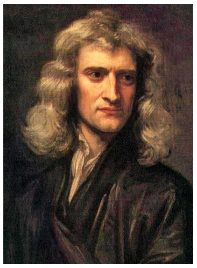
Das zeigt, dass das Verfahren lokal (in der Nähe von $\boldsymbol{x}^*$) bezüglich der euklidischen Norm quadratisch konvergent ist. Da sich alle Normen gegeneinander abschätzen lassen, gilt die Aussage auch für eine beliebige Norm (mit einer i.A. anderen Konstante). ❑

Wenn wir also ein quadratisch konvergentes Verfahren haben wollen, dann müssen wir dafür sorgen, dass die Ableitung $DF(\boldsymbol{x}^*)$ verschwindet. Im eindimensionalen Fall mit $F(x) = x + \lambda f(x)$ bedeutet das

$$0 = F'(x^*) = 1 + \lambda f'(x^*)\,;$$

man muss also $\lambda = -1/f'(x^*)$ setzen (was natürlich $f'(x^*) \neq 0$ voraussetzt; das bedeutet, dass $x^*$ eine einfache Nullstelle von $f$ ist). Das liefert die Vorschrift

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x^*)}\,.$$



Isaac Newton
1643–1727

Dabei haben wir aber das Problem, dass wir $x^*$ nicht kennen (wir wollen diese Zahl ja gerade erst berechnen) und deswegen die Ableitung $f'(x^*)$ nicht bestimmen können. Man behilft sich dadurch, dass man $f'(x^*)$ durch die Ableitung $f'(x_n)$ im aktuellen Näherungswert ersetzt. Wenn die Folge $(x_n)$ konvergiert, dann konvergiert auch $f'(x_n)$ gegen $f'(x^*)$, sodass man immer noch quadratische Konvergenz erwarten kann. Das ergibt das *Newton-Verfahren*

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\,.$$

**BSP**
$\sqrt{2}$
3. Versuch

8.8. **Beispiel.** Wir berechnen $\sqrt{2}$ mit dem Newton-Verfahren. Die Iterationsvorschrift ist

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{1}{2}\Big(x_n + \frac{2}{x_n}\Big) = \frac{x_n}{2} + \frac{1}{x_n}\,.$$

Wir erhalten die Iterationsfolge (auf 29 Nachkommastellen)

$$2\,,\quad \underline{1},5\,,\quad \underline{1},\underline{41}666666666666666666666666667\,,$$
$$\underline{1},\underline{41421}568627450980392156862745\,,\quad \underline{1},\underline{41421356237}4689910626295578 89\,,$$
$$\underline{1},\underline{41421356237309504880168}962350\,,\quad \underline{1},\underline{41421356237309504880168872421}\,;$$

der letzte Wert $x_6$ ist der auf 29 Nachkommastellen gerundete exakte Wert. ♣

**SATZ**
Newton-Verfahren konvergiert lokal quadratisch

8.9. **Satz.** *Sei $U \subset \mathbb{R}$ und sei $f\colon U \to \mathbb{R}$ dreimal stetig differenzierbar mit einer einfachen Nullstelle $x^* \in U$. Dann gibt es $\varepsilon > 0$, sodass für jedes $x_0 \in [x^*-\varepsilon, x^*+\varepsilon]$ die durch*

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

*gegebene Iterationsfolge quadratisch gegen $x^*$ konvergiert.*

*Beweis.* Sei $F(x) = x - f(x)/f'(x)$. Da $x^*$ nach Voraussetzung eine einfache Nullstelle von $f$ ist, gilt $f'(x^*) \neq 0$, und da $f$ insbesondere stetig differenzierbar ist, gilt das auch auf einer Umgebung von $x^*$. Damit ist $F$ in dieser Umgebung definiert und zweimal stetig differenzierbar. Nach Satz 8.7 genügt es nun zu zeigen, dass $F'(x^*) = 0$ ist:

$$F'(x^*) = 1 - \frac{f'(x^*)^2 - f(x^*)f''(x^*)}{f'(x^*)^2} = 1 - 1 + f(x^*)\frac{f''(x^*)}{f'(x^*)^2} = 0$$

wegen $f(x^*) = 0$. ❑

Wir haben angenommen, dass $f$ dreimal stetig differenzierbar ist, um Satz 8.7 verwenden zu können: In der Definition von $F$ tritt die Ableitung von $f$ auf, sodass wir eine Differenzierbarkeitsordnung verlieren. Tatsächlich ist die Aussage aber auch für nur *zweimal* stetig differenzierbares $f$ richtig. Es gilt dann nämlich

$$f(x^* + \Delta x) = f'(x^*)\Delta x + \frac{f''(\xi)}{2}(\Delta x)^2 \qquad \text{und} \qquad f'(x^* + \Delta x) = f'(x^*) + f''(\xi')\Delta x$$

mit $\xi, \xi'$ zwischen $x^*$ und $x^* + \Delta x$. Das liefert

$$F(x^* + \Delta x) = x^* + \frac{\frac{1}{2} f''(\xi) - f''(\xi')}{f'(x^*) + f''(\xi')\Delta x} (\Delta x)^2 ;$$

das ist die quadratische Konvergenz für $\Delta x$ hinreichend klein (und die Konstante $C$ liegt nahe bei $\frac{f''(x^*)}{2f'(x^*)}$).

Man kann zeigen, dass das Newton-Verfahren auch dann noch lokal konvergiert, wenn $x^*$ eine mehrfache Nullstelle von $f$ ist; allerdings ist die Konvergenz dann nur noch linear.

Das Newton-Verfahren lässt sich auch folgendermaßen interpretieren:

(1) Nach dem Motto „Lineare Probleme sind einfach, nichtlineare schwer“ ersetzt man $f(x)$ durch die beste lineare Approximation

$$f(x + \Delta x) \approx f(x) + f'(x)\, \Delta x$$

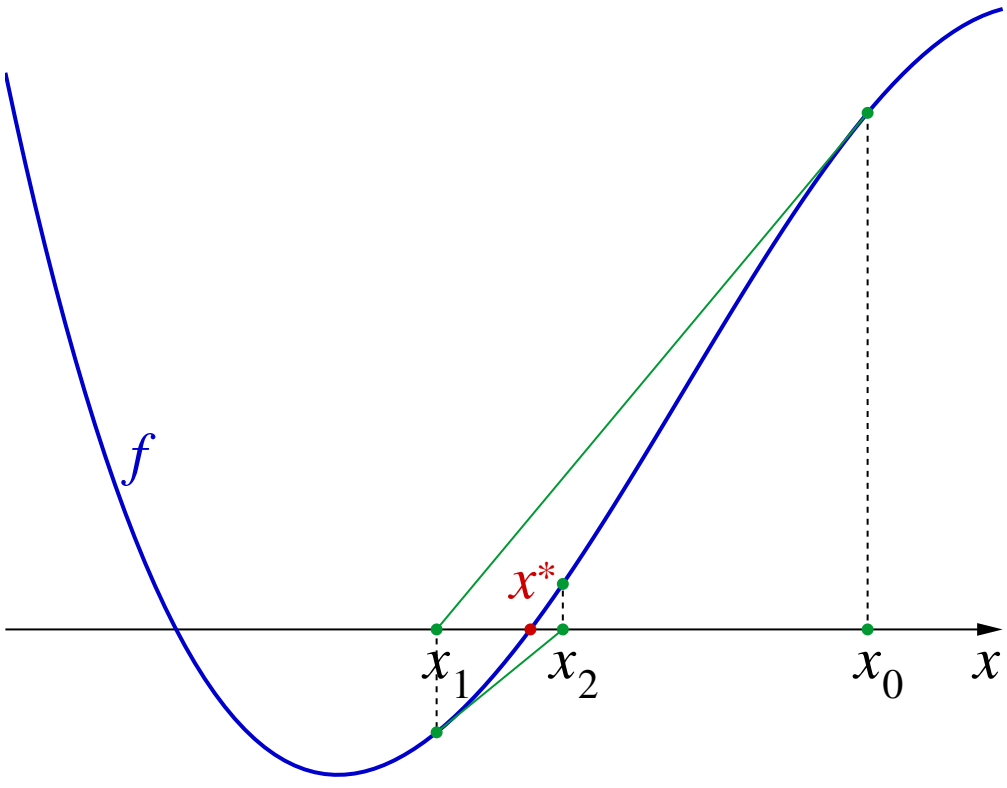
in der Nähe von $x$ und löst die entstehende *lineare* Gleichung

$$f(x) + f'(x)\, \Delta x = 0 \iff \Delta x = -\frac{f(x)}{f'(x)} .$$

Das liefert ausgehend von einer Näherung $x_n$ die neue Näherung

$$x_n + \Delta x = x_n - \frac{f(x_n)}{f'(x_n)} .$$

(2) Geometrisch bedeutet das, dass man die Tangente an den Graph von $f$ im Punkt $(x_n, f(x_n))$ betrachtet und dann $x_{n+1}$ als Schnittpunkt dieser Tangente mit der $x$-Achse definiert.



Gegenüber (z.B.) dem Bisektionsverfahren hat das Newton-Verfahren den großen Vorteil, dass es sehr viel schneller konvergiert. Dem gegenüber steht der Nachteil, dass die Konvergenz nur in der Nähe der Nullstelle garantiert ist, wobei man nicht unbedingt zu Beginn schon entscheiden kann, ob man nahe genug an der Nullstelle startet. In der Praxis verwendet man daher Kombinationen verschiedener Verfahren. Zum Beispiel kann man (wenn man ein $f$ mit Vorzeichenwechsel im betrachteten Intervall hat) mit dem Intervall-Mittelpunkt $x_0$ starten, einen Newton-Schritt ausführen und testen, ob $x_1$ im Intervall liegt und $|f(x_1)|$ kleiner als $|f(x_0)|$ ist. Falls das der Fall ist, kann man einen weiteren Newton-Schritt versuchen; anderenfalls macht man einen Bisektionsschritt.

Ein weiterer Nachteil des Newton-Verfahrens ist, dass man nicht nur $f$, sondern auch die Ableitung $f'$ auswerten muss, was aufwändig sein kann.

Abschließend wollen wir uns überlegen, wie man das Newton-Verfahren auf den höherdimensionalen Fall übertragen kann. Dabei halten wir uns an die Interpretation (1) von oben. Wir betrachten die beste lineare Approximation an $f$ im Punkt $\boldsymbol{x}$:

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + Df(\boldsymbol{x}) \cdot \Delta\boldsymbol{x}$$

(hier ist $Df(\boldsymbol{x})$ die Jacobi-Matrix $(\frac{\partial f_i}{\partial x_j}(\boldsymbol{x}))$ von $f$ in $\boldsymbol{x}$). Dann lösen wir das entstehende lineare Problem und nehmen das Ergebnis als neue Näherung an die Lösung. Das führt auf

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - Df(\boldsymbol{x}_n)^{-1} \cdot f(\boldsymbol{x}_n)$$

und den folgenden Algorithmus:

**ALGO** Newton-Verfahren im $\mathbb{R}^N$

---

**input:** $f\colon \mathbb{R}^N \to \mathbb{R}^N$ zweimal stetig differenzierbar; $\boldsymbol{x}_0 \in \mathbb{R}^N$; $\varepsilon > 0$
**output:** $\boldsymbol{x} \in \mathbb{R}^N$ mit $\|f(\boldsymbol{x})\| \le \varepsilon$

$\boldsymbol{x} := \boldsymbol{x}_0$
$\boldsymbol{b} := f(\boldsymbol{x})$
**while** $\|\boldsymbol{b}\| > \varepsilon$ **do**
  $A := Df(\boldsymbol{x})$
  $\boldsymbol{y} :=$ Lösung des LGS $A \cdot \boldsymbol{y} = \boldsymbol{b}$
  $\boldsymbol{x} := \boldsymbol{x} - \boldsymbol{y}$
  $\boldsymbol{b} := f(\boldsymbol{x})$
**end while**
**return** $\boldsymbol{x}$

---

Da das Verfahren je nach Startwert nicht unbedingt konvergiert, sollte man noch eine Obergrenze für die Zahl der Iterationsschritte festlegen und eine Fehlermeldung ausgeben, wenn sie erreicht wird. Bei ausreichend gutem Startwert haben wir aber wieder quadratische Konvergenz.

**SATZ** lokale Konvergenz von Newton im $\mathbb{R}^N$

8.10. **Satz.** *Sei $U \subset \mathbb{R}^N$ und sei $f\colon U \to \mathbb{R}^N$ zweimal stetig differenzierbar mit einer Nullstelle $\boldsymbol{x}^* \in U$, sodass $Df(\boldsymbol{x}^*)$ invertierbar ist. Dann gibt es $\varepsilon > 0$, sodass für jedes $\boldsymbol{x}_0 \in U$ mit $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \le \varepsilon$ die durch*

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - Df(\boldsymbol{x}_n)^{-1} \cdot f(\boldsymbol{x}_n)$$

*gegebene Iterationsfolge quadratisch gegen $\boldsymbol{x}^*$ konvergiert.*

*Beweis.* Der Beweis geht analog wie für Satz 8.9. Wir nehmen der Einfachheit halber an, dass $f$ sogar dreimal stetig differenzierbar ist; der Satz bleibt aber auch für nur zweimal stetig differenzierbares $f$ richtig. Wir definieren $F$ auf einer Umgebung von $\boldsymbol{x}^*$ durch $F(\boldsymbol{x}) = \boldsymbol{x} - Df(\boldsymbol{x})^{-1} \cdot f(\boldsymbol{x})$; das ist möglich, weil wegen der Stetigkeit von $Df$ die Matrix $Df(\boldsymbol{x})$ auf einer Umgebung von $\boldsymbol{x}^*$ invertierbar ist. Dann ist $F$ zweimal stetig differenzierbar. Wir rechnen wieder nach, dass $DF(\boldsymbol{x}^*) = \mathbf{0}$ ist. Wir haben $Df(\boldsymbol{x}) \cdot (F(\boldsymbol{x}) - \boldsymbol{x}) + f(\boldsymbol{x}) = \mathbf{0}$; das bedeutet ausgeschrieben

$$\sum_{k=1}^{N} \frac{\partial f_i}{\partial x_k}(\boldsymbol{x})\big(F_k(\boldsymbol{x}) - x_k\big) + f_i(\boldsymbol{x}) = 0 \qquad \text{für alle } 1 \le i \le N.$$

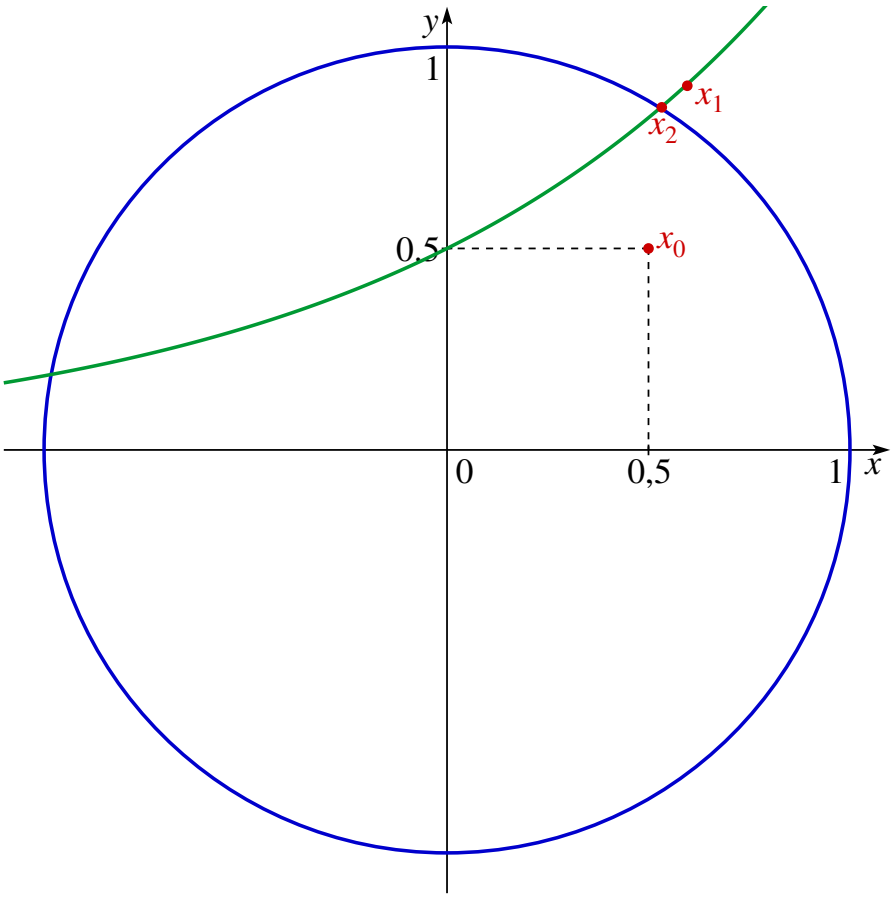
Wenn wir das nach $x_j$ differenzieren, dann bekommen wir

$$\sum_{k=1}^{N}\Big(\frac{\partial^2 f_i}{\partial x_j \partial x_k}(\boldsymbol{x})\big(F_k(\boldsymbol{x}) - x_k\big) + \frac{\partial f_i}{\partial x_k}(\boldsymbol{x})\Big(\frac{\partial F_k}{\partial x_j}(\boldsymbol{x}) - \delta_{jk}\Big)\Big) + \frac{\partial f_i}{\partial x_j}(\boldsymbol{x}) = 0\,.$$

Wegen $F(\boldsymbol{x}^*) = \boldsymbol{x}^*$ verschwindet der erste Term in der Summe für $\boldsymbol{x} = \boldsymbol{x}^*$. Die Summe über den zweiten Term zusammen mit dem Term außerhalb der Summe ergibt für alle $i, j$ (beachte, dass $\delta_{jk} = 1$ ist für $k = j$ und sonst 0)

$$0 = \sum_{k=1}^{N} \frac{\partial f_i}{\partial x_k}(\boldsymbol{x}^*)\frac{\partial F_k}{\partial x_j}(\boldsymbol{x}^*) - \sum_{k=1}^{N} \frac{\partial f_i}{\partial x_k}(\boldsymbol{x}^*)\delta_{jk} + \frac{\partial f_i}{\partial x_j}(\boldsymbol{x}^*) = \big(Df(\boldsymbol{x}^*) \cdot DF(\boldsymbol{x}^*)\big)_{i,j}\,.$$

Das bedeutet $Df(\boldsymbol{x}^*) \cdot DF(\boldsymbol{x}^*) = \mathbf{0}$, und weil $Df(\boldsymbol{x}^*)$ invertierbar ist, folgt $DF(\boldsymbol{x}^*) = \mathbf{0}$. ❑



**BSP**
Newton
2-dimensional

8.11. **Beispiel.** Wir wollen den Schnittpunkt im ersten Quadranten des Einheitskreises mit dem Graph von $x \mapsto e^x/2$ bestimmen. Das ist äquivalent zur Bestimmung der entsprechenden Nullstelle von $f\big((x,y)^\top\big) = (x^2+y^2-1, 2y-e^x)^\top$. Es ist mit $\boldsymbol{x} = (x,y)^\top$

$$Df(\boldsymbol{x}) = \begin{pmatrix} 2x & 2y \\ -e^x & 2 \end{pmatrix} \qquad \text{und damit}$$

$$Df(\boldsymbol{x})^{-1} \cdot f(\boldsymbol{x}) = \frac{1}{4x + 2e^x y} \begin{pmatrix} 2(x^2 - y^2 - 1 + ye^x) \\ e^x(x^2 + y^2 - 1 - 2x) + 4xy \end{pmatrix} .$$

Mit dem Startwert $(x_0, y_0) = (0{,}5, 0{,}5)$ erhalten wir die Folge

$(0{,}596274476244787912371629691194,\quad 0{,}903725523755212087628370308805)$
$(0{,}532953736384934024157195677721,\quad 0{,}850197107895674471334816155399)$
$(0{,}529014886030303299370008107887,\quad 0{,}848623145017623748864019553187)$
$(0{,}529003200643737276289277148623,\quad 0{,}848619828813955600066706991386)$
$(0{,}529003200545318154317624939810,\quad 0{,}848619828788374413157831492965)$
$(0{,}529003200545318154310693723062,\quad 0{,}848619828788374413156059528119)$
$(0{,}529003200545318154310693723062,\quad 0{,}848619828788374413156059528120)$;

das letzte Paar von Werten löst die Gleichungen im Rahmen der Rechengenauigkeit. ♣

# Teil II: Optimierung

Im Bereich der Optimierung werden wir uns exemplarisch mit der Linearen Optimierung befassen. Dabei geht es um die Modellierung, die qualitative und quantitative Analyse und die algorithmische Lösung von Entscheidungsproblemen, deren Ziel die Maximierung (oder Minimierung) einer linearen Zielfunktion unter linearen Nebenbedingungen ist.

## 9. Problemstellung

Als einfaches (und zugegebenermaßen etwas künstliches) Beispiel betrachten wir folgendes Problem:

**BSP** Vorbereitung auf das Staatsexamen

9.1. **Beispiel.** Sie schreiben in vier Wochen die Staatsexamensklausuren in Mathematik: Algebra und Analysis. Um sich den relevanten Stoff in Analysis anzueignen, brauchen Sie zwei Wochen, für den Stoff der Algebra drei. Die beste Note, die Sie dann jeweils bei maximaler Vorbereitungszeit erwarten können, sei eine 2 in Analysis und eine 1 in Algebra. Wir nehmen außerdem an, dass die Note linear von der jeweiligen Vorbereitungszeit abhängt: Ohne Vorbereitung bekommen Sie eine 6, bei einer Woche in Analysis eine 4 usw. Wie sollten Sie Ihre Vorbereitungszeit auf Algebra und Analysis aufteilen, um die beste Durchschnittsnote zu erzielen?

Als erster Schritt muss das Problem mathematisch modelliert werden. Im Beispiel gibt es zwei *Entscheidungsvariablen*, nämlich die Zeit (in Wochen), die Sie für die Vorbereitung auf das Algebra- bzw. das Analysis-Examen verwenden. Wir nennen diese Variablen $x_1$ und $x_2$. Diese Variablen können nicht beliebige Werte annehmen, sondern unterliegen gewissen Nebenbedingungen. Im Beispiel sind das

$$x_1 \geq 0\,, \quad x_2 \geq 0\,, \quad x_1 \leq 3\,, \quad x_2 \leq 2\,, \quad x_1 + x_2 \leq 4\,.$$

Die letzte Bedingung drückt zum Beispiel aus, dass Sie insgesamt nur vier Wochen Zeit haben. Wichtig ist hier, dass diese Nebenbedingungen allesamt *linear* sind: Es wird eine obere oder untere Schranke für den Wert einer Linearkombination der Entscheidungsvariablen festgelegt. Gleichungen wären hier ebenfalls zulässig. Schließlich gibt es die *Zielfunktion*, die optimiert werden soll. Im Beispiel ist sie gegeben als

$$\frac{1}{2}\Big(6 - \frac{5}{3}x_1\Big) + \frac{1}{2}\Big(6 - 2x_2\Big) = 6 - \frac{5}{6}x_1 - x_2$$

und soll minimiert werden (kleinerer Zahlenwert = bessere Note). Für die einzelnen Noten haben wir dabei die Punkte $(0,6)$ und $(3,1)$ (Algebra) bzw. $(0,6)$ und $(2,2)$ (Analysis) durch eine Gerade interpoliert. Äquivalent können wir die Funktion $\frac{5}{6}x_1 + x_2$ maximieren. Wichtig ist wieder, dass die Zielfunktion *linear* ist.

Unser Optimierungsproblem lautet also:

$$\begin{aligned} \text{maximiere} \quad & \tfrac{5}{6}x_1 + x_2 \\ \text{unter den Nebenbedingungen} \quad & x_1 \leq 3 \\ & x_2 \leq 2 \\ & x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

♣

Wir definieren allgemein:

**DEF** Standard-maximierungs-aufgabe der linearen Optimierung

9.2. **Definition.** Die *Standardmaximierungsaufgabe der linearen Optimierung* lautet

$$\begin{array}{rr} \text{maximiere} & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{unter den Nebenbedingungen} & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

Dabei ist $\boldsymbol{x} \in \mathbb{R}^n$ der Vektor der *Entscheidungsvariablen*, $\boldsymbol{c} \in \mathbb{R}^n$ ist der Vektor der Koeffizienten der *Zielfunktion* $\langle \boldsymbol{c}, \boldsymbol{x} \rangle$, die Matrix $A \in \mathrm{Mat}(m, n, \mathbb{R})$ liefert die Koeffizienten und der Vektor $\boldsymbol{b} \in \mathbb{R}^m$ die rechten Seiten der *Nebenbedingungen* $A\boldsymbol{x} \leq \boldsymbol{b}$; dazu gibt es die *Nichtnegativitätsbedingungen* $\boldsymbol{x} \geq \boldsymbol{0}$. Ungleichungen zwischen Vektoren sind komponentenweise zu interpretieren. Kompakter kann man schreiben

$$\max_{\boldsymbol{x}} \{ \langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \} .$$

Man nennt so eine Aufgabe auch ein *Lineares Programm*, kurz LP. ♢

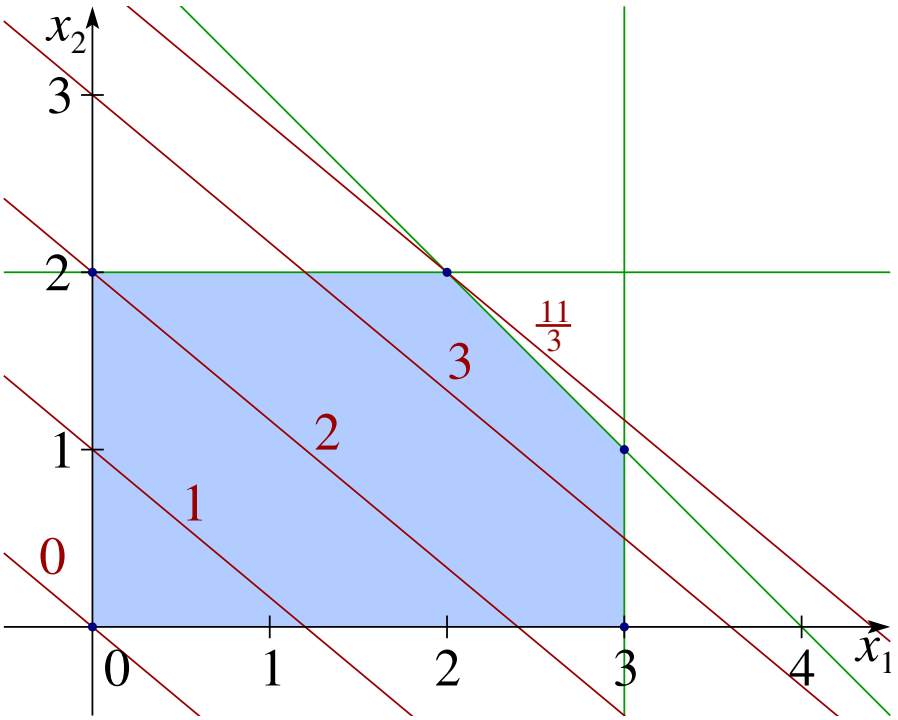
Man kann auch Ungleichungen $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ in die obige Form bringen, denn die Ungleichung ist äquivalent zu $\langle -\boldsymbol{a}, \boldsymbol{x} \rangle \leq -b$. Gleichungen $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ sind äquivalent zu einem Paar von Ungleichungen (mit „$\leq$“ und „$\geq$“).

Im Beispiel oben sind die Daten gegeben durch

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \qquad \boldsymbol{c} = \begin{pmatrix} \frac{5}{6} \\ 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{und} \quad \boldsymbol{b} = \begin{pmatrix} 3 \\ 2 \\ 4 \end{pmatrix} .$$

**BSP** Staatsexamen (Fortsetzung)

9.3. **Beispiel.** Wir wollen uns jetzt im Beispiel anschauen, wie man so eine Optimierungsaufgabe graphisch lösen kann. Dazu skizzieren wir die *zulässige Menge* (englisch *feasible set*) der Vektoren $\boldsymbol{x}$, die den Nebenbedingungen genügen. Zusätzlich zeichnen wir (in rot) Höhenlinien der Zielfunktion ein.



Wir sehen, dass die zulässige Menge hier ein konvexes Polygon ist; im Beispiel ist es nichtleer und beschränkt. Damit ist klar, dass ein Maximum existiert (stetige Funktion auf kompakter Menge). Die Höhenlinien der Zielfunktion sind parallele Geraden; der Wert der Zielfunktion steigt an, wenn man diese Geraden nach rechts oben verschiebt. Damit ist klar, dass das Maximum erreicht wird, wenn die betreffende Gerade mit der zulässigen Menge nichtleeren Schnitt hat, aber alle echt rechts oberhalb gelegenen parallelen Geraden die zulässige Menge nicht mehr treffen. Anschaulich ist klar, dass unter den „letzten“ Schnittpunkten ein *Eckpunkt* des Polygons sein muss. Im Beispiel ist dieser Punkt sogar eindeutig bestimmt; es ist der Punkt $(2,2)$, wo das Maximum $11/3$ der Zielfunktion angenommen wird. Für unser Problem heißt das also, dass Sie die Zeit hälftig zwischen Algebra und Analysis aufteilen sollten; Sie können dann mit einer Durchschnittsnote von $6 - \frac{11}{3} = \frac{7}{3} \approx 2{,}33$ rechnen. ♣

Allgemein gilt, dass eine Ungleichung $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \leq b$ (für $\boldsymbol{a} \neq \boldsymbol{0}$) einen *abgeschlossenen Halbraum* im $\mathbb{R}^n$ definiert, also die Menge der Punkte, die entweder auf der affinen Hyperebene $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ liegen oder auf einer Seite von ihr (und zwar auf der, in die der Normalenvektor $\boldsymbol{a}$ *nicht* zeigt). Das gilt auch für die Nichtnegativitätsbedingungen. Insgesamt ist die zulässige Menge

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \geq \boldsymbol{0}, A\boldsymbol{x} \leq \boldsymbol{b}\}$$

also der Durchschnitt von endlich vielen abgeschlossenen Halbräumen und damit ein Polyeder.

**DEF**
Polyeder
Polytop
Ecke

9.4. **Definition.** Ein *Polyeder* $P$ im $\mathbb{R}^n$ ist der Schnitt von endlich vielen abgeschlossenen Halbräumen:

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$$

für eine Matrix $A \in \mathrm{Mat}(m, n, \mathbb{R})$ und einen Vektor $\boldsymbol{b} \in \mathbb{R}^m$. Ein nichtleeres und beschränktes Polyeder heißt ein *Polytop*.

Sei $\boldsymbol{a}_i$ die $i$-te Zeile von $A$. Sei $\boldsymbol{x}_0 \in P$ und sei $I$ die Menge der Zeilenindizes $i$ mit $\langle \boldsymbol{a}_i, \boldsymbol{x}_0 \rangle = b_i$ (also die Nummern der Ungleichungen, für die in $\boldsymbol{x}_0$ Gleichheit gilt). Dann ist $\boldsymbol{x}_0$ eine *Ecke* von $P$, wenn der Rang der Matrix $A_{\boldsymbol{x}_0}$, deren Zeilen die $\boldsymbol{a}_i$ mit $i \in I$ sind, $n$ ist. ♢

Eine Ecke ist also der Schnittpunkt von $n$ das Polyeder begrenzenden affinen Hyperebenen, deren Normalenvektoren linear unabhängig sind.

Es kann passieren, dass die zulässige Menge leer ist; dann sind die Nebenbedingungen widersprüchlich. Zum Beispiel können die Bedingungen $x_1 \leq 3$, $x_2 \leq 2$ und $-x_1 - x_2 \leq -6$ nicht gleichzeitig erfüllt sein. Dann heißt das zugehörige LP *unzulässig* (englisch *infeasible*).

Es kann auch passieren, dass die zulässige Menge unbeschränkt ist. Ein einfaches Beispiel mit $n = 2$ wäre die einzige Nebenbedingung (außer der Nichtnegativität) $x_1 - x_2 \leq 17$. In diesem Fall ist es möglich, dass die Zielfunktion auf der zulässigen Menge nach oben unbeschränkt ist. Dann gibt es kein Maximum, aber man kann den Wert der Zielfunktion beliebig groß machen. Solche unbeschränkten Probleme kommen in der Praxis (wo es zum Beispiel um Gewinnmaximierung geht) normalerweise nicht vor — noch hat niemand eine Methode entdeckt, wie man unbegrenzt Geld verdienen kann.

Ist beides nicht der Fall, dann ist die zulässige Menge ein Polytop.

Der folgende Satz zeigt, dass allgemein gilt, was wir im Beispiel beobachtet haben.

**SATZ**
Maximum wird in Ecke angenommen

9.5. **Satz.** *Es sei eine Standardmaximierungsaufgabe*

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

*gegeben. Dann tritt genau einer der folgenden Fälle ein.*

(1) *Die zulässige Menge* $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ *ist leer.*

(2) *Die Zielfunktion ist auf der zulässigen Menge nach oben unbeschränkt.*

(3) *Das Maximum der Zielfunktion wird in einer Ecke der zulässigen Menge angenommen.*

*Beweis.* Es ist klar, dass sich die drei Fälle gegenseitig ausschließen.

Man kann zeigen, dass das Bild eines Polyeders unter einer linearen Abbildung wieder ein Polyeder ist. Insbesondere ist das Bild der zulässigen Menge $P$ unter der Zielfunktion ein Polyeder und damit ein abgeschlossenes (aber möglicherweise unbeschränktes) Intervall in $\mathbb{R}$. Ist die Zielfunktion $f$ auf $P$ nach oben beschränkt, dann ist also $\sup f(P) = \max f(P)$, d.h., das Maximum wird angenommen. Zu zeigen ist also nur, dass in diesem Fall das Maximum auch in einer Ecke angenommen wird. Sei dazu $N$ die maximale Zahl linear unabhängiger linker Seiten der definierenden Ungleichungen von $P$, die in einem maximierenden Punkt mit Gleichheit gelten. Wir müssen $N = n$ zeigen. Sei $\boldsymbol{x}_0$ ein maximierender Punkt mit $N$ linear unabhängigen Gleichungen. Ist $N < n$, dann hat die Matrix $A_{\boldsymbol{x}_0}$ dieser Gleichungen nichttrivialen Kern; sei $\boldsymbol{y} \neq \boldsymbol{0}$ ein Vektor in diesem Kern. Für jedes $\lambda \in \mathbb{R}$ erfüllt dann $\boldsymbol{x} = \boldsymbol{x}_0 + \lambda\boldsymbol{y}$ diese Gleichungen ebenfalls. Da die übrigen Ungleichungen in $\boldsymbol{x}_0$ strikt gelten, bleiben sie jedenfalls für kleines $|\lambda|$ ebenfalls erhalten. Die Zielfunktion hat für diese Punkte die Form $f(\boldsymbol{x}_0 + \lambda\boldsymbol{y}) = c_0 + c_1\lambda$. Da $\boldsymbol{x}_0$ ein maximierender Punkt ist, muss $c_1 = 0$ sein (sonst könnte man durch geeignete Wahl von $\lambda$ die Zielfunktion vergrößern), sodass die Zielfunktion auf der Geraden $\boldsymbol{x}_0 + \mathbb{R}\boldsymbol{y}$ konstant ist. Da die Koordinaten in $P$ alle nichtnegativ sind, kann nicht die gesamte Gerade in $P$ liegen, also gibt es ein maximales oder ein minimales $\lambda = \lambda_0$ (oder beides) mit $\boldsymbol{x} = \boldsymbol{x}_0 + \lambda_0\boldsymbol{y} \in P$. Für den Punkt $\boldsymbol{x}$ muss dann eine weitere linear unabhängige Gleichung gelten, im Widerspruch zur Wahl von $N$. ❑

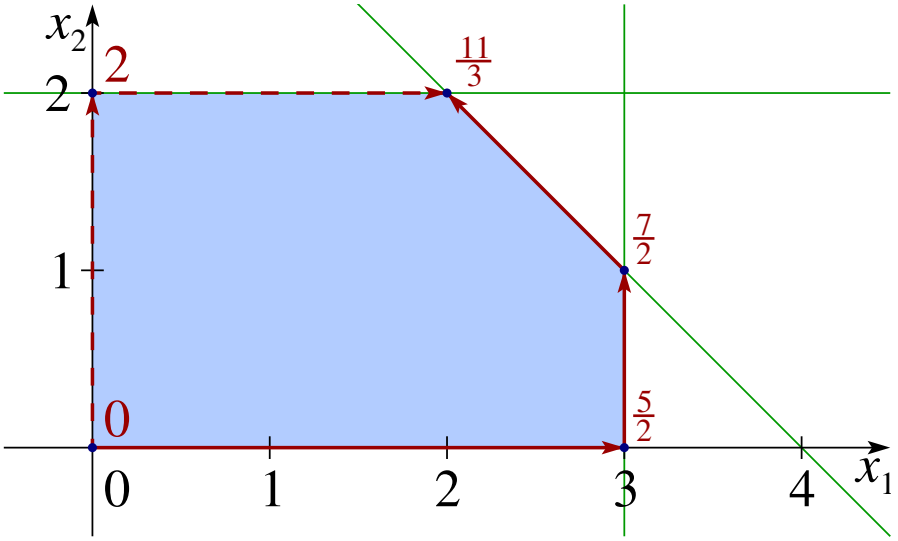
Wenn wir wissen, dass die Zielfunktion beschränkt ist, dann zeigt dieser Satz, dass die Lösung der Maximierungsaufgabe ein endliches Problem ist: Wir bestimmen alle Ecken der zulässigen Menge (davon gibt es höchstens $\binom{n+m}{n}$, denn jede Ecke ist durch $n$ linear unabhängige Gleichungen aus einer Menge von $n+m$ Gleichungen gegeben), werten die Zielfunktion dort aus und wählen eine Ecke mit dem maximalen Wert. Der Aufwand dafür wächst allerdings mit wachsenden $n$ und $m$ sehr stark an, so dass dieser Ansatz nicht praktikabel ist. Im nächsten Abschnitt werden wir ein Verfahren kennenlernen, das im Normalfall deutlich effizienter ist.

## 10. Das Simplexverfahren

Wie können wir das Optimum effizienter finden als dadurch, alle Ecken der zulässigen Menge zu finden und die Werte der Zielfunktion dort zu vergleichen? Eine Möglichkeit ist, in einer Ecke zu starten (in vielen Fällen ist $\mathbf{0}$ zulässig und dann in jedem Fall eine Ecke, da die $n$ Nichtnegativitätsbedingungen „scharf“ sind, die offensichtlich linear unabhängig sind) und dann entlang der Kanten zu benachbarten Ecken zu gehen, solange sich der Wert der Zielfunktion auf diese Weise vergrößern lässt. Ist das nicht mehr der Fall, dann muss in der zuletzt besuchten Ecke das Maximum angenommen werden.

10.1. **Beispiel.** Wir illustrieren das an unserem Beispiel.

**BSP**
Staatsexamen
(Kantenzug)




Wir starten in der Ecke $\mathbf{0}$. Dort gibt es zwei Kanten, und entlang beider wächst die Zielfunktion (Werte der Zielfunktion in den Ecken sind in rot angegeben). Wir haben also die Wahl. Wir könnten uns zum Beispiel für die Kante entscheiden, deren anderes Ende den höheren Wert der Zielfunktion aufweist. Dann würden wir zur Ecke $(3, 0)$ gehen (durchgezogene rote Pfeile), wo die Zielfunktion den Wert $5/2 = 2{,}5$ hat. Von dort gibt es genau eine Kante mit wachsender Zielfunktion; sie führt uns nach $(3, 1)$ mit dem Wert $7/2 = 3{,}5$. Von dort gibt es wiederum eine Kante mit wachsender Zielfunktion, und wir landen in $(2, 2)$ mit Wert $11/3 \approx 3{,}67$. Alle von dort ausgehenden Kanten verringern die Zielfunktion, also ist dort das Optimum.

Alternativ könnten wir zu Beginn die Kante auswählen, entlang derer die Zielfunktion am schnellsten wächst. Das ist die senkrechte Kante (gestrichelte rote Pfeile), denn der Koeffizient von $x_2$ in der Zielfunktion $\frac{5}{6}x_1 + x_2$ ist größer als der von $x_1$. Das führt uns über $(0, 2)$ mit Wert 2 zum Optimum $(2, 2)$. ♣

Wie kann man das in einen Algorithmus umsetzen?

Eine erste Überlegung ist, dass die Ecken durch eine Auswahl von $n$ linear unabhängigen Ungleichungen gegeben sind, die in der jeweiligen Ecke scharf sind. Für die Nichtnegativitätsbedingungen heißt das einfach, dass die entsprechende Variable verschwindet. Um die anderen Ungleichungen auf die gleiche Weise behandeln zu können, führt man sogenannte *Schlupfvariablen* ein: Die Ungleichung

$$\langle \boldsymbol{a}, \boldsymbol{x} \rangle \leq b$$

wird mit Hilfe einer neuen Variablen $w$ äquivalent umgeformt zu

$$\langle \boldsymbol{a}, \boldsymbol{x} \rangle + w = b\,, \qquad w \geq 0\,.$$

Die neue Variable $w$ misst den „Schlupf“ in der Ungleichung, gibt also an, wie weit die Ungleichung davon entfernt ist, scharf zu sein. Die Ungleichung ist insbesondere genau dann scharf, wenn $w = 0$ ist. Ist $\boldsymbol{w} \in \mathbb{R}^m$ der Vektor der Schlupfvariablen, dann ist unser LP

$$\max_{\boldsymbol{x}} \{ \langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \}$$

äquivalent zu

$$\max_{\boldsymbol{x},\boldsymbol{w}} \{ \langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} + \boldsymbol{w} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{w} \geq \boldsymbol{0} \}\,.$$

In dieser Form ist eine Ecke dadurch gegeben, dass $n$ Einträge des aus $\boldsymbol{x}$ und $\boldsymbol{w}$ gebildeten Vektors verschwinden. Diese Einträge haben die Eigenschaft, dass sich das lineare Gleichungssystem $A\boldsymbol{x} + \boldsymbol{w} = \boldsymbol{b}$ nach den übrigen Variablen auflösen lässt (das ist die Bedingung der linearen Unabhängigkeit). Diese übrigen Variablen werden *Basisvariablen* genannt, die anderen (deren Nullsetzen die Ecke definiert) entsprechend *Nichtbasisvariablen*. Der Übergang von einer Ecke zur nächsten entspricht dann dem Austausch von zwei Variablen zwischen diesen Mengen: Eine Basisvariable wird zur Nichtbasisvariablen und eine Nichtbasisvariable wird zur Basisvariablen.

Wenn wir mit $\boldsymbol{x}_B$ den Vektor der Basisvariablen und mit $\boldsymbol{x}_N$ den Vektor der Nichtbasisvariablen bezeichnen, dann lassen sich die relevanten Daten für eine Ecke schreiben als

$$\boldsymbol{x}_B = \tilde{\boldsymbol{b}} - \tilde{A}\boldsymbol{x}_N\,, \qquad z = z_0 + \langle \tilde{\boldsymbol{c}}, \boldsymbol{x}_N \rangle\,;$$

dabei steht $z$ für die Zielfunktion (und kann formal als eine weitere Basisvariable betrachtet werden, die jedoch niemals zur Nichtbasisvariablen werden kann). Diese Darstellung heißt ein *Verzeichnis* (englisch *dictionary*). Der durch das Verschwinden der Nichtbasisvariablen definierte Punkt ist genau dann zulässig, wenn $\tilde{\boldsymbol{b}} \geq \boldsymbol{0}$ ist, denn die Werte der Basisvariablen ergeben sich, wenn man $\boldsymbol{x}_N = \boldsymbol{0}$ setzt. Die Zielfunktion hat entsprechend in dieser Ecke den Wert $z_0$.

Sind die Koeffizienten von $\tilde{\boldsymbol{c}}$ alle $\leq 0$, dann kann $z$ durch Verlassen der aktuellen Ecke nicht mehr vergrößert werden (denn $\boldsymbol{x}_N \geq \boldsymbol{0}$); wir haben also ein Optimum gefunden. Anderenfalls wählen wir eine Nichtbasisvariable $x_j$ mit echt positivem Koeffizienten in $\tilde{\boldsymbol{c}}$. Durch Vergrößern dieser Variable vom Startwert 0 wird dann $z$ wachsen. Wir bestimmen das Maximum von $x_j$, sodass der entsprechende Punkt noch zulässig ist. Dies ist dadurch gegeben, dass erstmalig mindestens eine der Basisvariablen das Vorzeichen (von positiv nach negativ) wechselt. Eine dieser Basisvariablen wird dann die neue Nichtbasisvariable. Das Verzeichnis muss dann so umgeformt werden, dass alles durch die neuen Nichtbasisvariablen ausgedrückt wird.

10.2. **Beispiel.** Wir führen das Verfahren an unserem Beispiel durch. Wegen $\boldsymbol{b} \geq \boldsymbol{0}$ ist der Nullpunkt zulässig und damit eine Ecke der zulässigen Menge. Das zugehörige Verzeichnis sieht so aus (die Schlupfvariablen sind $w_1$, $w_2$, $w_3$):

**BSP**
Staatsexamen
(Simplexverf.)

$$\begin{array}{rcrcrcr}
z & = & & & \frac{5}{6}x_1 & + & x_2 \\
\hline
w_1 & = & 3 & - & x_1 & & \\
w_2 & = & 2 & & & - & x_2 \\
w_3 & = & 4 & - & x_1 & - & x_2
\end{array}$$

Beide Nichtbasisvariablen $x_1$ und $x_2$ haben positive Koeffizienten in der Zielfunktion. Wir müssen also eine Auswahl treffen. Wir entscheiden uns für $x_2$. Jetzt müssen wir feststellen, um wie viel wir $x_2$ erhöhen dürfen, ohne die zulässige Menge zu verlassen. Die Bedingungen dafür sind $w_1, w_2, w_3 \geq 0$, also (unter Beachtung von $x_1 = 0$)

$$0 \leq 3\,, \qquad x_2 \leq 2\,, \qquad x_2 \leq 4\,.$$

Die zweite Bedingung, die von $w_2$ kommt, ist die einschränkendste, also wird jetzt $x_2 = 2$ und $w_2 = 0$. Die neuen Nichtbasisvariablen sind $x_1$ und $w_2$. Wir können $x_2$ durch $w_2$ und die andere Nichtbasisvariable $x_1$ ausdrücken: $x_2 = 2 - w_2$. Das setzen wir in die übrigen Gleichungen ein; dann erhalten wir das neue Verzeichnis

$$\begin{array}{rcrcrcr} z & = & 2 & + & \frac{5}{6}x_1 & - & w_2 \\ \hline w_1 & = & 3 & - & x_1 & & \\ x_2 & = & 2 & & & - & w_2 \\ w_3 & = & 2 & - & x_1 & + & w_2 \end{array}$$

Jetzt ist $x_1$ die einzige Nichtbasisvariable mit positivem Koeffizienten in der Zielfunktion. Die Bedingungen an $x_1$ sind

$$x_1 \leq 3\,, \qquad 0 \leq 2\,, \qquad x_1 \leq 2\,.$$

Die letzte Bedingung (von $w_3$) ist entscheidend; wir setzen $x_1 = 2$, $w_3 = 0$. Mit $x_1 = 2 - w_3 + w_2$ bekommen wir dann das Verzeichnis

$$\begin{array}{rcrcrcr} z & = & \frac{11}{3} & - & \frac{5}{6}w_3 & - & \frac{1}{6}w_2 \\ \hline w_1 & = & 1 & + & w_3 & - & w_2 \\ x_2 & = & 2 & & & - & w_2 \\ x_1 & = & 2 & - & w_3 & + & w_2 \end{array}$$

Jetzt sind alle Koeffizienten der Zielfunktion negativ, also haben wir das Optimum erreicht, und zwar mit $(x_1, x_2; w_1, w_2, w_3) = (2, 2; 1, 0, 0)$ mit $z = \frac{11}{3}$. ♣

Das allgemeine Verfahren, von einem Verzeichnis

$$\boldsymbol{x}_B = \tilde{\boldsymbol{b}} - \tilde{A}\boldsymbol{x}_N\,, \qquad z = z_0 + \langle \tilde{\boldsymbol{c}}, \boldsymbol{x}_N \rangle$$

zum nächsten zu gelangen, sieht dann so aus. Wir schreiben $B$ für die Menge der Indizes der Basisvariablen und $N$ für die Menge der Indizes der Nichtbasisvariablen.

(1) Ist $\tilde{\boldsymbol{c}} \leq \boldsymbol{0}$, dann haben wir bereits das Maximum der Zielfunktion erreicht.

(2) Sonst wähle $j \in N$ mit $\tilde{c}_j > 0$ (*Spaltenpivot*). Variable $x_j$ wird von einer Nichtbasisvariablen zu einer Basisvariablen.

(3) Auf der Kante, an der entlang wir uns bewegen wollen, gilt $x_i = 0$ für alle $i \in N \setminus \{j\}$ und $x_j \geq 0$. Die Nebenbedingungen $x_k \geq 0$ für $k \in B$ lauten dann $\tilde{b}_k - \tilde{a}_{kj}x_j \geq 0$. Da $\tilde{\boldsymbol{b}} \geq \boldsymbol{0}$ ist (die Ecke ist zulässig), ist das nur dann eine Einschränkung, wenn $\tilde{a}_{kj} > 0$ ist; in diesem Fall ist $q_k = \tilde{b}_k/\tilde{a}_{kj} \geq 0$ eine obere Schranke für $x_j$. Sonst setzt man $q_k = +\infty$. $q_k$ heißt der *Engpassquotient* zu $x_k$.

(4) Sind alle $q_k = \infty$, dann ist das LP unbeschränkt, denn wir können $x_j$ und damit die Zielfunktion beliebig groß machen. Anderenfalls wählen wir ein $i \in B$ mit $q_i = \min_{k\in B} q_k$ (*Zeilenpivot*); $x_i$ wird von einer Basisvariablen zur Nichtbasisvariablen.

(5) Wir drücken $x_j$ durch $x_i$ und die alten Nichtbasisvariablen aus:

$$x_j = \tilde{a}_{ij}^{-1}\Big(\tilde{b}_i - x_i - \sum_{l\in N\setminus\{j\}} \tilde{a}_{il}x_k\Big)$$

und verwenden diese Relation, um $x_j$ aus den anderen rechten Seiten und der Zielfunktion zu eliminieren. Das sind im Wesentlichen Zeilenoperationen an der Matrix $\tilde{A}$.

Das Simplexverfahren mit Verzeichnissen sieht dann so aus:

**ALGO**
Simplex-verfahren mit Verzeichnissen

---

**input:** $A \in \mathrm{Mat}(m,n,\mathbb{R})$, $\boldsymbol{b} \in \mathbb{R}^m$ mit $\boldsymbol{b} \geq \boldsymbol{0}$, $\boldsymbol{c} \in \mathbb{R}^n$
**output:** $\boldsymbol{x} \in \mathbb{R}^n$ mit $A\boldsymbol{x} \leq \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$ und $\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ maximal

$B := \{n+1, \ldots, n+m\}$; $N := \{1, \ldots, n\}$
$z_0 := 0$
**while** $\boldsymbol{c} \not\leq \boldsymbol{0}$ **do**
  führe einen Simplex-Schritt aus ($\to B$, $N$, $A$, $\boldsymbol{b}$, $\boldsymbol{c}$, $z_0$ werden ersetzt)
**end while**
**for** $i = 1$ **to** $n$ **do**
  **if** $i \in N$ **then** $x_i := 0$ **else** $x_i := b_i$ **end if**
**end for**
**return** $(x_1, \ldots, x_n)^\top$

---

Die Voraussetzung $\boldsymbol{b} \geq \boldsymbol{0}$ stellt sicher, dass $\boldsymbol{x} = \boldsymbol{0}$ eine zulässige Ecke ist. Gilt dies nicht, muss man erst einmal eine zulässige Ecke finden (oder feststellen, dass die zulässige Menge leer ist). Methoden dafür werden wir später betrachten.

Wird im Simplex-Schritt festgestellt, dass das Problem unbeschränkt ist, sollte das Verfahren mit einer entsprechenden Meldung abbrechen.

Bevor wir das Verhalten des Simplexverfahrens weiter diskutieren, wollen wir uns eine etwas handlichere Version überlegen. Dazu packen wir die gesamte Information in eine Matrix: Dem linearen Gleichungssystem

$$A\boldsymbol{x} + \boldsymbol{w} + \boldsymbol{0}z = \boldsymbol{b}\,, \qquad \boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{0}^\top\boldsymbol{w} - z = 0$$

(wobei wir $z$ als weitere Variable betrachten) entspricht die erweiterte Koeffizientenmatrix

$$\left(\begin{array}{ccc|c} A & I_m & \boldsymbol{0}_m & \boldsymbol{b} \\ \hline \boldsymbol{c}^\top & \boldsymbol{0}_m^\top & -1 & 0 \end{array}\right).$$

Zusätzlich beschriften wir die Spalten mit den zugehörigen Variablen bzw. „RS“ für die rechte Seite und fügen je eine Spalte links („BV“ für die Basisvariablen) und rechts („EQ“ für die Engpassquotienten) an. Das so entstandene Schema heißt *Tableau*.

10.3. **Beispiel.** In unserem Standardbeispiel sieht das dann so aus:

**BSP**
Staatsexamen Start-Tableau

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS | EQ |
|---|---|---|---|---|---|---|---|---|
| $w_1$ | 1 | 0 | 1 | 0 | 0 | 0 | 3 | |
| $w_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| $w_3$ | 1 | 1 | 0 | 0 | 1 | 0 | 4 | |
| $z$ | $\frac{5}{6}$ | 1 | 0 | 0 | 0 | $-1$ | 0 | |

Mit den Engpassquotienten für $x_1$ (der Abwechslung halber) bekommt man

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS | EQ |
|---|---|---|---|---|---|---|---|---|
| $w_1$ | **1** | 0 | 1 | 0 | 0 | 0 | 3 | **3** |
| $w_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 2 | $\infty$ |
| $w_3$ | 1 | 1 | 0 | 0 | 1 | 0 | 4 | 4 |
| $z$ | $\frac{5}{6}$ | 1 | 0 | 0 | 0 | $-1$ | 0 | |

Zum Spaltenpivot $x_1$ gehört also das Zeilenpivot $w_1$. Um zum nächsten Tableau zu gelangen, wechselt die Markierung der ersten Zeile von $w_1$ zu $x_1$; wir müssen dann geeignete Vielfache der ersten Zeile von den anderen abziehen, um in der Spalte $x_1$ den entsprechenden Standard-Basisvektor $\mathbf{e}_1$ zu bekommen. Das liefert

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS | EQ |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 1 | 0 | 0 | 0 | 3 | $\infty$ |
| $w_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 2 |
| $w_3$ | 0 | **1** | $-1$ | 0 | 1 | 0 | 1 | **1** |
| $z$ | 0 | 1 | $-\frac{5}{6}$ | 0 | 0 | $-1$ | $-\frac{5}{2}$ | |

(schon mit den Engpassquotienten für das einzig mögliche Spaltenpivot $x_2$). Zeilenpivot ist dann $w_3$. Die dritte Zeile wird also statt $w_3$ jetzt neu $x_2$ zugeordnet und in der Spalte $x_2$ der entsprechende Standard-Basisvektor $\mathbf{e}_3$ erzeugt:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS | EQ |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 1 | 0 | 0 | 0 | 3 | 3 |
| $w_2$ | 0 | 0 | **1** | 1 | $-1$ | 0 | 1 | **1** |
| $x_2$ | 0 | 1 | $-1$ | 0 | 1 | 0 | 1 | $\infty$ |
| $z$ | 0 | 0 | $\frac{1}{6}$ | 0 | $-1$ | $-1$ | $-\frac{7}{2}$ | |

Jetzt ist $w_1$ Spaltenpivot mit dem Zeilenpivot $w_2$. Im nächsten Schritt wird das Maximum erreicht:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS | EQ |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 0 | $-1$ | 1 | 0 | 2 | |
| $w_1$ | 0 | 0 | 1 | 1 | $-1$ | 0 | 1 | |
| $x_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| $z$ | 0 | 0 | 0 | $-\frac{1}{6}$ | $-\frac{5}{6}$ | $-1$ | $-\frac{11}{3}$ | |

Man beachte, dass der Wert der Zielfunktion in dieser Darstellung *negativ* in der rechten unteren Ecke der Matrix auftritt. ♣

Das lässt sich jetzt relativ schlank als Algorithmus formulieren. Die Spalte „$z$“ wird niemals verändert, kann also auch weggelassen werden. Wir schreiben $M_{i,*}$ für die $i$-te Zeile der Matrix $M$.

**ALGO** Simplex-verfahren mit Tableaus

**input:** $A \in \mathrm{Mat}(m,n,\mathbb{R})$, $\boldsymbol{b} \in \mathbb{R}^m$ mit $\boldsymbol{b} \geq \mathbf{0}$, $\boldsymbol{c} \in \mathbb{R}^n$
**output:** $\boldsymbol{x} \in \mathbb{R}^n$ mit $A\boldsymbol{x} \leq \boldsymbol{b}$, $\boldsymbol{x} \geq \mathbf{0}$ und $\langle \boldsymbol{c}, \boldsymbol{x} \rangle$ maximal, oder „unbeschränkt“

$$M := \begin{pmatrix} A & I_m & \boldsymbol{b} \\ \boldsymbol{c}^\top & \mathbf{0}_m^\top & 0 \end{pmatrix}$$
$B := (n+1, \ldots, n+m)$ *// Indizes der Basisvariablen*
**while** $M_{m+1,*} \not\leq \mathbf{0}^\top$ **do** *// Optimalitätstest*
  wähle $j \leq m+n$ mit $M_{m+1,j} > 0$ *// Spaltenpivot*
  $I := \{i \in \{1, \ldots, m\} \mid M_{i,j} > 0\}$

**if** $I = \emptyset$ **then return** „unbeschränkt“ **end if**
$q := \min\{M_{i,m+n+1}/M_{i,j} \mid i \in I\}$ // *Engpassquotient*
wähle $i \in I$ mit $q = M_{i,m+n+1}/M_{i,j}$ // *Zeilenpivot*
$M_{i,*} := \frac{1}{M_{i,j}} M_{i,*}$ // *Eintrag* $M_{ij} = 1$ *machen*
**for** $k = 1$ **to** $m + 1$ **do** // *j-te Spalte ausräumen*
**if** $k \neq i$ **then** $M_{k,*} := M_{k,*} - M_{k,j} M_{i,*}$ **end if**
**end for**
$B_i := j$ // *neue Basisvariable notieren*
**end while**
**for** $j = 1$ **to** $n$ **do** $x_j := 0$ **end for**
**for** $i = 1$ **to** $m$ **do**
**if** $B_i \leq n$ **then** $x_{B_i} := M_{i,m+n+1}$ **end if**
**end for**
**return** $(x_1, \ldots, x_n)^\top$

---

Damit das Ganze wirklich ein Algorithmus wird, muss man festlegen, wie man Spalten- und Zeilenpivot auswählt, falls es mehrere Möglichkeiten gibt. Bei manchen Vorschriften dafür kann es vorkommen, dass sich der Algorithmus in einer Endlosschleife verfängt. (Beispiel als Übungsaufgabe.) In diesem Fall muss man in derselben Ecke verharren; es werden nur Variablen zwischen den Basis- und Nichtbasisvariablen hin- und hergeschoben, ohne dass sich die Werte ändern (d.h., der Wert ist für alle betroffenen Variablen null). Das bedeutet, dass der minimale Engpassquotient jeweils null ist. Man hat dann also eine Ecke, in der *mehr* als $n$ Nebenbedingungen scharf sind. Eine solche Ecke heißt auch *degeneriert*. Bei zufällig gewählten Maximierungsaufgaben kommen degenerierte Ecken praktisch nicht vor, bei Problemen aus der Praxis sind sie jedoch häufig.

Eine degenerierte Ecke eines Polytops im $\mathbb{R}^3$ kann zum Beispiel aussehen wie eine Ecke eines regulären Oktaeders (dort schneiden sich jeweils vier Begrenzungsebenen) oder eines regulären Ikosaeders (dort sind es sogar fünf). Endlosschleifen im Simplexverfahren sind allerdings erst ab vierdimensionalen Problemen möglich.

Um dieses Problem zu umgehen, gibt es zwei Möglichkeiten:

(1) Man „stört“ das Problem ein wenig, indem man die rechten Seiten leicht verändert. Das führt dazu, dass sich die degenerierten Ecken in mehrere normale Ecken aufspalten; damit sind keine Endlosschleifen mehr möglich. Man kann dabei die Störungen rein formal (als Variablen mit infinitesimal kleinen Werten) behandeln.

(2) Man verwendet eine Pivot-Regel, die Endlosschleifen vermeidet. Eine solche Regel ist zum Beispiel, immer die Variable mit dem kleinstmöglichen Index zu nehmen.

Da in allen anderen Fällen der Wert der Zielfunktion echt zunimmt und es nur endlich viele Ecken gibt, muss das Simplexverfahren terminieren, wenn es nicht in einer Ecke hängen bleibt.

Eine andere wichtige Frage ist die nach der Effizienz. In dem oben formulierten Algorithmus wird in jedem Simplex-Schritt die ganze Matrix $M$ „angefasst“; die Komplexität eines solchen Schritts ist also von der Ordnung $(m + 1)(m + n + 1)$ (das ist die Größe von $M$). Für die Effizienz entscheidend ist dann die Anzahl der Simplex-Schritte, die man durchführen muss.

Da es exponentiell viele Ecken geben kann (zum Beispiel wird ein Würfel im $\mathbb{R}^n$ durch $2n$ Ungleichungen beschrieben, hat aber $2^n$ Ecken), ist nicht von vornherein klar, dass das Simplexverfahren eine vernünftige Komplexität hat. Tatsächlich ist die schlechte Nachricht, dass man für praktisch jede Pivot-Regel ein Beispiel konstruieren kann, in dem das Verfahren exponentiell viele Ecken durchläuft. Wenn man als Spaltenpivot die Variable mit dem größten Koeffizienten in der Zielfunktion nimmt (was naheliegend ist), dann gibt es einen verzerrten Würfel im $\mathbb{R}^n$, für den das Simplexverfahren *alle* $2^n$ Ecken durchläuft (Klee und Minty 1972; das Simplexverfahren selbst wurde 1947 (noch bevor Computer gebräuchlich waren!) von Dantzig formuliert).

Die gute Nachricht ist, dass solch problematische Aufgaben in der Praxis nicht vorkommen, sodass das Simplexverfahren für die meisten Probleme recht gut funktioniert. Auf der anderen Seite gibt es auch andere Ansätze, die polynomiale (also der Form $O(m^k n^l)$) Komplexität haben, aber eben in der Praxis nicht durchweg besser sind. Diese Verfahren sind unter dem Namen *Innere-Punkte-Verfahren* bekannt; das erste solche Verfahren mit beweisbar polynomialer Laufzeit wurde 1984 von Karmarkar beschrieben. Die Grundidee dabei ist, die Nebenbedingungen durch „Strafterme“ in der Zielfunktion zu ersetzen. Statt des LPs

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

betrachtet man das nichtlineare Maximierungsproblem

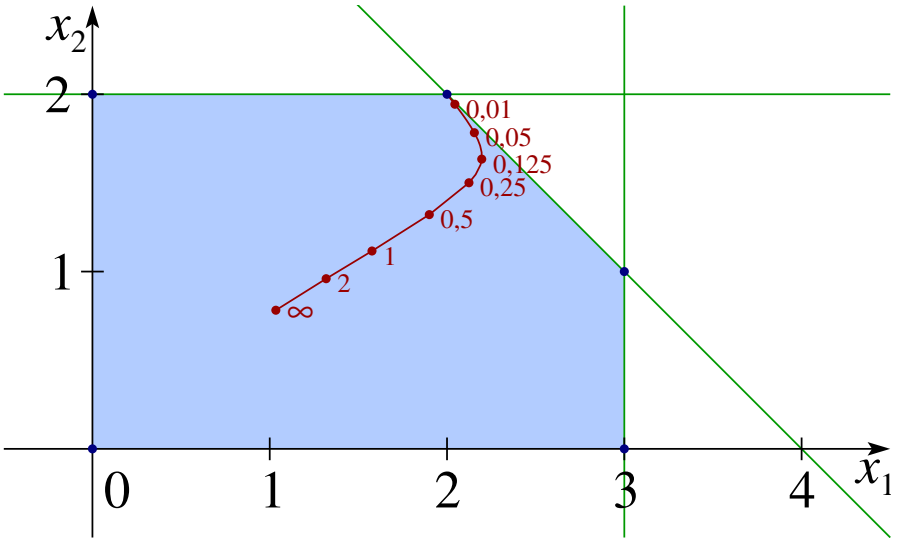
$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle + \mu \sum_j \log x_j \mid A\boldsymbol{x} = \boldsymbol{b}\}$$

mit einem Parameter $\mu > 0$. Da $\log x \to -\infty$ für $x \searrow 0$, wird die modifizierte Zielfunktion sehr klein, wenn man sich dem Rand der zulässigen Menge nähert; das Maximum wird also im Inneren der zulässigen Menge erreicht — daher der Name dieses Verfahrens. Außerdem ist die modifizierte Zielfunktion strikt konvex (nach oben), weshalb es eine eindeutige Lösung gibt. Sie kann mithilfe von Lagrange-Multiplikatoren (Stichwort „Extrema unter Nebenbedingungen“; die Nebenbedingungen sind hier $A\boldsymbol{x} = \boldsymbol{b}$) durch das Newton-Verfahren approximiert werden. Man kann zeigen, dass diese Lösung für $\mu \searrow 0$ gegen eine Lösung des ursprünglichen linearen Problems konvergiert. In der Praxis kombiniert man die Schritte des Newton-Verfahrens mit der Verkleinerung von $\mu$ und nähert sich so einem Optimum an.

Ein weiterer Vorteil des Simplexverfahrens und seiner Varianten ist, dass es einen „Warmstart“ ermöglicht, wenn zum Beispiel weitere Nebenbedingungen hinzugefügt werden. Mit dem dualen Simplexverfahren (siehe nächsten Abschnitt) kann man ausgehend von der bisherigen Optimallösung in wenigen Schritten eine Lösung für das neue Problem finden, während Innere-Punkte-Verfahren wieder von vorn beginnen müssen. Diese Hinzunahme weiterer Nebenbedingungen ist ein wesentlicher Bestandteil sogenannter Branch-and-Bound-Verfahren für die Lösung *ganzzahliger* linearer Optimierungsprobleme. Aus diesen Gründen ist das Simplexverfahren nach wie vor das in der Praxis am häufigsten eingesetzte Lösungsverfahren.

**BSP**
Staatsexamen
Innere-
Punkte-
Verfahren

10.4. **Beispiel.** In unserem Beispiel sieht die Approximation beim Innere-Punkte-Verfahren dann so aus:




Die roten Zahlen sind die Werte von $\mu$, die zu der jeweiligen angenäherten Lösung gehören. ♣

# 11. Dualität

Wenn man ein lineares Optimierungsproblem der Form

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

gegeben hat und eine vorgebliche Lösung $\boldsymbol{x}^*$ vorgelegt bekommt, dann stellt sich die Frage nach der Verifikation. Die Ungleichungen $A\boldsymbol{x}^* \leq \boldsymbol{b}$ und $\boldsymbol{x}^* \geq \boldsymbol{0}$ kann man leicht nachprüfen, aber wie stellt man fest, ob $\boldsymbol{x}^*$ wirklich die Zielfunktion maximiert?

Wenn wir gute obere Schranken für die Zielfunktion auf der zulässigen Menge bestimmen können, dann gibt uns das wenigstens ein Maß dafür, wie weit weg vom Optimum der Wert $\langle \boldsymbol{c}, \boldsymbol{x}^*\rangle$ höchstens entfernt sein kann. Im besten Fall stimmen der Wert und die obere Schranke überein; dann haben wir ein „Zertifikat“ für die Optimalität von $\boldsymbol{x}^*$.

Wie bekommt man obere Schranken für die Zielfunktion? Wir betrachten wieder unser Beispiel.

**BSP** Staatsexamen Schranken für die Zielfunktion

11.1. **Beispiel.** Wir erinnern uns an die Zielfunktion $\frac{5}{6}x_1 + x_2$ und die Nebenbedingungen $x_1 \leq 3$, $x_2 \leq 2$, $x_1 + x_2 \leq 4$, $x_1 \geq 0$, $x_2 \geq 0$.

Aus $x_1 \leq 3$ und $x_2 \leq 2$ folgt zum Beispiel $\frac{5}{6}x_1 + x_2 \leq \frac{5}{2} + 2 = \frac{9}{2} = 4{,}5$.

Wir können aber auch so abschätzen:

$$\tfrac{5}{6}x_1 + x_2 = \tfrac{5}{6}(x_1 + x_2) + \tfrac{1}{6}x_2 \leq \tfrac{10}{3} + \tfrac{1}{3} = \tfrac{11}{3}\,.$$

Das liefert sogar genau das Maximum!

Wie sieht die allgemeinste Schranke aus, die wir auf diese Weise bekommen können? Dazu schreiben wir alle Ungleichungen einheitlich als obere Abschätzungen:

$$\begin{aligned} x_1 &\leq 3\\ x_2 &\leq 2\\ x_1 + x_2 &\leq 4\\ -x_1 &\leq 0\\ -x_2 &\leq 0 \end{aligned}$$

Jede Linearkombination dieser Ungleichungen mit nichtnegativen Koeffizienten liefert dann eine Ungleichung, die auf der zulässigen Menge gilt. Wir suchen dann also eine solche Linearkombination mit der Eigenschaft, dass ihre linke Seite gleich der Zielfunktion ist, und so dass die rechte Seite möglichst klein ist. Wenn wir die Koeffizienten der Linearkombination mit $y_1, y_2, y_3, z_1, z_2$ bezeichnen, dann ist unsere allgemeinste Ungleichung

$$(y_1 + y_3 - z_1)x_1 + (y_2 + y_3 - z_2)x_2 \leq 3y_1 + 2y_2 + 4y_3\,.$$

Die Aufgabe lautet also:

---

$$\begin{aligned} \text{minimiere} \quad & 3y_1 + 2y_2 + 4y_3\\ \text{unter den Nebenbedingungen} \quad & y_1 + y_3 - z_1 = \tfrac{5}{6}\\ & y_2 + y_3 - z_2 = 1\\ & y_1, y_2, y_3, z_1, z_2 \geq 0 \end{aligned}$$

---

Hier kann man $z_1$ und $z_2$ als Schlupfvariable auffassen und die Aufgabe äquivalent so formulieren:

---

$$\begin{aligned} \text{minimiere} \quad & 3y_1 + 2y_2 + 4y_3 \\ \text{unter den Nebenbedingungen} \quad & y_1 + y_3 \geq \tfrac{5}{6} \\ & y_2 + y_3 \geq 1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

---

Das ist das zu unserem ursprünglichen Optimierungsproblem *duale* Problem. Es wird gelöst von $(y_1, y_2, y_3) = (0, \frac{1}{6}, \frac{5}{6})$ mit dem Zielfunktionswert $\frac{11}{3}$. ♣

Jetzt wollen wir uns überlegen, wie das allgemein funktioniert. Sei also

$$\max_{\boldsymbol{x}} \{\langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

eine lineare Maximierungsaufgabe. Die linken Seiten der Ungleichungen sind gegeben durch $A$ und durch $-I_n$, wenn wir alle Ungleichungen in der „$\leq$"-Form schreiben. Eine nichtnegative Linearkombination mit Koeffizienten $\boldsymbol{y}$ bzw. $\boldsymbol{z}$ hat dann die Form

$$\boldsymbol{y}^\top A\boldsymbol{x} - \boldsymbol{z}^\top \boldsymbol{x} \leq \boldsymbol{y}^\top \boldsymbol{b}\,.$$

Die linke Seite soll der Zielfunktion entsprechen; es muss also

$$\boldsymbol{y}^\top A - \boldsymbol{z}^\top = \boldsymbol{c}^\top \qquad \text{bzw.} \qquad A^\top \boldsymbol{y} - \boldsymbol{z} = \boldsymbol{c}$$

sein. Elimination der Schlupfvariablen $\boldsymbol{z}$ liefert $A^\top \boldsymbol{y} \geq \boldsymbol{c}$. Die rechte Seite soll minimiert werden, um die beste obere Schranke für die Zielfunktion zu liefern. Das ergibt das lineare Minimierungsproblem

$$\min_{\boldsymbol{y}} \{\langle \boldsymbol{b}, \boldsymbol{y} \rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}\,.$$

11.2. **Definition.** Das *duale LP* zur Standard-Maximierungsaufgabe **DEF** duales LP

$$\max_{\boldsymbol{x}} \{\langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

ist die Minimierungsaufgabe

$$\min_{\boldsymbol{y}} \{\langle \boldsymbol{b}, \boldsymbol{y} \rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}\,.$$

Das ursprüngliche Problem heißt in diesem Zusammenhang auch das *primale* Problem. ♢

Wenn das Originalproblem also durch die erweiterte Matrix

$$M = \left(\begin{array}{c|c} A & \boldsymbol{b} \\ \hline \boldsymbol{c}^\top & 0 \end{array}\right)$$

beschrieben wird, dann ist das duale Problem durch die transponierte Matrix

$$M^\top = \left(\begin{array}{c|c} A^\top & \boldsymbol{c} \\ \hline \boldsymbol{b}^\top & 0 \end{array}\right)$$

gegeben. Allerdings ist dabei zu beachten, dass sich die Richtung der Optimierung (Maximierung/Minimierung) und auch die Richtung der Ungleichungen in den Nebenbedingungen (außer bei den Nichtnegativitätsbedingungen) geändert hat.

Unsere Motivation, obere Schranken für die Zielfunktion zu finden, wird durch folgenden Satz präzisiert.

**SATZ** schwache Dualität

11.3. **Satz.** *Sei $P$ die Standard-Maximierungsaufgabe*

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

*und $D$ die dazu duale Minimierungsaufgabe*

$$\min_{\boldsymbol{y}}\{\langle \boldsymbol{b}, \boldsymbol{y}\rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}\,.$$

*Seien weiter $\boldsymbol{x}$ in der zulässigen Menge von $P$ und $\boldsymbol{y}$ in der zulässigen Menge von $D$. Dann gilt*

$$\langle \boldsymbol{c}, \boldsymbol{x}\rangle \leq \langle \boldsymbol{b}, \boldsymbol{y}\rangle\,.$$

Jede zulässige Lösung von $D$ liefert also eine obere Schranke für das Maximum der Zielfunktion von $P$ und jede zulässige Lösung von $P$ liefert eine untere Schranke für die Zielfunktion von $D$.

*Beweis.* Es gilt

$$\langle \boldsymbol{c}, \boldsymbol{x}\rangle = \boldsymbol{c}^\top \boldsymbol{x} \leq (A^\top \boldsymbol{y})^\top \boldsymbol{x} = \boldsymbol{y}^\top A\boldsymbol{x} = \boldsymbol{y}^\top (A\boldsymbol{x}) \leq \boldsymbol{y}^\top \boldsymbol{b} = \langle \boldsymbol{b}, \boldsymbol{y}\rangle\,.$$

Dabei haben wir die Bedingungen $\boldsymbol{x} \geq \boldsymbol{0}$, $\boldsymbol{y} \geq \boldsymbol{0}$, $A^\top \boldsymbol{y} \geq \boldsymbol{c}$ und $A\boldsymbol{x} \leq \boldsymbol{b}$ verwendet. ❑

Wir werden später sehen, dass tatsächlich sogar

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\} = \min_{\boldsymbol{y}}\{\langle \boldsymbol{b}, \boldsymbol{y}\rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}$$

gilt (wie wir das auch an unserem Beispiel gesehen haben), falls beide Probleme zulässige Lösungen haben („starke Dualität“). Um das Optimum zu finden, kann man also auch mit dem dualen Problem arbeiten. Das ist zum Beispiel dann nützlich, wenn die Zielfunktion negative Koeffizienten hat (also $\boldsymbol{c} \leq \boldsymbol{0}$) und der Nullpunkt nicht zulässig ist (also $\boldsymbol{b} \not\geq \boldsymbol{0}$). Dann kann man das Simplexverfahren auf das duale Problem anwenden, was darauf hinausläuft, die Optimalitätsbedingung ($\tilde{\boldsymbol{c}} \leq \boldsymbol{0}$) beizubehalten und die Zulässigkeit schrittweise zu verbessern, bis man einen Punkt in der zulässigen Menge erhält. Da die Matrizen, die die beiden Probleme beschreiben, zueinander transponiert sind, läuft das auf eine Vertauschung der Rollen von Zeilen und Spalten hinaus.

**BSP** duales Simplex-Verfahren

11.4. **Beispiel.** Wir betrachten das LP

| | |
|---|---|
| maximiere | $-x_1 - x_2$ |
| unter den Nebenbedingungen | $-x_1 - 3x_2 \leq -3$ |
| | $-2x_1 - x_2 \leq -2$ |
| | $-2x_1 - 3x_2 \leq -2$ |
| | $x_1, x_2 \geq 0$ |

Das zugehörige Simplex-Tableau sieht so aus:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|
| $w_1$ | $-1$ | $-3$ | 1 | 0 | 0 | 0 | $-3$ |
| $w_2$ | $-2$ | $-1$ | 0 | 1 | 0 | 0 | $-2$ |
| $w_3$ | $-2$ | $-3$ | 0 | 0 | 1 | 0 | $-2$ |
| $z$ | $-1$ | $-1$ | 0 | 0 | 0 | $-1$ | 0 |

Die *Zulässigkeitsbedingung* ist verletzt, da nicht alle rechten Seiten $\geq 0$ sind (hier sind sogar alle negativ). Die *Optimalitätsbedingung* ist hingegen erfüllt, denn die Koeffizienten der Zielfunktion sind alle $\leq 0$. Da wir jetzt an der Zulässigkeit arbeiten wollen, müssen wir zunächst eine *Pivotzeile* mit negativer rechter Seite wählen, hier zum Beispiel die erste Zeile. Welche Spalte müssen wir als Pivotspalte wählen? Zunächst einmal muss durch das Skalieren der Pivotzeile die rechte Seite positiv werden; dafür muss der Eintrag $a_{ij}$ der Matrix im Schnittpunkt von Pivotzeile und -spalte negativ sein. Damit die Einträge der Zielfunktionszeile bei der anstehenden Zeilenumformung nicht positiv werden, muss man dann eine Spalte $j$ auswählen, für die der Engpassquotient (der in diesem Zusammenhang auch *Profitquotient* heißt)

$$q_j = \begin{cases} \frac{c_j}{a_{ij}} & \text{falls } a_{ij} < 0, \\ \infty & \text{sonst} \end{cases}$$

minimal ist. Im Beispiel ist (mit $i = 1$) $q_1 = 1$, $q_2 = 1/3$, also wird die zweite Spalte Pivotspalte. Das neue Tableau ist also

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|
| $x_2$ | $1/3$ | $1$ | $-1/3$ | $0$ | $0$ | $0$ | $1$ |
| $w_2$ | $-5/3$ | $0$ | $-1/3$ | $1$ | $0$ | $0$ | $-1$ |
| $w_3$ | $-1$ | $0$ | $-1$ | $0$ | $1$ | $0$ | $1$ |
| $z$ | $-2/3$ | $0$ | $-1/3$ | $0$ | $0$ | $-1$ | $1$ |

Hier gibt es nur noch eine Zeile mit negativer rechter Seite (die zweite). Die Engpassquotienten sind $q_1 = 2/5$ und $q_3 = 1$. Damit ist die erste Spalte die Pivotspalte, und wir erhalten das folgende Tableau:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|
| $x_2$ | $0$ | $1$ | $-2/5$ | $1/5$ | $0$ | $0$ | $4/5$ |
| $x_1$ | $1$ | $0$ | $1/5$ | $-3/5$ | $0$ | $0$ | $3/5$ |
| $w_3$ | $0$ | $0$ | $-4/5$ | $-3/5$ | $1$ | $0$ | $8/5$ |
| $z$ | $0$ | $0$ | $-1/5$ | $-2/5$ | $0$ | $-1$ | $7/5$ |

Hier sind jetzt sowohl die Optimalitäts- als auch die Zulässigkeitsbedingung erfüllt. Also ist $(x_1, x_2) = (3/5, 4/5)$ eine Optimallösung mit Zielfunktionswert $-7/5$. ♣

Daraus lässt sich jetzt leicht ein Algorithmus für das duale Simplexverfahren ableiten. Es ist anwendbar, wenn $\boldsymbol{c} \leq \mathbf{0}$ ist (das ist die Optimalitätsbedingung). Abgesehen davon ist der einzige Unterschied zum primalen Simplexverfahren, dass zuerst die Pivotzeile gewählt wird (mit negativer rechter Seite) und dann über die Profitquotienten eine passende Pivotspalte. Sobald Pivotzeile und -spalte gewählt sind, ist das Vorgehen wie vorher auch: Pivotzeile skalieren, sodass der Eintrag im Schnittpunkt von Pivotzeile und -spalte 1 wird, und dann den Rest der Pivotspalte durch Zeilenoperationen zu null machen. Im Fall, dass alle Profitquotienten unendlich sind, ist das Problem unzulässig: Alle Einträge in der Pivotzeile sind $\geq 0$, außer der rechten Seite, die negativ ist. Das ergibt einen Widerspruch, da die Werte aller Variablen $\geq 0$ sein müssen.

In Anwendungen kommt es öfter vor, dass nachträglich Nebenbedingungen hinzugefügt werden (müssen), etwa weil bei der Modellierung relevante Einschränkungen vergessen worden sind. Ein anderer typischer Fall tritt bei der *ganzzahligen* linearen Optimierung auf (man sucht also ein Optimum unter der zusätzlichen Einschränkung, dass einige oder alle Variablen nur ganzzahlige Werte annehmen dürfen): Man löst das Problem zunächst ohne die Ganzzahligkeitsbedingung.

Wenn eine Variable $x_j$ in der Optimallösung nicht ganzzahlig ist, sondern etwa $n < x_j < n+1$ gilt mit $n \in \mathbb{Z}$, dann löst man zwei neue Probleme mit der zusätzlichen Nebenbedingung $x_j \leq n$ bzw. $x_j \geq n+1$, usw. In solchen Fällen erlaubt es die duale Simplexmethode, mit der Optimallösung des vorherigen Problems zu starten (die immer noch die Optimalitätsbedingung erfüllt, aber die Zulässigkeitsbedingung hinsichtlich der neuen Ungleichung verletzt) und in wenigen Schritten eine Optimallösung des neuen Problems zu finden.

Wir müssen noch die Frage beantworten, wie man vorgehen kann, wenn weder $\boldsymbol{b} \geq \mathbf{0}$ (Zulässigkeit) noch $\boldsymbol{c} \leq \mathbf{0}$ (Optimalität) für das gegebene LP erfüllt sind. Eine Möglichkeit dafür ist, ein Hilfs-LP zu formulieren und zu lösen, das eine zulässige Lösung liefert. Sei also das Standard-LP

$$\max_{\boldsymbol{x}} \{\langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \mathbf{0}\}$$

gegeben mit $\boldsymbol{b} \not\geq \mathbf{0}$. Wir setzen $M := 1 + \max\{-b_1, \ldots, -b_m\}$. Wir führen eine neue Variable $k$ ein und betrachten das LP

$$\max_{\boldsymbol{x},k} \{k \mid A\boldsymbol{x} + \mathbf{1}k \leq \boldsymbol{b} + \mathbf{1}M, k \leq M, \boldsymbol{x} \geq \mathbf{0}, k \geq 0\}\,. \tag{11.1}$$

Dabei steht $\mathbf{1}$ für einen Spaltenvektor, dessen Einträge alle 1 sind. Da nach Definition von $M$ gilt, dass $\boldsymbol{b} + \mathbf{1}M \geq \mathbf{1} \geq \mathbf{0}$ ist, ist das Start-Tableau für dieses LP zulässig. Da $k \leq M$ eine Nebenbedingung ist, ist das LP auch beschränkt; das Maximum existiert also. Wenn die zulässige Menge des ursprünglichen LP nicht leer ist, dann liefert jeder zulässige Vektor, ergänzt mit $k = M$, eine zulässige Lösung des neuen LP; in diesem Fall ist das Maximum also $M$. Das bedeutet, dass das ursprüngliche LP unzulässig ist, wenn das Maximum des neuen LP $< M$ ist. Umgekehrt gilt, dass eine Lösung mit $k = M$ des neuen LP durch Weglassen von $k$ zu einer zulässigen Lösung des ursprünglichen LP führt. Wir haben also folgenden Algorithmus für die *Phase I* des Lösungsverfahrens.

**ALGO**
Phase I

---

**input:** $A \in \mathrm{Mat}(m, n, \mathbb{R})$, $\boldsymbol{b} \in \mathbb{R}^m$

**output:** $\boldsymbol{x} \in \mathbb{R}^n$ mit $A\boldsymbol{x} \leq \boldsymbol{b}$, $\boldsymbol{x} \geq \mathbf{0}$, oder „unzulässig“

$M := 1 + \max\{-b_1, \ldots, -b_m\}$
$(\boldsymbol{x}, k) :=$ Lösung von (11.1)
**if** $k < M$ **then**
  **return** „unzulässig“
**else**
  **return** $\boldsymbol{x}$
**end if**

---

In der Praxis wird man (im zulässigen Fall) das End-Tableau des Simplexverfahrens für (11.1) passend modifizieren ($k$ entfernen, Zielfunktion durch die ursprüngliche Zielfunktion ersetzen) und dann gleich mit dem Simplexverfahren für das ursprüngliche Problem fortsetzen.

11.5. **Beispiel.** Als einfaches Beispiel betrachten wir folgendes Problem: Was ist der größtmögliche Wert von $x_1$, sodass es $x_2 \geq 0$ gibt mit $x_1 + x_2 \geq 3$ und $5x_1 + 2x_2 \leq 10$? Wir wollen also folgendes LP lösen:

**BSP**
Phase I

$$\begin{aligned} &\text{maximiere} && x_1 \\ &\text{unter den Nebenbedingungen} && -x_1 - x_2 \leq -3 \\ & && 5x_1 + 2x_2 \leq 10 \\ & && x_1, x_2 \geq 0 \end{aligned}$$

Das Tableau dazu ist

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $z$ | RS |
|---|---|---|---|---|---|---|
| $w_1$ | $-1$ | $-1$ | 1 | 0 | 0 | $-3$ |
| $w_2$ | 5 | 2 | 0 | 1 | 0 | 10 |
| $z$ | 1 | 0 | 0 | 0 | $-1$ | 0 |

Hier sind weder die Optimalitäts- noch die Zulässigkeitsbedingung erfüllt.

Nach dem Algorithmus oben setzen wir $M := 1+\max\{3, -10\} = 4$ und betrachten das neue LP

$$\begin{aligned} &\text{maximiere} && k \\ &\text{unter den Nebenbedingungen} && -x_1 - x_2 + k \leq 1 \\ & && 5x_1 + 2x_2 + k \leq 14 \\ & && k \leq 4 \\ & && x_1, x_2, k \geq 0 \end{aligned}$$

mit dem Tableau

| BV | $x_1$ | $x_2$ | $k$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|---|
| $w_1$ | $-1$ | $-1$ | 1 | 1 | 0 | 0 | 0 | 1 |
| $w_2$ | 5 | 2 | 1 | 0 | 1 | 0 | 0 | 14 |
| $w_3$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 4 |
| $z$ | 0 | 0 | 1 | 0 | 0 | 0 | $-1$ | 0 |

Die Zulässigkeitsbedingung ist erfüllt. Spaltenpivot ist $k$ (das ist die einzige Möglichkeit); Zeilenpivot ist $w_1$. Das liefert das Tableau

| BV | $x_1$ | $x_2$ | $k$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|---|
| $k$ | $-1$ | $-1$ | 1 | 1 | 0 | 0 | 0 | 1 |
| $w_2$ | 6 | 3 | 0 | $-1$ | 1 | 0 | 0 | 13 |
| $w_3$ | 1 | 1 | 0 | $-1$ | 0 | 1 | 0 | 3 |
| $z$ | 1 | 1 | 0 | $-1$ | 0 | 0 | $-1$ | $-1$ |

Jetzt haben wir die Wahl zwischen $x_1$ und $x_2$ als Spaltenpivots. Wir nehmen $x_2$; dann ist $w_3$ Zeilenpivot:

| BV | $x_1$ | $x_2$ | $k$ | $w_1$ | $w_2$ | $w_3$ | $z$ | RS |
|---|---|---|---|---|---|---|---|---|
| $k$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 4 |
| $w_2$ | 3 | 0 | 0 | 2 | 1 | $-3$ | 0 | 4 |
| $x_2$ | 1 | 1 | 0 | $-1$ | 0 | 1 | 0 | 3 |
| $z$ | 0 | 0 | 0 | 0 | 0 | $-1$ | $-1$ | $-4$ |

Jetzt ist die Optimalitätsbedingung erfüllt. Der Wert der Zielfunktion ist $4 = M$, also haben wir eine zulässige Lösung unseres ursprünglichen LP gefunden. Die Basisvariablen sind $x_2$ und $w_2$. Wir tauschen also im Starttableau des ursprünglichen

LP die Basisvariable $w_1$ gegen $x_2$:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $z$ | RS |
|---|---|---|---|---|---|---|
| $x_2$ | 1 | 1 | $-1$ | 0 | 0 | 3 |
| $w_2$ | 3 | 0 | 2 | 1 | 0 | 4 |
| $z$ | 1 | 0 | 0 | 0 | $-1$ | 0 |

Die Zulässigkeitsbedingung ist jetzt erfüllt. Von hier aus führt uns ein Simplex-Schritt (Spaltenpivot $x_1$, Zeilenpivot $w_2$) zur optimalen Lösung:

| BV | $x_1$ | $x_2$ | $w_1$ | $w_2$ | $z$ | RS |
|---|---|---|---|---|---|---|
| $x_2$ | 0 | 1 | $-5/3$ | $-1/3$ | 0 | $5/3$ |
| $x_1$ | 1 | 0 | $2/3$ | $1/3$ | 0 | $4/3$ |
| $z$ | 0 | 0 | $-2/3$ | $-1/3$ | $-1$ | $-4/3$ |

Das Maximum ist also $x_1 = 4/3$ (mit $x_2 = 5/3$). ♣

## 12. Farkas-Lemma und starker Dualitätssatz

Im letzten Abschnitt ist noch die Frage offen geblieben, ob es eine Lücke zwischen dem Maximum der Zielfunktion des primalen LPs

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

und dem Minimum der Zielfunktion des dualen LPs

$$\min_{\boldsymbol{y}}\{\langle \boldsymbol{b}, \boldsymbol{y}\rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}$$

geben kann, wenn beide existieren. Der schwache Dualitätssatz 11.3 garantiert, dass das Maximum nicht größer ist als das Minimum. Der starke Dualitätssatz wird zeigen, dass sie sogar übereinstimmen.

Zuerst beweisen wir eine Hilfsaussage, die eine einleuchtende geometrische Interpretation hat.

**LEMMA** Farkas-Lemma

12.1. **Lemma.** *Seien $A \in \mathrm{Mat}(m, n, \mathbb{R})$ und $\boldsymbol{b} \in \mathbb{R}^m$. Dann ist von den folgenden beiden Aussagen genau eine richtig:*

(1) $\exists \boldsymbol{x} \in \mathbb{R}^n\colon \boldsymbol{x} \geq \boldsymbol{0}$ *und* $A\boldsymbol{x} = \boldsymbol{b}$;

(2) $\exists \boldsymbol{y} \in \mathbb{R}^m\colon A^\top \boldsymbol{y} \geq \boldsymbol{0}$ *und* $\langle \boldsymbol{b}, \boldsymbol{y}\rangle < 0$.

*Beweis.* Ist (1) erfüllt, dann gilt für $\boldsymbol{x}$ wie in (1) und jedes $\boldsymbol{y} \in \mathbb{R}^m$

$$\langle \boldsymbol{b}, \boldsymbol{y}\rangle = \boldsymbol{y}^\top \boldsymbol{b} = \boldsymbol{y}^\top (A\boldsymbol{x}) = (A^\top \boldsymbol{y})^\top \boldsymbol{x} = \langle A^\top \boldsymbol{y}, \boldsymbol{x}\rangle \geq 0\,,$$

denn $A^\top \boldsymbol{y} \geq \boldsymbol{0}$ und $\boldsymbol{x} \geq \boldsymbol{0}$. Damit muss (2) falsch sein.

Gilt andererseits (1) nicht, dann liegt $\boldsymbol{b}$ außerhalb der abgeschlossenen und konvexen Menge $K = \{A\boldsymbol{x} \mid \boldsymbol{x} \geq \boldsymbol{0}\} \subset \mathbb{R}^m$ (das ist der Kegel im $\mathbb{R}^m$, der von den Spalten von $A$ aufgespannt wird). Es gibt dann eine Hyperebene $H$ durch $\boldsymbol{0}$, sodass $K$ in einem der beiden durch $H$ definierten abgeschlossenen Halbräumen und $\boldsymbol{b}$ im verbleibenden offenen Halbraum liegt. Man kann zum Beispiel den Punkt $\boldsymbol{a} \in K$ mit minimalem Abstand zu $\boldsymbol{b}$ betrachten und für $H$ die Hyperebene durch $\boldsymbol{a}$ nehmen, die senkrecht auf der Geraden durch $\boldsymbol{b}$ und $\boldsymbol{a}$ steht. Wäre $K$ nicht in dem „von $\boldsymbol{b}$ abgewandten“ abgeschlossenen Halbraum enthalten, dann gäbe es wegen der Konvexität von $K$ Punkte von $K$ beliebig nahe bei $\boldsymbol{a}$, die auf der „falschen“ Seite von $H$ liegen; die wären dann aber näher an $\boldsymbol{b}$ als $\boldsymbol{a}$, ein Widerspruch. Ähnlich sieht man, dass der Nullvektor in $H$ liegen muss, denn die ganze Halbgerade von $\boldsymbol{0}$ durch $\boldsymbol{a}$ liegt in $K$. Wäre $\boldsymbol{0} \notin H$, dann würde diese Halbgerade $H$ in $\boldsymbol{a}$ durchstoßen, und wir bekämen denselben Widerspruch wie eben.

Wir können $H$ durch eine Gleichung $\langle \boldsymbol{z}, \boldsymbol{y}\rangle = 0$ beschreiben (hier ist $\boldsymbol{y}$ der Koeffizientenvektor der Gleichung und $\boldsymbol{z}$ ein beliebiger Punkt im $\mathbb{R}^m$). Nach eventuellem Ersetzen von $\boldsymbol{y}$ durch $-\boldsymbol{y}$ gilt dann $\langle \boldsymbol{b}, \boldsymbol{y}\rangle < 0$ und $\langle A_j, \boldsymbol{y}\rangle \geq 0$ für alle $1 \leq j \leq n$, wobei $A_j$ für die $j$te Spalte von $A$ steht. Letzteres ist äquivalent zu $A^\top \boldsymbol{y} \geq \boldsymbol{0}$. Insgesamt folgt also, dass (2) gelten muss, wenn (1) nicht gilt. ❑

Gyula (auch Julius) Farkas war ein ungarischer Mathematiker, der von 1847 bis 1930 lebte und hauptsächlich in Klausenburg (Kolozsvár; heute Cluj-Napoca in Rumänien) wirkte. Sein Lemma veröffentlichte er 1902.

Letzten Endes beruht der Beweis des Lemmas darauf, dass man in einem endlichdimensionalen reellen Vektorraum eine abgeschlossene konvexe Menge $K$ von einem Punkt $\boldsymbol{b}$, der außerhalb liegt, durch eine (i.A. affine) Hyperebene trennen kann. Diese

Aussage gilt allgemeiner auch in beliebigen normierten reellen Vektorräumen (wie zum Beispiel dem Raum $\mathcal{C}([a,b])$ der stetigen Funktionen auf dem Intervall $[a,b]$ mit der Maximumsnorm). Das ist eine Konsequenz des Satzes von Hahn-Banach, der eine wichtige Grundlage der Funktionalanalysis darstellt. Sein Beweis ist allerdings nicht-konstruktiv: Er verwendet das Zornsche Lemma (ähnlich wie der Beweis, dass *jeder* Vektorraum eine Basis hat).

Für Anwendungen ist eine allgemeinere Version des Lemmas, die sich flexibler einsetzen lässt, von Vorteil.

**LEMMA** allgemeines Farkas-Lemma

12.2. **Lemma.** *Seien $k,l,m,n \in \mathbb{Z}_{\geq 0}$. Wir betrachten Matrizen $A \in \mathrm{Mat}(m,n,\mathbb{R})$, $B \in \mathrm{Mat}(m,k,\mathbb{R})$, $C \in \mathrm{Mat}(l,n,\mathbb{R})$ und $D \in \mathrm{Mat}(l,k,\mathbb{R})$ und Vektoren $\boldsymbol{a} \in \mathbb{R}^m$, $\boldsymbol{b} \in \mathbb{R}^l$. Dann ist genau eine der folgenden Aussagen richtig:*

(1) $\exists \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^k$:
$$A\boldsymbol{x} + B\boldsymbol{y} \leq \boldsymbol{a}, \quad C\boldsymbol{x} + D\boldsymbol{y} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0};$$

(2) $\exists \boldsymbol{u} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^l$:
$$A^\top \boldsymbol{u} + C^\top \boldsymbol{v} \geq \boldsymbol{0}, \quad B^\top \boldsymbol{u} + D^\top \boldsymbol{v} = \boldsymbol{0}, \quad \boldsymbol{u} \geq \boldsymbol{0}, \quad \langle \boldsymbol{a}, \boldsymbol{u} \rangle + \langle \boldsymbol{b}, \boldsymbol{v} \rangle < 0.$$

*Beweis.* Wir führen diese Aussage auf Lemma 12.1 zurück. In Aussage (1) können wir Schlupfvariablen $\boldsymbol{z}$ einführen und die nicht durch eine Nichtnegativitätsbedingung eingeschränkten Variablen $\boldsymbol{y}$ durch Differenzen $\boldsymbol{y} = \boldsymbol{y}^+ - \boldsymbol{y}^-$ nichtnegativer Variablen ersetzen. Dann hat (1) die Form

$$\exists \boldsymbol{x}, \boldsymbol{y}^+, \boldsymbol{y}^-, \boldsymbol{z}: \qquad \begin{pmatrix} A & B & -B & I_m \\ C & D & -D & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y}^+ \\ \boldsymbol{y}^- \\ \boldsymbol{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{pmatrix}, \qquad \boldsymbol{x}, \boldsymbol{y}^+, \boldsymbol{y}^-, \boldsymbol{z} \geq \boldsymbol{0}\,.$$

Nach Lemma 12.1 gilt also entweder (1) oder

$$\exists \boldsymbol{u}, \boldsymbol{v}: \qquad \begin{pmatrix} A^\top & C^\top \\ B^\top & D^\top \\ -B^\top & -D^\top \\ I_m & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} \geq \boldsymbol{0}\,, \qquad \langle \boldsymbol{a}, \boldsymbol{u} \rangle + \langle \boldsymbol{b}, \boldsymbol{v} \rangle < 0\,,$$

was unter Beachtung von

$$B^\top \boldsymbol{u} + D^\top \boldsymbol{v} \geq \boldsymbol{0} \ \wedge\ -B^\top \boldsymbol{u} - D^\top \boldsymbol{v} \geq \boldsymbol{0} \iff B^\top \boldsymbol{u} + D^\top \boldsymbol{v} = \boldsymbol{0}$$

genau die Aussage (2) ist. ❑

Wir können das nun verwenden, um den starken Dualitätssatz zu beweisen.

**SATZ** starke Dualität

12.3. **Satz.** *Seien das primale LP*

$$\max_{\boldsymbol{x}} \{ \langle \boldsymbol{c}, \boldsymbol{x} \rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \}$$

*und das zugehörige duale LP*

$$\min_{\boldsymbol{y}} \{ \langle \boldsymbol{b}, \boldsymbol{y} \rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0} \}$$

*gegeben. Dann gelten die folgenden Aussagen:*

(1) *Sind beide Probleme zulässig, dann haben beide optimale Lösungen, und für jedes Paar optimaler Lösungen stimmen die Werte der Zielfunktionen überein.*

(2) *Ist eines der Probleme zulässig und beschränkt, so ist das andere ebenfalls zulässig (sodass insbesondere die Zielfunktionen im jeweiligen Optimum denselben Wert annehmen).*

(3) *Ist eines der Probleme unbeschränkt, so ist das andere unzulässig.*

*Beweis.*

(1) Wenn beide Probleme zulässig sind, dann sind sie nach dem schwachen Dualitätssatz 11.3 auch beschränkt (denn der Wert der Zielfunktion in jedem beliebigen Punkt der zulässigen Menge des einen Problems liefert eine Schranke für die Zielfunktion des anderen Problems); damit existieren optimale Lösungen für beide Probleme. Der schwache Dualitätssatz zeigt dann auch die Ungleichung $\max_{\boldsymbol{x}}\langle \boldsymbol{c}, \boldsymbol{x}\rangle \leq \min_{\boldsymbol{y}}\langle \boldsymbol{b}, \boldsymbol{y}\rangle$. Es ist also nur noch zu zeigen, dass es zulässige $\boldsymbol{x}$ und $\boldsymbol{y}$ gibt mit $\langle \boldsymbol{c}, \boldsymbol{x}\rangle \geq \langle \boldsymbol{b}, \boldsymbol{y}\rangle$. Wäre dies nicht der Fall, dann wäre die Aussage

$$\exists \boldsymbol{x}, \boldsymbol{y}\colon \left\{ \begin{array}{rcl} A\boldsymbol{x} & \leq & \boldsymbol{b} \\ -A^\top \boldsymbol{y} & \leq & -\boldsymbol{c} \\ \langle -\boldsymbol{c}, \boldsymbol{x}\rangle + \langle \boldsymbol{b}, \boldsymbol{y}\rangle & \leq & 0 \\ \boldsymbol{x} & \geq & \boldsymbol{0} \\ \boldsymbol{y} & \geq & \boldsymbol{0} \end{array} \right.$$

falsch. Nach dem allgemeinen Farkas-Lemma 12.2 bedeutet das, dass die Aussage

$$\exists \boldsymbol{u}, \boldsymbol{v}, z\colon \left\{ \begin{array}{rcl} A^\top \boldsymbol{u} & \geq & z\boldsymbol{c} \\ A\boldsymbol{v} & \leq & z\boldsymbol{b} \\ \boldsymbol{u} & \geq & \boldsymbol{0} \\ \boldsymbol{v} & \geq & \boldsymbol{0} \\ z & \geq & 0 \\ \langle \boldsymbol{b}, \boldsymbol{u}\rangle & < & \langle \boldsymbol{c}, \boldsymbol{v}\rangle \end{array} \right.$$

wahr sein muss. Wir unterscheiden zwei Fälle.

(i) $z = 0$. Dann gibt es $\boldsymbol{u} \geq \boldsymbol{0}$ und $\boldsymbol{v} \geq \boldsymbol{0}$ mit

$$A^\top \boldsymbol{u} \geq \boldsymbol{0}\,, \quad A\boldsymbol{v} \leq \boldsymbol{0} \quad \text{und} \quad \langle \boldsymbol{b}, \boldsymbol{u}\rangle < \langle \boldsymbol{c}, \boldsymbol{v}\rangle\,.$$

Nach dem Farkas-Lemma bedeutet das, dass es keine $\boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}$ geben kann mit

$$A\boldsymbol{x} \leq \boldsymbol{b} \quad \text{und} \quad A^\top \boldsymbol{y} \geq \boldsymbol{c}\,,$$

was der vorausgesetzten Zulässigkeit der beiden Probleme widerspricht.

(ii) $z > 0$. Dann können wir alle Relationen durch $z$ teilen und so $z = 1$ annehmen. Die Aussage lautet dann

$$\exists \boldsymbol{u}, \boldsymbol{v}\colon \left\{ \begin{array}{rcl} A^\top \boldsymbol{u} & \geq & \boldsymbol{c} \\ A\boldsymbol{v} & \leq & \boldsymbol{b} \\ \boldsymbol{u} & \geq & \boldsymbol{0} \\ \boldsymbol{v} & \geq & \boldsymbol{0} \\ \langle \boldsymbol{b}, \boldsymbol{u}\rangle & < & \langle \boldsymbol{c}, \boldsymbol{v}\rangle\,, \end{array} \right.$$

was dem schwachen Dualitätssatz 11.3 widerspricht.

Wir erhalten also in jedem Fall einen Widerspruch. Damit muss die als falsch angenommene Aussage $\max_{\boldsymbol{x}}\langle \boldsymbol{c}, \boldsymbol{x}\rangle \geq \min_{\boldsymbol{y}}\langle \boldsymbol{b}, \boldsymbol{y}\rangle$ wahr sein, und (1) ist bewiesen.

(2) Aus Symmetriegründen genügt es zu zeigen, dass Zulässigkeit und Beschränktheit des primalen Problems die Zulässigkeit des dualen Problems impliziert. Wir führen wieder einen Widerspruchsbeweis. Wäre das duale Problem nicht zulässig, dann hieße das, dass es kein $\boldsymbol{y} \geq \mathbf{0}$ gibt mit $A^\top \boldsymbol{y} \geq \boldsymbol{c}$. Nach dem Farkas-Lemma würde das bedeuten, dass es $\boldsymbol{x} \geq \mathbf{0}$ gibt mit $A\boldsymbol{x} \leq \mathbf{0}$ und $\langle \boldsymbol{c}, \boldsymbol{x} \rangle > 0$. Da das primale Problem nach Voraussetzung zulässig ist, gibt es $\boldsymbol{x}_0 \geq \mathbf{0}$ mit $A\boldsymbol{x}_0 \leq \boldsymbol{b}$. Dann gilt für jedes $\lambda \geq 0$ auch

$$A(\boldsymbol{x}_0 + \lambda \boldsymbol{x}) = A\boldsymbol{x}_0 + \lambda A\boldsymbol{x} \leq \boldsymbol{b} + \lambda \mathbf{0} = \boldsymbol{b}$$

und $\boldsymbol{x}_0 + \lambda \boldsymbol{x} \geq \mathbf{0}$; außerdem ist

$$\langle \boldsymbol{c}, \boldsymbol{x}_0 + \lambda \boldsymbol{x} \rangle = \langle \boldsymbol{c}, \boldsymbol{x}_0 \rangle + \lambda \langle \boldsymbol{c}, \boldsymbol{x} \rangle \,,$$

was wegen $\langle \boldsymbol{c}, \boldsymbol{x} \rangle > 0$ für $\lambda \to \infty$ beliebig groß wird. Das ist aber ein Widerspruch zur Beschränktheit des primalen Problems.

(3) Das folgt aus dem schwachen Dualitätssatz 11.3: Ist eines der Probleme zulässig, dann ist der Wert seiner Zielfunktion in einem beliebigen Punkt der zulässigen Menge eine Schranke für die Zielfunktion des anderen Problems, sodass das andere Problem nicht unbeschränkt sein kann. ❑

Es sind also folgende Situationen möglich:

(1) Beide Probleme sind zulässig und beschränkt; die optimalen Werte der Zielfunktionen stimmen überein.

(2) Eines der Probleme ist unbeschränkt, das andere ist unzulässig.

(3) Beide Probleme sind unzulässig.

Dass der dritte Fall tatsächlich vorkommen kann, zeigt das folgende Beispiel.

**BSP** primale und duale Unzulässigkeit

12.4. **Beispiel.** Wir betrachten das primale LP

| | |
|---|---|
| maximiere | $x_1 + x_2$ |
| unter den Nebenbedingungen | $x_1 - x_2 \leq 0$ |
| | $-x_1 + x_2 \leq -1$ |
| | $x_1, x_2 \geq 0$ |

mit dem zugehörigen dualen LP

| | |
|---|---|
| minimiere | $-y_2$ |
| unter den Nebenbedingungen | $y_1 - y_2 \geq 1$ |
| | $-y_1 + y_2 \geq 1$ |
| | $y_1, y_2 \geq 0$ |

Addition der beiden Nebenbedingungen liefert für das primale Problem den Widerspruch $0 \leq -1$ und für das duale Problem den Widerspruch $0 \geq 2$; damit sind beide Probleme unzulässig. ♣

Sind beide Probleme zulässig und beschränkt, dann kann man zeigen, dass das Simplexverfahren für das primale Problem implizit auch das duale Problem löst und umgekehrt (siehe [V, Section 5.4]). Man erhält also optimale Lösungen für *beide* Probleme; die optimale Lösung des dualen Problems kann dann als Zertifikat

für die Lösung des primalen Problems dienen: Die Zulässigkeit beider Lösungen kann man leicht testen, und die Gleichheit der Zielfunktionswerte festzustellen ist noch einfacher.

Zum Abschluss dieses Abschnitts beweisen wir noch ein Optimalitätskriterium.

**SATZ** komplementärer Schlupf

12.5. **Satz.** *Seien das primale LP*

$$\max_{\boldsymbol{x}}\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

*und das zugehörige duale LP*

$$\min_{\boldsymbol{y}}\{\langle \boldsymbol{b}, \boldsymbol{y}\rangle \mid A^\top \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\}$$

*beide zulässig. Wir führen Schlupfvariable $\boldsymbol{w} = \boldsymbol{b} - A\boldsymbol{x}$ und $\boldsymbol{z} = A^\top \boldsymbol{y} - \boldsymbol{c}$ ein. Dann gilt: Ein Paar $(\boldsymbol{x}, \boldsymbol{w})$, $(\boldsymbol{y}, \boldsymbol{z})$ von zulässigen Lösungen ist genau dann optimal, wenn gilt*

$$\begin{aligned} x_j z_j &= 0 \quad \textit{für alle } 1 \leq j \leq n \qquad \textit{und} \\ w_i y_i &= 0 \quad \textit{für alle } 1 \leq i \leq m. \end{aligned}$$

Es muss also jeweils entweder eine Variable $x_j$ (bzw. $y_i$) oder die entsprechende Schlupfvariable $z_j$ (bzw. $w_i$) des anderen Problems verschwinden (oder beide).

*Beweis.* Wir erinnern uns an den Beweis des schwachen Dualitätssatzes 11.3:

$$\langle \boldsymbol{c}, \boldsymbol{x}\rangle = \boldsymbol{c}^\top \boldsymbol{x} \leq \boldsymbol{y}^\top A\boldsymbol{x} \leq \boldsymbol{y}^\top \boldsymbol{b} = \langle \boldsymbol{b}, \boldsymbol{y}\rangle .$$

Der starke Dualitätssatz 12.3 sagt uns, dass $\boldsymbol{x}$ und $\boldsymbol{y}$ genau dann optimal sind, wenn im schwachen Dualitätssatz Gleichheit gilt. Das bedeutet, dass beide Ungleichungen in der Ungleichungskette oben scharf sein müssen:

$$\boldsymbol{c}^\top \boldsymbol{x} = \boldsymbol{y}^\top A\boldsymbol{x} = \boldsymbol{y}^\top \boldsymbol{b}$$

oder äquivalent

$$0 = (\boldsymbol{y}^\top A - \boldsymbol{c}^\top)\boldsymbol{x} = \langle \boldsymbol{x}, \boldsymbol{z}\rangle \quad \text{und} \quad 0 = \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{y}\rangle .$$

Wegen $\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \geq \boldsymbol{0}$ heißt das genau, dass jeder Term in den beiden Summen

$$\langle \boldsymbol{x}, \boldsymbol{z}\rangle = x_1 z_1 + \ldots + x_n z_n$$

und

$$\langle \boldsymbol{w}, \boldsymbol{y}\rangle = w_1 y_1 + \ldots + w_m y_m$$

verschwinden muss. ❑

## 13. Ganzzahlige lineare Optimierung

Zum Abschluss unserer Diskussion der linearen Optimierung wollen wir uns noch etwas näher mit dem Fall beschäftigen, dass alle oder einige der Entscheidungsvariablen nur ganzzahlige Werte annehmen dürfen. Solche Einschränkungen treten in der Praxis häufig auf, da zum Beispiel bei der Organisation einer Fahrzeugflotte jeweils nur eine ganzzahlige Anzahl von Fahrzeugen eine bestimmte Route befahren kann. Auch viele Probleme der diskreten Optimierung lassen sich mit Hilfe eines ILP (*integer linear program*) formulieren, selbst wenn sie zunächst keine offensichtliche lineare Struktur haben.

**BSP**
Problem des Handlungsreisenden

13.1. **Beispiel.** (Siehe [V, Section 23.2].) Ein bekanntes Problem der diskreten Optimierung ist das Problem des Handlungsreisenden (*Traveling salesman problem* oder, politisch korrekt, *traveling salesperson problem*). Der oder die Handlungsreisende befindet sich in der Stadt $S_0$ und muss auf seiner/ihrer Runde die Städte $S_1, \dots, S_n$ besuchen und dann nach $S_0$ zurückkehren. Die Fahrt von $S_i$ nach $S_j$ ist mit Kosten $c_{ij} > 0$ verbunden. Das Problem besteht darin, eine Rundreise zu finden, die die Gesamtkosten minimiert. Da die Anzahl $n!$ der möglichen Rundreisen mit $n$ sehr stark wächst, ist das naive (oder brute-force) Verfahren, alle Möglichkeiten aufzuzählen und zu vergleichen, schon für recht kleine $n$ nicht mehr durchführbar.

Für die Formulierung als ILP führt man Entscheidungsvariablen $x_{ij} \in \{0, 1\}$ (für $0 \le i, j \le n$) ein mit der Bedeutung $x_{ij} = 1$ genau dann, wenn $S_j$ auf der Rundreise direkt nach $S_i$ besucht wird. Damit lässt sich die Zielfunktion (Kosten) schon einmal als lineare Funktion

$$z = \sum_{i,j} c_{ij} x_{ij}$$

schreiben. Wir müssen jetzt noch Nebenbedingungen formulieren, die sicherstellen, dass die $x_{ij}$ wirklich eine Rundtour beschreiben. Eine notwendige Bedingung ist sicherlich, dass man von genau einer Stadt nach $S_i$ kommt und von dort zu genau einer Stadt weiterreist. Das übersetzt sich in die Bedingungen

$$\forall i \in \{0, \dots, n\} \colon \sum_{j=0}^{n} x_{ji} = \sum_{j=0}^{n} x_{ij} = 1 \,.$$

Damit ist es aber immer noch möglich, dass die $x_{ij}$ mehrere disjunkte Rundtouren beschreiben. Um das auszuschließen, führt man weitere Variablen $t_0, \dots, t_n$ ein, sodass $t_i \in \{0, 1, \dots, n\}$ die Stellung von $S_i$ in der Rundtour angibt. Es ist also $t_0 = 0$ und $t_j = t_i + 1$, falls $x_{ij} = 1$ ist (falls $j > 0$; für $j = 0$ wäre die Bedingung $t_0 = t_i - n$). Wir müssen das jetzt noch in lineare Ungleichungen übersetzen. Das geschieht in der Form

$$\forall i \ge 0, j \ge 1 \colon t_j \ge t_i + 1 - (n+1)(1 - x_{ij}) = t_i + (n+1)x_{ij} - n \,.$$

Wegen $0 \le t_k \le n$ bedeutet das keine Einschränkung, wenn $x_{ij} = 0$ ist. Im Fall $x_{ij} = 1$ erhalten wir $t_j \ge t_i + 1$. Entlang der Tour müssen also die $t_i$ jeweils um mindestens 1 zunehmen; wegen der Beschränkung $0 \le t_k \le n$ muss das Wachstum dann sogar genau 1 betragen. Wir erhalten folgendes Problem.

$$
\begin{aligned}
\text{minimiere} \quad & \sum_{i,j=0}^{n} c_{ij} x_{ij} \\
\text{unter den Nebenbedingungen} \quad & \forall j \in \{0,\dots,n\}\colon \sum_{i=0}^{n} x_{ij} = 1 \\
& \forall i \in \{0,\dots,n\}\colon \sum_{j=0}^{n} x_{ij} = 1 \\
& t_0 = 0 \\
& \forall i \in \{0,\dots,n\}, j \in \{1,\dots,n\}\colon t_j \geq t_i + (n+1)x_{ij} - n \\
& \forall i \in \{1,\dots,n\}\colon t_i \leq n \\
& \forall i,j \in \{0,\dots,n\}\colon x_{ij} \in \{0,1\} \\
& \forall i \in \{1,\dots,n\}\colon t_i \geq 0
\end{aligned}
$$

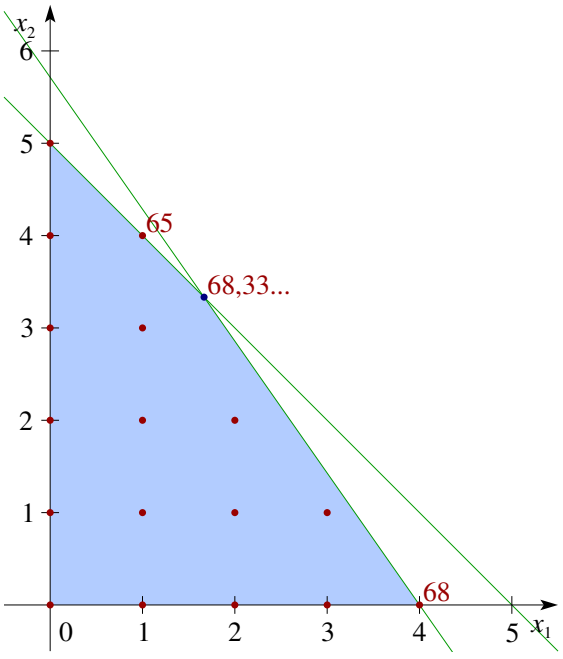
Das ist ein *gemischt-ganzzahliges lineares Optimierungsproblem* (MILP, *mixed integer linear program*), da nur die $x_{ij}$ ganzzahlig sein müssen (für die $t_i$ folgt es aus den Nebenbedingungen). Wir haben also das Problem des Handlungsreisenden für $n+1$ Städte in ein MILP mit $(n+1)^2$ ganzzahligen (oder genauer binären, also mit Werten in $\{0,1\}$) und $n+1$ reellen Variablen und ähnlich vielen Nebenbedingungen übersetzt. ♣

Wie kann man so ein (M)ILP lösen? Wir betrachten ein einfaches Beispiel (siehe [V, Section 23.5]).

13.2. **Beispiel.** **BSP** ILP

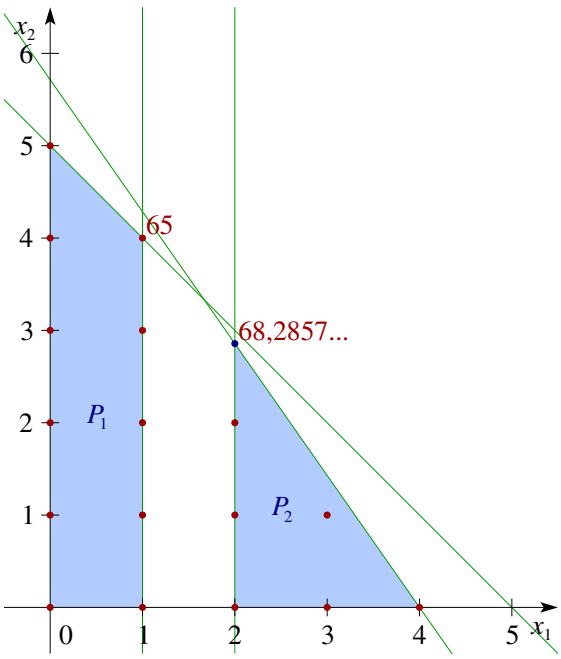
$$
\begin{aligned}
\text{maximiere} \quad & 17x_1 + 12x_2 \\
\text{u. d. Nebenbed.} \quad & 10x_1 + 7x_2 \leq 40 \\
& x_1 + x_2 \leq 5 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}
$$

Der durch die Ungleichungen definierte Bereich ist in der Skizze rechts hellblau ausgefüllt; die Punkte mit ganzzahligen Koordinaten darin sind rot markiert. Für einige Punkte ist der Wert der Zielfunktion (ebenfalls in rot) angegeben.




Man kann nun zunächst die *LP-Relaxierung* des ILPs betrachten; das ist das LP, das entsteht, wenn man die Ganzzahligkeitsbedingung weglässt.

In unserem Beispiel wird das Optimum im (dunkelblauen) Punkt $(\frac{5}{3}, \frac{10}{3})$ mit dem Wert $68\frac{1}{3}$ erreicht. Hätte dieser Punkt ganzzahlige Koordinaten, dann wäre er auch die optimale Lösung des ILPs. Da das aber nicht der Fall ist, spalten wir das Problem auf: Die $x_1$-Koordinate der optimalen Lösung des relaxierten Problems liegt zwischen 1 und 2. Für eine Lösung des ILPs muss aber $x_1 \in \mathbb{Z}$, also insbesondere entweder $x_1 \leq 1$ oder $x_1 \geq 2$ gelten. Wir betrachten also zwei neue LPs, nämlich $P_1$ mit der zusätzlichen Bedingung $x_1 \leq 1$ und $P_2$ mit der zusätzlichen Bedingung $x_1 \geq 2$.




Für $P_1$ finden wir als Optimum den Punkt $(1, 4)$ mit dem Zielfunktionswert 65. Dieser Punkt hat ganzzahlige Koordinaten, liefert also das Optimum von $P_1$ als ILP. Gleichzeitig wissen wir, dass das Optimum des ursprünglichen ILPs mindestens 65 ist.

Für $P_2$ ergibt sich als Optimum der Punkt $(2, \frac{20}{7})$ mit einem Zielfunktionswert von $68\frac{2}{7}$. Wäre dieser Wert $\leq 65$, dann wüssten wir, dass das Optimum nicht von $P_2$ kommen kann und wären fertig. Da dies aber nicht der Fall ist, müssen wir das Verfahren fortsetzen. Die $x_2$-Koordinate der optimalen Lösung des relaxierten $P_2$ liegt zwischen 2 und 3, also teilen wir weiter auf: $P_3$ ist das ILP $P_2$ mit der zusätzlichen Bedingung $x_2 \leq 2$, $P_4$ hat entsprechend die zusätzliche Bedingung $x_2 \geq 3$. An der Skizze sieht man, dass $P_4$ unzulässig ist. Das Optimum des relaxierten $P_3$ ist bei $(\frac{13}{5}, 2)$ mit dem Zielfunktionswert $68\frac{1}{5} > 65$. Da die $x_1$-Koordinate zwischen 2 und 3 liegt, generieren wir zwei neue ILPs $P_5$ (zusätzliche Bedingung $x_1 \leq 2$) und $P_6$ (zusätzliche Bedingung $x_1 \geq 3$). Das Optimum für das relaxierte $P_5$ liegt im ganzzahligen Punkt $(2, 2)$ mit dem Zielfunktionswert $58 < 65$; für $P_6$ ist das Optimum bei $(3, \frac{10}{7})$ mit dem Zielfunktionswert $68\frac{1}{7}$. Das führt zu zwei weiteren ILPs $P_7$ ($x_2 \leq 1$) und $P_8$ ($x_2 \geq 2$). $P_8$ ist unzulässig, und $P_7$ liefert $(\frac{33}{10}, 1)$ mit dem Zielfunktionswert $68\frac{1}{10}$. Weitere Unterteilung in $P_9$ ($x_1 \leq 3$) und $P_{10}$ ($x_1 \geq 4$) ergibt $(3, 1)$ mit Zielfunktionswert 63 für $P_9$ und das Optimum $(4, 0)$ mit Zielfunktionswert 68 für $P_{10}$. Da 68 größer ist als der bisher beste Wert 65 für einen Punkt mit ganzzahligen Koordinaten, ist $(4, 0)$ die optimale Lösung unseres Problems. ♣

Das im Beispiel skizzierte Verfahren heißt (wie schon früher erwähnt) *Branch-and-Bound*. „Branch“, weil man einen verzweigten Baum von immer weiter eingeschränkten ILPs bearbeiten muss, und „Bound“, weil man die jeweils beste bekannte ganzzahlige Lösung dazu benutzen kann, ganze Äste des Baums abzuschneiden, wenn das entsprechende relaxierte LP keine bessere Lösung liefern kann. Das lässt sich am elegantesten als ein *rekursiver* Algorithmus formulieren. Dafür nehmen wir an, dass „$\boldsymbol{x}, z :=$ löse_LP$(P)$“ für ein LP $P$ in $\boldsymbol{x}$ eine optimale Lösung und in $z$ den zugehörigen Zielfunktionswert liefert, oder aber in $\boldsymbol{x}$ den speziellen Wert „unzulässig“ und in $z$ den Wert $-\infty$.

**ALGO** Branch-and-Bound

**input:** ein ILP $P$
**output:** eine optimale Lösung $\boldsymbol{x} \in \mathbb{Z}^n$ von $P$, oder „unzulässig“

**function** rec($P$, $b$) // $P$ ein ILP, $b$ untere Schranke für das Optimum
  $\boldsymbol{x}^*, z^* :=$ löse_LP($P$) // LP-Relaxierung lösen
  **if** $z^* \leq b$ **then return** $-, -\infty$ **end if** // Ast kann kein Optimum liefern
  **if** $\boldsymbol{x}^* \in \mathbb{Z}^n$ **then return** $\boldsymbol{x}^*, z^*$ **end if** // Optimum gefunden
  Wähle $i$ mit $x_i^* \notin \mathbb{Z}$; $l := \lfloor x_i^* \rfloor$
  $\boldsymbol{x}', z' :=$ rec($P \cup \{x_i \leq l\}, b$) // erstes neues ILP
  $\boldsymbol{x}'', z'' :=$ rec($P \cup \{x_i \geq l+1\}, \max\{b, z'\}$) // zweites neues ILP
  **if** $\max\{z', z''\} = -\infty$ **then**
    **return** $-, -\infty$ // kein neues Optimum
  **else**
    **if** $z' > z''$ **then return** $\boldsymbol{x}', z'$ **else return** $\boldsymbol{x}'', z''$ **end if**
  **end if**
**end function**

$\boldsymbol{x}, z :=$ rec($P, -\infty$)
**if** $z = -\infty$ **then**
  **return** „unzulässig“
**else**
  **return** $\boldsymbol{x}$
**end if**

Die Hilfsfunktion rec, die sich rekursiv mit den jeweils erweiterten ILPs aufruft, gibt dabei entweder einen ganzzahligen Vektor $\boldsymbol{x}$ zurück, der das ILP $P$ optimal löst mit Zielfunktionswert $z > b$, oder einen undefinierten Wert „$-$“ für $\boldsymbol{x}$ und $-\infty$ für $z$.

Wie schon früher diskutiert, eignet sich der duale Simplex-Algorithmus sehr gut dafür, ausgehend von einer optimalen Lösung von $P$ in wenigen Schritten eine optimale Lösung für $P$ mit einer zusätzlichen Ungleichung zu finden. Wenn man den obigen Algorithmus implementiert, wird man das also entsprechend verwenden.

Um tatsächlich einen wohldefinierten Algorithmus zu bekommen, muss man noch festlegen, wie man die Koordinate $x_i$ wählt, nach der man das Problem weiter aufspaltet (und auch, welches der beiden neuen Probleme man zuerst löst; oben haben wir dafür der Einfachheit halber eine feste Reihenfolge vorgesehen). Wie man das macht, kann sich deutlich auf die Effizienz auswirken.

Man kann mit Innere-Punkte-Methoden lineare Optimierungsprobleme in Polynomzeit (approximativ) lösen. Wie sieht es mit der Effizienz (oder *Komplexität*) beim Lösen von *ganzzahligen* linearen Optimierungsproblemen aus? Dazu erst einmal ein paar allgemeine Begriffe.

**DEF P, NP**

13.3. **Definition.** Ein Problem $A$ gehört zur Klasse **P**, wenn es durch einen Algorithmus gelöst werden kann, dessen Laufzeit für Eingabedaten der (geeignet gemessenen) Länge $n$ von der Form $O(n^d)$ ist für eine Zahl $d$ („polynomial beschränkte Laufzeit“).

Ein Problem $A$ kann *polynomial* auf ein Problem $B$ *reduziert* werden, geschrieben $A \preccurlyeq_{\mathbf{P}} B$, wenn es einen Algorithmus gibt, der in polynomial in der Länge

der Eingabedaten von $A$ beschränkter Laufzeit aus diesen Daten Eingabedaten für $B$ generiert, und einen Algorithmus, der ebenfalls in polynomial beschränkter Laufzeit aus der Ausgabe von $B$ die zugehörige Ausgabe von $A$ erzeugt.

Ein Problem $A$ gehört zur Klasse **NP**, wenn seine Ausgabe (evtl. unter Hinzuziehung eines „Zertifikats“ polynomial beschränkter Länge) in polynomial beschränkter Zeit verifiziert werden kann. $A$ heißt **NP**-*schwer* (englisch **NP**-*hard*), wenn jedes Problem $B \in \mathbf{NP}$ polynomial auf $A$ reduziert werden kann. Ein **NP**-schweres Problem, das selbst in **NP** ist, heißt **NP**-*vollständig*. ♢

Genau genommen sind die obigen Begriffe für *Entscheidungsprobleme* definiert, also Probleme, deren Antwort „ja“ oder „nein“ lautet. Man kann aber viele Berechnungsprobleme auf Entscheidungsprobleme zurückführen und/oder umgekehrt; daher wird diese Unterscheidung im üblichen Sprachgebrauch gerne verwischt. Für das Problem des Handlungsreisenden wäre das Entscheidungsproblem etwa *Gibt es eine Rundtour mit Kosten* $< c$*?*.

Wie wir im Numerik-Teil gesehen haben, ist zum Beispiel das Lösen eines linearen Gleichungssystems in **P**. Ist $B \in \mathbf{P}$ und gilt $A \preccurlyeq_{\mathbf{P}} B$, dann ist auch $A \in \mathbf{P}$ (Übung).

Probleme in **P** werden üblicherweise als effizient lösbar angesehen. Meistens sind der Exponent $d$ und die implizierte Konstante in der Schranke für die Laufzeit relativ klein, sodass diese Gleichsetzung berechtigt ist. Allerdings kann ein polynomialer Algorithmus, wenn $d$ oder die Konstante vor dem $n^d$ groß sind, durchaus ineffizienter sein als einer, der exponentiell viel Zeit braucht, jedenfalls für nicht allzu großes $n$. Der Punkt ist, dass die exponentielle Laufzeit irgendwann sehr viel schneller wächst als die polynomiale, wenn $n$ groß wird.

Man beachte, dass **NP** *nicht* für „nicht polynomial“ steht. Vielmehr steht das **N** hier für „nicht-deterministisch“: Ein Problem ist in **NP**, wenn es nicht-deterministisch in Polynomzeit lösbar ist. „Nicht-deterministisch“ bedeutet dabei, dass der Algorithmus raten darf; das Problem ist in **NP**, wenn dabei in Polynomzeit das Richtige herauskommt, vorausgesetzt, der Algorithmus hat passend geraten. Die Formulierung in der Definition ist dazu äquivalent; das Zertifikat besteht in der Information darüber, wie man richtig raten muss. Klarerweise gilt $\mathbf{P} \subset \mathbf{NP}$: Die Lösung eines Problems in **P** kann ich in Polynomzeit verifizieren, indem ich sie mit dem Lösungsalgorithmus nachrechne.

Ein einigermaßen typisches Problem in **NP** ist das Faktorisierungsproblem $F$: *Sei* $N \in \mathbb{Z}_{\geq 2}$ *eine zusammengesetzte Zahl. Finde einen nichttrivialen Teiler von* $N$. Ist ein Teiler von $N$ gegeben, dann lässt sich diese Eigenschaft in Polynomzeit (durch Ausführen einer Division) verifizieren. Auf der anderen Seite ist kein Algorithmus mit polynomial (in $\log N$) beschränkter Laufzeit bekannt, der das Faktorisierungsproblem löst. Es ist also nicht bekannt, ob $F \in \mathbf{P}$ ist. (Dagegen ist bekannt, dass das Problem *Entscheide, ob* $N$ *eine Primzahl ist* in **P** ist. Diese Erkenntnis ist allerdings relativ neu.)

Es gibt also Probleme in **NP**, von denen man nicht weiß, ob sie in **P** sind. Es ist ein wichtiges ungelöstes Problem, ob „$\mathbf{P} = \mathbf{NP}$“ gilt oder nicht. Wer es löst, bekommt eine Million Dollar (es ist eines der „Millenniums-Probleme“ der Clay Foundation). Um zu zeigen, dass $\mathbf{P} = \mathbf{NP}$ ist, würde es genügen, für *ein* **NP**-vollständiges (oder **NP**-schweres) Problem $A$ einen polynomialen Algorithmus zu finden, denn jedes andere Problem in **NP** lässt sich (nach Definition) polynomial auf $A$ reduzieren, wäre dann also auch in **P**. Allerdings ist das bisher noch niemandem gelungen. Entsprechend würde es für den Nachweis, dass $\mathbf{P} \neq \mathbf{NP}$ ist, genügen, für *ein* Problem in **NP** nachzuweisen, dass es nicht in **P** ist. Auch das ist noch niemandem

gelungen. Die vorherrschende Meinung ist, dass **P** eine echte Teilklasse von **NP** ist. Ist das so, dann bedeutet die **NP**-Vollständigkeit (oder **NP**-Schwierigkeit) eines gegebenen Problems, dass es *nicht* effizient lösbar ist (jedenfalls nicht mit den üblichen Computern; bei Quantencomputern sieht das schon wieder anders aus).

Es gibt tatsächlich Probleme, von denen man zeigen kann, dass sie **NP**-vollständig sind. Hier findet man eine Liste von 21 solchen Problemen. Leider gehört das Lösen von ILPs mit binären Variablen dazu (und das Problem des Handlungsreisenden ist, wie auch das Lösen allgemeiner (M)ILPs, **NP**-schwer). Das bedeutet, dass es wahrscheinlich nicht möglich ist, beliebige ILPs effizient zu lösen. Auf der anderen Seite beziehen sich solche Aussagen immer auf den schlimmsten Fall. Viele Probleme, die in der Praxis auftreten, lassen sich durchaus einigermaßen effizient lösen: Die größten exakt gelösten Instanzen des Problems des Handlungsreisenden haben $n$ in der Größenordnung von einigen Zehntausend. In der Praxis genügt auch häufig eine Lösung, die nicht allzu weit vom Optimum entfernt ist; für solche approximativen Lösungen kann es dann wieder effiziente Verfahren geben. Für das „metrische“ Problem des Handlungsreisenden (mit $c_{ij} = c_{ji}$ und $c_{ik} \leq c_{ij} + c_{jk}$ (Dreiecksungleichung)) kann man zum Beispiel in Polynomzeit eine Lösung finden, die höchstens um den Faktor 1,5 schlechter ist als das Optimum.

# Teil III: Computeralgebra

## 14. Einführung und Grundlagen

Was ist Computeralgebra?

Die Computeralgebra ist Teil eines Gebiets, das als *Wissenschaftliches Rechnen* bezeichnet wird. Dabei geht es darum, mit Hilfe des Computers mathematische Probleme zu lösen. Die dabei verwendeten Verfahren lassen sich grob einteilen in *numerische Algorithmen* und *symbolische Algorithmen*. Mit den numerischen Verfahren beschäftigt sich die *Numerische Mathematik*, mit den symbolischen zu großen Teilen die *Computeralgebra*. Die wesentlichen Unterschiede sind etwa die folgenden.

Bei den numerischen Verfahren sind die zu Grunde liegenden Objekte *kontinuierlicher* Natur (etwa Vektorfelder), die durch (oft sehr große Anzahlen an) *Gleitkommazahlen* im Computer *näherungsweise* dargestellt werden. Relevante Probleme bei der Konstruktion und der Untersuchung von Algorithmen sind *Konvergenz* (kommen wir bei entsprechend hohem Aufwand der wahren Lösung beliebig nahe?), Kontrolle der *Rundungsfehler*, die bei Rechnungen mit Gleitkommazahlen auftreten, und natürlich die *Effizienz* der Verfahren. Häufig sind *große Mengen* von Daten zu verarbeiten, an denen in gleicher Weise und vielfach hintereinander relativ einfache Berechnungen durchgeführt werden. Man denke zum Beispiel an die numerische Lösung eines Systems von partiellen Differentialgleichungen, etwa bei der Wettervorhersage. Wir haben im ersten Teil dieser Vorlesung einen Einblick in Fragestellungen und Methoden der Numerischen Mathematik erhalten.

Symbolische Verfahren dagegen rechnen *exakt*; die zu Grunde liegenden Objekte sind *algebraischer* und damit *diskreter* Natur, etwa Polynome in mehreren Variablen mit rationalen Zahlen als Koeffizienten. Diese Objekte können sehr *komplex* sein, und diese Komplexität muss durch geeignete Datenstrukturen im Computer abgebildet werden. Die verwendeten Algorithmen sind dementsprechend ebenfalls *komplex*, und das Hauptproblem liegt darin, *effiziente* Algorithmen und Datenstrukturen zu finden. Häufig beruhen diese auf höchst nichttrivialen Resultaten der Algebra und Zahlentheorie. Typische Aufgaben sind etwa die Faktorisierung von ganzen Zahlen (das RSA-Kryptosystem beruht darauf, dass dafür kein wirklich effizientes Verfahren bekannt ist) oder das Auffinden von rationalen Lösungen polynomialer Gleichungssysteme.

Es gibt allerdings auch Mischformen. Man kann etwa partielle Differentialgleichungen symbolisch „vorverarbeiten“, um sie in eine für numerische Algorithmen besser geeignete Form zu bringen. Auf der anderen Seite kann es effizienter sein, eine symbolische Rechnung (etwa mit ganzen Zahlen) numerisch, also näherungsweise, durchzuführen, wenn man dadurch das exakte Resultat hinreichend gut approximieren kann.

**BSP**
symbolisch/
numerisch

14.1. **Beispiel.** Wir betrachten folgendes Problem:

$$f(X) = X^4 - a_3X^3 + a_2X^2 - a_1X + a_0$$

sei ein Polynom mit ganzzahligen Koeffizienten. Wir bezeichnen die Nullstellen von $f$ in $\mathbb{C}$ mit $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. Wir wollen das Polynom $g(X)$ berechnen, dessen Nullstellen die sechs verschiedenen möglichen Ausdrücke der Form $\alpha_i\alpha_j$ sind mit

$1 \leq i < j \leq 4$. Die Theorie der symmetrischen Polynome sagt uns, dass die Koeffizienten von $g$ Polynome in den $a_j$ mit ganzzahligen Koeffizienten sind. Explizit gilt (wie durch eine symbolische Rechnung nachgewiesen werden kann)

$$g(X) = X^6 - a_2X^5 + (a_1a_3 - a_0)X^4 + (2a_0a_2 - a_0a_3^2 - a_1^2)X^3 + a_0(a_1a_3 - a_0)X^2 - a_0^2a_2X + a_0^3\,.$$

Eine rein symbolische Lösung wäre, in diesen Ausdruck die Werte der Koeffizienten des gegebenen Polynoms einzusetzen. Alternativ kann man die Nullstellen $\alpha_i$ als komplexe Zahlen näherungsweise berechnen, daraus die Nullstellen $\beta_1, \ldots, \beta_6$ von $g$ bestimmen und dann $g(X)$ näherungsweise als

$$g(X) = (X - \beta_1)(X - \beta_2)\cdots(X - \beta_6)$$

erhalten. Wenn die Näherung gut genug ist, bekommen wir die wahren Koeffizienten (die ja ganze Zahlen sind) durch Runden. Ein wesentlicher Unterschied zum in der Numerik Üblichen ist hier, dass die numerische Rechnung unter Umständen mit sehr hoher Genauigkeit (also Anzahl an Nachkommastellen) durchgeführt werden muss, damit das Ergebnis nahe genug an der exakten Lösung liegt.

In diesem Beispiel ist das Einsetzen der Koeffizienten in die explizite Formel für $g$ nicht sehr aufwendig. Ein ähnlich gelagerter Fall tritt auf für $f$ vom Grad 6 und $g$ das Polynom mit den zehn Nullstellen der Form $\alpha_{i_1}\alpha_{i_2}\alpha_{i_3} + \alpha_{i_4}\alpha_{i_5}\alpha_{i_6}$ mit $\{i_1, \ldots, i_6\} = \{1, \ldots, 6\}$, das zu bestimmen für einen Algorithmus aus meinem Forschungsgebiet wichtig ist. Hier hat der explizite Ausdruck für $g$ über 160 Terme, und der Ansatz über eine numerische Näherung gewinnt an Charme. ♣

Für diesen Teil der Vorlesung werden Grundkenntnisse in Algebra (Theorie des euklidischen Rings $\mathbb{Z}$, Polynomringe, endliche Körper) vorausgesetzt, wie sie in den Vorlesungen „Einführung in die Zahlentheorie und algebraische Strukturen“ und „Einführung in die Algebra“ vermittelt werden. Wir werden typische Methoden und Fragestellungen der Computeralgebra am Beispiel der effizienten Multiplikation von Polynomen (und ganzen Zahlen) kennenlernen.

Als Literatur zum Thema ist das sehr schöne Buch [GG] von von zur Gathen und Gerhard zu empfehlen.

Bevor wir in die Materie wirklich einsteigen können, müssen wir erst einmal eine Vorstellung davon haben, wie die grundlegenden algebraischen Objekte, nämlich ganze Zahlen und Polynome, im Computer dargestellt werden können, und wie wir die Komplexität bzw. Effizienz unserer Algorithmen messen können.

Computer arbeiten mit *Datenworten,* die aus einer gewissen festen Anzahl an Bits bestehen (heutzutage meist 64). Da wir in der Computeralgebra häufig mit sehr großen ganzen Zahlen zu tun haben, reicht ein Wort im Allgemeinen nicht aus, um eine ganze Zahl zu speichern. Man wird also ein Array (eine geordnete Liste) von solchen Worten verwenden. Dann muss man zusätzlich noch wissen, wie lang das Array ist, und man muss das Vorzeichen in geeigneter Weise codieren. Mathematisch gesehen, schreiben wir eine ganze Zahl $N$ als

$$N = (-1)^s \sum_{j=0}^{n-1} a_j B^j$$

wobei $s \in \{0, 1\}$ und $B$ die der Wortlänge entsprechende Basis ist; wir nehmen im Folgenden $B = 2^{64}$ an. Die „Ziffern“ $a_j$ erfüllen $0 \leq a_j < B$. Wenn wir $n < 2^{63}$

annehmen (was realistisch ist, denn größere Zahlen würden jeglichen Speicherplatz sprengen), dann lässt sich $N$ im Computer darstellen als Folge

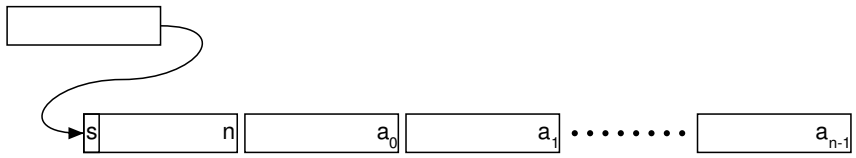
$$s \cdot 2^{63} + n, a_0, \ldots, a_{n-1}$$

von Worten. Es ist häufig sinnvoll, diese Darstellung zu *normalisieren* (also eindeutig zu machen), indem man fordert, dass $a_{n-1} \neq 0$ ist. Die Zahl 0 kann dann etwa durch das eine Wort 0 dargestellt werden (hat also $s = 0$ und $n = 0$). Die Zahl $n$ ist in diesem Fall die *Wortlänge* $\lambda(N)$ von $N$; es gilt für $N \neq 0$

**DEF** Wortlänge

$$\lambda(N) = \left\lfloor \frac{\log |N|}{\log B} \right\rfloor + 1 \,.$$

Man verwendet einen *Zeiger* auf das erste Wort der Darstellung von $N$ (also seine Adresse im Speicher), um $N$ im Computer zu repräsentieren:




Polynome in einer Variablen werden analog dargestellt:

$$f(X) = \sum_{j=0}^{n} a_j X^j \,,$$

wobei die Koeffizienten $a_j$ aus einem Ring $R$ kommen. Das erste Wort gibt wiederum die Länge ($n + 1$, wenn $n$ der Grad ist) an, es folgen die Koeffizienten $a_0, \ldots, a_n$ als Zeiger auf die entsprechenden Datenstrukturen (etwa für ganze Zahlen wie oben). Man wird verlangen, dass $a_n \neq 0$ ist; das Nullpolynom wird wieder durch das Nullwort dargestellt.

Nachdem die Datenstrukturen geklärt sind, müssen die grundlegenden arithmetischen Operationen implementiert werden. Für ganze Zahlen sind dies etwa Vergleich, Negation, Addition, Multiplikation und Division mit Rest. Für Polynome bleibt vom Vergleich nur der Test auf Gleichheit übrig, und bei der Division mit Rest nehmen wir an, dass der Divisor Leitkoeffizient 1 hat. (Da die Negation (Vorzeichenwechsel) im wesentlichen trivial ist, ist die Subtraktion als Negation des Subtrahenden plus Addition eingeschlossen.)

Geht man von normierten Darstellungen aus, dann sind zwei ganze Zahlen genau dann gleich, wenn ihre Darstellungen gleich sind (also gleiches Vorzeichen und gleiche Länge, und dann übereinstimmende „Ziffern“). Analog gilt für Polynome, dass sie genau dann gleich sind, wenn sie denselben Grad $n$ haben und ihre Koeffizienten $a_j$ jeweils gleich sind (wobei hier der Algorithmus zum Test der Gleichheit im Koeffizientenring $R$ verwendet wird). Hier ist eine recht ausführliche Version des Vergleichsalgorithmus für ganze Zahlen.

---

**ALGO** Vergleich in $\mathbb{Z}$

**function** compare($M$, $N$)

**input:** $M = (-1)^s \sum_{j=0}^{m-1} a_j B^j$, $N = (-1)^t \sum_{j=0}^{n-1} b_j B^j$

**output:** $-1$ falls $M < N$, 0 falls $M = N$, 1 falls $M > N$.

**if** $s \neq t$ **then return** $(-1)^s$ **end if**

// Ab hier haben $M$ und $N$ dasselbe Vorzeichen.

```
  if m > n then return (−1)^s end if
  if m < n then return −(−1)^s end if
  // Ab hier gilt m = n.
  for j = n − 1 to 0 by −1 do
    if a_j > b_j then return (−1)^s end if
    if a_j < b_j then return −(−1)^s end if
  end for
  // Wenn wir hier ankommen, gilt M = N.
  return 0
end function
```

---

Der Additionsalgorithmus hat einen ähnlichen Aufbau. Je nach Vorzeichen müssen die Beträge addiert oder subtrahiert werden. Wir verwenden den Additionsbefehl des Prozessors, der für die Eingabeworte $u$ und $v$ und das Ausgabewort $w$ der Relation

$$w + c' \cdot B = u + v + c$$

entspricht, wobei $c, c' \in \{0, 1\}$ den Wert des *Carry-Flags* (Übertragsbit) des Prozessors vor und nach der Ausführung bezeichnet. Für die Subtraktion gibt es analog einen Befehl entsprechend der Relation

$$w - c' \cdot B = u - v - c\,.$$

**ALGO**
Addition in $\mathbb{Z}$

---

```
function add(M, N)
input:  M = (−1)^s Σ_{j=0}^{m−1} a_j B^j, N = (−1)^t Σ_{j=0}^{n−1} b_j B^j.
output: M + N = (−1)^u Σ_{j=0}^{k−1} d_j B^j.
  if s = t then
    // M und N haben gleiches Vorzeichen: addieren.
    if m < n then vertausche M und N end if
    // Ab hier ist m ≥ n.
    k := m // Länge der Summe.
    u := s // Vorzeichen der Summe.
    c := 0 // Initialisierung Übertrag.
    for j = 0 to n − 1 do
      d_j + c · B := a_j + b_j + c // Addition mit Übertrag
    end for
    for j = n to m − 1 do
      d_j + c · B := a_j + 0 + c // Addition mit Übertrag
    end for
    if c = 1 then
      // Ergebnis wird länger.
      d_k := c
      k := k + 1
    end if
  else // M und N haben verschiedenes Vorzeichen: subtrahieren.
    ...
  end if
```

**return** $(u, k, d_0, d_1, \ldots, d_{k-1})$
**end function**

---

Was können wir über die Effizienz dieses Additionsalgorithmus sagen? Wie bereits im Numerik-Teil besprochen, misst man den Aufwand über die Anzahl der Rechenoperationen, die durchgeführt werden müssen, in Abhängigkeit von der Größe der Eingabe. Die Größe der Eingabedaten wird meistens in Datenworten gemessen; für die Addition wäre das also $\lambda(M) + \lambda(N)$ für die beiden Zahlen $M$ und $N$. Die Anzahl der Additionen mit Übertrag von einzelnen Datenworten ist $\max\{m, n\} = \max\{\lambda(M), \lambda(N)\}$; dazu kommen noch beschränkt viele weitere Operationen. Die Komplexität ist also in $\Theta(\lambda(M) + \lambda(N))$ und damit *linear* in der Länge der Eingabedaten. Es ist klar, dass *jeder* Algorithmus, der zwei ganze Zahlen in der hier beschriebenen Darstellung addiert, mindestens lineare Komplexität haben muss: Da jedes Wort der beiden zu addierenden Zahlen das Ergebnis beeinflussen kann, muss die komplette Eingabe „angefasst“ werden.

Für die Addition von Polynomen gilt Entsprechendes. Hier verwendet man meistens die Anzahl der Operationen im Koeffizientenring $R$ als Maß für die Komplexität. Wir setzen die Definition

$$\sum_{i=0}^{n} a_i X^i + \sum_{i=0}^{n} b_i X^i = \sum_{i=0}^{n} (a_i + b_i) X^i$$

in einen Algorithmus um (dabei sei $a_i = 0$, falls $i$ größer ist als der Grad des Polynoms, entsprechend für $b_i$). Wir haben also $n + 1$ Additionen von Koeffizienten durchzuführen; die Komplexität ist also wiederum *linear*. Die *Wortkomplexität* kann größer sein; sie hängt vom Typ der Koeffizienten ab. Sind die Koeffizienten ganze Zahlen vom Betrag $\leq M$, dann erhalten wir eine Wortkomplexität in $\Theta(n \log M)$.

## 15. Schnellere Multiplikation

Wir erinnern uns an die „Schulmethode“ für die Multiplikation von Polynomen bzw. Zahlen in Dezimalschreibweise:

$(2x^2 - x + 1) \cdot (3x^2 - 2x + 1)$ :

| | |
|---|---|
| $1$ | $2x^2 - x + 1$ |
| $-2x$ | $-4x^3 + 2x^2 - 2x$ |
| $3x^2$ | $6x^4 - 3x^3 + 3x^2$ |
| | $6x^4 - 7x^3 + 7x^2 - 3x + 1$ |

$1234 \cdot 567$ :

| | |
|---|---|
| $7$ | $8638$ |
| $60$ | $7404$ |
| $500$ | $6170$ |
| | $699678$ |

Als Pseudocode für den etwas übersichtlicheren Fall der Polynome (wo es beim Addieren keinen Übertrag gibt) sieht das so aus:

**ALGO**
Multiplikation von Polynomen

---

**function** multiply($p$, $q$)
**input:** $p = \sum_{i=0}^{m} a_i X^i$, $q = \sum_{i=0}^{n} b_i X^i$.
**output:** $p \cdot q = \sum_{i=0}^{m+n} c_i X^i$.

```
  for i = 0 to m + n do c_i := 0 end for // Initialisierung
  for j = 0 to m do
    for k = 0 to n do
      c_{j+k} := c_{j+k} + a_j · b_k
    end for
  end for
  return (m + n, c_0, ..., c_{m+n})
end function
```

---

Wenn wir die Operationen im Koeffizientenring $R$ zählen, die im obigen Multiplikationsalgorithmus ausgeführt werden, kommen wir auf

$$(m+1)(n+1) \text{ Multiplikationen} \quad \text{und} \quad (m+1)(n+1) \text{ Additionen.}$$

Die Komplexität der Schulmethode für die Multiplikation zweier Polynome vom Grad $n$ ist also in $\Theta(n^2)$: Das Verfahren hat *quadratische* Komplexität.

Für die Komplexität der Schulmethode für die Multiplikation zweier ganzer Zahlen $M$ und $N$ erhalten wir entsprechend $\Theta(\lambda(M) \cdot \lambda(N))$.

Wir werden uns jetzt überlegen, dass das nicht das letzte Wort ist: Es gibt eine recht einfache Möglichkeit, schneller als in quadratischer Zeit zu multiplizieren. Der Grundgedanke dabei ist „Divide And Conquer“ („Teile und herrsche“): Man führt das Problem auf analoge Probleme kleinerer (meistens etwa halber) Größe zurück.

Für die Multiplikation zweier linearer Polynome $aX + b$ und $cX + d$:

$$(aX + b)(cX + d) = acX^2 + (ad + bc)X + bd$$

braucht man im klassischen Algorithmus, der der Formel entspricht, *vier* Multiplikationen und eine Addition im Koeffizientenring. Man kann den Koeffizienten von $X$ im Ergebnis aber auch schreiben als

$$ad + bc = (a + b)(c + d) - ac - bd\,.$$

Da man $ac$ und $bd$ sowieso berechnen muss, kommt man mit insgesamt *drei* Multiplikationen aus; der Preis, den man dafür bezahlt, besteht in drei zusätzlichen Additionen/Subtraktionen. Da Additionen aber im allgemeinen deutlich billiger sind als Multiplikationen, kann man damit etwas gewinnen.

**Die Karatsuba-Multiplikation.**

Wir wollen natürlich nicht nur lineare Polynome multiplizieren. Seien also jetzt $a, b \in R[X]$ mit $\deg a, \deg b < 2n$. Dann schreiben wir

$$a = a_1X^n + a_0\,, \qquad b = b_1X^n + b_0$$

mit Polynomen $a_0, a_1, b_0, b_1$ vom Grad $< n$. Wie vorher gilt dann

$$ab = a_1b_1X^{2n} + \big((a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0\big)X^n + a_0b_0\,.$$

Wenn wir die Teilprodukte mit der klassischen Methode berechnen (zu Kosten von $n^2$ Multiplikationen und $n^2$ Additionen), dann bekommen wir hier (M: Multiplikationen, A: Additionen/Subtraktionen in $R$)

- 3 Multiplikationen von Polynomen vom Grad $< n$: $3n^2\,\mathsf{M} + 3n^2\,\mathsf{A}$
- 2 Additionen von Polynomen vom Grad $< n$: $2n\,\mathsf{A}$
- 2 Subtraktionen von Polynomen vom Grad $< 2n$: $4n\,\mathsf{A}$
- $2n$ Additionen von Koeffizienten beim „Zusammensetzen“: $2n\,\mathsf{A}$

Insgesamt also $3n^2$ Multiplikationen und $3n^2 + 8n$ Additionen oder Subtraktionen. Die klassische Methode braucht $4n^2$ Multiplikationen und $4n^2$ Additionen/Subtraktionen.

Wir können das Ganze natürlich auch rekursiv machen; dann kommt der Geschwindigkeitsvorteil erst richtig zur Geltung:

**ALGO**
Karatsuba I

**function** karatsuba($a$, $b$, $k$)

**input:** $a, b \in R[X]$, $\deg a, \deg b < 2^k$.

**output:** $a \cdot b$.

```
  if k = 0 then
    return a · b // konstante Polynome
  else
    a_1 X^{2^{k-1}} + a_0 := a
    b_1 X^{2^{k-1}} + b_0 := b
    c_2 := karatsuba(a_1, b_1, k − 1)
    c_0 := karatsuba(a_0, b_0, k − 1)
    c_1 := karatsuba(a_1 + a_0, b_1 + b_0, k − 1) − c_2 − c_0
    // c_0, c_1, c_2 sind Polynome vom Grad < 2^k.
    return c_2 X^{2^k} + c_1 X^{2^{k-1}} + c_0
  end if
end function
```

Sei $T(k)$ der Aufwand für einen Aufruf karatsuba($a$, $b$, $k$). Dann gilt

$$T(0) = \mathsf{M} \quad \text{und} \quad T(k) = 3T(k-1) + 4 \cdot 2^k\,\mathsf{A} \text{ für } k \geq 1.$$

Das folgende Lemma sagt uns, wie wir diese Rekursion auflösen können.

**LEMMA**
Rekursion

15.1. **Lemma.** *Sei $(a_n)_{n\geq 0}$ rekursiv definiert durch*

$$a_0 = \alpha\,, \qquad a_n = \lambda a_{n-1} + \beta\mu^n \quad \textit{für } n \geq 1.$$

*Dann gilt*

$$a_n = \begin{cases} \alpha\lambda^n + \frac{\beta\mu}{\mu-\lambda}(\mu^n - \lambda^n) & \textit{falls } \lambda \neq \mu, \\ (\alpha + \beta n)\lambda^n & \textit{falls } \lambda = \mu. \end{cases}$$

*Beweis.* Es ist klar, dass es genau eine Lösung $(a_n)$ gibt. Man prüft in beiden Fällen nach, dass die angegebene Folge eine Lösung ist. ❑

Wie kommt man auf diese Lösung der Rekursion?

**1. Möglichkeit:** Theorie linearer Gleichungen. Wir betrachten erst einmal die homogene Version $a_n = \lambda a_{n-1}$. Ihre Lösungen sind genau die Folgen der Form $a_n = \alpha'\lambda^n$ mit einer beliebigen Konstante $\alpha'$. Wir brauchen dann noch eine spezielle Lösung der inhomogenen Gleichung. Dazu machen wir den Ansatz $a_n = \gamma\mu^n$ und setzen in die Rekursion ein; das liefert $\gamma\mu^n = \lambda\gamma\mu^{n-1} + \beta\mu^n$. Im Fall $\lambda \neq \mu$ kann man das nach $\gamma$ auflösen und erhält $\gamma = \frac{\beta\mu}{\mu-\lambda}$. Die allgemeine Lösung der Rekursion ist damit

$$a_n = \alpha'\lambda^n + \frac{\beta\mu}{\mu-\lambda}\mu^n\,;$$

man findet $\alpha' = \alpha - \frac{\beta\mu}{\mu-\lambda}$ aus der Anfangsbedingung $a_0 = \alpha$. Den Fall $\lambda = \mu$ erhält man als Grenzübergang aus

$$\lim_{\mu\to\lambda} \mu\frac{\mu^n - \lambda^n}{\mu-\lambda} = n\lambda^n\,.$$

**2. Möglichkeit:** Methode der „erzeugenden Funktionen“. Man verpackt die Folge in eine Potenzreihe und gewinnt aus Anfangsbedingung und Rekursion eine Gleichung für die Potenzreihe:

$$A(z) = \sum_{n=0}^{\infty} a_n z^n = \alpha + \sum_{n=1}^{\infty}(\lambda a_{n-1} + \beta\mu^n)z^n = \alpha + \lambda z A(z) + \beta\frac{\mu z}{1-\mu z}\,.$$

(Der letzte Term ist eine geometrische Reihe.) Diese Gleichung kann man auflösen und bekommt

$$A(z) = \frac{1}{1-\lambda z}\Big(\alpha + \beta\frac{\mu z}{1-\mu z}\Big)\,.$$

Im Fall $\lambda \neq \mu$ führt Partialbruchzerlegung auf

$$\frac{z}{(1-\lambda z)(1-\mu z)} = \frac{1}{\mu-\lambda}\Big(\frac{1}{1-\mu z} - \frac{1}{1-\lambda z}\Big)\,;$$

man erhält die explizite Lösung aus der Formel für die geometrische Reihe. Im Fall $\lambda = \mu$ hat man

$$A(z) = \frac{\alpha}{1-\lambda z} + \beta\frac{\lambda z}{(1-\lambda z)^2}\,.$$

Mit

$$\sum_{n=0}^{\infty} n z^n = z\,\frac{d}{dz}\,\frac{1}{1-z} = \frac{z}{(1-z)^2}$$

bekommt man in diesem Fall die angegebene Lösung.

Aus Lemma 15.1 erhalten wir:

15.2. **Folgerung.** *Die Kosten für karatsuba(a, b, k) betragen $3^k$ Multiplikationen und $8(3^k - 2^k)$ Additionen oder Subtraktionen in R.*

**FOLG** Kosten für Karatsuba

*Im Grad $n = 2^k$ ausgedrückt bedeutet das: Wir können das Produkt zweier Polynome vom Grad $< n = 2^k$ mit einem Aufwand von $n^{\log_2 3}$ Multiplikationen und $< 8n^{\log_2 3}$ Additionen/Subtraktionen in R berechnen.*

Man beachte, dass $\log_2 3 \approx 1{,}585$ und damit deutlich kleiner als 2 ist.

In der Praxis ist es meistens so, dass die klassische Methode für Polynome von kleinem Grad schneller ist. Daher wird man statt „**if** $k = 0 \ldots$“ eine etwas höhere Schranke wählen (und dann das Resultat klassisch berechnen). Die optimale Wahl muss durch Ausprobieren ermittelt werden.

Meistens ist der Grad nicht von der Form $2^k - 1$. Man geht dann etwa so vor ($d$ ist die Gradschranke für den klassischen Algorithmus):

**ALGO** Karatsuba II

---

**function** karatsuba′($a$, $b$)
**input:** $a, b \in R[X]$.
**output:** $a \cdot b$.

```
n := deg a + 1; m := deg b + 1
if n < m then (a, b, n, m) := (b, a, m, n) end if
// Jetzt ist n ≥ m.
if m < d then
  return multiply(a, b) // Klassische Methode
end if
k := ⌈n/2⌉
a_1 X^k + a_0 := a
b_1 X^k + b_0 := b
if b_1 = 0 then
  c_1 := karatsuba′(a_1, b_0)
  c_0 := karatsuba′(a_0, b_0)
  return c_1 X^k + c_0
else
  c_2 := karatsuba′(a_1, b_1)
  c_0 := karatsuba′(a_0, b_0)
  c_1 := karatsuba′(a_1 + a_0, b_1 + b_0) − c_2 − c_0
  return c_2 X^{2k} + c_1 X^k + c_0
end if
end function
```

---

Die Alternative wäre, karatsuba($a$, $b$, $k$) zu verwenden mit dem kleinsten $k$, das $2^k > \deg a, \deg b$ erfüllt. Dabei verschenkt man aber zu viel (im schlimmsten Fall einen Faktor $\approx 3$).

### Karatsuba für ganze Zahlen.

Für ganze Zahlen (statt Polynome) funktioniert der Algorithmus im wesentlichen genauso: Seien $\lambda(a), \lambda(b) \leq 2N$ und schreibe

$$a = a_1 B^N + a_0 \quad \text{und} \quad b = b_1 B^N + b_0$$

(mit $B = 2^{64}$ wie üblich). Dann ist

$$a \cdot b = (a_1 b_1) B^{2N} + \big((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0\big) B^N + (a_0 b_0)\,.$$

Das einzige Problem ist (wie meistens), dass bei der Berechnung des Teilprodukts $(a_1 + a_0)(b_1 + b_0)$ die Faktoren $\geq B^N$ sein können, was für die Rekursion nicht so schön ist. Es gilt aber stets $a_1 + a_0, b_1 + b_0 < 2B^N$; es gibt also höchstens 1 Bit „Überlauf“, das man am besten separat verarztet.

An der Komplexität ändert sich nichts: Mit dem Karatsuba-Algorithmus kann man zwei Zahlen $a, b \in \mathbb{Z}$ mit $\lambda(a), \lambda(b) \leq n$ mit $O(n^{\log_2 3})$ Wortoperationen multiplizieren.

## 16. Diskrete Fourier-Transformation und FFT

Die *Fourier-Transformation* in der Analysis ist die Abbildung

$$f \longmapsto \left(\xi \mapsto \int_{-\infty}^{\infty} e^{-ix\xi} f(x)\, dx\right)$$

(für geeignete Klassen von Funktionen $f\colon \mathbb{R} \to \mathbb{C}$). Die *Diskrete Fourier-Transformation* ist eine diskrete Version davon. Sie hat die gleichen schönen Eigenschaften, aber den Vorteil, dass man sich keine Gedanken über Konvergenz machen muss.

Im Folgenden wird $0 \in R$ (evtl. abweichend von der Definition aus der „Einführung in die Zahlentheorie und algebraische Strukturen“) als Nullteiler betrachtet. Ist in diesem Sinne $a$ kein Nullteiler und $ab = 0$, dann folgt $b = 0$.

**DEF** primitive $n$te Einheitswurzel

16.1. **Definition.** Seien $R$ ein Ring und $n \in \mathbb{Z}_{\geq 1}$. Ein Element $\omega \in R$ heißt *$n$te Einheitswurzel*, wenn $\omega^n = 1$ ist. $\omega$ ist eine *primitive $n$te Einheitswurzel*, wenn zusätzlich $n \cdot 1_R$ in $R$ invertierbar ist, und wenn für keinen Primteiler $p$ von $n$ das Element $\omega^{n/p} - 1$ ein Nullteiler in $R$ ist. (Insbesondere muss $\omega^{n/p} \neq 1$ gelten.) ♢

**BSP** Einheitswurzeln

16.2. **Beispiele.** Ein endlicher Körper $\mathbb{F}_q$ enthält primitive $n$te Einheitswurzeln genau für $n \mid q - 1$.

Der Ring $\mathbb{Z}$ enthält keine primitiven $n$ten Einheitswurzeln für $n \geq 2$ (denn kein solches $n$ ist in $\mathbb{Z}$ invertierbar).

Der Körper $\mathbb{Q}$ enthält dem gegenüber auch die primitive zweite Einheitswurzel $-1$, aber keine primitiven $n$ten Einheitswurzeln für $n \geq 3$. ♣

Wir beweisen die wichtigste Eigenschaft von primitiven $n$ten Einheitswurzeln:

**LEMMA** primitive $n$te Einheitswurzeln

16.3. **Lemma.** *Sei $R$ ein Ring, seien $1 \leq \ell < n$, und sei $\omega \in R$ eine primitive $n$te Einheitswurzel. Dann gilt*

(1) *$\omega^\ell - 1$ ist kein Nullteiler in $R$;*

(2) $\displaystyle\sum_{j=0}^{n-1} \omega^{j\ell} = 0.$

*Es folgt, dass für jedes zu $n$ teilerfremde $\ell \in \mathbb{Z}$ das Element $\omega^\ell$ ebenfalls eine primitive $n$te Einheitswurzel ist. Die Aussage (2) gilt für jedes $\ell \in \mathbb{Z}$ mit $n \nmid \ell$.*

*Beweis.*

(1) Sei $g = \mathrm{ggT}(\ell, n) < n$. Dann gibt es einen Primteiler $p$ von $n$ mit $g \mid \frac{n}{p}$. Da $\omega^g - 1$ ein Teiler von $\omega^{n/p} - 1$ ist, ist $\omega^g - 1$ kein Nullteiler.
Jetzt schreiben wir $g = u\ell + vn$ mit $u, v \in \mathbb{Z}$. Dann ist
$$\omega^g - 1 = \omega^{u\ell}\omega^{vn} - 1 = \omega^{u\ell} - 1\,,$$
also ist $\omega^\ell - 1$ ein Teiler von $\omega^g - 1$ und damit ebenfalls kein Nullteiler.

(2) Es ist
$$(\omega^\ell - 1)\sum_{j=0}^{n-1} \omega^{j\ell} = \omega^{n\ell} - 1 = 0\,.$$
Da $\omega^\ell - 1$ kein Nullteiler ist, muss die Summe verschwinden.

Sei nun $\ell \in \mathbb{Z}$ beliebig. Wegen $\omega^n = 1$ können wir $\ell$ durch $r$ ersetzen, wobei $\ell \equiv r \bmod n$ und $0 \le r < n$ gilt. Sind $\ell$ und $n$ (und damit auch $r$ und $n$) teilerfremd, dann gibt es $k \in \mathbb{Z}$ mit $k\ell \equiv 1 \bmod n$, sodass $(\omega^\ell)^k = \omega$ ist. Es folgt, dass $(\omega^\ell)^{n/p} - 1$ ein Teiler von $\omega^{n/p} - 1$ ist; damit ist $(\omega^\ell)^{n/p} - 1$ kein Nullteiler. Da natürlich auch $(\omega^\ell)^n = 1$ ist (und $n \cdot 1_R \in R^\times$), ist $\omega^\ell$ eine primitive $n$te Einheitswurzel.

Die letzte Aussage folgt, da $p \nmid \ell$ zu $1 \le r < n$ äquivalent ist. ❑

Seien im Folgenden $R$ ein Ring, $n \ge 1$ und $\omega \in R$ eine primitive $n$te Einheitswurzel. Wir identifizieren ein Polynom

$$a = a_{n-1}X^{n-1} + \cdots + a_1X + a_0 \in R[X]$$

mit seinem Koeffizientenvektor $(a_0, a_1, \ldots, a_{n-1}) \in R^n$.

16.4. **Definition.** Die $R$-lineare Abbildung **DEF** DFT

$$\mathrm{DFT}_\omega : R^n \longrightarrow R^n\,, \qquad a \longmapsto \big(a(1), a(\omega), a(\omega^2), \ldots, a(\omega^{n-1})\big)$$

heißt *Diskrete Fourier-Transformation (DFT)* (bezüglich $\omega$). ♢

16.5. **Definition.** Für zwei Polynome $a, b \in R^n$ definieren wir die *Faltung* als das Polynom **DEF** Faltung

$$c = a * b = a *_n b = \sum_{j=0}^{n-1} c_j X^j \quad \text{mit} \quad c_j = \sum_{\substack{k,\ell \in \{0,\ldots,n-1\} \\ k+\ell \equiv j \bmod n}} a_k b_\ell\,. \qquad ♢$$

Wenn wir $R^n$ mit dem Restklassenring $R[X]/\langle X^n - 1\rangle$ identifizieren, dann ist die Faltung genau die Multiplikation in diesem Restklassenring. Es gilt nämlich (modulo $X^n - 1$)

$$\begin{aligned} a \cdot b &= \sum_{j=0}^{2n-2} \Big( \sum_{k+\ell=j} a_k b_\ell \Big) X^j \\ &= \sum_{j=0}^{n-1} \Big( \sum_{k+\ell=j} a_k b_\ell \Big) X^j + \sum_{j=n}^{2n-2} \Big( \sum_{k+\ell=j} a_k b_\ell \Big) X^j \\ &\equiv \sum_{j=0}^{n-1} \Big( \sum_{k+\ell=j} a_k b_\ell \Big) X^j + \sum_{j=0}^{n-2} \Big( \sum_{k+\ell=j+n} a_k b_\ell \Big) X^j \\ &= \sum_{j=0}^{n-1} \Big( \sum_{k+\ell \equiv j \bmod n} a_k b_\ell \Big) X^j\,. \end{aligned}$$

Der Zusammenhang zwischen den beiden Definitionen wird aus dem folgenden Lemma deutlich. Wir schreiben

$$(a_0, a_1, \ldots, a_{n-1}) \bullet (b_0, b_1, \ldots, b_{n-1}) = (a_0b_0, a_1b_1, \ldots, a_{n-1}b_{n-1})$$

für die komponentenweise Multiplikation zweier Vektoren in $R^n$.

**LEMMA** Faltung und DFT

16.6. **Lemma.** *Seien $a, b \in R^n$ und $\omega \in R$ eine primitive nte Einheitswurzel. Dann gilt*

$$\mathrm{DFT}_\omega(a * b) = \mathrm{DFT}_\omega(a) \bullet \mathrm{DFT}_\omega(b)\,.$$

*Beweis.* Gilt $f \equiv g \bmod (X^n - 1)$, dann ist $f(\omega^j) = g(\omega^j)$, denn $(X^n - 1)(\omega^j) = \omega^{nj} - 1 = 0$. Damit gilt für die $j$te Komponente des Vektors $\mathrm{DFT}_\omega(a * b)$:

$$\big(\mathrm{DFT}_\omega(a*b)\big)_j = (a*b)(\omega^j) = (a\cdot b)(\omega^j) = a(\omega^j)\cdot b(\omega^j) = \big(\mathrm{DFT}_\omega(a)\big)_j \cdot \big(\mathrm{DFT}_\omega(b)\big)_j\,.$$

❑

Die lineare Abbildung $\mathrm{DFT}_\omega$ ist sogar invertierbar.

**SATZ** Invertierbarkeit der DFT

16.7. **Satz.** *Seien $R$ ein Ring, $n \geq 1$, $\omega \in R$ eine primitive nte Einheitswurzel. Dann ist $\omega^{-1} = \omega^{n-1}$ ebenfalls eine primitive nte Einheitswurzel, und es gilt für $a \in R^n$:*

$$\mathrm{DFT}_\omega\big(\mathrm{DFT}_{\omega^{-1}}(a)\big) = na\,.$$

*Insbesondere ist $\mathrm{DFT}_\omega$ invertierbar, und es gilt $\mathrm{DFT}_\omega^{-1} = n^{-1}\,\mathrm{DFT}_{\omega^{-1}}$.*

*Beweis.* Es genügt, die Behauptung auf der Standardbasis nachzuprüfen. Es ist dann zu zeigen, dass $\sum_{j=0}^{n-1} \omega^{kj}\omega^{-mj} = n\delta_{k,m}$ ist. Nun gilt aber mit Lemma 16.3

$$\sum_{j=0}^{n-1} \omega^{kj}\omega^{-mj} = \sum_{j=0}^{n-1} \omega^{(k-m)j} = 0 \quad \text{für } k \neq m$$

(denn dann ist $k - m \not\equiv 0 \bmod n$) und

$$\sum_{j=0}^{n-1} \omega^{kj}\omega^{-kj} = \sum_{j=0}^{n-1} 1 = n\,.$$

Dass $\omega^{-1}$ eine primitive $n$te Einheitswurzel ist, wurde in Lemma 16.3 gezeigt. ❑

Die Aussagen von Lemma 16.6 und Satz 16.7 kann man zusammenfassen, indem man sagt, dass $\mathrm{DFT}_\omega$ ein Ringisomorphismus zwischen dem Restklassenring $R[X]/\langle X^n - 1\rangle$ und dem Produktring $R^n$ (mit komponentenweisen Operationen) ist.

Um zwei Polynome $a$ und $b$ mit $\deg(ab) < n$ zu multiplizieren, können wir also so vorgehen: Wir wählen $N \geq n$ und eine primitive $N$te Einheitswurzel $\omega$. Dann ist

$$a \cdot b = N^{-1}\,\mathrm{DFT}_{\omega^{-1}}\big(\mathrm{DFT}_\omega(a) \bullet \mathrm{DFT}_\omega(b)\big)\,,$$

wobei wir die üblichen Identifikationen von $R^N$ mit den Polynomen vom Grad $< N$ und mit dem Restklassenring $R[X]/\langle X^N - 1\rangle$ vorgenommen haben. Der Aufwand besteht in $N$ Multiplikationen in $R$ und den drei DFT-Berechnungen (plus Multiplikation mit $N^{-1}$). Die Komplexität wird dabei von den DFT-Berechnungen dominiert (alles andere ist linear in $N$).

**FFT — Fast Fourier Transform.**

Wir lernen jetzt eine Möglichkeit kennen, $\mathrm{DFT}_\omega$ sehr schnell zu berechnen, wenn $n = 2^k$ eine Potenz von 2 ist. Die Grundidee ist wiederum „Divide And Conquer“.

Sei zunächst nur vorausgesetzt, dass $n = 2m$ gerade ist, und sei $a \in R[X]$ ein Polynom vom Grad $< n$. Wir schreiben

$$a = a_1 X^m + a_0$$

mit Polynomen $a_0, a_1$ vom Grad $< m$. Dann gilt (unter Beachtung von $\omega^m = -1$: Es ist $(\omega^m + 1)(\omega^m - 1) = \omega^n - 1 = 0$, und $\omega^m - 1$ ist kein Nullteiler)

$$\begin{aligned} a(\omega^{2j}) &= a_1(\omega^{2j})(\omega^m)^{2j} + a_0(\omega^{2j}) = (a_0 + a_1)(\omega^{2j}) \qquad \text{und} \\ a(\omega^{2j+1}) &= a_1(\omega^{2j+1})(\omega^m)^{2j+1} + a_0(\omega^{2j+1}) = (a_0 - a_1)(\omega \cdot \omega^{2j})\,. \end{aligned}$$

Wir setzen $r_0(X) = a_0(X) + a_1(X)$ und $r_1(X) = (a_0 - a_1)(\omega X)$. $\omega^2$ ist eine primitive $m$te Einheitswurzel. Damit ist die Berechnung von $\mathrm{DFT}_\omega(a)$ äquivalent zur Berechnung von $\mathrm{DFT}_{\omega^2}(r_0)$ und von $\mathrm{DFT}_{\omega^2}(r_1)$. Die Berechnung von $r_0$ und $r_1$ aus $a$ erfordert dabei $n = 2m$ Additionen bzw. Subtraktionen und $m$ Multiplikationen mit Potenzen von $\omega$.

Wir erhalten folgenden Algorithmus. Wir nehmen dabei an, dass die Potenzen $\omega^j$ vorberechnet sind (das kostet einmalig $2^k$ Multiplikationen in $R$).

**ALGO**
FFT

---

**function** $\mathrm{fft}(a, k, \omega)$
**input:** $a = (a_0, \ldots, a_{2^k-1}) \in R^{2^k}$, $\omega \in R$ primitive $2^k$te Einheitswurzel.
**output:** $\mathrm{DFT}_\omega(a) \in R^{2^k}$.

```
  if k = 0 then return (a_0) end if
  m := 2^{k-1}
  for j = 0 to m - 1 do
    b_j := a_j + a_{m+j}
    c_j := (a_j - a_{m+j}) · ω^j
  end for
  (b_0, ..., b_{m-1}) := fft((b_0, ..., b_{m-1}), k - 1, ω^2)
  (c_0, ..., c_{m-1}) := fft((c_0, ..., c_{m-1}), k - 1, ω^2)
  return (b_0, c_0, b_1, c_1, ..., b_{m-1}, c_{m-1})
end function
```

---

Sei $T(k)$ der Aufwand für einen Aufruf $\mathrm{fft}(a, k, \omega)$, gemessen in Operationen in $R$. Dann gilt

$$T(k) = 2T(k-1) + 2^k\,\mathsf{A} + 2^{k-1}\,\mathsf{M} \quad \text{für } k \geq 1.$$

Dabei steht $\mathsf{A}$ wieder für Additionen und Subtraktionen und $\mathsf{M}$ für Multiplikationen in $R$ (hier sind das nur Multiplikationen mit Potenzen von $\omega$). Lemma 15.1 sagt uns dann, dass gilt

$$T(k) \in \big(\mathsf{A} + \tfrac{1}{2}\,\mathsf{M}\big) k 2^k + O(2^k) \subset O(k2^k)\,.$$

Mit $n = 2^k$ gilt $O(k2^k) = O(n \log n)$.

16.8. **Folgerung.** *Seien $R$ ein Ring und $\omega \in R$ eine primitive $n$te Einheitswurzel mit $n = 2^k$. Dann kann man die Faltung zweier Polynome in $R[X]$ vom Grad $< n$ mit $O(n \log n)$ Operationen in $R$ berechnen.* **FOLG** schnelle Faltung

*Beweis.* Wir verwenden folgenden Algorithmus:

**ALGO** Faltung

---

**function** convolution($a$, $b$, $k$, $\omega$)
**input:** $a, b \in R[X]$, $\deg a, \deg b < 2^k$.
**output:** $a *_{2^k} b \in R[X]$.

```
// Wir identifizieren Polynome und ihre Koeffizientenvektoren.
â := fft(a, k, ω)
b̂ := fft(b, k, ω)
n := 2^k
for j = 0 to n − 1 do
   ĉ_j := â_j · b̂_j
end for
c := fft(ĉ, k, ω^{-1})
d := n^{-1} // (in R)
return d · c
```
**end function**

---

Der Aufwand dafür besteht in drei FFTs ($O(n \log n)$), der punktweisen Multiplikation von $\hat{a}$ und $\hat{b}$ ($O(n)$) und schließlich der Multiplikation des Ergebnisses mit $d$ ($O(n)$). Die Komplexität der FFTs dominiert den Gesamtaufwand, der damit ebenfalls in $O(n \log n)$ ist. ❑

16.9. **Satz.** *Sei $R$ ein Ring, in dem es primitive $2^k$te Einheitswurzeln gibt für jedes $k$. Dann kann man zwei Polynome $a, b \in R[X]$ mit $\deg ab < n$ mit einem Aufwand von $O(n \log n)$ Operationen in $R$ multiplizieren.* **SATZ** schnelle Multiplikation

*Beweis.* Wähle $k$ mit $2^k \geq n$. Dann ist $a *_{2^k} b = a \cdot b$, und wir rufen einfach convolution($a, b, k$) auf. Man beachte, dass (mit dem minimalen $k$) $2^k < 2n \in O(n)$ gilt. ❑

### „Three Primes"-FFT für die Multiplikation ganzer Zahlen.

Wir können Satz 16.9 benutzen, um einen schnellen Multiplikationsalgorithmus für ganze Zahlen zu konstruieren. Dieser wird zwar nur für ganze Zahlen beschränkter Länge funktionieren, aber die Beschränkung liegt weit jenseits dessen, was mit dem Computer überhaupt verarbeitbar ist.

Seien $a, b \in \mathbb{Z}_{>0}$ mit $\lambda(a), \lambda(b) \leq n$, und $B = 2^{64}$. Wir schreiben

$$a = \tilde{a}(B) \quad \text{und} \quad b = \tilde{b}(B)$$

mit Polynomen

$$\tilde{a} = a_{n-1}X^{n-1} + \cdots + a_1 X + a_0 \quad \text{und} \quad \tilde{b} = b_{n-1}X^{n-1} + \cdots + b_1 X + b_0\,,$$

wobei die Koeffizienten $a_j, b_j$ Wortlänge haben, also in $[0, B-1]$ liegen. Für die Koeffizienten $c_j$ des Produktes $\tilde{a} \cdot \tilde{b}$ gilt dann $c_j \leq n(B-1)^2$. Wir können drei

Primzahlen $p, q, r$ wählen mit $p, q, r \equiv 1 \bmod 2^{57}$ und $2^{63} < p, q, r < 2^{64}$, zum Beispiel $p = 95 \cdot 2^{57} + 1$, $q = 108 \cdot 2^{57} + 1$ und $r = 123 \cdot 2^{57} + 1$. Wir setzen voraus, dass $n(B-1)^2 < pqr$ ist (das gilt für $n \leq 2^{63}$; für eine solche Zahl wäre der Speicherplatzbedarf $2^{63} \cdot 8 = 2^{66}$ Byte — ein Gigabyte sind nur $2^{30}$ Byte!). Dann lässt sich $c_j$ nach dem Chinesischen Restsatz aus den Resten von $c_j$ modulo $p$, $q$ und $r$ bestimmen. In den endlichen Körpern $\mathbb{F}_p$, $\mathbb{F}_q$, $\mathbb{F}_r$ gibt es jeweils primitive $2^{57}$te Einheitswurzeln, etwa $\omega_p = 55$, $\omega_q = 64$, $\omega_r = 493$. Wir können also mittels FFT Polynome vom Grad $< 2^{56}$ über diesen Körpern multiplizieren. Dies reduziert die Schranke für $n$ auf $2^{56}$ (was immer noch weit ausreicht).

Das führt auf folgenden Algorithmus. Wir schreiben $x$ rem $m$ für die Zahl $a \in \mathbb{Z}$ mit $x \equiv a \bmod m$ und $0 \leq a < m$. Als Vorberechnung bestimmen wir (mit dem Erweiterten Euklidischen Algorithmus) Zahlen $u, v, w \in \mathbb{Z}$ mit $0 \leq u, v, w < pqr$ und $qr \mid u$, $u \equiv 1 \bmod p$, $pr \mid v$, $v \equiv 1 \bmod q$, $pq \mid w$, $w \equiv 1 \bmod r$. Dann gilt für $x = (au + bv + cw)$ rem $pqr$:

$$0 \leq x < pqr\,, \quad \text{sowie} \quad x \equiv a \bmod p\,, \quad x \equiv b \bmod q \quad \text{und} \quad x \equiv c \bmod r\,.$$

**ALGO**
3-Primes FFT

---

**function** fast_multiply($a$, $b$)
**input:** $a, b \in \mathbb{Z}_{>0}$ mit $\lambda(a), \lambda(b) < 2^{56}$.
**output:** $a \cdot b$.

$n := \max\{\lambda(a), \lambda(b)\}$
$\tilde{a} := a_{n-1}X^{n-1} + \cdots + a_0$ wie oben
$\tilde{b} := b_{n-1}X^{n-1} + \cdots + b_0$ wie oben
$\tilde{a}_p := \tilde{a}$ rem $p$; $\tilde{a}_q := \tilde{a}$ rem $q$; $\tilde{a}_r := \tilde{a}$ rem $r$
$\tilde{b}_p := \tilde{b}$ rem $p$; $\tilde{b}_q := \tilde{b}$ rem $q$; $\tilde{b}_r := \tilde{b}$ rem $r$
$k := \lceil \log_2 n \rceil + 1$
$\tilde{c}_p := \text{convolution}(\tilde{a}_p, \tilde{b}_p, k, \omega_p^{2^{57-k}})$
$\tilde{c}_q := \text{convolution}(\tilde{a}_q, \tilde{b}_q, k, \omega_q^{2^{57-k}})$
$\tilde{c}_r := \text{convolution}(\tilde{a}_r, \tilde{b}_r, k, \omega_r^{2^{57-k}})$
$\tilde{c} := u \cdot \tilde{c}_p + v \cdot \tilde{c}_q + w \cdot \tilde{c}_r$
$\tilde{c} := \tilde{c}$ rem $pqr$
**return** $\tilde{c}(2^{64})$
**end function**

---

Was kostet das? Die Reduktion von $\tilde{a}$ und $\tilde{b}$ modulo $p, q, r$ kostet $O(n)$ Wortoperationen (pro Koeffizient ein Vergleich und eventuell eine Subtraktion, denn die Koeffizienten sind $< 2p$). Die Multiplikation als Faltung mittels FFT kostet jeweils $O(k2^k) = O(n \log n)$ Wortoperationen (Operationen in $\mathbb{F}_p$ usw. sind in wenigen Wortoperationen zu erledigen, da $p, q, r < B$ sind). Die Rekonstruktion von $\tilde{c}$ kostet wieder einen Aufwand von $O(n)$ Wortoperationen, und das gleiche gilt für die Auswertung am Ende. Die Komplexität ist also

$$O(n \log n) \quad \text{Wortoperationen}\,.$$

Da wir $n$ beschränkt haben, hat diese Aussage eigentlich keinen Sinn. In der Praxis ist dieser Algorithmus aber tatsächlich schneller als Karatsuba für Zahlen einer Größe, die gelegentlich vorkommen kann.

**Schnelle Multiplikation in $R[X]$ für beliebige Ringe $R$.**

Ist $R$ ein beliebiger Ring (also zum Beispiel einer, der keine primitiven $2^k$ten Einheitswurzeln enthält), kann man wie folgt vorgehen. Wir nehmen erst einmal an, dass 2 in $R$ invertierbar ist. Dann kann man sich eine künstliche primitive $2^k$te Einheitswurzel schaffen, indem man in $R[y]$ modulo $y^{2^{k-1}}+1$ rechnet. Wenn man das geschickt genug macht, verliert man nicht allzu viel, und man erhält einen Algorithmus, der Polynome vom Grad $< n$ in $O(n \log n \log\log n)$ Operationen in $R$ multipliziert.

Wenn 2 nicht invertierbar ist, kann man immer noch, indem man die Multiplikation mit $n^{-1}$ am Ende des Algorithmus im Beweis von Folgerung 16.8 weglässt, ein Vielfaches $2^\kappa ab$ des Produktes von $a$ und $b$ berechnen. Analog gibt es eine „3-adische FFT“, mit deren Hilfe man $3^\lambda ab$ mit vergleichbarem Aufwand berechnen kann. Da $2^\kappa$ und $3^\lambda$ teilerfremd sind, kann man (schnell) $u, v \in \mathbb{Z}$ berechnen mit $u \cdot 2^\kappa + v \cdot 3^\lambda = 1$. Dann ist $ab = u \cdot 2^\kappa ab + v \cdot 3^\lambda ab$. Es folgt:

**SATZ** schnelle Mult. in $R[X]$ für allg. $R$

16.10. **Satz.** *Sei $R$ ein beliebiger (kommutativer) Ring (mit 1). Es gibt einen Algorithmus, der das Produkt zweier Polynome in $R[X]$ vom Grad $< n$ mit*

$$O(n \log n \log\log n) \quad \textit{Operationen in } R$$

*berechnet.*

Wer genau wissen möchte, wie das geht, kann es in [GG, § 8] nachlesen.

Eine Variante (eigentlich die ursprüngliche Version) des oben angedeuteten Ansatzes führt auf das folgende berühmte Ergebnis von Schönhage und Strassen (1971):

**SATZ** schnelle Mult. in $\mathbb{Z}$

16.11. **Satz.** *Es gibt einen Algorithmus, der das Produkt zweier ganzer Zahlen $a$ und $b$ mit $\lambda(a), \lambda(b) \leq n$ in*

$$O(n \log n \log\log n) \quad \textit{Wortoperationen}$$

*berechnet.*

Dieses Resultat ist hauptsächlich von theoretischem Interesse, denn der Algorithmus ist so komplex, dass er für Zahlen, wie sie in der Praxis vorkommen (oder überhaupt verarbeitbar sind), langsamer ist als Karatsuba oder die Three-Primes FFT. Letztere veranschaulicht aber recht gut, dass die wesentliche Komplexitätsaussage auch in der Praxis relevant ist. Der Faktor $\log\log n$ wächst so langsam („The function $\log\log x$ tends to infinity, but has never been observed to do so“), dass er für praktische Zwecke als konstant anzusehen ist, wie die Three-Primes-Variante recht schön zeigt, die eine Komplexität von $n \log n$ über einen sehr großen Bereich von Werten von $n$ hat. (Für $n$ zwischen $2^{32}$ und $2^{64}$ liegt $\log_2 \log_2 n$ zwischen 5 und 6, verändert sich also kaum.) Inzwischen gibt es theoretische Verbesserungen, die den Faktor $\log\log n$ in der Asymptotik noch verringern. Genaueres kann man im Wikipedia-Eintrag für den Schönhage-Strassen-Algorithmus nachlesen.

In jedem Fall beantwortet der Satz von Schönhage und Strassen die Frage

*Was ist das Infimum der Exponenten $\varepsilon > 0$, sodass es einen Multiplikationsalgorithmus der Komplexität $O(n^\varepsilon)$ Wortoperationen für ganze Zahlen der Länge $n$ gibt?*

mit „1“. Demgegenüber ist die Antwort auf die analoge Frage

*Was ist das Infimum der Exponenten $\varepsilon > 0$, sodass es einen Multiplikationsalgorithmus der Komplexität $O(n^\varepsilon)$ Operationen in $R$ für $n \times n$-Matrizen über $R$ gibt?*

noch unbekannt. Die beste bekannte obere Schranke (von 2012) ist $\approx 2{,}3727$, die beste untere Schranke ist die triviale, nämlich 2 (die $n^2$ Einträge des Ergebnisses müssen bestimmt werden).

Um die Notation zu vereinfachen und uns auf das Wesentliche zu konzentrieren, führen wir folgende „Soft-O"-Schreibweise ein.

**DEF**
Soft-O
quasi-linear

16.12. **Definition.** Seien $f, g\colon \mathbb{Z}_{>0} \to \mathbb{R}_{>0}$ zwei Funktionen. Wir schreiben

$$f(n) \in \tilde{O}\big(g(n)\big)\,,$$

wenn $g(n) \to \infty$ für $n \to \infty$ und es Konstanten $C > 0$ und $k$ gibt, so dass $f(n) \le Cg(n)(\log g(n))^k$ gilt für alle hinreichend großen $n$.

Im Fall $g(n) = n$, also $f(n) \in \tilde{O}(n)$, sagen wir auch, $f$ wachse *quasi-linear*. ♢

Aus dem Satz von Schönhage-Strassen folgt dann etwa, dass man zwei ganze Zahlen $a$ und $b$ mit $\lambda(a), \lambda(b) \le n$ mit einem Aufwand von $\tilde{O}(n)$ Wortoperationen (also quasi-linearem Aufwand) multiplizieren kann.

Man kann die Newton-Iteration (siehe den Numerik-Teil dieser Vorlesung) zusammen mit der schnellen Multiplikation dazu verwenden, eine schnelle Division mit Rest zu implementieren, deren Komplexität mit der der schnellen Multiplikation vergleichbar ist. Auf diese Weise (und mit einer schnellen Variante des Erweiterten Euklidischen Algorithmus) kann man in Restklassenringen der Form $\mathbb{Z}/n\mathbb{Z}$ und insbesondere in den endlichen Körpern $\mathbb{F}_p$ die vier Grundrechenarten mit quasi-linearem (in $\lambda(n)$) Aufwand an Wortoperationen durchführen.

## 17. Modulare Arithmetik: Berechnung der Determinante

Wir haben bereits bei der „Three-Primes-FFT“ gesehen, dass es vorteilhaft sein kann, modulo einer oder mehrerer Primzahlen zu rechnen. Wir wollen diese Beobachtung jetzt anhand des Beispiels der Berechnung der Determinante einer Matrix mit ganzzahligen Einträgen weiter vertiefen.

Wir erinnern uns zunächst an das Standard-Verfahren zur Berechnung einer Determinante für Matrizen mit Einträgen aus einem Körper $K$. Es beruht darauf, dass man das Verhalten der Determinante unter Zeilen- (oder Spalten-)umformungen kennt, und dass die Determinante einer Dreiecksmatrix durch das Produkt ihrer Diagonaleinträge gegeben ist.

**ALGO**
Determinante klassisch

---

**function** $\det(A)$
**input:** $A = (a_{ij}) \in \mathrm{Mat}(n, K)$
**output:** $\det(A) \in K$.

```
  if n = 0 then return 1 end if // Abbruchbedingung
  // Eintrag ≠ 0 in erster Spalte finden
  p := 1
  while a_{p1} = 0 and p ≤ n do p := p + 1 end while
  if p > n then return 0 end if // 1. Spalte ist Nullspalte
  if p > 1 then
    // vertausche Zeilen 1 und p in A
    for j = 1 to n do (a_{1j}, a_{pj}) := (a_{pj}, a_{1j}) end for
    s := −a_{11} // Vorzeichenwechsel der Determinante
  else
    s := a_{11}
  end if
  // erste Zeile durch a_{11} teilen
  for j = 1 to n do a_{1j} := a_{1j}/a_{11} end for
  // erste Spalte ausräumen
  for i = 2 to n do
    for j = 2 to n do
      a_{ij} := a_{ij} − a_{i1}a_{1j}
    end for
  end for
  return s · det((a_{ij})_{2≤i,j≤n})
end function
```

---

Der Aufwand für eine $n \times n$-Matrix besteht in $\Theta(n^2)$ Operationen (aus der zweifach geschachtelten Schleife; der restliche Aufwand ist $O(n)$ und daher vernachlässigbar) plus dem Aufwand für eine $(n-1) \times (n-1)$-Matrix; insgesamt kommt man auf $\Theta(n^3)$ Operationen in $K$ (das ist analog zum Aufwand für das Lösen eines linearen Gleichungssystems, vergleiche den Numerik-Teil der Vorlesung).

Wir wollen jetzt den Fall betrachten, dass die Einträge von $A$ im Ring $\mathbb{Z}$ liegen. Um $\det(A)$ zu berechnen, können wir natürlich auch in $\mathbb{Q}$ rechnen. Wenn wir obigen Algorithmus verwenden, dann werden wir im Verlauf der Rechnung in der Regel verglichen mit den Einträgen der ursprünglichen Matrix recht komplizierte (d.h., mit langem Zähler und Nenner) Brüche zu verarbeiten haben.

17.1. **Beispiel.** Wir betrachten **BSP** det/$\mathbb{Q}$

$$A = \begin{pmatrix} 9 & 7 & 4 \\ 8 & 0 & 3 \\ 7 & 5 & 2 \end{pmatrix} .$$

Der obige Algorithmus läuft dann so ab:

$$\begin{pmatrix} 9 & 7 & 4 \\ 8 & 0 & 3 \\ 7 & 5 & 2 \end{pmatrix} \to \begin{pmatrix} 1 & \frac{7}{9} & \frac{4}{9} \\ * & -\frac{56}{9} & -\frac{5}{9} \\ * & -\frac{4}{9} & -\frac{10}{9} \end{pmatrix} \to \begin{pmatrix} 1 & \frac{7}{9} & \frac{4}{9} \\ * & 1 & \frac{5}{56} \\ * & * & -\frac{15}{14} \end{pmatrix}$$

und die Determinante ergibt sich als

$$9 \cdot \Big(-\frac{56}{9}\Big) \cdot \Big(-\frac{15}{14}\Big) = 60 . \qquad \clubsuit$$

Für eine zufällige $n \times n$-Matrix $A$ mit Einträgen der Länge $\ell$ stellt man fest, dass die Einträge der $(n-k) \times (n-k)$-Matrix, die man nach $k$ Schritten des Verfahrens erhält, rationale Zahlen mit Zähler und Nenner etwa der Länge $(k+1)\ell$ sind (siehe Übungen.) Für den Aufwand in Wortoperationen erhält man daraus eine Komplexität von $\tilde{O}(n^4\ell)$, wenn man schnelle Multiplikation und Division verwendet.

Ein Ansatz, das zu verbessern, beruht auf den folgenden Einsichten.

(1) Die Determinante von $A = (a_{ij})$ ist ein *Polynom* in den Einträgen $a_{ij}$ mit ganzzahligen Koeffizienten, gegeben durch die „Leibnizformel“

$$\det(A) = \sum_{\sigma \in S_n} \varepsilon(\sigma) a_{1,\sigma(1)} \cdots a_{n,\sigma(n)} .$$

(2) Damit ist die Determinante mit Ringhomomorphismen verträglich. Insbesondere gilt $\det\big((\bar{a}_{ij})\big) = \overline{\det(A)}$, wenn der Überstrich die Reduktion modulo $N$ bedeutet (für $N \in \mathbb{Z}_{>1}$).

(3) Wir können $|\det(A)|$ abschätzen, zum Beispiel durch $n^{n/2}M^n$, wenn die Einträge betragsmäßig durch $M$ beschränkt sind (Hadamardsche Ungleichung).

(4) Wenn $a \in \mathbb{Z}$ ist mit $|a| < N/2$ und wir die Restklasse von $a$ mod $N$ kennen, dann können wir $a$ bestimmen (als betragskleinsten Repräsentanten der Restklasse).

Wir können das ausnutzen, indem wir eine Primzahl $p$ wählen, die größer als $2n^{n/2}M^n$ ist, dann die Determinante der mod $p$ reduzierten Matrix in $\mathbb{F}_p$ berechnen und schließlich die Determinante von $A$ aus dem Ergebnis rekonstruieren. Der Aufwand beträgt dann $\Theta(n^3)$ Operationen in $\mathbb{F}_p$, die nach dem im vorigen Abschnitt Gesagten jeweils einen Aufwand von $\tilde{O}(\lambda(p)) = \tilde{O}(n \log M)$ Wortoperationen erfordern. Der Gesamtaufwand ist damit $\tilde{O}(n^4 \log M)$ Wortoperationen. Das ist vergleichbar mit dem Aufwand für die Rechnung über $\mathbb{Q}$. Dabei ist allerdings noch nicht der Aufwand berücksichtigt, der erforderlich ist, um eine geeignete Primzahl $p$ zu finden. Wenn man sich hier mit einer „Pseudo-Primzahl“ zufrieden gibt (die höchstwahrscheinlich prim ist, aber nicht mit Sicherheit), dann kann man eine gegebene Zahl $p$ mit einem Aufwand von $\tilde{O}(\lambda(p)^2)$ Wortoperationen auf Primalität testen. (Wir kommen darauf am Ende der Vorlesung zurück.) Da eine Zahl in der Nähe von $N$ mit einer Wahrscheinlichkeit von etwa $1/\log N$ prim ist, muss man im Schnitt etwa $\log p = \Theta(\lambda(p))$ Zahlen testen, bis man eine

(Pseudo-)Primzahl gefunden hat. Das ergibt dann einen erwarteten Aufwand der Größenordnung $\tilde{O}(n^3(\log M)^3)$. Wenn $(\log M)^2$ deutlich größer ist als $n$, dann ist das Auffinden von $p$ sogar der Schritt, der den Aufwand dominiert! Mit diesem Ansatz ist also noch nicht wirklich etwas gewonnen.

Wie können wir es besser machen? Der wesentliche Grund dafür, dass das Verfahren mittels der Reduktion mod $p$ nicht schneller ist, liegt darin, dass man immer noch mit Zahlen derselben Größenordnung rechnen muss wie die Zähler und Nenner beim Rechnen in $\mathbb{Q}$. Wenn wir die Berechnung effizienter machen wollen, dann sollten wir versuchen, mit kürzeren Zahlen zu rechnen. Was wir dafür brauchen, ist der Chinesische Restsatz, den wir hier für ganze Zahlen formulieren.

**SATZ** Chinesischer Restsatz

17.2. **Satz.** *Seien $m_1, m_2, \ldots, m_n \in \mathbb{Z}_{\geq 1}$ paarweise teilerfremd. Dann gibt es für beliebige $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ stets eine ganze Zahl $x$ mit $x \equiv a_j \bmod m_j$ für alle $j \in \{1, 2, \ldots, n\}$, und $x$ ist modulo $m_1 m_2 \cdots m_n$ eindeutig bestimmt.*

*Genauer gilt, dass der kanonische Homomorphismus*

$$\mathbb{Z} \longrightarrow \frac{\mathbb{Z}}{m_1\mathbb{Z}} \times \frac{\mathbb{Z}}{m_2\mathbb{Z}} \times \cdots \times \frac{\mathbb{Z}}{m_n\mathbb{Z}}$$

*surjektiv ist mit Kern $m_1 m_2 \cdots m_n \mathbb{Z}$.*

Insbesondere kann eine ganze Zahl $a$ mit $|a| < m_1 m_2 \cdots m_n/2$ eindeutig aus ihren Restklassen modulo $m_1$, $m_2$, …, $m_n$ rekonstruiert werden. Wir können also wie folgt vorgehen:

**ALGO** Determinante mit kleinen Primzahlen

---

**function** $\det'(A)$
**input:** $A = (a_{ij}) \in \mathrm{Mat}(n, \mathbb{Z})$
**output:** $\det(A) \in \mathbb{Z}$.

$M := \max\{|a_{ij}| \mid 1 \leq i, j \leq n\}$
Bestimme paarweise verschiedene Primzahlen $p_1, \ldots, p_m$ mit $p_1 \cdots p_m > 2n^{n/2} M^n$
**for** $k = 1$ **to** $m$ **do**
  // Determinante mod $p_k$ berechnen
  $d_k := \det\big((a_{ij} + p_k\mathbb{Z})_{1 \leq i,j \leq n}\big)$
**end for**
$d := \mathrm{crt}\big((p_1, \ldots, p_m), (d_1, \ldots, d_m)\big)$
**if** $2d > p_1 \cdots p_m$ **then** $d := d - p_1 \cdots p_m$ **end if**
**return** $d$
**end function**

---

Dabei soll $\mathrm{crt}\big((m_1, \ldots, m_n), (a_1, \ldots, a_n)\big)$ (*Chinese Remainder Theorem*) für paarweise teilerfremde ganze Zahlen $m_j$ und ganze Zahlen $a_j$ die eindeutig bestimmte Lösung von $x \equiv a_j \bmod m_j$ für alle $j$ mit $0 \leq x < m_1 \cdots m_n$ berechnen.

Wenn wir davon ausgehen, dass alle $p_k$ in ein Datenwort passen (für praktische Zwecke gibt es mehr als genug Primzahlen zwischen $2^{63}$ und $B = 2^{64}$), dann brauchen Operationen in $\mathbb{F}_{p_k}$ konstante Zeit. Die Berechnung von $d_k$ erfordert dann einen Aufwand von $O(n^3)$ Wortoperationen. Wenn wir den Aufwand zur Berechnung der Reste der $a_{ij}$ modulo der $p_k$ und für den Aufruf von crt erst einmal beiseite lassen, dann haben wir $O(n^3 m)$ Wortoperationen. Wie groß muss $m$ sein?

Wegen $p_k \approx B$ brauchen wir etwa $\lambda(n^{n/2}M^n) = O(n(\log n + \log M))$ Primzahlen. Damit kommt man wieder auf $\tilde{O}(n^4 \log M)$ Wortoperationen. Trotzdem ist diese Version schneller (theoretisch und praktisch), denn in der Version mit einer großen Primzahl (oder beim Rechnen in $\mathbb{Q}$) steckt in dem $\tilde{O}$ ein zusätzlicher logarithmischer Faktor, der daher kommt, dass die Multiplikationen und Divisionen quasi-linearen und nicht linearen Aufwand erfordern. Auch der konstante Faktor ist kleiner.

Wir müssen uns allerdings noch überlegen, dass auch die anderen Schritte schnell genug sind. Für das Finden der Primzahlen gilt das, da man für jede weitere Primzahl nur konstanten Aufwand hat. Wie sieht es mit dem Reduzieren mod $p_k$ und der Rekonstruktion mit dem Chinesischen Restsatz aus? Wenn man die $a_{ij}$ mod $p_k$ „naiv“ berechnet, dann braucht man jeweils $\leq \lambda(M)$ Wortoperationen, insgesamt also $n^2 m\lambda(M) \in O(n^3(\log M)^2)$ Wortoperationen, was je nach Verhältnis von $\lambda(M)$ und $n$ durchaus spürbar sein kann. Es gibt allerdings eine bessere Alternative — „Divide and Conquer“ lässt grüßen.

**ALGO**
parallele Reduktion

---

**function** red$\big((m_1, \ldots, m_n), a\big)$
**input:** $m_1, \ldots, m_n \in \mathbb{Z}_{\geq 1}$ paarweise teilerfremd, $a \in \mathbb{Z}$.
**output:** $(a \text{ rem } m_1, \ldots, a \text{ rem } m_n) \in \mathbb{Z}^n$.

```
if n = 1 then return (a rem m_1) end if // Abbruchbedingung
k := ⌊n/2⌋
return red((m_1, ..., m_k), a rem m_1 ··· m_k)
          cat red((m_{k+1}, ..., m_n), a rem m_{k+1} ··· m_n)
end function
```

---

Dabei hängt „cat“ zwei Listen aneinander. Die Produkte $m_1 \cdots m_k$ und $m_{k+1} \cdots m_n$ und die ähnlichen Produkte, die in den rekursiven Aufrufen benötigt werden, kann man zu Beginn ebenfalls rekursiv berechnen und abspeichern. Wenn wir der Einfachheit halber annehmen, dass $n = 2^m$ ist und alle $m_j$ dieselbe Länge $\ell$ haben, dann braucht das $2^{m-1}$ Multiplikationen von Zahlen der Länge $\ell$, $2^{m-2}$ Multiplikationen von Zahlen der Länge $2\ell$, …, $2^{m-k}$ Multiplikationen von Zahlen der Länge $2^{k-1}\ell$, …, eine Multiplikation von Zahlen der Länge $2^{m-1}\ell$. Das ergibt

$$\sum_{k=0}^{m-1} 2^k \tilde{O}(2^{m-1-k}\ell) = \tilde{O}(2^m \ell)$$

Wortoperationen. Wenn wir $|a| < m_1 \cdots m_n$ annehmen, dann ist der Aufwand für die Reduktion analog, nur dass man Divisionen statt Multiplikationen hat. Für unsere Anwendung reduziert das den Aufwand auf $\tilde{O}(n^3 \log M)$ Wortoperationen.

Für die Rekonstruktion gilt Ähnliches. Es gibt eine schnelle Variante des Erweiterten Euklidischen Algorithmus (xgcd: *eXtended Greatest Common Divisor*), die ebenfalls in quasi-linearer Zeit läuft. Damit hat die Funktion crt2 im Algorithmus unten quasi-lineare Komplexität.

**ALGO**
Rekonstruktion

---

**function** crt2$(m_1, m_2, a_1, a_2)$
**input:** $m_1, m_2 \in \mathbb{Z}_{\geq 1}$ teilerfremd, $a_1, a_2 \in \mathbb{Z}$.
**output:** $x \in \mathbb{Z}$, $0 \leq x < m_1 m_2$ mit $x \equiv a_1 \bmod m_1$, $x \equiv a_2 \bmod m_2$.

$(g, s_1, s_2) := \mathrm{xgcd}(m_1, m_2)$ // $g = 1 = s_1 m_1 + s_2 m_2$
**return** $(a_1 s_2 m_2 + a_2 s_1 m_1) \bmod (m_1 m_2)$
**end function**

**function** $\mathrm{crt}\big((m_1, \ldots, m_n), (a_1, \ldots, a_n)\big)$
**input:** $m_1, \ldots, m_n \in \mathbb{Z}_{\geq 1}$ paarweise teilerfremd, $a_1, \ldots, a_n \in \mathbb{Z}$.
**output:** $x \in \mathbb{Z}$, $0 \leq x < m_1 \cdots m_n$ mit $x \equiv a_j \bmod m_j$ für alle $1 \leq j \leq n$.

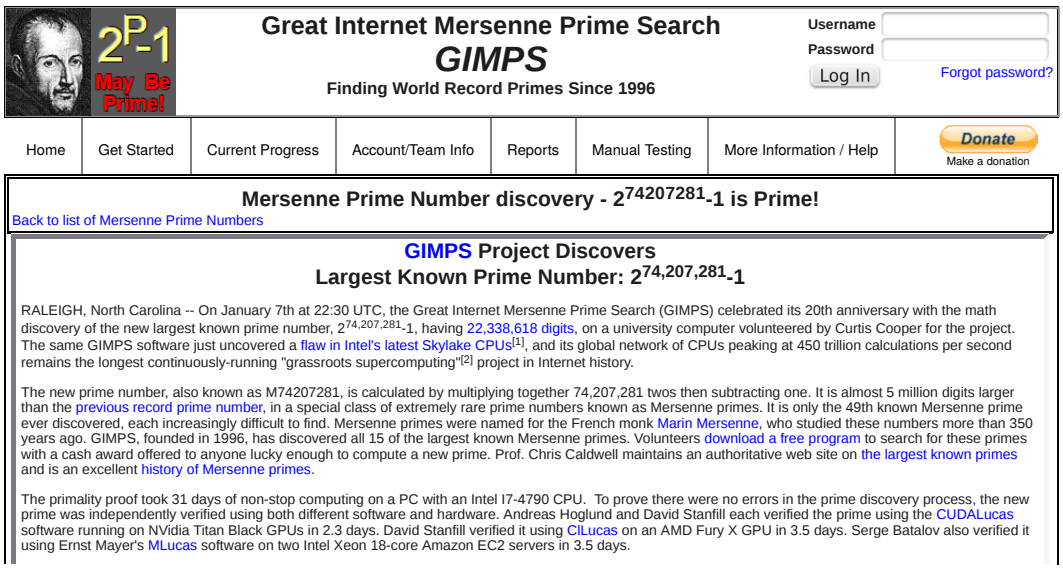
**if** $n = 1$ **then return** $a_1 \bmod m_1$ **end if** // Abbruchbedingung
$k := \lfloor n/2 \rfloor$
$b_1 := \mathrm{crt}\big((m_1, \ldots, m_k), (a_1, \ldots, a_k)\big)$
$b_2 := \mathrm{crt}\big((m_{k+1}, \ldots, m_n), (a_{k+1}, \ldots, a_n)\big)$
**return** $\mathrm{crt2}(m_1 \cdots m_k, m_{k+1} \cdots m_n, b_1, b_2)$
**end function**

---

Wir können wieder die vorberechneten Produkte der $m_j$ verwenden. Der Aufwand für crt2 ist quasi-linear in der Länge der Eingaben; wir erhalten dann insgesamt wieder einen Aufwand von $\tilde{O}(n\ell)$. Für unsere Determinantenberechnung (mit $\ell = 1$ und $m \in O(n \log M)$ statt $n$) ergibt das $\tilde{O}(n \log M)$, fällt also nicht ins Gewicht. (Der Unterschied zur Reduktion am Anfang ist, dass hier nur ein Element rekonstruiert werden muss und nicht $n^2$.)

# 18. Primzahltests

(Auch) aus aktuellem Anlass:




möchte ich in diesem letzten Abschnitt noch etwas auf Primzahltests eingehen.

Wir wollen also folgendes Problem (effizient) lösen:

*Stelle fest, ob eine gegebene natürliche Zahl $N$ eine Primzahl ist!*

Dieses Problem ist durchaus von praktischer Relevanz, da viele moderne kryptographische Verfahren wie zum Beispiel RSA große Primzahlen (mit mehreren Hundert Dezimalstellen) benötigen.

Man wird sich vielleicht zuerst an die Definition einer Primzahl erinnern als einer Zahl, die ($> 1$ ist und) außer 1 und sich selbst keine Teiler hat. Das führt auf den folgenden Algorithmus:

**ALGO**
Probedivision

---

**function** isprime($N$)
**input:** $N \in \mathbb{Z}_{\geq 2}$
**output:** **true**, wenn $N$ Primzahl ist, sonst **false**.

```
  for d = 2 to ⌊√N⌋ do
    if N rem d = 0 then return false end if
  end for
  return true
end function
```

---

Die Schranke $\lfloor\sqrt{N}\rfloor$ kommt daher, dass für jeden Teiler $d$ von $N$ auch $N/d$ ein Teiler ist; wenn es also einen nichttrivialen Teiler gibt, dann gibt es auch einen, der $\leq \sqrt{N}$ ist.

Der Aufwand dafür beträgt im Fall, dass $N$ tatsächlich prim ist (dann wird die Schleife komplett durchlaufen) $\sqrt{N}$ Divisionen von Zahlen der Länge $\lambda(N)$, also grob $\tilde{O}(\sqrt{N})$ Wortoperationen. Das ist keine polynomiale Komplexität, denn die Größe der Eingabe ist $\lambda(N) = \Theta(\log N)$ und $\sqrt{N} = e^{(\log N)/2}$ ist exponentiell in $\log N$.

Wie kann man es schneller hinbekommen? (Dass es offenbar geht, haben wir im letzten Abschnitt schon verwendet.) Da es bisher keinen Algorithmus gibt, der Zahlen in Polynomzeit faktorisieren kann, kann der Ansatz, die Nicht-Existenz eines nichttrivialen Teilers nachzuweisen, nicht zum Ziel führen. Stattdessen hilft es, sich zu überlegen, wie man zeigen kann, dass eine Zahl *nicht* prim ist, ohne sie zu faktorisieren. Dafür kann man Aussagen verwenden, die für Primzahlen gelten, aber für zusammengesetzte Zahlen im Allgemeinen nicht. Eine solche Aussage ist der kleine Satz von Fermat:

**SATZ** kleiner Satz von Fermat

18.1. **Satz.** *Seien $p$ eine Primzahl und $a \in \mathbb{Z}$ mit $p \nmid a$. Dann ist $a^{p-1} \equiv 1 \bmod p$.*

Im Umkehrschluss gilt dann: Ist $N \in \mathbb{Z}_{\geq 2}$ und $a \in \mathbb{Z}$ mit $0 < a < N$ und $a^{N-1} \not\equiv 1 \bmod N$, dann kann $N$ keine Primzahl sein.

Bevor wir das in einen Algorithmus umsetzen, überlegen wir uns noch, dass wir Reste von Potenzen der Form $a^e$ rem $N$ effizient berechnen können.

**LEMMA** Potenzen mod $N$

18.2. **Lemma.** *Seien $N \in \mathbb{Z}_{\geq 2}$, $0 \leq a < N$ und $e \in \mathbb{Z}_{\geq 2}$. Man kann $a^e$ rem $N$ mit einem Aufwand von $O(\log e)$ Operationen in $\mathbb{Z}/N\mathbb{Z}$, also $\tilde{O}\big((\log e)(\log N)\big)$ Wortoperationen, berechnen.*

*Beweis.* Wir verwenden die Relationen

$$a^{2m} \text{ rem } N = (a^m \text{ rem } N)^2 \text{ rem } N \qquad \text{und}$$
$$a^{2m+1} \text{ rem } N = a \cdot (a^{2m} \text{ rem } N) \text{ rem } N\,,$$

die aus den Rechenregeln $a^{2m} = (a^m)^2$ und $a^{2m+1} = a \cdot a^{2m}$ folgen. Das liefert den folgenden rekursiven Algorithmus:

**ALGO** Potenzen mod $N$

---

**function** expmod$(a, e, N)$
**input:** $N \in \mathbb{Z}_{\geq 2}$, $0 \leq a < N$, $e \geq 0$
**output:** $a^e$ rem $N$.

```
  if e = 0 then return 1 end if
  b := expmod(a, ⌊e/2⌋, N)
  b := b^2 rem N
  if e rem 2 = 1 then
    b := (a · b) rem N
  end if
  return b
end function
```

---

Die Anzahl der rekursiven Aufrufe ist $\lfloor \log_2 e \rfloor + 1 \in O(\log e)$. In jedem Aufruf werden eine oder zwei Multiplikationen und Divisionen von Zahlen der Länge $\lambda(N)$ ausgeführt; der Aufwand dafür ist jeweils $\tilde{O}(\log N)$ Wortoperationen. ❑

Damit können wir die Relation $a^{N-1} \equiv 1 \bmod N$, die zu $a^{N-1}$ rem $N = 1$ äquivalent ist, mit einem Aufwand von $\tilde{O}((\log N)^2)$ Wortoperationen testen.

Der kleine Satz von Fermat ergibt dann den „Fermat-Test“:

**ALGO** Fermat-Test

**function** fermat$(N, m)$
**input:** $N \in \mathbb{Z}_{\geq 2}$, $m \geq 1$: Anzahl der Tests
**output:** „zusammengesetzt“ oder „möglicherweise prim“.

```
  for i = 1 to m do
    a := random(2, N − 1)
    if expmod(a, N − 1, N) ≠ 1 then
      return „zusammengesetzt“
    end if
  end for
  return „möglicherweise prim“
end function
```

Hierbei soll random$(A, B)$ für ganze Zahlen $A \leq B$ eine Zufallszahl aus $[A, B] \cap \mathbb{Z}$ liefern. Der Parameter $m$ gibt an, für wie viele Werte von $a$ die Aussage des kleinen Fermatschen Satzes getestet wird.

Wenn wir annehmen, dass der Aufwand für die Bestimmung einer Zufallszahl von der Größenordnung $O(\log N)$ ist („Zufallszahlen“ werden im Computer durch Pseudo-Zufallszahlen-Generatoren erzeugt, die bei jedem Aufruf eine gewisse Folge von Operationen auf Zahlen einer festen Länge ausführen; damit bekommt man eine feste Zahl von zufälligen Bits in konstanter Zeit, also $\lambda(N)$ zufällige Worte in $O(\lambda(N))$ Wortoperationen), dann ist der Aufwand für den Fermat-Test $\tilde{O}(m(\log N)^2)$ Wortoperationen.

Jetzt stellt sich die Frage, ob dieser Test auch zuverlässig ist: Falls $N$ zusammengesetzt ist, wird der Test das für geeignetes $a$ auch nachweisen? Und ist der Anteil der geeigneten $a$ unabhängig von $N$ durch eine positive Konstante nach unten beschränkt? (Dann kann man $m$ so wählen, dass der Test mit einer Wahrscheinlichkeit beliebig nahe bei 1 feststellt, dass $N$ zusammengesetzt ist.)

Dazu geben wir den „schlechten“ Zahlen $N$ erst einmal eine Bezeichnung.

**DEF** Pseudoprimzahl, Carmichael-Zahl

18.3. **Definition.** Sei $a \in \mathbb{Z}$. Eine ganze Zahl $N \geq 2$ heißt *Pseudoprimzahl* zur Basis $a$, wenn $N$ nicht prim ist, aber $a^{N-1} \equiv 1 \bmod N$ gilt.

$N$ heißt *Carmichael-Zahl*, wenn $N$ Pseudoprimzahl zur Basis $a$ ist für alle zu $N$ teilerfremden $a \in \mathbb{Z}$. ♢

Jede Zahl ist Pseudoprimzahl zur Basis 1, und jede ungerade Zahl ist Pseudoprimzahl zur Basis $-1$. Es sind also nur Zahlen $a$ mit $|a| \geq 2$ interessant.

**BSP** Pseudoprimzahlen

18.4. **Beispiele.** Es gibt Pseudoprimzahlen zu Basen $a \neq \pm 1$. Die kleinste Pseudoprimzahl zur Basis 2 ist zum Beispiel $N = 341 = 11 \cdot 31$. Die kleinste Pseudoprimzahl zur Basis 3 ist $N = 91 = 7 \cdot 13$.

Es gibt tatsächlich auch Carmichael-Zahlen. Die kleinste ist $N = 561 = 3 \cdot 11 \cdot 17$. Es gibt sogar unendlich viele davon; das wurde von Alford, Granville und Pomerance 1994 bewiesen.[1] ♣

[1] W.R. Alford, A. Granville, C. Pomerance: *There are infinitely many Carmichael numbers*, Ann. Math. **139** (1994), 703–722.

Die Existenz von Carmichael-Zahlen zeigt, dass der Fermat-Test noch nicht befriedigend ist, weil er Carmichael-Zahlen nicht als zusammengesetzt erkennen kann. Man muss ihn also noch verfeinern. Dazu verwenden wir eine weitere Eigenschaft von Primzahlen:

18.5. **Lemma.** *Sei $p > 2$ prim. Dann hat die Kongruenz $x^2 \equiv 1 \bmod p$ genau zwei Lösungen $0 \le x < p$.*

**LEMMA** $x^2 \equiv 1 \bmod p$

*Beweis.* Es gibt stets die Lösungen $x = 1$ und $x = p - 1$; wegen $p > 2$ sind sie verschieden. Da $p$ eine Primzahl ist, ist $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ ein Körper. Also hat das Polynom $x^2 - 1$ höchstens zwei Nullstellen in $\mathbb{F}_p$. Das ist dazu äquivalent, dass die Kongruenz höchstens zwei Lösungen zwischen 0 und $p-1$ (einschließlich) hat. ❑

Wenn wir also eine Zahl $a$ finden mit $a^2 \equiv 1 \bmod N$, aber $a \not\equiv \pm 1 \bmod N$, dann kann $N$ nicht prim sein. Wir nutzen das wie folgt: Wir können annehmen, dass $N$ ungerade ist (sonst ist entweder $N = 2$ oder $N$ ist offensichtlich zusammengesetzt). Dann ist $N - 1$ gerade, und wir schreiben $N - 1 = q2^e$ mit $q$ ungerade und $e \ge 1$. Wir können $a^{N-1}$ rem $N$ berechnen, indem wir

$$b_0 = a^q \text{ rem } N\,, \quad b_1 = b_0^2 \text{ rem } N\,, \quad b_2 = b_1^2 \text{ rem } N\,, \quad \ldots\,, \quad b_e = b_{e-1}^2 \text{ rem } N$$

setzen; dann ist $b_e = a^{N-1}$ rem $N$. Das liefert uns einige Gelegenheiten, das Kriterium von Lemma 18.5 zu überprüfen: Wenn $N$ den Fermat-Test besteht, dann ist $b_{e-1}^2 \equiv b_e = 1 \bmod N$. Falls also $b_{e-1} \not\equiv \pm 1 \bmod N$ ist, dann kann $N$ nicht prim sein. Falls $b_{e-1} = 1$ ist (und $e \ge 2$), dann können wir $b_{e-2}$ testen, und so weiter. Wir erhalten auf diese Weise den Miller-Rabin-Test:

**ALGO** Miller-Rabin-Test

**function** MillerRabin$(N, m)$
**input:** $N \in \mathbb{Z}_{\ge 5}$ ungerade, $m \ge 1$: Anzahl der Tests
**output:** „zusammengesetzt“ oder „wahrscheinlich prim“.

```
  q := N − 1; e := 0
  while q rem 2 = 0 do
    q := q/2; e := e + 1
  end while    // jetzt ist N − 1 = 2^e q mit q ungerade
  for i = 1 to m do
    a := random(2, N − 2)
    b := expmod(a, q, N)
    if b = 1 then continue end if    // Test OK, nächstes a
    for j = 1 to e do
      if b = N − 1 then break end if    // Test OK, nächstes a
      if j = e then return „zusammengesetzt“ end if
      b := b^2 rem N
      if b = 1 then return „zusammengesetzt“ end if
    end for
  end for
  return „wahrscheinlich prim“
end function
```

Die Anweisung **continue** bewirkt, dass der innerste Schleifendurchlauf abgebrochen wird und der Programmablauf mit dem nächsten Durchlauf fortgesetzt wird.

Die Anweisung **break** bewirkt, dass der innerste Schleifendurchlauf abgebrochen wird und der Programmablauf nach dem Schleifenende fortgesetzt wird.

Wenn $a^q \equiv 1 \bmod N$ ist, dann sind alle $b_j = 1$, und es gibt keinen Widerspruch dazu, dass $N$ prim sein könnte („**if** $b = 1$ **then continue end if**“). Wenn $b_j = N-1$ ist für ein $0 \le j < e$, dann gilt $b_{j+1} = \ldots = b_e = 1$; das liefert ebenfalls keinen Widerspruch. Ist $b_j = 1$ für $j > 0$ mit $j$ minimal und ist $b_{j-1} \neq N-1$, dann sagt Lemma 18.5, dass $N$ zusammengesetzt sein muss. Wenn wir $b_e$ berechnen müssten und $b_{e-1} \neq 1, N-1$ ist, dann kann $N$ ebenfalls keine Primzahl sein, denn entweder ist $b_e \neq 1$, womit $N$ den Fermat-Test nicht besteht, oder $b_e = 1$ ist das Quadrat modulo $N$ einer Zahl, die $\not\equiv \pm 1$ ist, womit das Kriterium aus Lemma 18.5 zieht.

Der Aufwand für den Miller-Rabin-Test entspricht dem für den Fermat-Test (oder ist sogar geringer, da eventuell ein Teil der Rechnung übersprungen wird), da im Wesentlichen nur die Potenz $a^{N-1}$ rem $N$ berechnet wird.

Der große Vorteil des Miller-Rabin-Tests gegenüber dem Fermat-Test ist seine Zuverlässigkeit:

**SATZ**
M-R-Test ist zuverlässig

18.6. **Satz.** *Sei $N \in \mathbb{Z}_{\ge 15}$ ungerade und zusammengesetzt. Dann gibt es höchstens $(N-11)/4$ Zahlen $2 \le a \le N-2$, sodass $N$ den Miller-Rabin-Test mit der Basis $a$ besteht (also nicht als zusammengesetzt erkannt wird).*

*Beweis.* Hier ist eine Beweisskizze. Man betrachtet die Menge $M$ der Restklassen $[a]$, die „schlecht“ sind in dem Sinn, dass der Test mit $a$ die Zahk $N$ nicht als zusammengesetzt erkennt. Man zeigt dann, dass $M$ in einer Untergruppe $G$ der Einheitengruppe $(\mathbb{Z}/N\mathbb{Z})^\times$ enthalten ist, deren Index $\ge 4$ ist. Daraus folgt die Behauptung. ❑

Man kann den Satz so interpretieren, dass bei zufälliger Wahl von $a$ eine zusammengesetzte Zahl $N$ mit Wahrscheinlichkeit $\ge 3/4$ auch als zusammengesetzt erkannt wird. (Für die allermeisten Zahlen ist der Index von $M$ deutlich größer als 4 und damit die Wahrscheinlichkeit noch deutlich größer.) Wenn wir also den Miller-Rabin-Test mit $m$ unabhängig voneinander zufällig gewählten Zahlen $a$ durchführen, wird $N$ mit einer Wahrscheinlichkeit von mindestens $1 - (1/4)^m$ als zusammengesetzt erkannt. Für $m = 20$, was ein üblicher Wert ist, ist $(1/4)^m = 2^{-40} < 10^{-12}$. Eine Zahl $p$, die diesen Test bestanden hat, ist also sehr wahrscheinlich prim. Wie wahrscheinlich? Die Dichte von Primzahlen in der Nähe von $N$ ist nach dem Primzahlsatz etwa $1/\log N$. Da wir nur ungerade Zahlen betrachten (und die geraden alle zusammengesetzt sind), erhöht sich die Dichte auf $2/\log N$. Nach dem Satz von Bayes aus der elementaren Stochastik ergibt sich dann die bedingte Wahrscheinlichkeit dafür, dass eine Zahl $N$, die den Test überstanden hat, tatsächlich prim ist, zu $\ge 1 - \frac{1}{2}10^{-12}\log N$. Für $N \approx 10^{1000}$ ist $\log N \approx 1000 \log 10 \approx 2300$; das ergibt als grobe Schätzung eine Wahrscheinlichkeit von $\le 10^{-9}$ dafür, dass eine tausendstellige „Miller-Rabin-Primzahl“ $N$ nicht prim ist. Auf meinem (inzwischen fast fünf Jahre alten) Laptop dauert ein Miller-Rabin-Test mit $m = 20$ und $N \approx 10^{1000}$ prim (sodass der Test komplett durchlaufen wird) eine knappe halbe Sekunde.

Als Mathematiker möchte man sich aber vielleicht nicht mit einer sehr hohen Wahrscheinlichkeit zufrieden geben, sondern mit Sicherheit wissen wollen, ob $N$ prim ist oder nicht. Wir brauchen also noch eine effiziente Möglichkeit, von einer Zahl $N$, von der wir stark vermuten, dass sie prim ist (etwa weil sie den Miller-Rabin-Test bestanden hat), auch zu *beweisen*, dass sie prim ist. Eine Möglichkeit

basiert auf einer Art „Umkehrung“ des kleinen Satzes von Fermat. Dazu erst einmal eine Hilfsaussage. Zur Erinnerung: „$a \perp b$“ steht für „$a$ und $b$ sind teilerfremd“.

**LEMMA** Hilfsaussage für PL-Test

18.7. **Lemma.** *Seien $N > 0$ eine ganze Zahl und $p$ ein Primteiler von $N-1$. Sei weiter $a_p \in \mathbb{Z}$ mit*

$$a_p^{N-1} \equiv 1 \bmod N \qquad \text{und} \qquad (a_p^{(N-1)/p} - 1) \perp N\,. \tag{18.1}$$

*Sei außerdem $p^{e_p}$ die höchste Potenz von $p$, die $N-1$ teilt. Dann gilt für jeden (positiven) Teiler $d$ von $N$, dass $d \equiv 1 \bmod p^{e_p}$ ist.*

*Beweis.* Wir können uns auf Primteiler $d$ beschränken. Da $a_p \perp N$, also auch $a_p \perp d$, folgt $a_p^{d-1} \equiv 1 \bmod d$. Andererseits ist $a_p^{(N-1)/p} \not\equiv 1 \bmod d$, da nach Voraussetzung $(a_p^{(N-1)/p} - 1) \perp N$. Sei $n$ die Ordnung von $a_p$ mod $d$; dann folgt $n \mid d-1$, $n \mid N-1$ (denn $a_p^{N-1} \equiv 1 \bmod d$), aber $n \nmid (N-1)/p$. Aus den letzten beiden Eigenschaften folgt $p^{e_p} \mid n$, aus der ersten dann $p^{e_p} \mid d-1$. ❑

Wenn wir über die Faktorisierung von $N-1$ gut genug Bescheid wissen, können wir dieses Ergebnis nutzen, um zu beweisen, dass $N$ prim ist.

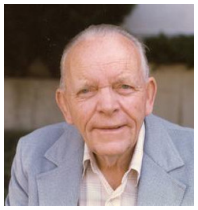
**FOLG** Kriterium für Primzahl

18.8. **Folgerung.** *Sei $N > 0$ eine ganze Zahl, sei $N-1 = F \cdot U$ mit $F \geq \sqrt{N}$, und alle Primteiler von $F$ seien bekannt.*

*$N$ ist genau dann prim, wenn es für jeden Primteiler $p$ von $F$ eine Zahl $a_p \in \mathbb{Z}$ gibt, die* (18.1) *erfüllt.*

*Beweis.* Sei zunächst $N$ prim, und sei $g$ eine Primitivwurzel mod $N$ (d.h. so dass (das Bild von) $g$ die zyklische Gruppe $(\mathbb{Z}/N\mathbb{Z})^\times$ erzeugt). Dann hat $a_p = g$ die Eigenschaft (18.1).

Seien nun umgekehrt für alle $p \mid F$ Zahlen $a_p$ mit (18.1) gegeben. Aus Lemma 18.7 folgt dann, dass jeder Teiler $d$ von $N$ die Kongruenz $d \equiv 1 \bmod F$ erfüllt. Insbesondere ist $d = 1$ oder $d > F \geq \sqrt{N}$. Wenn $N$ zusammengesetzt wäre, hätte $N$ einen nichttrivialen Teiler $\leq \sqrt{N}$, was wir gerade ausgeschlossen haben, also ist $N$ prim. ❑



D.H. Lehmer
1905 – 1991

Aus diesem Ergebnis lässt sich direkt ein Primzahltest ableiten, der *Pocklington-Lehmer*[2]*-Test*. Er kann nachweisen, dass eine Zahl prim ist, aber nicht, dass eine Zahl $N$ zusammengesetzt ist. Man wird also erst den Miller-Rabin-Test verwenden, um ausreichend sicher zu sein, dass $N$ prim ist, und dann den Pocklington-Lehmer-Test, um dies endgültig nachzuweisen.

Der Test basiert auf der Verwendung der zyklischen Gruppe $(\mathbb{Z}/N\mathbb{Z})^\times$ der Ordnung $N-1$ (wenn $N$ prim ist). Sein Nachteil ist, dass er eine gute Kenntnis der Faktorisierung von $N-1$ erfordert, was in der Praxis ein großes Hindernis sein kann. Für Zahlen spezieller Form lässt der Test sich aber gut verwenden.

18.9. **Beispiel.** Eine Zahl der Form $F_n = 2^{2^n} + 1$ heißt *Fermat-Zahl*. Fermat hatte behauptet, dass alle Fermat-Zahlen Primzahlen sind. (Es ist leicht zu sehen, dass $2^m + 1$ nicht prim sein kann, wenn $m$ keine Zweierpotenz ist.) Tatsächlich sind die Zahlen

**BSP** Fermat-Zahlen

$$F_0 = 3, \quad F_1 = 5, \quad F_2 = 17, \quad F_3 = 257, \quad F_4 = 65537$$

alle prim, aber Euler hat gezeigt, dass $F_5$ durch 641 teilbar und damit nicht prim ist. Tatsächlich ist bis heute keine weitere Fermatsche Primzahl bekannt, aber von $F_5, \ldots, F_{32}$ (und etlichen weiteren) ist bekannt, dass sie nicht prim sind.

Für Fermat-Zahlen lässt sich der Pocklington-Lehmer-Test noch etwas vereinfachen:

*Für $n \geq 1$ ist $F_n$ genau dann prim, wenn $3^{2^{2^n-1}} \equiv -1 \bmod F_n$ ist.*

Ist $F_n$ prim, dann ist 3 ein quadratischer Nichtrest mod $F_n$; das folgt mit dem Quadratischen Reziprozitätsgesetz daraus, dass $F_n = 4^{2^{n-1}} + 1 \equiv 2 \bmod 3$ ist. Die Aussage ergibt sich dann aus dem Euler-Kriterium. Umgekehrt ist $a_2 = 3$ eine passende Zahl für den Pocklington-Lehmer-Test (und 2 ist der einzige Primteiler von $F_n - 1$).

Zum Beispiel erhalten wir für $n = 4$ (mit $2^n - 1 = 15$):

| $m$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $3^{2^m} \bmod F_4$ | 3 | 9 | 81 | 6561 | −11088 | −3668 | 19139 | 15028 |
| $m$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $3^{2^m} \bmod F_4$ | 282 | 13987 | 8224 | −8 | 64 | 4096 | −256 | −1 |

♣

Man kann diesen Ansatz variieren, indem man statt $\mathbb{F}_N^\times$ die Untergruppe der Ordnung $N + 1$ von $\mathbb{F}_{N^2}^\times$ benutzt. Dabei braucht man dann Informationen über die Faktorisierung von $N + 1$. Das führt zum Beispiel zum bekannten *Lucas-Lehmer-Test* für Mersennesche Primzahlen $2^p - 1$ (siehe unten). Eine weitere Alternative besteht darin, sogenannte „Elliptische Kurven“ zu verwenden. Sie stellen ebenfalls eine Gruppe der Größe etwa $N$ zur Verfügung und können in ähnlicher Weise genutzt werden, haben dabei aber den großen Vorteil, dass viele verschiedene Kurven mit unterschiedlichen Gruppenordnungen zur Verfügung stehen, sodass man mit vernünftiger Wahrscheinlichkeit eine Kurve findet, deren Gruppenordnung gut faktorisierbar ist. Das führt auf den ECPP-Algorithmus („Elliptic curve primality proof“) von Goldwasser und Kilian, der in der Praxis in den meisten Fällen der zurzeit effizienteste Algorithmus zur Verifikation von Primzahlen ist.

Eine Diskussion von Primzahltests wäre nicht vollständig, ohne den deterministischen Polynomzeit-Algorithmus von Agrawal, Kayal und Saxena[3] zu erwähnen. Dieses Resultat löst ein altes Problem, denn bis dahin war kein Verfahren bekannt, das für eine beliebige natürliche Zahl deterministisch (d.h. ohne Zufallszahlen zu verwenden wie etwa der Miller-Rabin-Test) und in polynomialer Laufzeit feststellt, ob sie prim ist oder zusammengesetzt. Dieser Durchbruch hat sich aus einem Bachelorprojekt der beiden Studenten Kayal und Saxena entwickelt. Dieser Algorithmus ist allerdings bisher (auch nach einigen Verbesserungen) in der Praxis noch langsamer als Miller-Rabin plus ECPP.

[2]Bild: Oberwolfach Photo Collection, Copyright: George M. Bergman, Berkeley

[3]Manindra Agrawal, Neeraj Kayal, Nitin Saxena. *PRIMES is in P,* Annals of Mathematics **160** (2004), no. 2, 781–793.


F.É.A. Lucas
1842 – 1891

Zum Abschluss möchte ich (aus aktuellem Anlass, siehe den Beginn dieses Abschnitts) noch erklären, wie man *Mersenne-Zahlen*, das sind Zahlen der Form $M_p = 2^p - 1$ mit $p$ prim, auf Primalität testen kann. (Ist der Exponent $n$ von 2 keine Primzahl, dann kann $2^n - 1$ nicht prim sein.) Dazu verwendet man, wie oben angedeutet, eine Variante des Pocklington-Lehmer-Tests, die mit einer Gruppe der Ordnung $N + 1$ arbeitet. (Sei $N$ prim, dann kann man den Körper $\mathbb{F}_{N^2}$ betrachten. Seine multiplikative Gruppe $\mathbb{F}_{N^2}^\times$ ist zyklisch und hat die Ordnung $N^2 - 1 = (N-1)(N+1)$, hat also eine ebenfalls zyklische Untergruppe der Ordnung $N+1$.) Sein Spezialfall für Mersenne-Zahlen ist der Lucas-Lehmer-Test, der auf folgender Aussage beruht:

**LEMMA**
Lucas-Lehmer-Test

18.10. **Lemma.** *Sei $p \geq 3$ prim. Wir definieren die Folge $(S_n)$ durch $S_0 = 4$ und $S_{n+1} = S_n^2 - 2$. $M_p = 2^p - 1$ ist genau dann eine Primzahl, wenn $S_{p-2}$ durch $M_p$ teilbar ist.*

*Beweis.* Man zeigt zuerst durch Induktion, dass

$$(2+\sqrt{3})^{2^m} + (2-\sqrt{3})^{2^m} = S_m \qquad \text{und} \qquad (2+\sqrt{3})^{2^m}(2-\sqrt{3})^{2^m} = 1$$

gilt. Aus $S_{p-2} \equiv 0 \bmod M_p$ folgt $S_{p-1} \equiv -2 \bmod M_p$ und umgekehrt, und die Kongruenz bedeutet gerade, dass $[2+\sqrt{3}]^{2^{p-1}} = -1$ ist in $\mathbb{Z}/M_p\mathbb{Z}[\sqrt{3}]$, wobei $R[\sqrt{3}]$ für $R[x]/\langle x^2-3\rangle$ steht.

Ist $M_p$ prim, dann ist 3 (wegen $M_p \equiv 7 \bmod 12$; hier benutzen wir $p \geq 3$) ein quadratischer Nichtrest mod $M_p$ und daher $\mathbb{Z}/M_p\mathbb{Z}[\sqrt{3}] \cong \mathbb{F}_{M_p^2}$. Man kann zeigen, dass $[2+\sqrt{3}]$ zwar ein Quadrat in $\mathbb{F}_{M_p^2}$ ist, aber kein Quadrat eines Elements in der Untergruppe der Ordnung $M_p + 1$. Damit hat $[2+\sqrt{3}] \in \mathbb{F}_{M_p^2}^\times$ Ordnung $M_p + 1 = 2^p$; also muss $[2+\sqrt{3}]^{2^{p-1}} = -1$ sein, und das Kriterium ist erfüllt.

Sei umgekehrt das Kriterium erfüllt. $M_p$ hat (wieder wegen $M_p \equiv 7 \bmod 12$) einen Primteiler $q \equiv 5$ oder $7 \bmod 12$, sodass 3 ein quadratischer Nichtrest mod $q$ ist. Das Kriterium zeigt, dass $[2+\sqrt{3}] \in \mathbb{F}_{q^2}^\times$ Ordnung $2^p$ hat (denn $[2+\sqrt{3}]^{2^{p-1}} = -1$). Da $[2+\sqrt{3}]$ in einer Untergruppe der Ordnung $q+1$ liegt, folgt $q + 1 \geq 2^p$, also $q \geq M_p$ und damit ist $M_p = q$ eine Primzahl. ❑

Wenn man den Test implementiert, dann berechnet man die Folge $(S_n)$ modulo $M_p$, setzt also $S_{n+1} = (S_n^2 - 2) \operatorname{rem} M_p$, und testet am Ende, ob $S_{p-2} = 0$ ist.

Der Aufwand für den Test beträgt im Wesentlichen $p - 1$ Quadrierungen modulo $M_p$, das sind $\tilde{O}(p^2) = \tilde{O}(\lambda(M_p)^2)$ Wortoperationen. Damit ist dieser Test deutlich schneller als jeder allgemein anwendbare Primzahltest.

**BSP**
LL-Test

18.11. **Beispiele.** Ist $M_{11} = 2^{11} - 1 = 2047$ prim? Wir machen den Test:

| $m$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_m \bmod M_{11}$ | 4 | 14 | 194 | 788 | 701 | 119 | −170 | 240 | 282 | −311 |

Da $S_9$ nicht durch $M_{11}$ teilbar ist, ist $M_{11}$ nicht prim (tatsächlich hat man die Faktorisierung $M_{11} = 23 \cdot 89$).

Ist $M_{13} = 2^{13} - 1 = 8191$ prim? Wir machen den Test:

| $m$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_m \bmod M_{13}$ | 4 | 14 | 194 | −3321 | 3953 | −2221 | 1857 | 36 | 1294 | 3470 | 128 | 0 |

Da $S_{11}$ durch $M_{13}$ teilbar ist, ist $M_{13}$ prim. ♣

# Literatur

[GG] Joachim von zur Gathen und Jürgen Gerhard: *Modern computer algebra,* 2nd edition, Cambridge University Press, 2003.

[Gr] Lars Grüne: *Einführung in die numerische Mathematik,* fünfte Auflage vom Wintersemester 2013/14, Vorlesungsskript. Online verfügbar unter http://num.math.uni-bayreuth.de/de/team/Gruene_Lars/lecture_notes/index.html

[Ku] Sascha Kurz: *Angewandte Mathematik (für Lehramt an Gymnasien),* Vorlesungsskript Wintersemester 2013/14 bzw. 2014/15.

[SK] Hans Rudolf Schwarz und Norbert Köckler: *Numerische Mathematik,* 8. Auflage, Vieweg+Teubner 2011.

[V] Robert J. Vanderbei: *Linear programming. Foundations and extensions.* Fourth edition. International Series in Operations Research & Management Science, **196**. Springer, New York, 2014.