Kapitel 2

Algebraische Gleichungen

Das Wort „Algebra“ wird auf eine Aufgabensammlung aus dem 9. Jahrhundert zurückgeführt; es entstand aus ihrem Titel „Hisâb al-jabr w'al-muqâbalah“ („Wissenschaft der Reduktion und des gegenseitigen Aufhebens“). Verfasst wurde dieses Buch von Muhammad ibn Musa al Hwârizmí, aus dessen Namen im Laufe der Zeit auch das Wort „Algorithmus“ entstanden sein soll.
Algebra in ihrem ursprünglichen Sinn ist so als die Lehre der Auflösung von Gleichungen zu verstehen. Damit beschäftigten sich bereits Mathematiker früher Kulturen, etwa im Zusammenhang mit Problemen geometrischer Art, wie sie in der Landvermessung auftraten. Es gibt Keilschrifttafeln aus dem 2. Jahrtausend v.u.Z., auf denen lineare Gleichungssysteme mit mehreren Unbekannten und sogar Systeme höheren Grades gelöst werden.

Wir haben bereits im 1. Kapitel gesehen, dass die Frage nach der Auflösung von Gleichungen immer wieder Anlass zur Erweiterung des Zahlbegriffs gab. Die ganzen Zahlen erhalten wir als Lösung des Problems, wie zu gegebenen natürlichen Zahlen a und b eine Zahl x gefunden werden kann, für die $a + x = b$ ist. Das ist allgemein möglich, wenn anstelle der natürlichen die ganzen Zahlen betrachtet werden.
Eine ähnliche Situation ergibt sich für die Multiplikation.
Die Gleichung $a \cdot x = b$ mit $a, b \in \mathbb{Z}$ und $a \neq 0$ besitzt erst nach Übergang zum Körper $\mathbb{Q}$ der rationalen Zahlen stets eine Lösung $x \in \mathbb{Q}$.
Wie schon früher erwähnt, entsteht durch die Konstruktion des Körpers $\mathbb{C}$ der komplexen Zahlen ein Zahlbereich, in dem jedes nichtkonstante Polynom mindestens eine Nullstelle besitzt. Wir wollen allerdings nicht übersehen, dass es vom Wissen über die Existenz einer Nullstelle bis zu ihrem Auffinden ein weiter Weg sein kann. Es gibt sogar Gleichungen, für die es prinzipiell unmöglich ist, eine Lösung im Sinne einer Formel anzugeben, in der außer den Grundrechenarten nur Wurzelausdrücke vorkommen, so wie wir dies z.B. von quadratischen Gleichungen kennen.
Ein Gegenbeispiel (wir geben hier keinen Beweis an) ist die recht einfach anmutende Gleichung $x^5 - 10x - 2 = 0$, die nach dem Zwischenwertsatz der Analysis sogar eine reelle Lösung besitzt. Dieses Phänomen werden wir erst recht zu beachten haben, wenn wir Systeme von Gleichungen höheren Grades mit mehreren Variablen lösen.
Der Verzicht auf Exaktheit beim Rechnen mit Näherungen wirft dagegen neue Probleme auf.

Die Naturwissenschaften und die Technik stellen gerade in der Gegenwart allerhöchste Anforderungen an die Mathematik hinsichtlich der Auflösung von Gleichungssystemen. Dies ist keine einseitige Entwicklung. Die Nutzung von Computern hatte zweifellos einen entscheidenden Einfluß auf die Entwicklung der letzten Jahrzehnte des 20. Jahrhunderts, wovon auch die „reine“ Mathematik nicht unberührt blieb: Die Auflösung polynomialer Gleichungen konnte in gewisser Weise automatisiert werden. Das symbolische Rechnen (auch *Computeralgebra* genannt) wurde nicht nur zu einer Schlüsseltechno-

logie der heutigen Informationsgesellschaft, es eröffnet der mathematischen Forschung den Zugang zur Nutzung von Algorithmen, die vergangenen Generationen von Mathematikern zwar in Ansätzen bekannt waren, wegen ihrer Komplexität aber ohne die aktuellen technischen Hilfsmittel nicht genutzt werden konnten.
Wir sollten jedoch nicht erwarten, dass eines Tages Maschinen unsere Arbeit erledigen, sondern müssen uns im Gegenteil bewusst sein, dass wir vor Herausforderungen stehen, wie es sie in der Geschichte der Mathematik noch nicht gegeben hat. Bruno Buchberger, der für die Computeralgebra fundamentale Arbeit geleistet hat, bemerkte dazu, wer heute Mathematik studiere, befinde sich „. . . im ‚Auge des Hurricans' der modernen Entwicklung und nicht irgendwo in einem Hinterzimmer" . . . „Je algorithmischer und dann je effizienter man mathematische Probleme lösen will, umso mehr mathematische Theorie und umso schwierigere Beweise sind nötig und nicht umgekehrt" (vgl. [DMV][1]).

Das vorliegende Kapitel kann nicht mehr sein als ein Einstieg in die Frage nach den Lösungen polynomialer Gleichungssysteme. Am Schluss wird klar, dass die Struktur der Lösungsmengen sehr viel subtiler ist, als es auf den ersten Blick vermutet werden mag. Die mathematische Disziplin, die sich insbesondere den qualitativen Aspekten des Studiums der Lösungsmengen widmet, ist die algebraische Geometrie. Die numerische Mathematik befasst sich dagegen mit dem Auffinden von Näherungslösungen; auch dazu können wir hier nur Andeutungen machen.

2.6 Symbolisches Rechnen

Mathematisches Verständnis drückt sich nicht zuletzt darin aus, dass wir eine Eigenschaft erwerben, die wohl am besten durch das Wort „Routine" beschrieben wird: Für gewisse, immer wieder vorkommende Typen von Problemen erkennen wir auf den ersten Blick ein Lösungsverfahren, einen *Algorithmus*, der uns das „Denken" abnimmt. Tatsächlich haben wir unsere geistigen Anstrengungen bereits zuvor in den Algorithmus investiert. Was in allen Einzelheiten verstanden ist, kann auch technischen Geräten überlassen werden, durch die wir mehr Zeit für den kreativen Teil unserer Arbeit gewinnen. In einfachster Form geschah das schon in einem frühen Stadium der Mathematik. Der Abakus wird noch heute in manchen Gegenden der Welt von Händlern benutzt. Ein elektronischer Taschenrechner erfüllt heute oft ähnliche Aufgaben. Kaum jemand käme auf die Idee, in der Verwendung des Taschenrechners einen Wettbewerb zwischen Mensch und Maschine zu sehen, führt doch dieser nur gedankenlos aus, was ihm durch Konstruktion vorgegeben wurde. 2/6/1

Die Idee, maschinell auch mit abstrakten Größen zu rechnen, Formeln zu manipulieren, in denen Unbestimmte auftreten, ist viel älter als die technische Möglichkeit Geräte zu bauen, die das auch wirklich leisten. Im 19. Jahrhundert musste die „analytic engine" von Charles Babbage (vgl. z.B. [Deck][2]) eine Vision bleiben. Heute sind Computer, die solche Aufgaben übernehmen, aus der Arbeit des Mathematikers nicht mehr hinwegzudenken.

Dieser Abschnitt ist gewissermaßen eine Belohnung für Leser, die zuvor erläuterte Algorithmen zur Lösung von Gleichungssystemen so weit verin-

[1] *Mitteilungen der Deutschen Mathematiker-Vereinigung*, 2/2000
[2] *Some Introductory Remarks on Computer Algebra*, in Casacuberta, Carles (ed.) et al., 3rd European congress of mathematics (ECM), Barcelona, July 10-14, 2000. Volume II. Birkhäuser, Basel [u.a.] Prog. Math. 202, 121-142 (2001)

nerlicht haben, dass es an der Zeit ist, Routineaufgaben „rechnen zu lassen“. Während in einfachen Fällen der Aufwand für das Eingeben der Ausgangs-Daten noch unverhältnismäßig groß sein mag und eine Rechnung „mit der Hand“ vorzuziehen ist, wird beim Lösen insbesondere von Gleichungssystemen höheren Grades die Grenze des Zumutbaren schnell erreicht. Dann ist der Computer oft ein unverzichtbares Hilfsmittel; wir sehen uns nach geeigneter Software um.

Welches System?

Die Frage ist nicht leicht zu beantworten, da im konkreten Fall sowohl mathematische als auch außer-mathematische Gründe eine Rolle spielen werden.

Machen Sie sich mit den Internetseiten einiger Computeralgebrasysteme vertraut!

Zunächst sollten wir uns von der Vorstellung verabschieden, es ginge um eine Art Taschenrechner, der für jedes unserer Probleme einen Knopf vorsieht, mit dem es gelöst wird. Natürlich wird jedes System eine Reihe abrufbarer Funktionen anbieten, und es ist nahe liegend, dies bei der Auswahl zu berücksichtigen. Für komplexe Anwendungen sind Aufgaben in der jeweiligen Skriptsprache zu formulieren, so dass ein Teil der gewünschten Funktionalität im Allgemeinen durch den Nutzer selbst zu schaffen ist.

Hier folgt eine alphabetisch geordnete Auswahl einiger Computeralgebrasysteme (CAS) mit ihren aktuellen Internet-Adressen.

Axiom	`http://www.nag.co.uk/symbolic_software.asp`
CoCoA	`http://cocoa.dima.unige.it/`
GAP	`http://www.gap-system.org/`
Macaulay 2	`http://www.math.uiuc.edu/Macaulay2/`
Magma	`http://magma.maths.usyd.edu.au/magma/`
Maple	`http://www.maplesoft.com/`
MAS	`http://www.fmi.uni-passau.de/algebra/`
Mathematica	`http://www.wri.com/`
MuPAD	`http://www.mupad.de/`
Reduce	`http://www.uni-koeln.de/REDUCE/`
Singular	`http://www.singular.uni-kl.de/`

Nachdem (ohne Anspruch auf Vollständigkeit) diese kleine Liste zusammengestellt ist, haben wir die Qual der Wahl:

„General Purpose“ oder „Special Purpose“?

Während Systeme wie Maple, Mathematica, MuPAD erfolgreich darauf abzielen, weite Teile der Mathematik zu überdecken, weisen hochspezialisierte Systeme wie CoCoA, Singular eine Funktionalität auf, die auf die Gebiete der kommutativen Algebra und algebraischen Geometrie ausgerichtet ist, bestechen durch große Leistungsfähigkeit und verfügen über Skriptsprachen, die den konkreten Aufgaben ihres Gebiets besonders nahe stehen. Zentrale Algorithmen dienen der Bestimmung von Gröbnerbasen und damit der Auflösung von Gleichungssystemen. Das hat ansatzweise dazu geführt, dass durch Aufruf aus anderen Systemen heraus die Nutzung solcher *Special-Purpose-Systeme* auch für andere Gebiete der Mathematik möglich ist.

„Open Source“?

Wir befinden uns erst am Anfang einer Entwicklung, in deren Verlauf zunehmend anspruchsvolle Sätze der theoretischen Mathematik mit Unterstützung durch Computer bewiesen werden, beispielsweise durch das Abarbeiten einer endlichen, großen Anzahl von Spezialfällen, wenn diese für eine allgemeine Verifikation zunächst auszuschließen sind. Es stellt sich die Frage, inwiefern ein solcher Satz aus mathematischer Sicht wirklich als bewiesen angesehen werden kann. Geringe Akzeptanz dürfte ein Beweis finden, der auf der Korrektheit des nicht-publizierten Quellcodes eines CAS beruht.
Für offene Quellcodes spricht nicht zuletzt auch, dass bei einem System mit einer breiten Gemeinschaft von Anwendern gute Aussicht auf intensive Weiterentwicklung der Algorithmen und Korrektur möglicher Fehler besteht.
Durch Autoren werden allerdings durchaus nicht immer die verwendeten Systeme zitiert oder Skripts angegeben, mit denen ein Resultat erhalten wurde. Das muss kein Ausdruck von Geringschätzung sein – es mag oft nicht leicht fallen zu entscheiden, was eigentlich das Wort „trivial“ bedeuten soll. Wann wäre ein Ergebnis „trivialerweise falsch“?

Kommunikation mit anderen Systemen?

Dem Daten-Tausch zwischen verschiedenen Systemen dürfte künftig größere Bedeutung zukommen, da nicht zu erwarten ist, dass alle anderen von einem einzigen verdrängt werden. Obwohl es erfolgversprechende Ansätze gibt, wird noch einige Zeit vergehen, bis verbindliche Standards existieren. Der Anwender wird kaum davor bewahrt bleiben, sich mit verschiedenen Systemen vertraut zu machen: die Stärken und Schwächen ihrer Algorithmen kennen zu lernen, die entsprechende Skriptsprache zu beherrschen, sich mit gelegentlich wechselnder Syntax verschiedener Versionen desselben Systems auseinander zu setzen oder mit Beschränkungen vertraut zu machen, die bestimmten Objekten auferlegt werden (wie groß darf z.B. ein Index oder ein Exponent sein?).

Dokumentation?

Dazu gibt es Ansätze, die den guten Willen erkennen lassen. Von Software darf erwartet werden, dass sie sich weitgehend selbst erklärt. Wer mit den entsprechenden mathematischen Begriffen vertraut ist, sollte – nach, sagen wir, einer Stunde – die grundlegenden Konzepte eines CAS anwendungsbereit kennen gelernt haben und künftig (auch ohne zusätzliche Literatur) über eine verständliche Online-Hilfe Zugang zu allen Funktionen erhalten. Darüber hinaus kann erwartet werden, dass mathematische Begriffe und Bezeichnungen, die nicht Allgemeingut sind oder unterschiedlich verwendet werden, ebenfalls in einer solchen Dokumentation erklärt sind.
Dann bleibt es der eigenen Entscheidung überlassen, nach dem Studium weiterführender Literatur oder (mit manchmal besserem Lerneffekt) durch eigene Überlegung Prozeduren zu entwickeln, die das bearbeitete Problem lösen.

„Copyright“ oder „Copy left“?

„Free Software is a matter of free speech, not of free beer.“ (Richard Stallman)

Besonders Nutzer-freundlich ist die GNU General Public Licence der Free Software Foundation (bekannt auch im Zusammenhang mit dem Betriebssystem Linux von Linus Torvalds), durch die Weitergabe und sogar Weiterentwicklung sehr großzügig geregelt sind. Wenn ein System unter diesen Bedingungen verwendet werden darf, kann in der Regel keine Unterstützung bei der Installation verlangt werden, obwohl in der Praxis oft Hilfe durch Autoren oder Nutzer-Gruppen gewährt wird.

Bei anderen Systemen ist die Verwendung an strenge Bedingungen geknüpft, insbesondere an den Erwerb einer Lizenz, für die recht hohe Kosten entstehen können; es gibt eine Vielzahl von Varianten.
Da sich Copyright-Bedingungen ändern können (z.B. in Abhängigkeit von der Versions-Nr.), sind hier gemachte Angaben im Einzelfall zu prüfen.

Nun wird in zwei konkreten Fällen angedeutet, wie die ersten Schritte mit einem CAS aussehen können. Keinesfalls wird auf den gesamten Funktionsumfang eingegangen. Vielmehr soll an ausgewählten Beispielen das Rechnen mit bereits vertrauten Begriffen erläutert und eine Grundlage für die selbstständige Aneignung weiterer Funktionen geschaffen werden.

Das Computeralgebrasystem SINGULAR[3] 2/6/2

Vorausgesetzt wird, dass das genannte System (das in der Version 2.04 zur Verfügung stand) auf unserem Rechner installiert ist. Eine freie Kopie mit Installationsanleitung und Dokumentation ist unter der zuvor angegebenen www-Adresse erhältlich.
Der Aufruf von SINGULAR unter Verwendung eines Dateinamens als Parameter führt dazu, dass die in der Datei aufgeführten Befehle der Reihe nach abgearbeitet werden. Bei Eingabe von Befehlen über die interaktive Shell werden diese ausgeführt und ihre Resultate auf dem Bildschirm ausgegeben.

```
int n=7;
```

definiert eine ganze Zahl `n`, der der Wert `7` zugewiesen ist. Mit dem Befehl

```
n;
```

wird der Wert von `n` (nach Betätigen der Eingabetaste) angezeigt; alternativ dazu können wir `n` in eine Datei schreiben, z.B. durch

```
write("Resultat",n);
```

in die Datei mit dem Namen „Resultat“.

```
help int;
```

ruft die Online-Hilfe zum Befehl „int“ auf. Dort sind auch Beispiele und präzise Angaben zur Syntax enthalten, so dass wir uns hier kurz fassen können.
Für die meisten Rechnungen ist ein Grundkörper erforderlich, der stets zusammen mit einem Polynomring über diesem eingeführt wird. Wir rechnen vorzugsweise über dem Primkörper (der bei Bedarf erweitert wird – beispielsweise dann, wenn nicht alle Koeffizienten der gegebenen Polynome darin enthalten sind).

```
ring r=5,X,lp;
```

definiert den Polynomring $\mathbb{F}_5[X]$ mit der lexikographischen Ordnung `lp`; die Primzahl `5` legt die Charakteristik des Grundkörpers fest. Ein Polynom lässt sich nun z.B. durch

```
poly f=X99-1;
```

definieren; dies ist $\texttt{X}^{99}-1$, wobei für indizierte Variable stets der Exponent durch das Zeichen „`^`“ abzutrennen ist, im Zweifelsfall wird also

```
poly f=X^99-1;
```

eingegeben. Offensichtlich gilt `f(1)=0`, wir sehen das auch unter Verwendung des Substitutionsbefehls

[3] Greuel, G.-M., Pfister, G., Schönemann, H., SINGULAR *Reference Manual*. Reports On Computer Algebra 12, May 1997, Center for Computer Algebra, Universität Kaiserslautern

```
subst(f,X,1);
```

Einfache Operationen mit Polynomen werden auf nahe liegende Weise ausgeführt, z.B. durch

```
poly g=2*X+3;
f+g; f*g; 2*f-g; g^12;
```

Wir fragen nach der Faktorzerlegung für das zuvor eingegebene Polynom f. Als Teiler kommen nur Polynome vom Grad ≤ 99 infrage. Da der Grundkörper endlich ist, ergibt sich durch Unterscheidung einer endlichen Zahl von Fällen die Zerlegung in irreduzible Faktoren. Die Idee aus Beispiel 3.* in 2/4/10 ergibt auch über dem Körper der rationalen Zahlen eine prinzipielle Möglichkeit der Faktorzerlegung. Nun soll nicht behauptet werden, dass moderne CAS auf diese Weise rechnen - das Auffinden schneller Algorithmen zur Faktorisierung erfordert bereits richtig interessante Mathematik. SINGULAR liefert das Resultat mit dem Befehl

```
factorize(f);
```

Dabei wird zuerst eine Liste der irreduziblen Faktoren und weiter die Liste ihrer Vielfachheiten ausgegeben (wobei der konstante Faktor am Anfang in unserem Sinn zu ignorieren ist).

Der $\mathbb{Q}$-Algebrahomomorphismus `h:S→R`, `S:=`$\mathbb{Q}$`[Y,Z]`, `R=` $\mathbb{Q}$`[X]` mit `h(Y)= X-1` und `h(Z)=`X^2 lässt sich so einführen:

```
ring S=0,(Y,Z),lp; ring R=0,X,lp;
map h=S,X-1,X2;
```

Damit ergibt sich nach Wahl des jeweils aktuellen Ringes

```
setring S; poly g=YZ; setring R;
h(g);
```

das erwartete Resultat `h(g)=`X^3-X^2. ❏

Für Matrizen-Operationen stehen umfangreiche Hilfsmittel zur Verfügung. Zunächst definieren wir die Matrix

$$\mathtt{A} = \begin{pmatrix} \mathtt{X}^2 & 2\mathtt{X} & 3 \\ 4 & 5 & 6 \end{pmatrix} \in \mathrm{M}(2,3;\mathbb{Q}[\mathtt{X}])$$

durch den SINGULAR-Befehl

```
matrix A[2][3]=X2,2*X,3,4,5,6;
```

(wobei die Einträge fortlaufend zeilenweise einzugeben sind). Eine weitere Matrix erklären wir durch

```
matrix B[2][2]=1,1,1,2;
```

und erhalten das erwartete Resultat für das Produkt mit

```
A*B;
```

oder in gefälligerer Form mittels

```
print(A*B);
```

Manche Funktionen erfordern das Laden zusätzlicher Bibliotheken, beispielsweise zur linearen Algebra, durch

```
LIB "linalg.lib";
```

Für die invertierbare Matrix `B` erhalten wir nun $\mathtt{B}^{-1}$ mittels

```
print(inverse(B));
```
❏

Natürlich erwarten wir, dass auch das Rechnen mit Näherungen möglich ist. Dafür werden Datentypen eingeführt, die eine gewisse Ähnlichkeit mit Polynomalgebren über den reellen Zahlen aufweisen. So wird durch die Vereinbarung

2/6/3

```
ring r=(real,30),U,lp;
```

das Rechnen mit Zahlen in 30-stelliger Gleitkommadarstellung möglich, wobei intern nochmals dieselbe Stellenzahl hinzugenommen wird, um (möglichst) die Stabilität der Operationen zu gewährleisten. Obwohl Rechnungen dieser Art nicht exakt sind und in die Irre führen können, lässt sich bei sorgfältiger Fehlerabschätzung meist ein brauchbares Resultat gewinnen.

Vgl. auch das erste Beispiel in 2/3/2.

Beispiel. (*Iterationsverfahren für lineare Gleichungssysteme*)

$(*)$ `A x = b` mit $\mathtt{A} \in \mathrm{M}(n;\mathbb{R})$, $\mathtt{b} \in \mathrm{M}(n,1;\mathbb{R})$

sei ein lineares Gleichungssystem mit einer regulären Koeffizientenmatrix `A`. Wir wissen bereits, dass dann eine eindeutig bestimmte Lösung $x \in \mathrm{M}(n,1;\mathbb{R})$ existiert. Ist die Zahl n groß, dann kann das System aufgrund des hohen Rechenaufwands und zahlreicher Rundungsfehler „praktisch unlösbar“ werden. Daher ist es ratsam so zu rechnen, dass nicht zu viele Zwischenschritte erforderlich sind und eventuelle Fehler sich möglichst automatisch korrigieren; die folgende Methode kann dabei unter Umständen helfen.

Es sei $\mathtt{A} = \mathtt{B} + \mathtt{C}$ mit einer ebenfalls regulären Matrix $\mathtt{B} \in \mathrm{M}(n;\mathbb{R})$, die jedoch „leichter“ zu invertieren ist als die ursprüngliche Matrix `A`. Dann ist die Bedingung $(*)$ offensichtlich äquivalent zum System

Idee: Nehmen wir an, wir hätten eine konvergente Folge (x_i) mit $x_{i+1} = f(x_i)$.
Dann folgt nach Wahl von f zunächst $x_{i+1} = -B^{-1}Cx_i + B^{-1}b$.
Sofern die Matrizen-Operationen mit der Grenzwertbildung für $i \to \infty$ vertauschbar sind, ergibt die letztere Gleichung mit dem Grenzwert $x = \lim_{i\to\infty} x_i$ offenbar $x = f(x)$.

$\binom{*}{*}$ $\mathtt{x} = -\mathtt{B}^{-1}\mathtt{C}\mathtt{x} + \mathtt{B}^{-1}\mathtt{b}$,

das zunächst viel schlechter aussieht als das erste. Wir betrachten die Abbildung

$$\mathtt{f} : \mathrm{M}(n,1;\mathbb{R}) \to \mathrm{M}(n,1;\mathbb{R}),\ \ \mathtt{x} \mapsto -\mathtt{B}^{-1}\mathtt{C}\mathtt{x} + \mathtt{B}^{-1}\mathtt{b}.$$

`x` ist genau dann eine Lösung von $\binom{*}{*}$, wenn $\mathtt{f}(\mathtt{x}) = \mathtt{x}$. Unter einer geeigneten Beschränktheitsbedingung für $\mathtt{B}^{-1}\mathtt{C}$ folgt aus einem Satz der Analysis (*banachscher Fixpunktsatz*), dass jede Folge $(\mathtt{x}_i)_{i\in\mathbb{N}}$, $\mathtt{x}_i \in \mathrm{M}(n,1;\mathbb{R})$, für die $\mathtt{f}(\mathtt{x}_i) = \mathtt{x}_{i+1}$ ist, gegen die eindeutig bestimmte Lösung `x` des gegebenen Systems konvergiert. Wir werden das hier nicht weiter erläutern, sondern begnügen uns mit einem numerischen Experiment, das durch das Auffinden von `x` mit $\mathtt{f}(\mathtt{x}) = \mathtt{x}$ gerechtfertigt wird. Wir wählen

$$\mathtt{A} = \begin{pmatrix} 1 & 0 & 0.1 \\ 0 & 1 & 0.1 \\ 0.1 & 0.1 & 1 \end{pmatrix},\ \ \mathtt{b} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix};$$

dies entspricht der Eingabe

```
matrix A[3][3]=1,0,0.1,0,1,0.1,0.1,0.1,1;
matrix b[3][1]=1,0,1;
```

Nun wird eine reguläre Matrix `B` so gewählt, dass die Differenz $\mathtt{C} = \mathtt{A} - \mathtt{B}$ „klein“ ist; für die Einheitsmatrix $\mathtt{B} = \mathrm{E}_3$ scheint dies plausibel:

```
matrix B[3][3]; B=B+1;
```

(SINGULAR-Definition einer Diagonalmatrix, deren Diagonalelemente 1 sind).

```
matrix C[3][3]=A-B;
```

Zur Bequemlichkeit beschreiben wir die oben definierte Abbildung `f` durch eine Prozedur

```
proc f (x) {return(-C*x+b);}
```

und definieren `x` als

```
matrix x[3][1];
```

Ersetzen des Anfangswerts von `x` (hier ist das die Nullspalte) durch `f(x)` drückt sich im Befehl

```
x=f(x);
```

aus. In der Hoffnung, dass die nach diesem Muster gebildete Folge konvergiert, lassen wir die Substitution wiederholen, solange `x` und `f(x)` verschieden sind,

```
while (x!=f(x)) {x=f(x);}
```

(`!=` bedeutet $\neq$). Was geschieht hier? Zunächst wird geprüft, ob $\mathtt{x} \neq \mathtt{f(x)}$. Ist dies der Fall, so wird `x` durch `f(x)` ersetzt und der erste Schritt wiederholt; anderenfalls ist die Rechnung beendet.
Das schnell erhaltene Resultat lässt sich durch

```
print(x);
```

anzeigen; es ist $\mathtt{x} = \begin{pmatrix} 0.90816326530612244897959183673 5 \\ -0.09183673469387755102040816326 53 \\ 0.09183673469387755102040816326 53 \end{pmatrix}$.

Wer Lust dazu hat, kann leicht mit dem gaußschen Algorithmus die exakte Lösung des Systems bestimmen und sich davon überzeugen, dass sie bis auf die angegebene Stellenzahl mit x übereinstimmt.

Dieses Verfahren funktioniert in vielen Fällen. Ist die Matrix `B` in $\binom{*}{*}$ eine Diagonalmatrix, so wird die hier erläuterte Methode als *Jacobi-Verfahren* bezeichnet, bei Wahl einer unteren Dreiecksmatrix für `B` als *Gauß-Seidel-Verfahren*. ❑

Der zentrale Algorithmus in SINGULAR dient der Bestimmung von *Standardbasen* für Ideale in verschiedenen Typen von Ringen; wir haben sie für Polynomringe unter dem Namen Gröbnerbasen kennen gelernt. Die Syntax ist einfach. 2/6/4

```
ring r=0,(X1,X2,X3),lp;

ideal i=X1^5+X2^5-X3^5,X1^7+X2^7-X3^7;

option(redSB); std(i);
```

Wir erhalten eine reduzierte Gröbnerbasis aus 21 Polynomen. Auch die Entscheidung, ob das Polynom $\mathtt{f} = \mathtt{X}_1^9 + \mathtt{X}_2^9 - \mathtt{X}_3^9$ in dem gegebenen Ideal liegt, wird uns leicht gemacht:

```
poly f=X1^9+X2^9-X3^9; reduce(f,std(i));
```

Der so erhaltene Rest bei Division durch `std(i)` ist von Null verschieden, d.h. `f` liegt nicht in dem gegebenen Ideal.

Nun kann es interessant sein, anstelle der lexikographischen Ordnung `lp` dieselbe Rechnung mit anderen Monomordnungen auszuführen.

```
ring r=0,(X1,X2,X3),M(1,1,1,1,0,1,1,2,3);
```

definiert einen Polynomring mit der Monomordnung, die durch die reguläre Matrix

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

gegeben ist. Wie zuvor erhalten wir eine Standardbasis, die in diesem Fall aus nur 6 Polynomen besteht.

```
ideal i=X1^5+X2^5-X3^5,X1^7+X2^7-X3^7;
std(i);
```

Eine häufig verwendete Monomordnung ist die graduiert invers-lexikographische Ordnung, die in SINGULAR mit `dp` bezeichnet wird; Rechnungen mit dieser Ordnung erweisen sich oft als besonders effektiv, sie ist jedoch nicht direkt für Elimination von Unbestimmten verwendbar.
Unter Verwendung von `dp` im angegebenen Beispiel entsteht eine weitere Gröbnerbasis für das gegebene Ideal, ebenfalls aus 6 Polynomen. ❑

Das Multipurpose-System MUPAD 2/6/5

Beim Aufruf von MuPAD (das in der Version 2.5 vorlag) erhalten wir eine interaktive Shell zur Eingabe von Befehlen. Die Hilfefunktion ermöglicht den Zugang zur Syntax wichtiger Funktionen des Systems einschließlich entsprechender Beispiele, vgl. auch [MuP][4]. So sind z.B. Informationen über das Einlesen von Dateien mit

```
help("read")
```

erhältlich. Steht HyTEX zur Verfügung (etwa bei Nutzung von XMUPAD auf UNIX- bzw. LINUX-Systemen), so können Beispiele durch „Knopfdruck" direkt aus dem Hilfetext übernommen werden, wodurch Experimente mit der Syntax erleichtert werden.

Es wird kein Anspruch auf Vollständigkeit erhoben. Sie können auch `info("read")` verwenden.

Wir beginnen mit einfachen Operationen für Zahlen.

```
1+3*5
```

ergibt erwartungsgemäß die Zahl 16, durch

```
2/5+1/6
```

erhalten wir den Bruch $\frac{17}{30}$.
Bezeichner (*Identifyer*), ganze Zahlen und daraus gebildete Ausdrücke werden mit Operationen $+$, $*$ als kommutativer Ring angesehen (wobei $-$ und `^` in üblicher Weise verwendet werden können). Das bedeutet, es wird insbesondere im Polynomring $\mathbb{Z}[F]$ über den ganzen Zahlen gerechnet, wobei F die endliche Familie der vom Benutzer eingeführten Bezeichner ist. Formales Rechnen mit Symbolen und aus diesen gebildeten Ausdrücken ist so zunächst generell möglich; wir bilden etwa

```
a*b-c^2
```

und weisen `d` diesen Ausdruck zu:

```
d:=a*b-c^2
```

Die Auswertung daraus gebildeter weiterer Ausdrücke erfolgt dann in bekannter Weise automatisch, etwa bei Eingabe von

```
d^2+a
```

Das vertraute Ausgabeformat entsteht, indem mittels

[4] Creutzig, C., Gerhard, J., Oevel, W., Wehmeier, S., *Das MuPAD Tutorium. Plattformunabhängige Einführung ab Version 2.0.* Springer Berlin [u.a.] 2002

```
PRETTY_PRINT:=FALSE:
```

der Wert einer entsprechenden Umgebungsvariablen gesetzt wird.

Nun wissen wir noch nicht, welche mathematischen Objekte mit so allgemeinen Operationen verwendet werden können. MuPAD stellt das sehr allgemeine Konzept einer Datenstruktur (Domain) zur Verfügung, das über eine Reihe (bereits im Kern) fest vorgegebener Typen hinaus auch die Möglichkeit bietet, eigene Datentypen einschließlich zugehöriger Operationen zu definieren (objektorientierte Programmierung). Der interessierte Leser muss dazu auf die Dokumentation verwiesen werden (vgl. auch die unter 2/6/1 angegebene Internet-Seite).

Wir lassen uns durch die Funktion `domtype()` anzeigen, wie sich dies in den obigen Beispielen verhält.

```
domtype(a); domtype(d)
```

zeigen, dass `a` im Datenbereich `DOM_IDENT` der Bezeichner liegt, `b` dagegen im Bereich `DOM_EXPR` der Ausdrücke (*Expressions*). Zu den von MuPAD vorgegebenen Datenbereichen gehören darüber hinaus Faktorringe der ganzen Zahlen, Polynome, Matrizen, Listen, Mengen, Gruppen, Strings, ...

Wir definieren ein Polynom über dem Restklassenring $\mathbb{F}_2 = \mathbb{Z}/(2)$ der ganzen Zahlen durch

```
f:=poly(X^2*Y-X+1, [X,Y], IntMod( 2 ))
```

Dies ist offenbar ein Tripel; es besteht aus einem Ausdruck, gebildet aus der an zweiter Position angegebenen Liste `[X,Y]` von Unbestimmten und dem Grundring `IntMod( 2 )`. Fehlt die letzte Angabe, so wird der Ring aller MuPAD-Ausdrücke angenommen. Den Ausdruck von `f` (hier aus `DOM_EXPR`) können wir durch

```
expr(f)
```

zurückgewinnen, während `f` selbst zum Bereich `DOM_POLY` der Polynome gehört; wir überzeugen uns davon mit

```
domtype(f)
```

Bei Operationen ist nun etwas Vorsicht geboten. Die Eingabe

```
f+f
```

führt erwartungsgemäß auf das Nullpolynom. Multiplikation mit der Zahl 3 (entsprechend der $\mathbb{Z}$-Algebrastruktur) ist jedoch nicht mit der Eingabe `3*f` zu erreichen – dies wird (wie wir uns leicht überzeugen) lediglich als formaler Ausdruck verstanden. Das erwartete Resultat gewinnen wir mit

```
poly(3, [X,Y], IntMod( 2 ))*f
```

Die Zerlegung des Polynoms `X^99-1` in irreduzible Faktoren ist durch

```
factor(poly(X^99-1, [X]))
```

möglich.

Mit Matrizen lässt sich in vertrauter Weise rechnen. Wir definieren durch

```
A:=matrix(2, 3, [[5,1,1],[2,0,3]])
```

eine Matrix

$$\begin{pmatrix} 5 & 1 & 1 \\ 2 & 0 & 3 \end{pmatrix} \in \mathrm{M}(2;\mathbb{Q})$$

und überzeugen uns durch

```
A+A; 2*A
```

```
B:=matrix(2, 2, [[5,1],[2,0]]); B*A
```

davon, dass die angegebenen Operationen die erwarteten Resultate liefern.

Beispiel. Zum Vergleich soll das bereits unter 2/6/3 verwendete Iterationsverfahren mit MuPAD ausgeführt werden. Natürlich wird nicht behauptet, dass es im angegebenen Fall ohne Alternative wäre. 2/6/6

```
DIGITS :=30:
```

erzwingt eine Voreinstellung von 30 Nachkommastellen in der Gleitkommadarstellung. Durch

```
A:=matrix(3, 3, [[1,0,0.1],[0,1,0.1],[0.1,0.1,1]])
```

```
b:=matrix(3, 1, [1,0,1])
```

ist ein lineares Gleichungssystem mit der erweiterten Koeffizientenmatrix (`A,b`) gegeben. Wird noch

```
B:=matrix(3, 3, 1, Diagonal); C:=A-B
```

gesetzt, eine einfache Prozedur `f` mittels

```
f := x -> -C*x+b
```

erzeugt sowie für

```
x:=matrix(3, 1)
```

das aus Nullen gebildete Spaltentupel gewählt, dann ergibt sich durch

```
while x <> f(x) do
  x:=f(x);
end_while
```

der bereits früher errechnete Näherungswert für die Lösung `x` des Systems. ❑

Gröbnerbasen lassen sich unter Benutzung der MuPAD-Bibliothek `groebner` bestimmen. Unter Verwendung der Zahlen aus 2/4/6 setzen wir 2/6/7

```
PRETTYPRINT := FALSE:
```

```
f:=poly(X1^5+X2^5-X3^5, [X1,X2,X3])
```

```
g:=poly(X1^7+X2^7-X3^7, [X1,X2,X3])
```

und bestimmen die reduzierte Gröbnerbasis in der graduiert invers-lexikographischen Ordnung mittels

```
groebner::gbasis([f,g])
```

Die reduzierte Gröbnerbasis in der lexikographischen Ordnung erhalten wir durch

```
Basis:=groebner::gbasis([f,g], LexOrder)
```

Wird nun

```
g:=poly(X1^9+X2^9-X3^9, [X1,X2,X3])
```

gesetzt, so erhalten wir durch

```
groebner::normalf(h, Basis, LexOrder)
```

den Rest bei Division durch die Gröbnerbasis `Basis` (vgl. 2/6/4).
Für die Matrixordnung mit

```
A:=matrix(3, 3, [[1,1,1],[1,0,1],[1,2,3]])
```

setzen wir

```
Ord:=Dom::MonomOrdering(Matrix(A))
```

und erhalten mittels

```
groebner::gbasis([f,g], Ord)
```

die (bis auf Reihenfolge) schon unter 2/6/4 gefundenen Erzeugenden des Ideals (f, g). ❑

Der vorliegende Abschnitt sollte eine Anregung sein, bei der weiteren Lektüre auch die Möglichkeiten der Nutzung eines CAS in Betracht zu ziehen. Der Umgang mit dem Computer kann das selbstständige Rechnen nicht ersetzen, sondern nur ergänzen; vom Leser wird erwartet, dass er (bis auf ausdrücklichen Hinweis) in der Lage ist, auch die weiteren Übungsaufgaben ohne technische Mittel zu bearbeiten. Nach den vorherigen Erläuterungen sollte es jedoch möglich sein, nahezu alle angegebenen Rechenaufgaben durch ein entsprechendes Skript zu lösen. Der Vergleich mit einem zuvor ohne Hilfsmittel erhaltenen Resultat kann dann vorteilhaft sein und bietet ein nützliches Training im Umgang mit dem Computer.

Schwerpunkte zum gewählten Stoff

- Machen Sie sich mit den Internet-Seiten einiger Computeralgebrasysteme vertraut [2/6/1]
- Rechnen mit dem Computeralgebra-System SINGULAR [2/6/2 – 2/6/4]
- Rechnen mit dem Multipurpose-System MUPAD [2/6/5 – 2/6/6]

Sachverzeichnis