SINGULAR Version 1.3.5
Universität Kaiserslautern
Fachbereich Mathematik und Zentrum für Computeralgebra
Autoren: G.–M. Greuel, G. Pfister, H. Schönemann

KURZEINFÜHRUNG IN SINGULAR

THOMAS KEILEN

INHALTSVERZEICHNIS

Die vorliegende Kurzeinführung in das Computeralgebrasystem SINGULAR erhebt keinerlei Anspruch auf Vollständigkeit. Ziel ist es, die Teilnehmer der Übungen zu den Vorlesungen *Lineare Algebra* und *Einführung in die Computeralgebra* schrittweise mit den notwendigen Begriffen und Befehlen vertraut zu machen, die notwendig sind, das Programm im Rahmen der Übungen einzusetzen. Dies bedingt insbesondere, daß der Einführung eine strikte Systematik fehlt. Wir verweisen deshalb für eine systematische und vollständige Dokumentation von SINGULAR auf das Handbuch [GPS99].

Zwei Gründe haben uns bewogen, SINGULAR auch in der Linearen Algebra zur Lösung kleinerer Probleme einzusetzen: zum einen ist das Programm SINGULAR kostenlos und für alle gängigen Computerplattformen erhältlich, zum anderen ist die Programmiersprache von SINGULAR der Sprache C angelehnt, was im weiteren Verlauf des Studiums von Vorteil sein mag. Wer das Programm SINGULAR auf dem eigenen Rechner installieren möchte, findet die Sourcen sowie Installationshinweise über die Singular Home Page:

`http://www.mathematik.uni-kl.de/~zca/Singular/`

1. Erste Schritte

1.1. **Notationen.** In der vorliegenden Einführung wollen wir uns auf folgende Notationen einigen:

- SINGULAR-Ein- und Ausgaben sowie reservierte Worte werden in der Schriftart Typewriter gesetzt, etwa `exit;` oder `help`.
- Das Symbol $\mapsto$ leitet SINGULAR-Ausgaben ein, z. B.:
  ```
  int i=5;
  i;
  ↦ 5
  ```
- Eckige Klammern bezeichnen Teile der Syntax, die optional sind, also fehlen können. Z. B.
  ```
  pmat(M,[n]);
  ```
  Der obigeBefehl, eine Prozedur der Bibliothek `matrix.lib` dient zur Ausgabe einer Matrix `M` als formatierte Matrix. Der optionale Parameter `n` gibt die Breite der Spalten in Zeichen an. Wenn er fehlt, wird ein Standardwert genommen.
- Tasten werden ebenfalls durch die Schriftart Typewriter gekennzeichnet, etwa:
  `n` (drücke die Taste `n`),
  `RETURN` (drücke die Eingabetaste),
  `CTRL-P` (drücke die Control-Taste und die Taste `P` zugleich).

1.2. **Singular aufrufen und beenden.** Es versteht sich von selbst, daß die erste Frage die ist, wie man das Programm startet und wie man es wieder beendet. Auf den am Fachbereich vorhanden Rechnern ist die jeweils aktuellste Version von SINGULAR durch Eingabe des Kommandos `Singular` auf der Kommandozeile des Systems zu starten. Die Eingabe von `Singular -v` sorgt dafür, daß beim Start des Programms die Versionsnummer von SINGULAR angezeigt wird.

Nach dem Start liefert SINGULAR einen Eingabeprompt, ein >, zurück und steht dem Nutzer fortan zur interaktiven Nutzung zur Verfügung. Sobald selbiger Nutzer

von dieser Möglichkeit nicht länger Gebrauch machen möchte, empfiehlt es sich, das Programm zu beenden. Hierzu stehen ihm drei Befehle zur Verfügung: `exit;`, `quit;` oder, für die ganz Schreibfaulen, `$`.

Man beachte hierbei, daß die Semikola im letzten Satz keine deplazierten Satzzeichen darstellen, sondern zu den SINGULAR-Befehlen gehören.

Generell schließt *jeder* Befehl in Singular durch ein Semikolon ab!

Das Semikolon teilt dem Rechner mit, er möge die just eingegebene Befehlfolge doch bitte *interpretieren* und, sollte er dabei erfolgreich sein, auch *ausführen*. Das Programm meldet sich entsprechend mit dem Ergebnis (bzw. einer Fehlermeldung), gefolgt von einem neuen Eingabeprompt wieder. Sollte man das Semikolon vergessen haben oder eine geschweifte Klammer geöffnet und nicht wieder geschlossen haben, so zeigt einem Singular dies dadurch an, daß als Eingabeprompt ein `.`, sprich ein Punkt, erscheint und die Möglichkeit zu weiteren Eingaben, etwa dem fehlenden Semikolon, gibt. Auf diese Weise besteht die Möglichkeit, längere Befehlsfolgen über mehrere Zeilen zu strecken.

1.3. **Die Online-Hilfe `help`.** Neben dem Starten und Beenden des Programms ist die nächstwichtigste Information die, wie man sich Hilfe besorgt, wenn man feststeckt. Hierzu steht in Singular der Befehl `help`, oder kurz `?`, zur Verfügung. Gibt man den Befehl `help` gefolgt von einem SINGULAR-Befehl, einem SINGULAR-Funktions-/Prozedurnamen oder eine SINGULAR-Bibliothek, so werden Informationen zum jeweiligen Objekt angezeigt. Bei Bibliotheken erhält man eine Auflistung der darin enthaltenen Prozeduren, bei Befehlen, Funktionen und Prozeduren erfährt man ihren Zweck und findet die allgemeine Syntax sowie, ganz wichtig, Beispiele für ihre Anwendung.

Beispiele:

```
help exit;
help standard.lib;
help printf;
```

Man kann sich die Hilfe auf verschiedenen Ausgabemedien, Browsern, anzeigen lassen. Standardmäßig wird dies bei SINGULAR 1.3.5 Netscape sein. Das heißt, daß SINGULAR nach Eingabe etwa von `help exit;` Netscape startet und den zu `exit;` gehörenden Hilfetext dort anzeigt. (Über selbsterklärende Buttons steht damit das ganze Handbuch zur Verfügung.) Neben Netscape stehen noch weitere Browser zur Verfügung, von denen hier nur `info` und `builtin` genannt sein sollen. Ersterer dürfte den Nutzern von Unix-Systemen vertraut sein, letzterer zeigt den Hilfetext schlicht auf der laufenden SINGULAR-Seite an und hat den Vorteil, auf allen Computerplattformen und ohne zusätzliche Programme (wie Netscape oder Info) zu funktionieren.

Mittels des Befehls `system("browsers");` erfährt man, welche Browser SINGULAR kennt, und durch `system("--browser","builtin");` wechselt man den Browser von Netscape zu `builtin` – für andere Browser entsprechend. Ferner besteht die Möglichkeit, bereits beim Start von SINGULAR einen Browser zu wählen, indem das Programm etwa durch den Befehl `Singular --browser=builtin` gestartet wird.

Während die Bedienung der Netscape-Hilfe selbsterklärend ist, benötigt man Hinweise, wenn man mit Info noch nicht gearbeitet hat. Wer nicht mit Info arbeitet, kann zu 1.6 gehen. Um sich innerhalb von Info fortzubewegen verwende man die unten aufgeführten Kommandos, die alle aus einzelnen Buchstaben bestehen. Beachte, man benutze *niemals* `RETURN` oder die Pfeiltasten! Einige Befehle lesen anschließend

weitere Eingaben von der Kommandozeile am unteren Rand des Bildschirms. Hierbei steht die `TAB` Taste zur Vervollständigung eines teilweise eingegebenen Befehls zur Verfügung.

Einige wichtige Info-Kommandos:

`q`	Verlassen der Online-Hilfe
`n`	Vorwärtsblättern zum nächsten Menüpunkt
`p`	Rückwärtsblättern zum vorhergehenden Menüpunktunkt
`m`	Auswählen eines durch Namen spezifizieren Menüpunktes
`f`	Aufrufen eines Querverweises
`l`	Aufrufen des zuletzt besuchten Menüpunktes
`b`	Zurückblättern zum Beginn des Menüpunktes
`e`	Vorwärtsblättern zum Ende des Menüpunktes
`SPACE`	Vorwärtsscrollen um eine Seite
`DEL`	Zurückscrollen um eine Seite
`h`	Aufrufen der Info-Einführung
`CTRL-H`	Aufruf eines Kurzüberblicks über die Online-Hilfe
`s`	Durchsuchen des Handbuches nach einem bestimmten String
`1,...,9`	Aufrufen des i-ten Unterpunktes eines Menüs

1.4. **Singular unterbrechen.** Unter Unix-ähnlichen Systemen und unter Windows NT besteht die Möglichkeit, SINGULAR durch die Tastenkombination `CTRL-C` zur Unterbrechung seiner Tätigkeit zu bewegen. (Funktioniert nicht bei `ESingular`!) SINGULAR reagiert durch Ausgabe des derzeit auszuführenden Befehls und erwartet weitere Anweisungen. Hier stehen folgende Optionen zur Auswahl:

`a`	SINGULAR führt den aktuellen Befehl noch aus und kehrt dann zum Toplevel zurück,
`c`	SINGULAR fährt fort,
`q`	das Programm SINGULAR wird beendet.

1.5. **Eingaben editieren.** Wer sich bei einem Kommando einmal verschrieben hat, oder ein früheres Kommando noch einmal benötigt, der muß sich nicht unbedingt die Mühe machen, alles noch einmal neu einzugeben. Vorhandener SINGULAR-Text kann editiert werden. Hierzu unterhält SINGULAR eine History aller Befehle einer SINGULAR-Sitzung. Wir wollen nur eine Auswahl der zur Verfügung stehenden Tastenkombinationen zum Editieren des Textes geben:

`TAB`	automatische Vervollständigung von Funktions- und Dateinamen
←	
`CTRL-B`	bewegt den Cursor nach links
→	
`CTRL-F`	bewegt den Cursor nach rechts
`CTRL-A`	bewegt den Cursor zum Zeilenanfang
`CTRL-E`	bewegt den Cursor zum Zeilenende
`CTRL-D`	löscht das Zeichen unter dem Cursor - nie auf leere Zeile anwenden!
`BACKSPACE`	
`DEL`	
`CTRL-H`	löscht das Zeichen vor dem Cursor
`CTRL-K`	löscht alles vom Cursor bis zum Zeilenende
`CTRL-U`	löscht alles vom Cursor bis zum Zeilenanfang
↓	
`CTRL-N`	liefert die nächste Zeile aus der History

↑	
`CTRL-P`	liefert die vorherige Zeile aus der History
`RETURN`	schickt die gegenwärtig Zeile zum SINGULAR-Parser

1.6. **Prozeduren.** Hat man ein konkretes Problem, das mit SINGULAR zu lösen ist, ruft man das Programm auf, gibt die Befehlsfolge ein und erhält ein Ergebnis. Häufig möchte man gleiche Rechnungen aber mit unterschiedlichen Eingaben durchführen. Dann ist es sinnvoll, die Befehlsfolge als Prozedur zu schreiben, der man die gewünschten Eingaben als Argumente übergibt und die die Lösungen zurückgibt.

Die Syntax einer Prozedur ist recht einfach:

```
proc PROZEDURNAME [PARAMETERLISTE]
{
  PROZEDURKÖRPER
}
```

Als `PROZEDURNAME` kommt jede noch nicht anderweitig vergebene Buchstabenfolge in Frage. In der `PARAMETERLISTE` sind die Typen und Namen der Argumente, die der Prozedur übergeben werden, festgelegt. Dabei ist die `PARAMETERLISTE` durch runde Klammern einzuschließen. Der `PROZEDURKÖRPER` enthält eine Abfolge von zulässigem SINGULAR-Code. Soll die Prozedur ein Ergebnis zurückliefern, so sollte dieses Ergebnis in einer Variablen `ergebnis` gespeichert werden und die Prozedur sollte mit dem Befehl `return(ergebnis);` enden.

Ein Beispiel sagt meist mehr als tausend Worte:

```
proc permcol (matrix A, int c1, int c2)
{
  matrix B=A;
  B[1..nrows(B),c1]=A[1..nrows(A),c2];
  B[1..nrows(B),c2]=A[1..nrows(A),c1];
  return(B);
}
```

Die Prozedur `permcol` soll zwei Spalten einer Matrix vertauschen. Hierfür erwartet sie drei Argumente. Das erste Argument erhält den Namen `A` und ist vom Typ `matrix`, die beiden folgenden heißen `c1` und `c2` und sind vom Typ `integer`. Es folgen SINGULAR-Anweisungen und das Ergebnis wird in der Variablen `B` vom Typ `matrix` gespeichert, die dann mit `return(B);` zurückgegeben wird. Das bedeutet insbesondere, daß das Ergebnis der Prozedur vom Typ `matrix` ist (siehe Abschnitt 2).

Eine Prozedur wird aufgerufen, indem man den Prozedurnamen, gefolgt von den Argumenten in runden Klammern eingibt. Z. B.

```
LIB "matrix.lib"; LIB "inout.lib"; ring r=0,(x),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
pmat(A,2);
↦ 1  2  3
   4  5  6
   7  8  9
matrix B=permcol(A,2,3);
pmat(B,2);
↦ 1  3  2
   4  6  5
   7  9  8
```

Variablen, die innerhalb einer Prozedur definiert werden, sind nur dort bekannt, und können deshalb durchaus die gleichen Namen haben, wie Objekte, die außerhalb der Prozedur definiert sind.

1.7. **Bibliotheken.** Um Prozeduren für mehr als eine SINGULAR-Sitzung verfügbar zu machen, ist es sinnvoll, sie in Dateien abzuspeichern, die später von Singular wieder eingelesen werden können – sogenannten Bibliotheken (= Libraries). Die Namen der Bibliotheken lassen meist Rückschlüsse auf die enthaltenen Prozeduren zu, und tragen grundsätzlich die Endung `.lib`. Bibliotheken werden in SINGULAR eingelesen durch den Befehl `LIB` gefolgt von dem in Anführungszeichen " eingeschachtelten Bibliotheksnamen, etwa

```
LIB "KeilenT.lib";
```

(Bibliotheksnamen sollten nach Möglichkeit nur *acht* Zeichen lang sein, um die Kompatibilität mit Betriebssystemen wie Dos zu gewährleisten!) Sofern es sich nicht um SINGULAR-eigene Bibliotheken handelt, sollten sie sich in dem Verzeichnis befinden, von dem aus SINGULAR gestartet wurde.

Jeder Übungsteilnehmer sollte für die Prozeduren, die im Laufe des Semesters zu schreiben sind, eine Bibliothek anlegen – vorzugsweise unter dem eingenen Namen, etwa `KeilenT.lib`.

Natürlich muß auch eine Bibliothek gewissen Syntaxregeln entsprechen, und Prozeduren, die in Bibliotheken abgespeichert werden, sollten um zwei erläuternde Zusätze erweitert werden. Wir verdeutlichen dies an einem Musterbeispiel.

```
////////////////////////////////////////////////////////////////////
version="1.0";

info="
     LIBRARY:     KeilenT.lib LOESUNGEN ZU DEN LA-UEBUNGEN

     AUTHOR:      Thomas Keilen, email:  keilen@mathematik.uni-kl.de

     SEE ALSO:    matrix.lib, linalg.lib

     KEYWORDS:    Lineare Algebra, Gauß-Algorithmus

     PROCEDURES:
       permcol(matrix,int,int)   vertauscht Spalten der Matrix
       permrow(matrix,int,int)   vertauscht Zeilen der Matrix
```

```
";
////////////////////////////////////////////////////////////////////////
LIB "inout.lib";
////////////////////////////////////////////////////////////////////////
proc permcol (matrix A, int c1, int c2)
 "USAGE:    permcol(A,c1,c2); A matrix, c1,c2 positive integers
 RETURN:    matrix, A being modified by permuting column c1 and c2
 NOTE:      Platz für wichtige Anmerkungen,
            auch über mehrere Zeilen gestreckt
 KEYWORDS:  matrix, gaussnf, Lineare Algebra
 EXAMPLE:   example permcol; shows an example"
{
  matrix B=A;
  B[1..nrows(B),c1]=A[1..nrows(A),c2];
  B[1..nrows(B),c2]=A[1..nrows(A),c1];
  return(B);
}
example
{
"EXAMPLE:";
echo = 2;
ring r=0,(x),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
pmat(A);
pmat(permcol(A,2,3));
}
```

⋮

Taucht innerhalb einer Zeile ein Doppelslash `//` auf, so wird der Rest der Zeile als Kommentar interpretiert und ignoriert.

Der erste Abschnitt, der zwischen den beiden Kommentarzeilen steht, ist sozusagen der Kopf der Bibliothek. Die erste Zeile enthält das reservierte Wort `version`, durch das die Versionsnummer der Bibliothek festgelegt wird. Dem reservierten Wort `info` folgen allgemeine Informationen zur Bibliothek. `SEE ALSO:` und `KEYWORDS:` dienen der Erstellung von Verweisen im Handbuch. Man sollte beachten, daß unter dem Punkt `PROCECURES:` alle Prozedurnamen, die in der Bibliothek enthalten sind, mit einer maximal einzeiligen Beschreibung aufgeführt werden. Dieser Teil wird von SINGULAR angezeigt, wenn die Hilfe zur entsprechenden Bibliothek angefordert wird, etwa

```
help KeilenT.lib;
```

Man beachte auch, daß sowohl `version`, als auch `info` durch das Gleichheitszeichen, `=`, Strings zugewiesen werden, so daß die Anführungszeichen `"`, die sie einschachteln ebenso erforderlich sind, wie das Semikolon am Ende der Zeile!

Abschnitt zwei dient dem Einladen von anderen Bibliotheken, deren Prozeduren im bei den eigenen Prozeduren benötigt werden. Im Beispiel die Bibliothek `inout.lib`, deren Prozedur `pmat` im `example`-Teil der Prozedur `permcol` verwendet wird.

Im dritten Abschnitt folgen die Prozeduren, schlicht aneinander gereiht. (Dabei ist darauf zu achten, daß der Befehl `proc` stets am Beginn einer neuen Zeile steht!) Es empfiehlt sich, die in Abschnitt 1.6 angegebene Syntax für Prozeduren um zwei Abschnitte zu erweitern. Zwischen Prozedurkopf und -körper kann man, in Anführungszeichen " eingeschachtelt, einen Kommentarblock einschieben, der gewisse Schlüsselworte gefolgt von zugehöriger Information enthält. Unter `USAGE:` sollte eingegeben werden, wie der Befehl aufgerufen wird, und ggf. von welchem Typ die Argumente sind. `RETURN:` sollte Informationen darüber enthalten, von welchem Typ die Rückgabe ist, und ggf. weitere Informationen. `NOTE:` dient dazu wichtige Hinweise, zur Prozedur, ihrer Bedienung, etc. zu geben. Die unter `KEYWORDS:` angegebenen Worte werden ins Inhaltsverzeichnis des SINGULAR-Handbuches aufgenommen, sofern die Bibliothek SINGULAR offiziell angegliedert wird. `EXAMPLE:` gibt schließlich einen Hinweis darauf, wie man sich unter SINGULAR ein Beispiel zeigen lassen kann. Der hier erläuterte Kommentarblock enthält die Information, die angezeigt wird, wenn man unter SINGULAR Hilfe zu einer Prozedur anfordert, etwa durch

```
help permcol;
```

Der zweite Zusatzabschnitt am Ende der Prozedur wird durch das reservierte Wort `example` eingeleitet, gefolgt von einem Abschnitt in geschweiften Klammern, der SINGULAR-Code enthält. Ziel ist es, ein Beispiel für die Wirkung der Prozedur zu geben, die dem Nutzer die Verwendung erleichtert. Der Nutzer erhält das Beispiel durch Eingabe von `example PROZEDURNAME;`.

1.8. **Ausgabe in Dateien / Einlesen aus Dateien.** Der Befehl `write` bietet die Möglichkeit, die Werte von Variablen oder beliebige Strings in einer Datei abzuspeichern. Hierzu werden die Variablenwerte in Strings umgewandelt. Die folgenden Zeilen speichern Variablenwerte bzw. einen String in der Datei `hallo.txt`:

```
int a=5;
int b=4;
write("hallo.txt",a,b);
write("hallo.txt","Das ist Singular.");
```

Es können also auch mehrere Variablen oder Strings durch Kommata getrennt angegeben werden. Ihre Werte werden jeweils in eine neue Zeile geschrieben.

Daten, die sich in einer Datei befinden, können mit dem Befehl `read` eingelesen werden. Dabei werden sie jedoch als Strings interpretiert, z. B.

```
read("hallo.txt");
↦ 5
  4
  Das ist Singular.
```

Soll SINGULAR-Code, der aus einer Datei eingelesen wird, auch als solcher erkannt werden, dann muß der `read`-Befehl an den Befehl `execute` übergeben werden. Enthalte etwa die Datei `hallo.txt` die folgenden Zeilen,

```
4*5-3;
6/3;
```

dann führt der Befehl

```
execute(read("hallo.txt"));
```

zu folgendem SINGULAR-Output:

```
↦ 17
  2
```

Eine Kurzform für `execute(read(...))` ist <, z. B.

```
< "hallo.txt";
```

Wer eine SINGULAR-Sitzung zur Sicherheit in einer Datei, etwa `hallo.txt`, dokumentieren will, kann dies mit dem Befehl `monitor` tun, z. B.

```
monitor("hallo.txt","io");
```

Die Option `"io"` bewirkt, daß sowohl Eingaben (input), als auch Ausgaben (output) gespeichert werden. Entsprechend führt das Weglassen eines der beiden Buchstaben dazu, daß nur Eingaben oder nur Ausgaben gespeichert werden. Die Option `monitor` erweist sich dann als sehr hilfreich, wenn man unter einem Betriebssystem arbeitet, auf dem `Singular` instabil läuft bzw. auf dem man keinen gut handhabbaren Editor zur Verfügung hat. Man beachte, daß `monitor` eine Datei öffnet, aber nicht wieder schließt. Dies kann explizit durch folgende Eingabe erreicht werden:

```
monitor("");
```

2. Datentypen in Singular und Ringe

SINGULAR arbeitet mit einer ganzen Reihe von unterschiedlichen Strukturen, die als verschiedene Datentypen vorliegen. Will man ein Objekt in SINGULAR definieren, sprich eine Variable einführen, so ist es notwendig, ihr von Beginn an einen Datentyp zuzuweisen.

In SINGULAR sind die Datentypen, bis auf die Ausnahmen `string`, `int`, `intvec` und `intmat`, von einer Metastruktur abhängig, dem sogenannten Ring, über dem sie leben. (Es ist Teil der Vorlesung Lineare Algebra, zu definieren, was ein Ring ist, und welche Ringe in Singular zur Verfügung stehen.) Will man eine Rechnung in SINGULAR durchführen, ist es deshalb stets unabdingbar, zunächst den Ring zu definieren, über dem man arbeitet. Für die Lineare Algebra werden wir zu Beginn mit den folgenden Ringdefinitionen auskommen:

`ring r=0,(x),lp;`	Die Menge der Polynome in der Variablen x mit Koeffizienten in den rationalen Zahlen $\mathbb{Q}$.
`ring r=(0,a,b),(x,y,z),lp;`	Die Menge der Polynome in den Variablen x, y, z, wobei die Koeffizienten rationale Ausdrücke in den Variablen a und b sind. Natürlich können statt a, b bzw. x, y, z auch beliebige andere Variablen stehen. Wesentlich ist, daß die Variablen in der ersten Klammer im Nenner von Brüchen auftauchen dürfen, die in der zweiten Klammer nicht.

Wir sehen also, daß wir zunächst über den rationalen Zahlen $\mathbb{Q}$ rechnen werden. Reelle Zahlen als Dezimalzahlen (floating point numbers) oder gar komplexe Zahlen werden wir erst zu einem späteren Zeitpunkt zur Verfügung haben.

Im folgenden geben wir ein Liste der in `Singular` verfügbaren Datentypen, und wir geben auch jeweils ein Beispiel an, indem wir eine Variable des entsprechenden Typs definieren und ihr einen Wert zuweisen, durch den Operator `=`. Für die Lineare Algebra werden wir zunächst mit den ersten acht Typen auskommen. Die anderen sind nur der Vollständigkeit halber aufgeführt worden.

`int i=1;` — Der Datentyp `integer` repräsentiert die Maschinenzahlen (= ganze Zahlen). Außerdem werden Wahrheitswerte (= `boolean`) als `integers` repräsentiert, `0 = FALSE`, `1 = TRUE`.

`string s="Hallo";` — `strings` sind beliebige Zeichenketten. Stets durch Anführungszeichen eingegrenzt.

`intvec iv=1,2,3,4;` — Ein Vektor aus `integers`.

`intmat im[2][3]=1,2,3,4,5,6;` — Eine Matrix mit 2 Zeilen und 3 Spalten mit `integer`-Einträgen, hier $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

`ring R=(0,a),(x,y),lp;` — Der Ring $\mathbb{Q}(a)[x,y]$ mit lexikographischer Ordnung. Für weitere Erläuterungen konsultiere man das Handbuch [GPS99].

`number n=4/6;` — `numbers` sind die Elemente des Körpers, der dem Ring zugrunde liegt. Bei `ring r=0,(x),lp;` also die rationalen Zahlen, bei `ring r=(0,a),(x),lp;` auch Brüche von Polynomen in a mit ganzzahligen Koeffizienten, etwa $\frac{a^2+1}{a-1}$.

`list l=n,iv,s;` — Eine Liste kann Objekte ganz unterschiedlicher Typen enthalten. Auf den zweiten Eintrag von `l` kann durch `l[2]` zugegriffen werden.

`matrix m[2][3]=1,2,3,4,5,6;` — Eine Matrix mit 2 Zeilen und 3 Spalten, bei der die Einträge entweder vom Typ `poly` oder vom Typ `number` sind, wie hier $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

`vector v=[1,2,3];` — Ein Vektor im Modul $\mathtt{R}^3$. Sind die Einträge sämtlich `number`, können wir ihn aber auch als Vektor über dem Grundkörper auffassen.

`proc` — Der Datentyp `procedure` ist in Kapitel 1.6 ausführlich besprochen.

`poly f=x2+2x+1;` — Ein Polynom in den Veränderlichen des Rings mit `numbers` als Koeffizienten, hier $f = x^2 + 2x + 1$. Beachte, daß Zahlen vor den Monomen als Koeffizienten interpretiert werden, wohingegen SINGULAR Zahlen nach einzelnen Variablen als Exponenten interpretiert.

`ideal i=f,x3;` — Das von f und x^3 erzeugt Ideal in `R`.

`qring Q=i;` — Der Quotientenring `R/i`.

`map g=R,x;` — Die Abbildung von `R` nach `Q`, die durch $x \mapsto \overline{x}$ definiert wird.

`module mo=v,[x,x2,x+1];` — Der von den Vektoren `v` und $(x, x^2, x+1)^t$ in $\mathtt{R}^3$ aufgespannte Modul.

`def j;` — Will man sich zum Zeitpunkt der Definition einer Variablen noch nicht festlegen, welchen Typ sie haben soll, so definiert man sie als `def`. Die erste Zuweisung, mit der der Variablen ein Wert zugewiesen wird, legt dann auch den Datentyp fest.

`link`	Für den Datentyp `link` verweisen wir auf das Handbuch [GPS99].
`resolution`	Für den Datentyp `resolution` verweisen wir auf das Handbuch [GPS99].

Auf den ersten Blick mag es erscheinen, als ob die Matrizen `im` und `m` identisch seien. Für SINGULAR ist das jedoch nicht der Fall, da sie von unterschiedlichem Typ sind!

Will man mit Dezimalzahlen rechnen, also gleichsam den Grundkörper $\mathbb{R}$ zur Verfügung haben, so muß man in der Definition des Rings die "Charakteristik" 0 durch `real` ersetzen (bzw. `(real,50)`, wenn man mit 50 Nachkommastellen rechnen will), z. B.

```
ring r=(real,10),x,lp;
```

Sogar die komplexen Zahlen sind verfügbar, indem man `real` durch `complex` ersetzt. `i` bezeichnet dann die imaginäre Einheit, d. h. die Quadratwurzel aus -1.

3. Einige Elemente der Programmiersprache Singular

3.1. **Zuweisungen.** Will man in Singular Berechnungen durchführen, ist es in aller Regel unabdingbar, daß man Variablen Werte zuweist. Dies geschieht durch den Operator `=`. Man kann einer Variablen bereits bei ihrer Definition einen Wert zuweisen,

```
int i=1;
```

oder dies zu einem späteren Zeitpunkt nachholen,

```
int i;
⋮
i=2;
```

3.2. **Schleifen.** Es gibt zwei Typen von Schleifen, die `for`- und die `while`-Schleifen.

Die `for`-Schleife wird typischerweise verwand, wenn man eine Befehlssequenz mehrfach ausführen möchte und die Anzahl bereits vor Eintritt in die Schleife bekannt ist. Z. B.

```
int s=0;
int i;
for (i=1; i<=10; i=i+1)
{
  s=s+i;
}
```

In gescheiften Klammern steht die Befehlssequenz, die iterativ ausgeführt werden soll; in runden befinden sich Anweisungen, wie häufig die Schleife durchlaufen werden soll. Der erste Eintrag legt die Laufvariable (vom Typ `integer`) fest; der zweite Eintrag gibt die Abbruchbedingung, d. h. die Schleife wird nur solange durchlaufen, wie der Ausdruck den Wert `TRUE` ergibt; der dritte Eintrag legt fest, wie sich die Laufvariable in jedem Durchgang ändern soll. Das Beispiel berechnet somit die Summe der ersten zehn natürlichen Zahlen.

`while`-Schleifen bieten sich an, wenn die Anzahl der Durchläufe nicht a priori klar ist. Z. B.

```
int s=10000;
int i=1;
while (s > 50)
{
  i=i*i;
  s=s-i;
}
```

Wieder folgt in geschweiften Klammern die Befehlssequenz, während in runden Klammern nur die Abbruchbedingung steht. Solange diese den Wert `TRUE` liefert, wird die Schleife ausgeführt.

Bei beiden Schleifen wird die Abbruchbedingung vor dem ersten Eintritt in die Schleife überprüft!

3.3. **Verzweigungen.** SINGULAR bietet als Verzweigung die `if-else`-Anweisung, wobei der `else`-Anteil fehlen kann. Z. B.

```
int i=10;
int s=7;
if (i<5 or s<10)
{
  s=5;
}
else
{
  s=0;
}
```

Wieder stehen die Befehlssequenzen als Block in geschweiften Klammern, während sich in runden Klammern die Verzweigungsbedingung findet.

3.4. **Vergleichsoperatoren.** In Singular gibt es die Vergleichsoperatoren `==` und `!=`, mit denen man Objekte vom gleichen Datentyp (etwa `int`, `string`, `matrix`, etc.) miteinander vergleichen kann. `==` testet auf Gleichheit und liefert mithin den Wert `1`, wenn die Objekte gleich sind, und ansonsten `0`. `!=` testet auf Ungleichheit. Den gleichen Effekt hat `<>`.

Für die Datentypen `int`, `number`, `poly` und `vector` stehen zudem die Operatoren `<`, `>`, `<=` und `>=` zur Verfügung. Ihre Bedeutung für `integers` ist klar. Für die anderen Datentypen verweisen wir auf das Handbuch [GPS99].

3.5. **Einige weitere ausgewählte Operatoren in Singular.** Wie bereits gesehen, hängen die Operatoren, die zur Verfügung stehen, vom jeweiligen Datentyp ab. Wir beschränken uns auf die in der Linaren Algebra zunächst benötigten Datentypen.

3.5.1. *`boolean`.* Für `boolean`-Ausdrücke sind die Verknüpfungsoperatoren `and` und `or` sowie der Negierungsoperator `not` definiert.

```
not ((1==0) or (1!=0));
↦ 0
```

3.5.2. *`int`.* Für `integers` sind die Operationen `+`, `-` und `*` gänzlich unproblematisch. `^` bedeutet potenzieren

```
int i=4;
i^3;
↦ 64
```

Etwas schwieriger sind die Befehle `div` und `mod`, wobei ersterer synomym zu `/` ist. Führt man für zwei ganze Zahlen Division mit Rest durch, so liefert `mod` den Rest, und `div` das Ergebnis ohne Rest. Z. B. $7 = 2 * 3 + 1$, also

```
7 div 3;
↦ 2
7 mod 3;
↦ 1
```

3.5.3. *list.* Für den Datentyp `list` gibt es die folgenden Operatoren:

`+`	Fügt die Elemente von zwei Listen zu einer zusammen.
`delete`	Löscht ein Element aus einer Liste, `delete(l,3)` löscht das dritte Element der Liste `l`.
`insert`	Fügt einer Liste ein Element hinzu. `insert(l,4)` fügt der Liste `l` das Element `4` an der ersten Stelle hinzu, `insert(l,4,2)` an der zweiten.

3.5.4. *matrix.* Die Operatoren `+`, `-` und `*` stehen mit ihren offensichtlichen Bedeutungen zur Verfügung.

Wir verdeutlichen an Beispielen, wie auf einzelne Einträge einer Matrix bzw. ganze Zeilen oder Spalten einer Matrix zugegriffen werden kann:

```
matrix m[2,3]=1,2,3,4,5,6;
print(m);
↦  1,2,3,
   4,5,6
m[1,2];
↦ 2;
m[1,1..3];
↦ 1 2 3
m[1..2,3];
↦ 3 6
```

4. Einige ausgewählte Funktionen in Singular

Singular verfügt über ein recht beachtliches Arsenal an Funktionen, die zum Teil im Singular-Kern integriert sind, zum Teil über Bibliotheken zur Verfügung gestellt werden. Wir wollen hier nur eine kleine Auswahl von Funktionsnamen geben, die für die Bearbeitung der Übungsblätter in Linearer Algebra von Nutzen sein können. Über ihre Syntax sollte man sich mittels `help` oder im Handbuch informieren.

4.1. **Funktionen, die mit dem Datentyp `matrix` in Zusammenhang stehen.** `ncols, nrows, print, size, transpose, det`, als Funktionen im Kern von Singular. Ferner die Funktionen der Bibliothek `matrix.lib`, insbesondere `permrow, permcol, multrow, multcol, addrow, addcol, concat, unitmat, gauss_row, gauss_col, rowred, colred`. Auch die Funktion `pmat` aus der Bibliothek `inout.lib` ist interessant.

4.2. **Funktionen, die mit dem Datentyp `int` in Zusammenhang stehen.** `random, gcd, prime` als Funktionen im Kern von `Singular`.

5. ESingular - oder der Editor Emacs

Es gibt viele Editoren, in denen man Singular-Prozeduren und Bibliotheken schreiben kann. Unter Unix-ähnlichen Systemen bietet sich der Editor Emacs (oder Xemacs) an, da er den eingegebenen Code durch entsprechende farbige Unterlegung der Schlüsselworte leichter überschaubar macht und eine Vielzahl von Optionen bietet, die das Editieren und finden von Fehlern erleichtern.

Aber noch aus einem weiteren Grund empfiehlt es sich, Emacs zu benutzen. Singular kann in einem speziellen Emacs-Modus gestartet werden, nämlich als `ESingular`. Dies bedeutet, daß zunächst der Editor Emacs gestartet wird, und dann innerhalb von Emacs das Programm Singular. Der Vorteil besteht darin, daß neben der vollen Funktionalität des Editors Emacs für das Editieren von Dateien eine Reihe weiterer Optionen zur Verfügung gestellt werden, die die Bedienung vereinfachen – insbesondere für den unerfahrenen Nutzer, dem Pulldown-Menüs zur Verfügung stehen. Mittels

```
ESingular --emacs=xemacs
```

besteht die Möglichkeit, die Version von Emacs, die benutzt werden soll, festzulegen, in diesem Fall der Xemacs. Alternativ kann der Standard mittels der Environment-Variablen `EMACS` verändert werden.

6. Aufgaben

Aufgabe 1

Schreibe eine Prozedur `binomi`, die zwei natürliche Zahlen `n` und `k` einliest und den Binomialkoeffizienten $\binom{n}{k}$ zurückgibt. (Vereinbarung: falls $k < 0$ oder $k > n$, dann $\binom{n}{k} = 0$.)

Aufgabe 2

Schreibe eine Prozedur `quadratsumme`, die eine natürliche Zahl `n` einliest und die Summe der Quadratzahlen $1^2, 2^2, 3^2, \ldots, n^2$ ausgibt.

Aufgabe 3

Schreibe eine Prozedur `minimum`, die einen Vektor von natürlichen Zahlen einliest und das Minimum der Zahlen ausgibt.

Aufgabe 4

Schreibe Prozeduren `zeilensummennorm`, `maximumsnorm` und `q_eukl_norm`, die eine $(m \times n)$-Matrix `A` von reellen Zahlen einlesen und

1. die Zeilensummennorm von `A` (d. h. $\max_{i=1,\ldots,m} \left(\Sigma_{j=1}^{n} |A_{ij}|\right)$),
2. die Maximumsnorm von `A` (d. h. $\max\left(|A_{ij}| \;\middle|\; i = 1, \ldots, m,\ j = 1, \ldots, n\right)$), respektive
3. das Quadrat der euklidischen Norm berechnen (d. h. $\Sigma_{i,j} |A_{ij}|^2$).

Für den Absolutbetrag verwende die Funktion `abs` aus der Bibliothek `linalg.lib`.

Aufgabe 5

Schreibe eine Singular-Prozedur `gauss_reduction`, die eine Matrix A einliest und die mittels Gauß-Elimination ermittelte Zeilen-Stufen-Form der Matrix ausgibt. Die Einträge der Matrizen sollen vom Typ `number` sein. Es dürfen die in `matrix.lib` definierten Prozeduren `multrow`, `addrow` und `permrow` verwendet werden. Teste Deine Ergebnisse mit der Prozedur `gnf` aus der Bibliothek `linalg.lib`.

Aufgabe 6

Schreibe eine Prozedur `ebenenschnitt`, die die Koeffizienten zweier Ebenengleichungen einliest und den Schnitt der Ebenen ausgibt. Verwende die Prozedur `gauss_reduction` aus Aufgabe 5.

7. Lösungen

Lösung zu Aufgabe 1

```
proc binomi (int n, int k)
"USAGE: binomi(n,k); int n, int k
RETURN: int, Binomialkoeffizient n ueber k
KEYWORDS: Binomialkoeffizient
EXAMPLE: example binomi; zeigt ein Beispiel"
{
  if ((k < 0) or (k > n))
  {
   return(0);
  }
  else
  {
   int i;
   int nenner,zaehler1,zaehler2 = 1,1,1;
   for (i=1;i<=n;i++)
   {
    nenner = nenner * i;
   }
   for (i=1;i<=k;i++)
   {
    zaehler1 = zaehler1 * i;
   }
   for (i=1;i<=n-k;i++)
   {
    zaehler2 = zaehler2 *i;
```

```
    }
    return (nenner / (zaehler1 * zaehler2));
  }
}
example
{
  "Beispiel:";
  echo = 2;
  binomi(5,2);
  binomi(7,5);
}
```

Lösung zu Aufgabe 2

```
proc quadratsumme (int n)
"USAGE: quadratsumme(n); int n
RETURN: int, Summe der ersten n Quadratzahlen
KEYWORDS: Quadratzahlen
EXAMPLE: example quadratsumme; zeigt ein Beispiel"
{
  if (n < 0)
    {
     return (0);
    }
    else
    {
     int i;
     int ergebnis = 0;
     for (i=1;i<=n;i++)
     {
      ergebnis = ergebnis + i*i;
     }
     return (ergebnis);
    }
}
example
{
  "Beispiel:";
  echo = 2;
  quadratsumme(3);
  quadratsumme(5);
}
```

Lösung zu Aufgabe 3

```
proc minimum (intvec iv)
"USAGE: minimum(iv); iv intvector
RETURN: int, the minimum of the entries in iv
KEYWORDS: minimum
EXAMPLE: example minium; shows an example"
{
  int i;
  int k=size(iv);
  int ergebnis=iv[1];
  for (i=2;i<=k;i++)
   {
    if (iv[i] < ergebnis)
    {
     ergebnis=iv[i];
    }
   }
  return(ergebnis);
}
example
{
  "EXAMPLE:";
  echo=2;
  intvec iv=3,2,5,2,1;
  print(iv);
  minimum(iv);
  iv =-3,4,5,3,-6,7;
  print(iv);
  minimum(iv);
}
```

Lösung zu Aufgabe 4

1. Wir schreiben zunächst eine eigene kurze Prozedur zum Berechnen des Absolutbetrages.
```
proc abs_val (poly r)
"USAGE: abs_val(r); poly r - eine rationale/reelle Zahl
RETURN: poly, gibt den Absolutbetrag von r wieder, falls r eine reelle
Zahl
KEYWORDS: Absolutbetrag
EXAMPLE: example abs_value; zeigt ein Beispiel"
{
  if (r < 0)
```

```
  {
   return(-r);
  }
  else
  {
   return(r);
  }
}
example
{
  "Beispiel:";
  echo = 2;
  ring r=real,x,lp;
  abs_val(-5.45);
  ring s=0,x,lp;
  abs_val(-4/5);
}

proc zeilensummennorm (matrix A)
"USAGE: zeilensummennorm(A); matrix A mit rationalen/reellen Eintraegen
RETURN: poly, gibt die Zeilensummennorm von A wieder
KEYWORDS: Matrixnorm, Zeilensummennorm
EXAMPLE: example zeilensummennorm; zeigt ein Beispiel"
{
  int i,j;
  int n,m = ncols(A),nrows(A);
  poly r,s = 0,0;
  for (i=1;i<=m;i++)
  {
   for (j=1;j<=n;j++)
   {
    r = r + abs(A[i,j]);
   }
   if (r > s)
   {
    s = r;
   }
   r = 0;
  }
  return (s);
```

```
}
example
{
  "Beispiel:";
  echo = 2;
  ring r=real,x,lp;
  matrix A[3][2]=-3,-2,-1,3,-4,2;
  print(A);
  zeilensummennorm(A);
  ring r=0,x,lp;
  matrix B[3][2]=-7,0,0,3,-4,2;
  print(B);
  zeilensummennorm(B);
}
```

2. proc maximumsnorm (matrix A)

```
"USAGE: maximumsnorm(A); matrix A mit rationalen/reellen Eintraegen
RETURN: poly, gibt die Zeilensummennorm von A wieder
KEYWORDS: Matrixnorm, Maximumsnorm
EXAMPLE: example maximumsnorm; zeigt ein Beispiel"
{
  int i,j;
  int n,m = ncols(A),nrows(A);
  poly r = 0;
  for (i=1;i<=m;i++)
  {
   for (j=1;j<=n;j++)
   {
    if (abs(A[i,j]) > r)
    {
     r = abs(A[i,j]);
    }
   }
  }
  return(r);
}
example
{
  "Beispiel:";
  echo = 2;
  ring r=real,x,lp;
```

```
  matrix A[3][2]=-3,-2,-1,3,-4,2;
  print(A);
  maximumsnorm(A);
  ring r=0,x,lp;
  matrix B[3][2]=-7,0,0,3,-4,2;
  print(B);
  maximumsnorm(B);
}
```

3.
```
proc q_eukl_norm (matrix A)
"USAGE: q_eukl_norm(A); matrix A mit rationalen/reellen Eintraegen
RETURN: poly, gibt das Quadrat der euklidischen Norm von A wieder
KEYWORDS: Matrixnorm, Euklidische Norm
EXAMPLE: example q_eukl_norm; zeigt ein Beispiel"
{
  int i,j;
  int n,m = ncols(A),nrows(A);
  poly r = 0;
  for (i=1;i<=m;i++)
  {
   for (j=1;j<=n;j++)
   {
    r = r + abs(A[i,j]) * abs(A[i,j]);
   }
  }
  return (r);
}
example
{
  "Beispiel:";
  echo = 2;
  ring r=real,x,lp;
  matrix A[3][2]=-3,-2,-1,3,-4,2;
  print(A);
  q_eukl_norm(A);
  ring r=0,x,lp;
  matrix B[3][2]=-7,0,0,3,-4,2;
  print(B);
  q_eukl_norm(B);
}
```

Lösung zu Aufgabe 6

```
proc ebenenschnitt (matrix E1, matrix E2)
"USAGE: ebenenschnitt(E1,E2); matrix E1, matrix E2 - Koeffizienten von zwei
Ebenengleichungen
RETURN: list, (string,[vc,vt]), Informationen zum Schnitt von E1 und E2, ggf.
       eine Parametrisierung der Schnittgeraden
KEYWORDS: Ebenenschnitt, Schnittgerade
EXAMPLE: example ebenenschnitt; zeigt ein Beispiel"
{
  matrix m[2][4] = E1[1,1..4],E2[1,1..4];
  list l;
  matrix A[2][4] = gauss_reduction(m);
  if ((A[2,2] == 0) and (A[2,3] == 0) and (A[2,4] == 0))
  {
   l="Die beiden Ebenen sind gleich!";
   return(l);
  }
  else
  {
   if ((A[2,2] == 0) and (A[2,3] == 0) and (A[2,4] != 0))
   {
    l="Die beiden Ebenen sind parallel!";
    return(l);
   }
   else
   {
    if (A[2,2] != 0)
    {
     vector vt = [-A[1,3]/A[1,1]+(A[2,3]*A[1,2])/(A[1,1]*A[2,2]),-A[2,3]/A[2,2],1];
     vector vc = [A[1,4]/A[1,1]-A[2,4]/(A[1,1]*A[2,2]),A[2,4]/A[2,2],0];
     l="Der Schnitt der Ebenen ist die Gerade [2] + t * [3]",vc,vt;
    }
    else
    {
     if (A[1,1] != 0)
     {
      vector vt = [-A[1,2]/A[1,1],1,0];
      vector vc = [A[1,4]/A[1,1]-(A[1,3]*A[2,4])/(A[1,1]*A[2,3]),0,A[2,4]/A[2,3]];
      l="Der Schnitt der Ebenen ist die Gerade [2] + t * [3]",vc,vt;
     }
```

```
      else
      {
       vector vt = [1,0,0];
       vector vc = [0,A[1,4]/A[1,2]-(A[1,3]*A[2,4])/(A[1,2]*A[2,3]),A[2,4]/A[2,3]];
       l="Der Schnitt der Ebenen ist die Gerade [2] + t * [3]",vc,vt;
      }
     }
    }
    return (l);
  }
}
example
{
  "Beispiel:";
  echo = 2;
  ring r=0,(x,y,z),lp;
  matrix E1[1][4]=1,0,0,0;
  print(E1);
  matrix E2[1][4]=0,1,0,0;
  print(E2);
  list le=ebenenschnitt(E1,E2);
  print(le);
  matrix F1[1][4]=1,2,3,1;
  print(F1);
  matrix F2[1][4]=1,2,0,-1;
  print(F2);
  list lf=ebenenschnitt(F1,F2);
  print(lf);
  matrix I1[1][4]=0,2,3,1;
  print(I1);
  matrix I2[1][4]=0,2,0,-1;
  print(I2);
  list li=ebenenschnitt(I1,I2);
  print(li);
  matrix G1[1][4]=1,0,0,0;
  print(G1);
  matrix G2[1][4]=1,0,0,1;
  print(G2);
  list lg=ebenenschnitt(G1,G2);
  print(lg);
```

```
  matrix H1[1][4]=1,0,0,0;
  print(H1);
  matrix H2[1][4]=2,0,0,0;
  print(H2);
  list lh=ebenenschnitt(H1,H2);
  print(lh);
}
```

Literatur

[GPS99] Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, Singular *Manual, Version 1.3.4*, Fachbereich Mathematik und Zentrum für Computeralgebra, Universität Kaiserslautern, D-67653 Kaiserslautern, July 1999.

Thomas Keilen, Fachbereich Mathematik, Universität Kaiserslautern, 67553 Kaiserslautern

E-mail address: keilen@mathematik.uni-kl.de