Teil I

Einführung in die Programmierung

Kapitel 1

Das Betriebssystem Linux

Allgemeines

Die praktischen Programmierübungen im Computer–Pool werden an PCs stattfinden, auf denen das Betriebssystem Linux installiert ist. Linux ist neben WINDOWS das am meisten verbreitete Betriebssystem. Die Vorteile von Linux sind

- höhere Sicherheit durch strenge Unterteilung der Zugriffsrechte,
- Preis,
- offener Code.

Die Nachteile sind, dass

- manche Dinge nicht so komfortabel wie unter WINDOWS sind,
- WINDOWS–Software auf Linux nicht läuft und man sich umgewöhnen muss, wenn man die entsprechende Linux–Software nutzt.

Bezüglich des ersten Nachteils wurde allerdings im Laufe der vergangenen Jahre viel getan, dass oft kein Unterschied zu WINDOWS mehr vorhanden ist.

Der Name Linux tauchte erstmals 1991 nach der Veröffentlichung des ersten Linux–Kernels durch Linus Torvalds auf. Dieses Betriebssystem wird vor allem auf Servern eingesetzt, so zum Beispiel laufen die Server von Google und Wikipedia unter Linux. Der Einsatz auf Desktops ist vor allem im universitären Bereich zu finden, und auch da vor allem bei Mathematikern und Informatikern. Für den Desktop gibt es viele unterschiedliche Linux–Distributionen:

- SuSe, ist im deutschsprachigen Raum am meisten verbreitet,
- RedHat, ist im amerikanischen Raum am meisten verbreitet,
- Debian, läuft im Computer–Pool.

Informationen zu Unterschieden, Vor– und Nachteilen der einzelnen Distributionen findet man im Internet. Von einem Hörer der vergangenen Vorlesung ist das Buch [Bar04] zur Einarbeitung in Linux empfohlen wurden.

Zugriffsrechte

Jeder Nutzer (user) ist in Linux einer Gruppe (group) zugeordnet. Die Zugriffsrechte jedes Files und Verzeichnisses in in Linux sind lesen (read, r), schreiben (write, w) und ausführen (execute, x). Diese Zugriffsrechte sind getrennt gesetzt für den user, die group und alle übrigen. Zum Beispiel, besagt

```
-rw-r----- 1 john users 29444 2007-10-13 12:22 tmp.txt
```

dass das File `tmp.txt` vom Nutzer john gelesen und geschrieben werden kann (Zeichen 2–4), von der Gruppe users nur gelesen werden kann (Zeichen 5–7) und alle übrigen dürfen mit diesem File nichts machen (Zeichen 8–10). Bei einem Verzeichnis

sieht diese Information zum Beispiel wie folgt aus

```
drwxr-xr-x 3 john users 312 2007-10-12 18:26 MOD_PROG
```

Das `d` am Anfang besagt, dass es sich um ein Verzeichnis (directory) handelt, der Nutzer john darf im Verzeichnis lesen, schreiben und in das Verzeichnis wechseln, die Gruppe users und alle übrigen dürfen nur lesen und in das Verzeichnis wechseln, dort aber nichts ändern (schreiben). Das Setzen und Ändern der Zugriffsrechte geschieht mit dem Befehl `chmod`. Wenn man zum Beispiel das File `tmp.txt` für alle les– und schreibbar machen will, so kann man das mit

```
chmod a+rw tmp.txt
```

und erhält danach die Information

```
-rw-rw-rw- 1 john users 29444 2007-10-13 12:22 tmp.txt
```

Man kann die Zugriffsrechte nur von Dateien und Verzeichnissen ändern, die einem gehören.
Das heißt, die Zugriffsrechte regeln wer was machen darf. Ist der Rechner ordentlich administriert, hat ein Virus oder ein Wurm keine Möglichkeit wichtige Dinge zu verändern, da er dazu nicht die Rechte hat. Die Rechte müssen so gesetzt sein, dass das nur der Administrator, der in Linux *root* heißt, machen darf und um als Administrator zu arbeiten, muss man ein entsprechendes Password eingeben. Aus Sicherheitsgründen sollte ein Password immer so gewählt sein, dass der Nutzernamen nicht ein Teil des Passwords ist und das Password auch Ziffern enthält.

Werkzeuge

Man hat in Linux die Möglichkeit mit graphischen Benutzteroberflächen (wie in WINDOWS) oder auch auf der Kommandozeilenebene (Shell) zu arbeiten. Die Shell ist ein Kommandointerpreter, der von der Kommandozeile die Anweisungen einliest, diese auf Korrektheit überprüft und ausführt. Wenn man sich von außen auf einem Computer mit Betriebssystem Linux einloggt, kann man oft keine graphischen Oberflächen öffnen, zum Beispiel um in das Netz der Mathematik von zu Hause zu gelangen, muss man folgenden Weg gehen:

PC zu Hause → `contact.math.uni-sb.de` → PC im Büro

Dann ist man gezwungen, mit der Shell zu arbeiten. Ein Ziel dieser Veranstaltung besteht darin, dass die wichtigsten Shell–Kommandos vermittelt und praktiziert werden. Eine Liste wichtiger Kommandos wird am Ende des Kapitels gegeben.

Wichtige Werkzeuge, die wir brauchen sind:

- Konquerer (erfüllt etwa die Aufgaben wie der Explorer in Windows)
- Editoren: Kate, Kile, emacs, Xemacs (sind selbsterklärend), `vi` ist ein Kommandoline–Editor, der dann nützlich ist, wenn man keine graphische Oberflächen öffnen kann (wenn man beispielsweise von außerhalb auf einem Linux–Rechner eingeloggt ist)
- `matlab`, Software zum Programmieren von Verfahren, dieses Programm gehört nicht zur Linux–Distribution und muss gekauft werden.
- `gcc`, der Gnu–C–Compiler,
- `latex`, Programmiersprache zur Textverarbeitung.

Standardsoftware wie `Firefox, acroread` läuft natürlich auch unter Linux. Es gibt natürlich noch viel mehr nützliche Dinge, siehe Literatur, Internet, Handbücher, Übungen.

Dateinamen, Baumstruktur

Dateinamen:

- Linux unterscheidet Groß– und Kleinschreibung,
- Der Schrägstrich / darf nicht verwendet werden, da er zur Trennung von Verzeichnisnamen dient,
- Sonderzeichen (Leerzeichen, Umlaute, &, ...) sollten vermieden werden, da einige eine spezielle Bedeutung besitzen.

Linux besitzt eine hierarchische, baumstrukturierte Verzeichnisstruktur, siehe Abbildung 1.1. Ausgangspunkt ist die Wurzel `/` (root). Die Position eines beliebigen (Unter)–Verzeichnisses ist durch den Pfad gegeben. Der Pfad gibt an, wie man von der Wurzel zu der gewünschten Datei beziehungsweise zum gewünschten Verzeichnis gelangt, zum Beispiel

```
pwd
```

(pwd – print name of current/working directory) gibt

```
/home/john
```

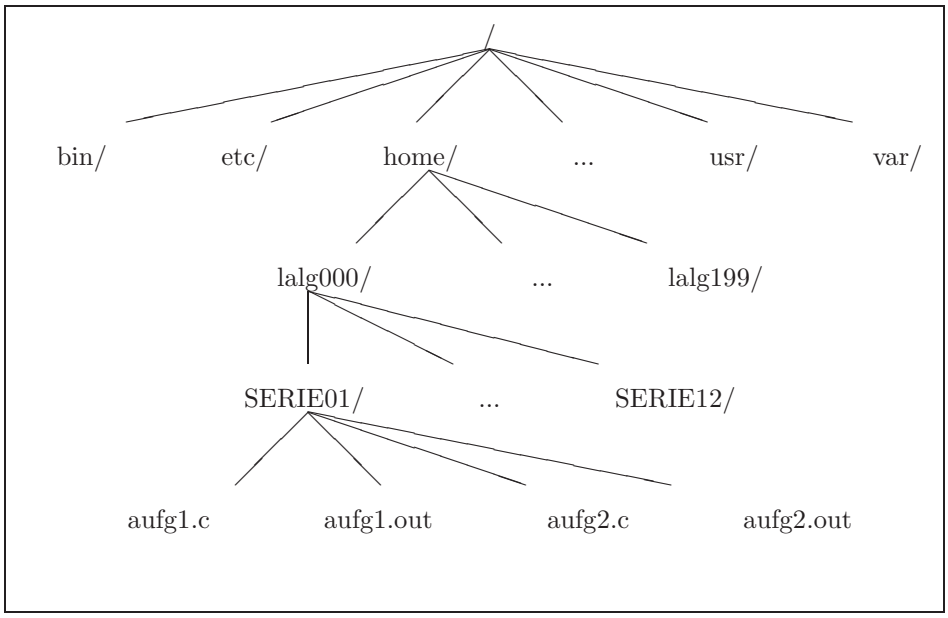

Abbildung 1.1: Beispiel für einen Verzeichnisbaum in Linux.

Pfade, die mit / beginnen, heißen absolute Pfade. Daneben gibt es noch die relativen Pfade, die relativ zur gegenwärtigen Position im Verzeichnisbaum sind. Wichtige relative Pfadbezeichner sind

- `.` : aktuelle Position im Verzeichnisbaum (Arbeitsverzeichnis),
- `..` : das übergeordnete Verzeichnis.

Es ist möglich, mehrere Dateien oder Verzeichnisse gleichzeitig anzusprechen, mit Hilfe sogenannter Wildcards

- `*` : ersetzt beliebig viele Zeichen (auch keines)
- `?` : ersetzt genau ein Zeichen
- `[zeichen1 - zeichen2]` : Auswahlsequence; alle Zeichen zwischen `zeichen1` und `zeichen2` werden ersetzt.

So kann zum Beispiel

```
ls tmp*
```

die Ausgabe

```
tmp tmp.txt
```

geben.

Liste von Kommandos

Nähere Informationen zu den Kommandos erhält man mit dem `man` Befehl

```
man Befehlsname
```

Sucht man Befehle, die mit einem Schlüsselbegriff zusammenhängen, so verwende man

```
man -k Schlüsselbegriff
```

Befehl	Bemerkungen
`cd`	wechsle Verzeichnis; zurück zum vorherigen Verzeichnis : `cd -` ins Homeverzeichnis: `cd`
`chmod`	verändere Zugriffsrechte; z.B. Schreibrecht für alle Nutzer `chmod a+w` *filename*
`cp`	kopiere Datei
`df`	gibt den belegten und freien Speicherplatz (Harddisc) zurück
`env`	zeigt die Umgebungsvariablen an
`find`	findet Dateien, z.B. um all Dateien mit Namen `core` zu finden `find / -name core -print`
`grep`	sucht nach einem Muster in einer Datei, z.B. um alle `printf`–Befehle in `test.c` zu finden `grep 'printf' test.c | more`
`gzip`	komprimiert Dateien
`gunzip`	dekomprimiert Dateien *filename.gz*
`kill`	beendet Prozesse
`ll`	zeigt Inhalt von Verzeichnissen
`ls`	zeigt Inhalt von Verzeichnissen, wichtige Optionen`-ali`
`man`	Handbuch, z.B. `man man`
`mkdir`	Anlegen von Verzeichnissen
`more`	Ansehen von Dateien
`mv`	verschieben von Dateien
`passwd`	Veränderung des Passwords
`ps`	zeigt laufende Prozesse an, z.B. vom Nutzer *abc* `ps -u abc`
`pwd`	zeigt Pfadnamen zum gegenwärtigen Verzeichnis
`rm`	löscht Dateien, **nutze besser immer** `rm -i`
`rmdir`	löscht Verzeichnisse
`tail`	zeigt die letzten Zeilen einer Datei
`tar`	Erstellung von Archiven
`typeset`	setzen von Umgebungsvariablen, z.B. `typeset -x PATH=$PATH:$HOME/bin`
`who`	zeigt wer im System eingeloggt ist
`which`	lokalisiert ausführbares Programm

Kapitel 2

Algorithmen

Unter einem Algorithmus versteht man eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder eines bestimmten Typs von Problemen. Eine exakte Definition dieses Begriffes ist nicht trivial und erfolgt in Informatikvorlesungen. Wir werden Algorithmen zur Lösung mathematischer Aufgabenstellungen verwenden.

Beispiel 2.1 *Quadratische Gleichung.* Zur Lösung der quadratischen Gleichung

$$0 = x^2 + px + q, \quad p, q \in \mathbb{R},$$

im Bereich der reellen Zahlen kann man folgenden Algorithmus verwenden:

Algorithmus 2.2 Lösung der quadratischen Gleichung.

- berechne
$$a := -\frac{p}{2}$$
- berechne
$$D := a^2 - q$$
- falls $D \geq 0$, dann
$$x_1 := a + \sqrt{D}, \quad x_2 := a - \sqrt{D}$$
sonst Ausgabe *keine reelle Lösung*

Man kann aber auch einen anderen Algorithmus verwenden, der sich im letzten Schritt unterscheidet und die Vieta[1]sche Wurzelformel nutzt:

Algorithmus 2.3 Lösung der quadratischen Gleichung.

- berechne
$$a := -\frac{p}{2}$$
- berechne
$$D := a^2 - q$$
- falls $D < 0$, dann Ausgabe *keine reelle Lösung*
sonst
  - falls $a < 0$, setze
$$x_1 := a - \sqrt{D}, \quad x_2 := q/x_1$$
sonst falls $a > 0$, setze
$$x_1 := a + \sqrt{D}, \quad x_2 := q/x_1$$
sonst
$$x_1 := \sqrt{-q}, \quad x_2 := -x_1.$$

[1]Francois Vieta (Viète) 1540 – 1603

Beide Algorithmen liefern mathematisch dasselbe Ergebnis. In den Übungen wird man sehen, dass das auf einem Computer im allgemeinen nicht mehr der Fall ist.

Diese Algorithmen enthalten bereits ein wichtiges Merkmal, nämlich Verzweigungen. Man muss an hand von berechneten Werten sich zwischen zwei oder mehr Möglichkeiten entscheiden, welcher Teil des Algorithmus abzuarbeiten ist. □

Beispiel 2.4 *Summe von Zahlen.* Man soll die Summe der natürlichen Zahlen von 1000 bis 10000 bilden. Ein möglicher Algorithmus ist, diese nacheinander aufzuaddieren:

Algorithmus 2.5

- setze $summe := 1000$
- setze für $i = 1001$ bis $i = 10000$

$$summe := summe + i$$

Hierbei ist *summe* als Variable zu verstehen, die einen bestimmten Speicherplatz im Computer einnimmt. Der neue Wert wird somit auf demselben Platz gespeichert, wie der bisherige Wert. Man muss zur Berechnung der Summe 9000 Additionen durchführen.

Auch dieser Algorithmus hat ein charakteristisches Merkmal, nämlich eine Schleife. Es gibt unterschiedliche Arten von Schleifen, siehe später.

Ein anderer Algorithmus nutzt die bekannte Summationsformel von Gauß[2]

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

Damit erhält man

$$\sum_{i=1000}^{10000} i = \sum_{i=1}^{10000} i - \sum_{i=1}^{999} i = \frac{10000 \cdot 10001}{2} - \frac{999 \cdot 1000}{2} = 5000 \cdot 10001 - 500 \cdot 999.$$

Damit kann man das Ergebnis mit zwei Multiplikationen und einer Subtraktion berechnen.

Man sieht, es kann billige (effiziente) und teure Algorithmen geben, die zum gleichen Ergebnis führen. Ziel ist es natürlich, den jeweils effizientesten Algorithmus zu verwenden. □

Zur Beschreibung von Algorithmen nutzt man die folgenden Grundstrukturen:
- Anweisung,
- bedingte Verzweigung (Alternative),
- Wiederholung (Schleife, Zyklus).

Die Zusammenfassung mehrerer Anweisungen wird auch Sequenz genannt.

Beispiel 2.6 Am Anfang von Algorithmus 2.2 kommt die folgende Sequenz

```
a = -p/2;
D = a*a-q;
```

(Hier wird MATLAB–Notation genutzt.) □

Bei der Alternative unterscheidet man die einfache und die vollständige Alternative.

Beispiel 2.7 Die einfache Alternative haben wir bereits in den Algorithmen 2.2 und 2.3 gesehen:

[2] Johann Carl Friedrich Gauss (1777 – 1855)

```
if D<0
   disp('Keine reelle Loesung')
else
   Anweisungen zur Berechnung der Lösung
end
```

Im Algorithmus 2.3 findet man auch eine vollständige Alternative:

```
if a<0
   x_1 = a-sqrt(D);
   x_2 = q/x_1;
elseif a>0
   x_1 = a+sqrt(D);
   x_2 = q/x_1;
else
   x_1 = sqrt(-q);
   x_2 = -x_1;
end
```

Möglichkeiten zur einfachen Behandlung mehrerer Alternativen sind in MATLAB und C ebenfalls vorhanden: `switch`–Anweisung, siehe später. □

In einer Schleife oder einem Zyklus ist eine Sequenz wiederholt abzuarbeiten. Die Anzahl der Durchläufe muss dabei vorab nicht unbedingt bekannt sein, sie bestimmt sich oft im Laufe der Abarbeitung. Man unterscheidet abweisende (kopfgesteuerte) Zyklen und nichtabweisende (fußgesteuerte) Zyklen:

- abweisender Zyklus: Bedingung für die Abarbeitung wird vor Beginn des Zyklus getestet, es kann passieren, dass diese beim ersten Mal schon nicht erfüllt ist und der Zyklus wird nie abgearbeitet,
- nichtabweisender Zyklus: Bedingung für die Abarbeitung wird am Ende des Zyklus getestet, damit wird der Zyklus mindestens einmal abgearbeitet.

Beispiele dafür wird es in den Übungen geben. Ist die Anzahl der Durchläufe bekannt, dann wird der Zyklus durch einen Zähler gesteuert.

Beispiel 2.8 Die Steuerung durch einen Zähler hatten wir bereits in Algorithmus 2.5:

```
summe = 1000;
for i=1001:10000
   summe = summe + i;
end
```

□

Jeder Zyklus erfordert Vorbereitungen, das heißt, vor Eintritt in den Zyklus sind entsprechende Variablen zu belegen. Im obigen Beispiel ist `summe` = `1000` zu setzen. Im Zykluskörper selbst muss so auf die Abbruchbedingung eingewirkt werden, dass der Abbruch nach endlich vielen Durchläufen garantiert ist. Hat man dabei einen Fehler gemacht, dann kann das Programm den Zyklus nicht verlassen und es hilft nur, das Programm abzubrechen und den Fehler zu suchen.

Kapitel 3

Einführung in MATLAB

3.1 Allgemeines

MATLAB ist eine kommerzielle mathematische Software zur Lösung mathematischer Probleme und zur graphischen Darstellung der Ergebnisse. Die Verfahren in MATLAB beruhen auf Matrizen (MATrix LABoratory).

MATLAB ist leider nicht ganz billig. Im Computer–Pool kann man das Programm mit

```
/usr/local/matlab/bin/matlab
```

starten.

Zur Einarbeitung in MATLAB gibt es viele Bücher, siehe `www.amazon.de`. Das Buch [DS04] ist so eine Art Klassiker, der einen kurzen und knappen Überblick gibt (man muss wissen, wonach man suchen soll). Eine Uraltversion von [DS04] ist im Internet (siehe Homepage, auf der die Übungen stehen) verfügbar. Weitere frei verfügbare Beschreibungen findet man auf der gleichen Homepage und im Internet. Diese Dokumentationen beruhen zwar auf älteren Versionen von MATLAB, sind aber für diese Vorlesung vollkommen ausreichend. Es gibt eine umfangreiche und gute Hilfe innerhalb von MATLAB, Aufruf mit `help`.

Man programmiert in MATLAB mit einer plattformunabhängigen Programmiersprache, die auf der jeweiligen Maschine interpretiert wird. Durch den einfachen, mathematisch orientierten Syntax der MATLAB–Skriptsprache und durch umfangreiche vorhandene Funktionsbibliotheken ist die Erstellung von Programmen wesentlich einfacher möglich als beispielsweise unter C. Man braucht sich vor allem nicht um die Organisation des Speichers kümmern. Allerdings sind MATLAB–Programme im allgemeinen bedeutend langsamer als C–Programme.

Man kann sein Programm direkt in das MATLAB–Befehlfenster eintippen. Sinnvoller ist es jedoch, es in eine separate Datei zu tun und diese vom MATLAB–Befehlfenster aus aufzurufen. *Vorlesung: an Summe der ersten 100 Zahlen demonstrieren.* Mit dem Befehl `edit` wird ein Editor geöffnet, in dem man die Datei erstellen kann. MATLAB–Befehlsdateien besitzen die Endung `.m`, (M–Files). Mit dem Befehl `what` kann man sich die im gegenwärtigen Verzeichnis vorhandenen M–Files ansehen. Sie werden ausgeführt, indem sie im MATLAB–Befehlfenster einfach aufgerufen werden (die benötigten Parameter müssen natürlich übergeben werden). Weitere wichtige allgemeine MATLAB–Befehle sind `ls, cd, pwd`. Sie haben die gleiche Bedeutung wie in LINUX. Des weiteren sind die Befehle

`clear;` löscht alle Variablen
`clf;` löscht alle Bildfenster
`who;` zeigt alle Variablen an

wichtig, damit bei einem wiederholten Starten von Programmen nicht alte Belegungen die Ergebnisse verfälschen. Die Ausgabe von Text erfolgt mit `disp`. Die Formatierung mit `format`.

Die Nutzung von MATLAB ist an vielen Hochschulen Standard im Rahmen von Vorlesungen, die sich mit numerischen Verfahren beschäftigen. Hier werden nur die wichtigsten Befehle vorgestellt. Ansonsten gilt, was für jede Programmiersprache gilt: *Learning by doing.*

3.2 Bemerkungen zu Vektoren und Matrizen

Vektoren sind aus der Schule bekannt. Man unterscheidet

$$\mathbf{a} = (a_1, \ldots, a_n),$$

einen n–dimensionalen Zeilenvektor und

$$\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix},$$

einen n–dimensionalen Spaltenvektor. Die Anzahl der Komponenten eines Vektors nennt man Dimension (hier n, in der Schule im allgemeinen $n \in \{2, 3\}$).

Wandelt man einen Zeilenvektor in einen Spaltenvektor mit den gleichen Einträgen um (oder Spalten– in Zeilenvektor), so nennt man diese Operation transponieren. Der transponierte Vektor des obigen Zeilenvektors ist

$$\mathbf{a}^T = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}.$$

Das Skalarprodukt zweier Spaltenvektoren ist aus der Schule für $n = 3$ bekannt. Für zwei n–dimensionale Spalenvektoren ist es

$$\mathbf{a} \cdot \mathbf{b} = (\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b} = \sum_{i=1}^{n} a_i b_i.$$

Die Norm oder Länge eines Vektors ist

$$\|\mathbf{a}\| = (\mathbf{a} \cdot \mathbf{a})^{1/2} = \left(\sum_{i=1}^{n} a_i^2 \right)^{1/2}.$$

Matrizen und ihre tiefere Bedeutung sowie ihre Eigenschaften werden am Ende von Lineare Algebra I behandelt. Hier ist es nur wichtig, dass es zwei–dimensionale Felder sind, mit m Zeilen und n Zeilen:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}.$$

In diesem Sinne sind n–dimensionale Zeilenvektoren $1 \times n$ Matrizen und m–dimensionale Spaltenvektoren $m \times 1$ Matrizen.

3.3 Matrizen

MATLAB rechnet mit Matrizen. Dabei ist zum Beispiel eine Zahl eine 1×1–Matrix und ein Spaltenvektor eine $n \times 1$–Matrix. Operationen mit Matrizen sind nur dann wohldefiniert, wenn ihre Dimensionen entsprechende Bedingungen erfüllen, zum Beispiel müssen bei der Addition die Dimensionen gleich sein. Die Dimensionskontrolle wird in MATLAB streng durchgeführt. So ist es zum Beispiel nicht möglich, einen Zeilen– und einen Spaltenvektor der gleichen Länge zu addieren. Die Indizierung von Matrizen beginnt in MATLAB mit 1.

MATLAB rechnet auch mit komplexen Zahlen. Wichtige Konstanten sind:

- `pi`
- `i`, `j`, imaginäre Einheit.

Die Eingabe beziehungsweise Erzeugung von Matrizen kann auf verschieden Art und Weisen erfolgen:

- Eingabe der Matrixeinträge,
- Laden aus externen Dateien,
- Erzeugen mit vordefinierten Funktionen,
- Erzeugen mit eigenen Funktionen.

Beispiel 3.1 Die Matrix

$$A = \begin{pmatrix} 2 & 4 & 7 \\ -5 & 4 & 2 \end{pmatrix}$$

kann wie folgt eingegeben werden

```
A = [2 4 7; -5, 4, 2]
```

Man kann auch jedes Element einzeln angeben:

```
A(1,1) = 2
A(1,2) = 4
A(1,3) = 7
A(2,1) = -5
A(2,2) = 4
A(2,3) = 2
```

Die Einheitsmatrix der Dimension $n \times n$ erhält man mit

```
B = eye(n)
```

wobei `n` vorher mit einer positiven ganzen Zahl belegt sein muss. Analog erhält man eine Matrix mit Nullen durch

```
C = zeros(n)
```

Eine $(m \times n)$–Matrix mit zufälligen Einträgen erhält man mit

```
D = rand(m,n)
```

Will man die Ausgabe der Matrizen auf dem Bildschirm unterdrücken, so beendet man die Befehle mit einem Semikolon. □

Wichtige Operationen mit Matrizen:

- Die transponierte Matrix A^T einer gegebenen Matrix `A` erhält man mit

  ```
  B = A'
  ```

  Man kann die transponierte Matrix auch auf dem Speicherplatz der ursprünglichen Matrix speichern

  ```
  A = A'
  ```

- Die Dimension von `A` erhält man mit

  ```
  [m,n] = size(A)
  ```

  Hierbei ist `m` die Anzahl der Zeilen und `n` die Anzahl der Spalten.
- Die Teilmatrix mit den Zeilenindizes $i1, \ldots, i2$ und den Spaltenindizes $j1, \ldots, j2$ einer Matrix `A` erhält man mit

```
B = A(i1:i2,j1:j2)
```

Wird der Doppelpunktoperator ohne vorderes Argument gebraucht, so wird mit dem ersten Index begonnen; ohne hinteres Argument, wird mit dem letzten Index aufgehört. So erhält man die erste Zeile von `A` durch

```
B = A(1,:)
```

- Addition zweier Matrizen `A` und `B`, Subtraktion sowie Multiplikation werden mit den üblichen Symbolen bezeichnet

  ```
  C = A+B      C = A-B      C = A*B
  ```

- Multiplikation, Division und Potenzierung einer Matrix `A` mit einem Skalar a ((1×1)–Matrix) werden mit den üblichen Symbolen bezeichnet

  ```
  B = a*A      B = A/a      B=A^a
  ```

- Elementweise Multiplikation und Division werden wie folgt durchgeführt

  ```
  C = A.*B      C = A./B
  ```

  Beispiel:

$$\mathtt{A} = (1,5), \quad \mathtt{B} = \begin{pmatrix} 3 \\ -2 \end{pmatrix}, \qquad \mathtt{A} * \mathtt{B} = -7, \quad \mathtt{A}. * \mathtt{B}' = (3, -10)$$

Die folgende Liste enthält wichtige Befehle, die man für Matrizen in MATLAB zur Verfügung hat. Die Einfachheit dieser Befehle sollte nicht darüber hinwegtäuschen, dass innerhalb von MATLAB zum Teil komplizierte Verfahren zur Berechnung der Ergebnisse ablaufen (siehe Vorlesung Praktische Mathematik). Vergleichbare Befehle stehen zum Beispiel in C nicht zur Verfügung. Daraus erklärt sich die Einfachheit, mit der man Programme in MATLAB erstellen kann. Für die angegebenen Befehle gibt es teilweise alternative Aufrufe, siehe MATLAB–Hilfe.

- der Rang einer $n \times n$–Matrix `A`:

  ```
  r = rank(A)
  ```

- die Inverse einer regulären $n \times n$–Matrix `A`:

  ```
  B = inv(A)
  ```

- die Determinante einer $n \times n$–Matrix `A`:

  ```
  d = det(A)
  ```

- die Spektralnorm einer $m \times n$–Matrix `A`:

  ```
  d = norm(A)
  ```

  Andere Normen können ebenfalls berechnet werden, siehe MATLAB–Hilfe. Ist `A` ein Vektor, dann ist die Spektralnorm die Euklidische Vektornorm.

- die Eigenwerte und Eigenvektoren einer $n \times n$–Matrix `A`:

  ```
  [u,v] = eig(A);
  ```

  Dabei enthält `v` eine Diagonalmatrix mit den Eigenwerten und die Matrix `u` enthält die zugehörigen Eigenvektoren.

- die Lösung eines linearen Gleichungssystems `Ax = b` mit einer regulären $n \times n$–Matrix `A` erhält man mit

  ```
  x = A\b
  ```

3.4 Wichtige Funktionen in MATLAB

Befehl	Bedeutung
`atan`	Arcus–Tangens
`cos`	Kosinus
`exp`	Exponentialfunktion
`log`	Logarithmus naturalis ln !

`log10`	Logarithmus zur Basis 10
`sin`	Sinus
`sqrt`	Wurzel
`tan`	Tangens

Für Einzelheiten, zum Beispiel was bei matrixwertigen Argumenten passiert, siehe MATLAB–Hilfe. Weitere Funktionen findet man ebenso in der MATLAB–Hilfe.

3.5 Ablaufkontrolle

Die Ablaufkontrolle beinhaltet Alternativen und Zyklen.

Die Alternative wird wie folgt programmiert:

```
if expr
   sequenz
elseif expr
   sequenz
else
   sequenz
end
```

Dabei gibt *expr* einen Wahrheitswert (true oder false) zurück. Folgende Relationen und wichtige logische Operatoren stellt MATLAB zum Vergleich von Ausdrücken zur Verfügung:

Befehl	Bedeutung
`<`	kleiner
`<=`	kleiner oder gleich
`>`	größer
`>=`	größer oder gleich
`==`	gleich
`~=`	ungleich
`&&`	logisches und
`||`	logisches oder
`~`	logisches nicht
`xor(·,·)`	exklusives oder (entweder oder)

Soll zum Beispiel kontrolliert werden, ob eine Matrix ein Zeilen– oder Spaltenvektor ist, so kann man das mit

```
[m,n] = size(a)
if (m==1) || (n==1)
```

tun.

Innerhalb der `if`–Anweisung sind mehr als eine `elseif`–Abfrage möglich. Eine Alternative für Mehrfachverzweigungen bildet die `switch`–Anweisung:

```
switch switch expr
   case expr,
      sequenz
   case expr,
      sequenz
   case expr,
```

```
      sequenz
   otherwise,
      sequenz
end
```

Hier wird der Ausdruck nach dem `switch`–Befehl ausgewertet und dann der entsprechende Teil der Anweisung abgearbeitet, bei welcher der Ausdruck hinter dem `case`–Befehl mit der Auswertung des `switch`–Befehls übereinstimmt.

Mit einer `for`–Schleife wird eine vorgegebene Anzahl von Zyklen durchlaufen, etwa

```
for i=1:100
   sequenz
end
```

Eine `while`–Schleife wiederholt eine Sequenz so oft, bis ein logisches Abbruchkriterium erfüllt ist, etwa

```
i=1;
while i<100
   i= input('i ');
end
```

Diese Schleife wird erst abgebrochen, wenn eine Zahl $i \geq 100$ eingegeben wird.

Sowohl die `for`– als auch die `while`–Schleife sind abweisend. Ein vorzeitiges Verlassen eine Schleife ist mit dem `break`–Befehl möglich.

3.6 Graphik

Eine Stärke von MATLAB ist die Graphik, mit der man sich schnell und unkompliziert die berechneten Ergebnisse ansehen kann. Wie generell in MATLAB, werden bei der Graphik Daten gezeichnet, die in Vektoren und Matrizen gespeichert sind.

Zweidimensionale Graphiken erhält man mit dem `plot`–Befehl:

```
for i=1:101
   x(i) = (i-1)/100;
   y(i) = x(i)^3-0.5;
end
plot(x,y)
```

Damit wird die Funktion $x^3 - 0.5$ im Intervall $[0, 1]$ gezeichnet. Übergibt man nur ein Argument an `plot`, dann sind die Werte auf der x–Achse die Indizes der Vektoreinträge. Für verfügbare Linienarten und –farben sei auf `help plot` sowie auf die Literatur verwiesen. Von der Hilfe zu `plot` wird man dann auch zu weiteren Befehlen geführt, mit denen man eine Graphik verschönern kann, wie `legend, xlabel, axis`. Man kann die Graphiken auch interaktiv editieren.

Die graphische Darstellung von Flächen über der Ebene geht mit dem `mesh`–Befehl:

```
mesh(x,y,Z)
```

Dabei sind `x,y` die Vektoren mit den x– und y–Koordinaten und in der Matrix `Z` stehen die zugehörigen Funktionswerte: $Z(i, j)$ ist der Funktionswert im Punkt $(x(i), y(j))$.

Will man in eine vorhandene Graphik weitere Bildelemente einfügen, dann nutzt man den Befehl `hold`:

```
hold on
```

Damit werden die vorhandenen Bildelemente nicht gelöscht.