Einführung in Matlab

Eberhard Luik

Vorwort: Ziel dieser Einführung ist es, den Studierenden die Grundlagen von Matlab zu vermitteln, um die in den verschiedenen Numerik-Vorlesungen behandelten Aufgabenstellungen selbständig am Computer zu lösen. Diese Grundkenntnisse werden in den einzelnen Lehrveranstaltungen weiter ausgebaut und vertieft.

In dieser Einführung unterstellen wir eine Rechnersituation, wie sie derzeit im CIP-Raum V203 gegeben ist: Linux als Betriebssystem und Matlab 6.1 (Release 12.1).

Inhaltsverzeichnis

1 Das Betriebssystem Linux

1.1 Einleitung

Wenn Sie in den Rechnerraum kommen, finden Sie in der Regel eingeschaltete Rechner vor, und das Betriebssystem fordert Sie zur Anmeldung am Rechner auf. Eventuell haben Sie einen schwarzen Bildschirm (wird der Rechner längere Zeit nicht benutzt, so wird der Bildschirm abgedunkelt). Dann bewegen Sie die Maus, und der Bildschirm wird aktiviert. Es kann auch vorkommen, daß Sie einen abgeschalteten Bildschirm vorfinden. Schalten Sie diesen dann ein (durch Betätigen des rechten Knopfes am Bildschirm).

Achtung: **Auf keinen Fall dürfen Sie den Rechner ausschalten oder mit der Reset-Taste zurücksetzen, denn dies kann zur Beschädigung des Betriebssystems oder der Anwendungssoftware führen.**

Ein Rechnersystem besteht im allgemeinen aus drei Komponenten:

Hardware,
Betriebssystem,
Anwendungssoftware.

Die *Hardware* ist der Teil, den Sie vor sich sehen: der eigentliche Rechner mit seinen Komponenten wie Prozessor, Speicherchips, Festplatte, Diskettenlaufwerk, CD-Laufwerk, usw., die Tastatur, der Bildschirm, die Maus, eventuell ein Drucker.

Die Hardware wird von gewissen Programmen gesteuert, welche als *Betriebssystem* bezeichnet werden. Außerdem sorgt das Betriebssystem dafür, daß andere Programme (z.B. ein von uns erstelltes Matlab-Programm) auf dem Rechner ausgeführt werden. Das Betriebssystem ist das Bindeglied zwischen Hardware und Anwendungssoftware. Wir verwenden **Linux** als Betriebssystem.

Die *Anwendungssoftware* dient im allgemeinen dazu, dem PC-Benutzer bestimmte Arbeiten zu erleichtern oder ganz abzunehmen. Dazu gehören zum Beispiel Programme zur Textverarbeitung, Tabellenkalkulation, Programmiersprachen oder auch Matlab.

1.2 Anmelden am Rechner

Zugangberechtigung

Um am Rechner zu arbeiten, benötigen Sie eine Zugangsberechtigung. Diese besteht aus einer *Jobnummer (user account)* und einem *Paßwort*. **Geben Sie diese auf keinen Fall weiter**. Alle Aktionen auf dem Rechner werden samt Jobnummer auf einem separaten Rechner (einem sogenannten Server) protokolliert, so daß Mißbrauch zurückverfolgt werden kann. Für alle Aktionen, die unter Ihrer Zugangsberechtigung durchgeführt werden, sind Sie verantwortlich.

Anmelden

Wenn am Bildschirm die Aufforderung zur Anmeldung (*login*) erscheint, so ist der Rechner bereit für einen Benutzer. Geben Sie dann über die Tastatur Ihre Jobnummer ein, und betätigen Sie die $\boxed{\hookleftarrow}$ – Taste (*Enter-Taste*). Danach wird Ihr Paßwort abgefragt. Dieses geben Sie nun ein und betätigen wieder die $\boxed{\hookleftarrow}$ – Taste. Aus Sicherheitsgründen erscheint das Paßwort nicht auf dem Bildschirm.

Falls Ihre Anmeldung korrekt war, erhalten Sie nun einen Bildschirm mit Fenstern und

Menüpunkten zum Anklicken. Besitzt ein Fenster die Überschrift `Konsole`, so können Sie in diesem Fenster Linux-Kommandos eingeben.

Paßwort änderen

Nach dem erstmaligen Anmelden am Rechner sollten sie zuerst Ihr Paßwort (das Sie von den Systembetreuern bekommen haben) abändern, damit niemand außer Ihnen das Paßwort kennt. Dazu überlegen Sie sich eine Buchstaben-Ziffern-Kombination (mindestens 9 Zeichen), welche kein sinnvolles Wort darstellt und welche von einem Dritten nicht "erraten" werden kann. Insbesondere sollten Sie nicht Ihren Vornamen, Ihr Geburtsdatum oder Ihre Telefonnummer als Paßwort auswählen. Merken Sie sich das neue Paßwort gut.

Bewegen Sie mit der Maus den Mauszeiger (schwarzer Pfeil) auf die oberste Zeile eines `Konsole` – Fensters und klicken Sie mit der linken Maustaste die Zeile mit dem Wort `Konsole` an. Damit wird dieses Fenster das aktive Fenster, d.h. eingegebener Text erscheint in diesem Fenster, und zwar an Stelle des kleinen schwarzen Rechtecks (genannt *Kursor*).

Um nun Ihr Paßwort zu ändern, geben Sie

```
passwd
```

ein. Dann wird zuerst nochmal Ihr bisheriges Paßwort abgefragt und danach zweimal das neue. Die eingegebenen Paßworte erscheinen nicht auf dem Bildschirm.

1.3 Speichermedien

Der Rechner hat verschiedene Speichermedien, die sich hinsichtlich ihrer Verfügbarkeit unterscheiden.

Hauptspeicher

Am schnellsten erfolgt der Zugriff auf den *Hauptspeicher*. Deshalb werden dorthin die aktuell verwendeten Programme kopiert (sofern der Platz dafür ausreicht). Auch werden z.B. die Zwischenergebnisse unseres gestarteten Matlab-Programmes dort gespeichert. Beim Ausschalten des Rechners oder bei Stromausfall gehen alle Daten des Hauptspeichers verloren.

Festplatte

Etwas langsamer ist der Zugriff auf die *Festplatte*. Dafür hat sie eine wesentlich größere Kapazität, und die Daten bleiben auch nach Ausschalten des Rechners bzw. bei Stromausfall erhalten. Eine Festplatte hat mittlerweile eine Kapazität von ca. 50 Gigabyte, d.h. es können rund 50 000 000 000 Zeichen gespeichert werden. Die Festplatte ist im Rechner eingebaut.

Disketten, CDs, Magnetbänder

Weitere Speichermedien sind eventuell *Disketten, CDs* und *Magnetbänder*. Der Zugriff ist hier zum Teil erheblich langsamer als auf die Festplatte. Dafür können diese ausgetauscht werden und bieten so beliebig viel Speicherkapazität. Geeignet sind diese Medien für Daten, die nur selten benötigt werden, oder auch für Sicherungskopien. Bei Bedarf werden die Daten dann auf die Festplatte kopiert.

1.4 Das Dateisystem

Datei und Dateiname

Eine *Datei* (engl. *file*) ist die kleinste Einheit, die wir auf einem Speichermedium ansprechen können. Sie enthält eine Menge von zusammengehörenden Daten (z.B ein Matlab-Programm). Jede Datei erhält einen Namen, den sogenannten *Dateinamen* (engl. *filename*). Dieser besteht aus maximal 255 Zeichen, dabei wird zwischen Groß- und Kleinbuchstaben unterschieden. Dateinamen sind zum Beispiel

```
aufgabe1.m
Aufgabe1.f90
eingabe.DAT
uebung1_ws2000.tex
```

Grundsätzlich ist die Wahl des Dateinamens beliebig. Es empfiehlt sich aber, den Dateinamen so zu wählen, daß an ihm bereits der Inhalt der Datei erkennbar ist. So bedeutet bei uns der Zusatz `.m` in dem obigen Beispiel, daß diese Datei ein Matlab-Programm enthält.

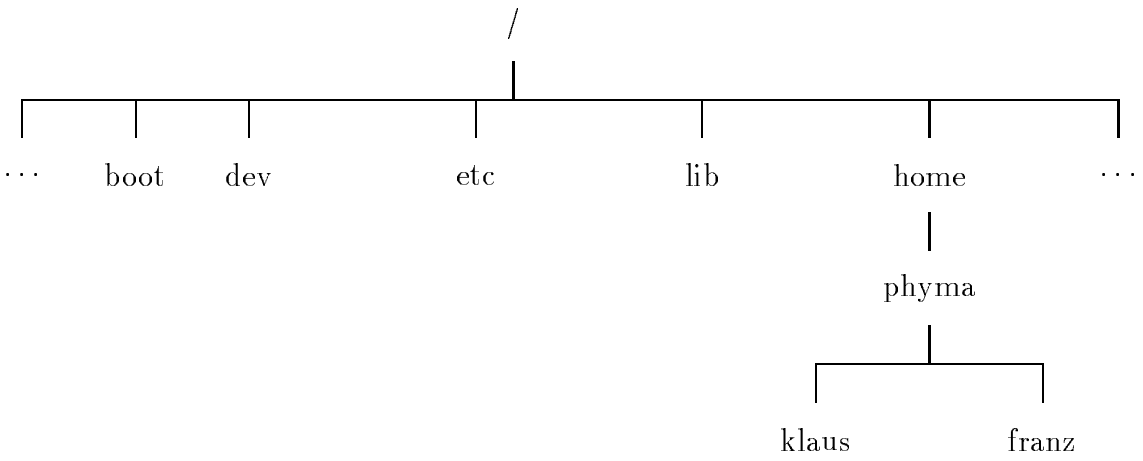
Neben dem Dateinamen besitzt eine Datei weitere Informationen, die vom Betriebssystem verwaltet werden. Dazu gehören Dateigröße (Anzahl der enthaltenen Zeichen), Datum, Besitzer, Zugriffsrechte (wer darf die Datei bearbeiten) und Art der Datei.

Die Gesamtheit aller Dateien wird als *Dateisystem* (engl. *file system*) bezeichnet.

Verzeichnisse

Um eine gewisse Ordnung in die Vielzahl von Dateien zu bringen, wird das Dateisystem unterteilt in *Verzeichnisse* (engl. *directories*), welche hierarchisch angeordnet werden. Jedes Verzeichnis besitzt einen Namen; es gelten dieselben Regeln wie für Dateinamen. Ein Verzeichnis besteht aus Dateien oder weiteren Verzeichnissen.

Das Dateisystem besitzt also eine Baumstruktur von Verzeichnissen, die zum Beispiel so aussehen kann:



Ganz oben erkennen Sie das Zeichen `/` . Dies ist das sogenannte *Wurzelverzeichnis*, das in der Hierarchie oberste Verzeichnis.

Nach dem Anmelden am Rechner landen Sie automatisch in einer Stelle der Baumstruktur, dem sogenannten *Home-Verzeichnis*. Wenn Sie den Usernamen `klaus` haben, so lautet Ihr Homeverzeichnis `/home/phyma/klaus` .

Pfade

Der Weg, den man bei der Suche nach einer Datei im Verzeichnisbaum zurücklegt, wird als *Pfad* (engl. *path*) bezeichnet. Es wird zwischen *absolutem* und *relativem* Pfad unterschieden. Der absolute Pfand beginnt immer mit dem Wurzelverzeichnis, der relative Pfad beginnt mit dem Verzeichnis, in dem man sich gerade befindet (dem sogenannten *aktuellen Verzeichnis*).

Befindet sich im obigen Beispiel im Verzeichnis `klaus` die Datei `prog1.m` und wir befinden uns bereits im Verzeichnis `/home/phyma`, so haben wir

absoluter Pfad: `/home/phyma/klaus`
relativer Pfad: `klaus`

Das in der Hierarchie höhere Verzeichnis wird zuerst genannt; die Verzeichnisse werden durch `/` getrennt.

Will man eine Datei ansprechen, welche sich nicht im aktuellen Verzeichnis befindet, so muß man den Pfad mit angeben. Im obigen Fall haben wir dann

`/home/phyma/klaus/prog1.m` .

1.5 Umgang mit Verzeichnissen

Aktuelles Verzeichnis abfragen

Das Kommando

`pwd`

liefert das aktuelle Verzeichnis. Eventuell wird Ihnen das aktuelle Verzeichnis automatisch gezeigt (hängt von der Rechnereinstellung ab), und zwar in der Zeile vor dem Kursor. Finden Sie dort zum Beispiel den Eintrag

`klaus@phyma25:~/programme >` ,

so bedeutet dies, daß sich der Benutzer `klaus` am Rechner mit dem Namen `phyma25` und dort im Verzeichnis `/home/phyma/klaus/programme` befindet. Dabei steht das Zeichen `~` als Abkürzung für das Homeverzeichnis.

Neues Verzeichnis anlegen

Durch das Kommando

`mkdir` <*name*>

wird im aktuellen Verzeichnis ein neues Verzeichnis mit dem Namen <*name*> angelegt, falls es noch nicht existiert. Befinden wir uns im Verzeichnis `/home/phyma/klaus` und geben dann

`mkdir ball`

ein, so gibt es anschließend das Verzeichnis `/home/phyma/klaus/ball`.

Verzeichnis wechseln

Man kann vom aktuellen Verzeichnis in das in der Hierarchie darüberliegende Verzeichnis (das sogenannte *Elternverzeichnis*) wechseln durch den Befehl

`cd ..` .

In das eine Stufe darunterliegende Verzeichnis gelangen wir durch

`cd` <*name*> .

Befinden wir uns zum Beispiel im Verzeichnis `/home/phyma/klaus`, so gelangen wir mit

`cd ball`

ins Verzeichnis `/home/phyma/klaus/ball` . Durch Angabe des kompletten Pfades kann man an jede Stelle des Verzeichnisbaumes springen. So gelangen wir durch

```
cd /usr/local/bin
```

sofort in das entsprechende Verzeichnis.

Verzeichnisse löschen

Ein Verzeichnis, welches keine Dateien oder Verzeichnisse erhält, kann vom Elternverzeichnis aus durch

`rmdir` <*name*>

gelöscht werden.

Inhalt eines Verzeichnisses anschauen

Mit dem Befehl

```
ls -al
```

wird der Inhalt des aktuellen Verzeichnisses angezeigt, d.h. alle Dateien und weitere Verzeichnisse sowie deren Eigenschaften. Wir erhalten dann Informationen der folgenden Form:

```
drwxr-xr-x   2 klaus    numerik      1024 Mai 28 14:13 ball
drwxr-xr-x   2 klaus    numerik      1024 Jun 14 09:19 programme
-rw-r-----   1 klaus    numerik     87326 Jul 11 11:15 prog1.f90
-rw-r-----   1 klaus    numerik      1703 Mai  9 09:32 readme
-rw-r-----   1 klaus    numerik      1337 Jul  3 10:11 help.txt
```

Enthält das aktuelle Verzeichnis viele Dateien und Verzeichnisse, so bewirkt das Kommando

```
ls -al | more
```

eine seitenweise Ausgabe des Inhalts. Das Weiterblättern erfolgt mit der Leertaste.

Den Datei- bzw. Verzeichnisnamen findet man in der rechten Spalte. Steht in der linken Spalte als erstes Zeichen ein `d`, so handelt es sich um ein Unterverzeichnis. Im obigen Fall sind also `ball` und `programme` Unterverzeichnisse des aktuellen Verzeichnisses, während `prog1.f90, readme` und `help.txt` Dateien darstellen.

Weitere Informationen sind Zugriffsrechte (vgl. unten), Eigentümer der Datei, Gruppenname (die einzelnen Benutzer werden zu Gruppen zusammengefaßt), Größe der Datei und Datum der letzten Änderung.

Soll der Inhalt eines anderen Verzeichnisses angezeigt werden, so lautet das Kommando

`ls -al` <*pfad/verzeichnis*>

1.6 Umgang mit Dateien

Zugriffsrechte

Jede Datei (und jedes Verzeichnis) ist mit Zugriffsrechten versehen. Sie regeln, ob ein Benutzer die Datei lesen (Abkürzung `r` für *read*), beschreiben (`w` für *write*) oder das enthaltene Programm auf dem Rechner starten darf (`x` für *execute*). Im obigen Beispiel haben wir die die Zeile

```
-rw-r-----   1 klaus    numerik      1337 Jul  3 10:11 help.txt
```

Die Dateizugriffsrechte stehen in der ersten Spalte. Die Zeichen 2, 3 und 4 geben die Berechtigungen für den Eigentümer (Benutzer) an (Lesezugriff, Schreibzugriff, Ausführungsberechtigung). Ein `-` bedeutet keine Berechtigung.
Die Zeichen 5, 6 und 7 geben die Berechtigungen für die Gruppenangehörigen an, die Zeichen 8, 9 und 10 die Berechtigungen für alle anderen Benutzer.

Im obigen Beispiel darf der Eigentümer (Benutzer) `klaus` die Datei `help.txt` lesen und beschreiben (und damit auch löschen). Alle Benutzer, die zur Gruppe `numerik` gehören, dürfen die Datei `help.txt` nur lesen, während die restliche Benutzer keine Berechtigungen für diese Datei haben.

Der Eigentümer einer Datei kann die Zugriffsrechte ändern mit Hilfe des Kommandos `chmod`, worauf wir hier jedoch nicht weiter eingehen.

Datei kopieren

Mit dem Befehl

`cp` *<datei1>* *<datei2>*

wird im aktuellen Verzeichnis die Datei mit dem Namen *<datei1>* in eine zweite Datei mit dem Namen *<datei2>* kopiert. Befindet sich die zu kopierende Datei in einem anderen Verzeichnis, so ist der Pfad mit anzugeben, also

`cp` *<pfad/datei1>* *<datei2>*

So wird zum Beispiel durch das Kommando

```
cp /usr/local/bin/readme help.txt
```

die Datei `readme` aus dem Verzeichnis `/usr/local/bin` in das aktuelle Verzeichnis kopiert und erhält dort den Namen `help.txt`.

Datei löschen

Durch das Kommando

`rm` *<datei>*

wird im aktuellen Verzeichnis die entsprechende Datei gelöscht (falls der Benutzer für diese Datei eine Schreibberechtigung hat). Gehen Sie bitte mit diesem Befehl sehr vorsichtig um, denn der Inhalt einer gelöschten Datei ist verloren und kann nicht rekonstruiert werden. Mit dem Befehl

```
rm *.txt
```

werden im aktuellen Verzeichnis alle Dateien mit dem Zusatz `.txt` gelöscht.

Datei umbenennen

Durch das Kommando

`mv` *<datei1>* *<datei2>*

wird im aktuellen Verzeichnis die Datei mit dem Namen *<datei1>* umbenannt in *<datei2>* . Befindet sich die Datei in einem anderen Verzeichnis, so ist der Pfad mit anzugeben.

Datei lesen

Den Inhalt einer Datei kann man durch den Befehl

`more` *<datei1>*

anschauen. Die Ausgabe erfolgt seitenweise, umgeblättert wird mit der Leertaste.

Mehr Möglichkeiten bietet das Kommando

`less` <*datei1*> .

Hier kann man sich mit den Pfeiltasten im Text vorwärts bzw. rückwärts bewegen. Ferner kann nach Wörtern im Text gesucht werden.

Es gibt auch Dateien, welche keinen (für den Benutzer) lesbaren Text enthalten, sondern einen vom Rechner interpretierbaren Code. Diese können natürlich mit den obigen Kommandos auch nicht gelesen werden.

Datei beschreiben bzw. ändern

Zum Beschreiben bzw. Abändern einer Datei gibt es spezielle Programme, nämlich die sogenannten *Editoren*. Dieses Thema ist Gegenstand des nächsten Paragraphen.

Selbstverständlich kann man zum Lesen einer Datei auch einen Editor verwenden.

Datei drucken

Eine Datei (welche normalen Text enthält) wird mit dem Kommando

`a2ps` <*datei*>

auf dem im CIP-Raum V203 vorhandenen Laserdrucker ausgegeben. Geben Sie bitte dieses Kommando nur einmal ein, selbst wenn die Datei nicht sofort gedruckt wird (weil zum Beispiel der Drucker nicht betriebsbereit ist), denn Ihr Druckauftrag wird auf dem Server gespeichert und auf jeden Fall ausgeführt. Eine Postskript-Datei (zu erkennen an dem Zusatz `.ps` oder `.eps`) wird durch den Befehl

`lpr` <*datei*>

ausgegeben.

1.7 Diskettenlaufwerk ansprechen

In Linux wird eine Diskette (Floppy) mit den sogenannten *mtools* behandelt, welche den DOS-Befehlen entsprechen und sich von diesen durch ein vorgestelltes `m` unterscheiden.

Zum Kopieren von Dateien wird der Befehl `mcopy` verwendet. So wird durch

```
mcopy a:aufg1.m help.m
```

die Datei `aufg1.m` von der Diskette ins aktuelle Verzeichnis der Festplatte kopiert und erhält dort den Namen `help.m`. Umgekehrt wird durch

```
mcopy prog1.m a:aufg1.m
```

die Datei `prog1.m` auf die Diskette unter dem Namen `aufg1.m` kopiert.

Die folgenden weiteren Kommandos beziehen sich immer auf das Diskettenlaufwerk:

`mcd` <*verzeichnis*>	Wechsel in ein anderes Verzeichnis
`mdel` <*datei*>	Löschen einer Datei
`mmd` <*verzeichnis*>	Anlegen eines Unterverzeichnisses
`mrd` <*verzeichnis*>	Löschen eines Unterverzeichnisses
`mdir`	Anzeigen des Inhaltes der Diskette

1.8 Prozesse

Unter einem *Prozeß* versteht man ein auf dem Rechner gestartetes Programm oder Kommando. Zum Beispiel wird jedes Fenster durch einen Prozeß gesteuert. Auf dem Rechner laufen in der Regel sehr viele Prozesse gleichzeitig, die wenigsten davon sind von Ihnen.

Mit dem Kommando

```
ps uax
```

werden alle zur Zeit laufenden Prozesse aufgelistet. Wir erhalten Informationen der folgenden Art:

```
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
....
at         181  0.0  0.2  1264   552  ?  S    09:14   0:00 /usr/sbin/atd
bin        122  0.0  0.1  1152   404  ?  S    09:14   0:00 /sbin/portmap
klaus      276  0.0  0.5  2276  1320   1 S    09:15   0:00 -bash
klaus      349  0.0  0.8  4028  2108   1 S    09:15   0:00 xterm -ls
klaus      880  0.0  0.1  1540   476  ?  R    10:47   0:15 aufgabe1
root         2  0.0  0.0     0     0  ?  SW   09:13   0:00 (kflushd)
root         3  0.0  0.0     0     0  ?  SW   09:13   0:00 (kupdate)
....
```

Sie erkennen zum Beispiel, daß der Benutzer `klaus` das Programm `aufgabe1` gestartet hat und der Computer dieses Programm zur Zeit ausführt. Außerdem hat dieses Programm die Prozeßnummer 880.

Im Normalfall hat der Computer nach einer gewissen Zeit alle Berechnungen des Programms durchgeführt. Damit wird dann auch der zugehörige Prozeß beendet. Bedingt durch einen Programmierfehler kann es vorkommen, daß der zugehörige Prozeß nie beendet wird. In diesem Fall muß der Benutzer (oder der für den Computer zuständige Betreuer, der sogenannte *Systemadministrator*) den Prozeß von Hand löschen. Dazu dient das Kommando

`kill -9` *<PID>* .

Dabei ist *<PID>* die Prozeßnummer des zu löschenden Prozesses.

Im obigen Beispiel wird durch das Kommando

```
kill -9 880
```

der Prozeß gelöscht, welcher das Programm `aufgabe1` ausführt. Der Benutzer darf nur die ihm gehörenden Prozesse löschen.

Achtung: Auf keinen Fall dürfen Sie den Rechner ausschalten oder mit der Reset-Taste den Rechner neu starten. Denn dadurch würden alle laufenden Prozesse gelöscht, was zu Datenverlust und im schlimmsten Fall zur Beschädigung des Betriebssystems führen kann.

1.9 Weitere Informationen zu Linux

Sie erhalten am Rechner ausführliche Informationen (in englisch) zu jedem Linux-Kommando durch Eingabe von

`man` *<kommando>* .

Insbesondere lassen die oben geschilderten Linux-Kommandos viele Varianten (sogenannte *Optionen*) zu, die mit dem `man` - Kommando aufgelistet werden. So erhalten Sie zum Beispiel durch

```
man chmod
```

Informationen zum Ändern der Dateizugriffsrechte.

Für Linux-Anfänger gut geeignet (und preislich günstig) ist das Lehrbuch von Göstenmeier und Rehm: Das Einsteigerseminar Linux, bhv-Verlag.

2 Der Editor namens Emacs

Mit einem Editor können wir eine neue Datei eröffnen und in diese etwas hineinschreiben oder eine bereits existierende Datei lesen bzw. ändern. Inzwischen gibt es eine Reihe von Editoren. Wir verwenden den Editor namens **Emacs**, welcher im Lieferumfang von Linux enthalten ist. Es gibt verschiedene Varianten, wir wählen den **GNU Emacs**.

Die Eingabe von Befehlen nach dem Starten des Editors kann sowohl mit der Maus als auch mit Hilfe von Tastenkombinationen erfolgen. Wir beschreiben hier hauptsächlich die Verwendung der Maus.

2.1 Aufruf des Editors

Der Aufruf des Editors erfolgt durch den Befehl

`emacs` <*name*>

oder

`emacs` <*name*> &

Dabei ist <*name*> der Name der zu bearbeitenden Datei. Im zweiten Fall wird der Editor im Hintergrund gestartet. Das Fenster, aus dem der Emacs gestartet wird, steht weiterhin für die Eingabe von Linux-Befehlen zur Verfügung. Im ersten Fall steht dagegen das betreffende Fenster erst nach Beendigung des Editors wieder zur Verfügung.

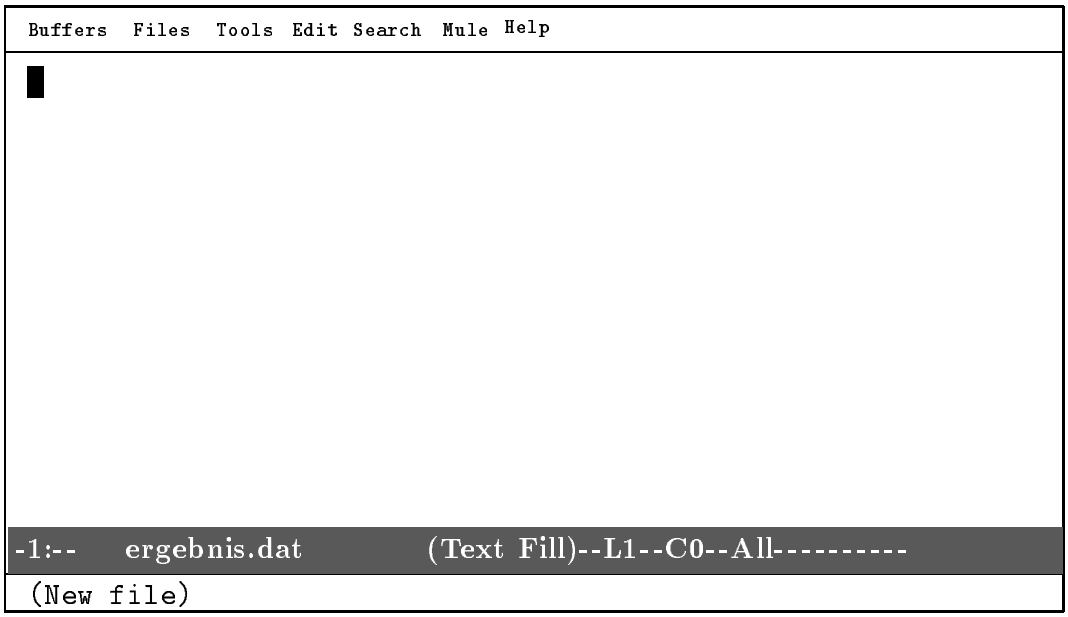
Soll beispielsweise die Datei `ergebnis.dat` bearbeitet werden, so lautet der Aufruf

```
emacs ergebnis.dat
```

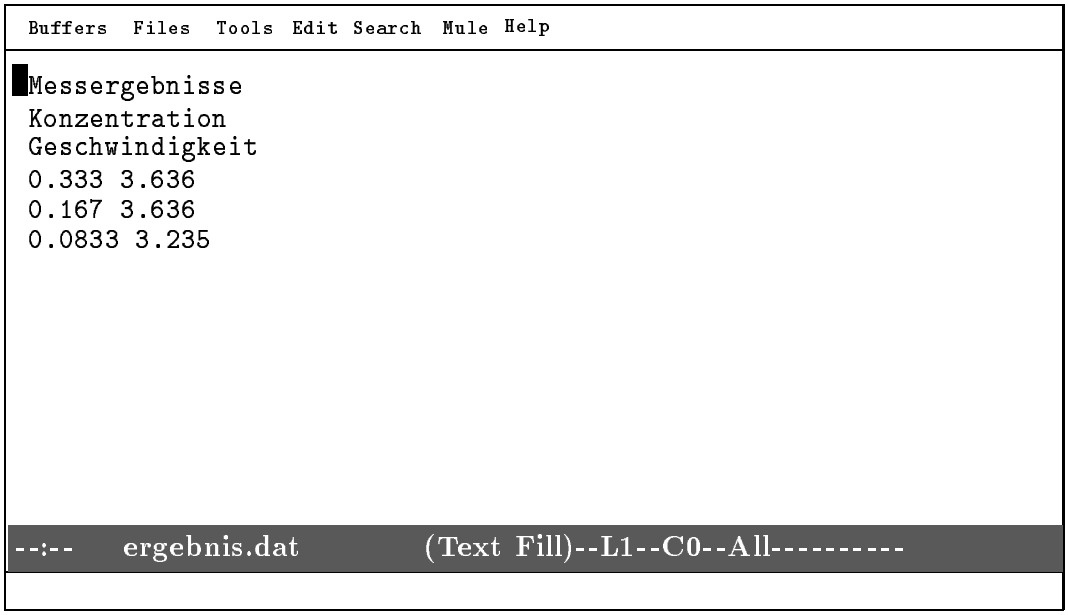
beziehungsweise

```
emacs ergebnis.dat &
```

Existiert im aktuellen Verzeichnis bisher die Datei mit diesem Namen noch nicht, so wird sie angelegt. Wir erhalten dann ein neues Fenster (von uns als Emacs-Fenster bezeichnet), welches folgende Gestalt hat:



Existiert beim Editoraufruf bereits eine Datei mit dem Namen `ergebnis.dat`, so wird sie uns zur Bearbeitung (Korrektur) vorgelegt; der Bildschirm hat dann die folgende Gestalt:



Wir erkennen am oberen Rand des Fensters eine eingerahmte Zeile (die sogenannte Kopfzeile) mit einigen Begriffen wie `Buffers, Files`, ... Durch Anklicken mit der linken Maustaste wird ein Untermenü eröffnet, welches uns eine Liste von möglichen Befehlen angibt. Durch Anklicken mit der linken Maustaste wird dann der entsprechende Befehl ausgeführt (siehe unten).

Danach folgt ein großes weißes Feld (falls die zu bearbeitende Datei noch nicht existiert) mit einem kleinen schwarzen Rechteck am linken oberen Rand, welches als Kursor bezeichnet wird. An dieser Stelle werden die Zeichen eingefügt, welche über die Tastatur eingegeben werden. Existiert beim Emacs-Aufruf die Datei bereits, so wird in diesem Feld ihr Inhalt angezeigt.

Unten erkennen wir eine schwarz unterlegte Zeile mit einigen Informationen, z.B. den Dateinamen der zu bearbeitenden Datei, den Bearbeitungsmodus (hier Text Fill), die Kursorposition (hier L1 für Zeile 1 und C0 für Spalte 0). Das `All` bedeutet, daß der gesamte Inhalt der Datei im Emacs-Fenster angezeigt wird.

Ganz unten befindet sich noch eine weitere Zeile (ein sogenannter Mini-Puffer). Diese wird von Emacs für Informationen und Fehlermeldungen verwendet.

2.2 Kursorbewegungen

Mit den Pfeiltasten [↑] [↓] [→] [←] wird der Kursor auf dem Bildschirm bewegt. Man kann den Kursor auch mit Hilfe der Maus plazieren. Dazu wird der Mauszeiger (schwarzer Pfeil) durch Bewegen der Maus an die gewünschte Stelle gebracht; durch Klicken der linken Maustaste wird der Kursor an diese Stelle gesetzt.

Mit den Tasten [Bild ↓] und [Bild ↑] wird auf dem Bildschirm eine ganze Seite vor- bzw. zurückgeblättert (unter der Voraussetzung, daß die zu bearbeitende Datei entsprechend groß ist).

Die [Ende] – Taste bewegt den Kursor an das Ende und die [Pos1] – Taste an den Anfang der betreffenden Zeile.

2.3 Einfügen und Überschreiben

Wird nun über die Tastatur Text eingetippt, so wird dieser an der Kursorposition eingefügt (Einfügemodus). Betätigen der [Einfg] – Taste bewirkt Überschreiben des Textes. In der schwarz unterlegten Fußzeile erscheint dann die Anzeige **(Text Ovwrt Fill)** . Nochmaliges Betätigen der [Einfg] – Taste bewirkt wieder ein Umschalten auf den Einfügemodus.

Das Betätigen der [↵] – Taste bei der Texteingabe bewirkt, daß der Kursor in die nächste Zeile springt.

2.4 Löschen eines Zeichens

Durch Eingabe von [Entf] wird das Zeichen gelöscht, welches an der Kursorposition steht. Durch Betätigen von [←] (Backspace-Taste) wird das Zeichen links vom Kursor gelöscht; gleichzeitig wird der Kursor und der nachfolgende Text um ein Zeichen nach links geschoben. Zum Löschen größerer Textbereiche gibt es weitere Möglichkeiten (vgl. 2.11).

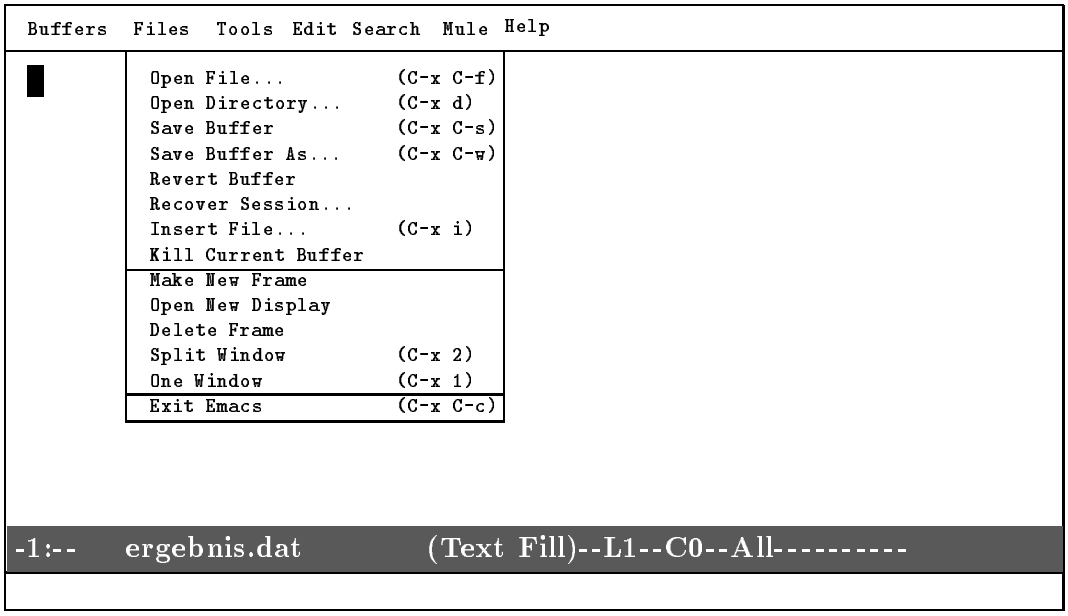
2.5 Puffer

Emacs kann mehrere Dateien gleichzeitig laden. Dazu werden Puffer (engl. *buffers*) verwendet. In jedem Puffer befindet sich eine Datei. Der Name des Puffers ist mit dem Dateinamen identisch. Der Inhalt des aktuellen Puffers wird im Emacs-Fenster angezeigt. Um nun einen anderen Puffer (eine andere geladene Datei) in das Emacs-Fenster zu holen, wird mit der linken Maustaste in der Kopfzeile das Feld `Buffers` angeklickt (die linke Maustaste bleibt gedrückt). Es erscheint eine Liste von möglichen Puffern. Wir bewegen nun den Mauszeiger (bei gedrückter linker Maustaste) auf den gewünschten Namen und lassen dann die linke Maustaste los. Damit wird dieser Puffer zum aktuellen Puffer und erscheint im Emacs-Fenster.

2.6 Änderungen speichern

Die Texteingaben bzw. Änderungen erfolgen im Puffer. Die Datei auf der Festplatte wird dadurch noch nicht verändert. Bei Verlassen des Editors oder bei Stromausfall gehen diese Änderungen verloren. Deshalb muß man den Inhalt der Puffer in die entsprechenden Dateien auf der Festplatte speichern. Dazu klicken wir mit der linken Maustaste in der Kopfzeile den Menüpunkt `Files` an und lassen diese Taste gedrückt. Es erscheint ein Untermenü, und das Emacs-Fenster hat die auf der folgenden Seite abgebildete Gestalt.

Wir erkennen eine Reihe möglicher Emacs-Kommandos, die mit der Maus ausgewählt und ausgeführt werden können. Bei einigen Befehlen stehen in Klammer noch die Tasten-Kombinationen, welche den entsprechenden Befehl ohne Verwendung der Maus ausführen (vergleiche 2.13).



Um nun unseren geänderten Text zu speichern, wählen wir bei gedrückter Maustaste mit dem Mauszeiger das Feld

```
Save Buffer      (C-x C-s)
```

aus und lassen dann die Maustaste los. Dadurch wird der Inhalt des aktuellen Puffers in die (in der Fußzeile) angegebene Datei auf der Festplatte gespeichert. Der ursprüngliche Inhalt der Datei wird dabei überschrieben und steht somit nicht mehr zur Verfügung.

Soll der ursprüngliche Dateiinhalt nicht gelöscht werden, so muß der aktuelle Puffer in eine andere (neue) Datei gespeichert werden. Dazu wählen wir mit dem Mauszeiger das Feld

```
Save Buffer As...      (C-x C-w)
```

.

Der Kursor springt dann in die unterste Zeile des Emacs-Fensters und fordert uns auf, einen Dateinamen einzugeben. Unter diesem Dateinamen wird dann der Inhalt des aktuellen Puffers auf der Festplatte gespeichert.

2.7 Weitere Datei laden

Um eine weitere Datei von der Festplatte in den Emacs-Editor zu laden, klickt man in dem Untermenü zu `Files` (vgl. oben) das Feld

```
Open File...      (C-x C-f)
```

an. Der Kursor springt in die unterste Zeile und verlangt die Eingabe eines Dateinamens. Diese Datei wird dann in einen neuen Puffer geladen und im Emacs-Fenster angezeigt.

2.8 Andere Datei in aktuellen Text einfügen

Um den Inhalt einer anderen Datei in den aktuellen Text (an der Kursor-Position) einzufügen, klickt man in dem Untermenü zu `Files` (vgl. oben) das Feld

```
Insert File...      (C-x i)
```

an. Der Kursor springt in die unterste Zeile und verlangt die Eingabe eines Dateinamens.

2.9 Beenden von Emacs

Hierzu klickt man in dem Untermenü zu `Files` (vgl. oben) das Feld

```
Exit Emacs      (C-x C-c)
```

an. Falls das Speichern vergessen wurde, so erfolgt eine Warnung, und man kann es an dieser Stelle nachholen.

2.10 Unterfenster

Das Emacs-Fenster kann in Unterfenster unterteilt werden. Dazu dienen im Untermenü zu `Files` die Felder

```
Split Window      (C-x 2)
One Window      (C-x 1)
```

Einige Emacs-Befehle nehmen automatisch eine Fensterunterteilung vor und benutzen ein Unterfenster für Informationen. Um diese Unterteilung wieder aufheben, wird der Kursor in dem gewünschten Unterfenster plaziert und dann der zweite Befehl angeklickt.

2.11 Textbereiche kopieren, löschen oder versetzen

Häufig besteht beim Bearbeiten einer Datei der Wunsch, einen bestimmten Teil des Textes zu markieren (z.B. beim Kopieren, Löschen oder Verschieben).

Dazu bewegt man den Mauszeiger auf das erste Zeichen des zu markierenden Textes. Während die linke Maustaste gedrückt bleibt, wird der Mauszeiger auf das Ende des zu markierenden Textes bewegt und die Maustaste dann losgelassen. Der markierte Text erscheint grau unterlegt. Erstreckt sich der zu markierende Text über mehrere Zeilen, so werden immer ganze Zeilen markiert.

Zur Bearbeitung von markierten Textbereichen dient in der Kopfzeile das Feld `Edit`, welches wieder ein Untermenü liefert.

Zum Löschen des markierten Bereiches verwendet man den Untermenüpunkt

```
Cut      (C-w)
```

Damit wird der markierte Text entfernt und in den sogenannten *Kill-Puffer* gespeichert, ist also noch nicht verloren. Allerdings wird dann der bisher im Kill-Puffer gespeicherte Text gelöscht. Mit dem Untermenüpunkt

```
Undo      (C-_)
```

kann man das Löschen wieder rückgängig machen (dies gilt nur für den zuletzt gelöschten Bereich).

Mit dem Untermenüpunkt

```
Copy
```

wird der markierte Bereich in den Kill-Puffer geschrieben, bleibt aber an der bisherigen Stelle erhalten.

Der markierte (bzw. gelöschte) Bereich kann an einer anderen Stelle des Textes eingefügt werden. Dazu wird der Kursor an der einzufügenden Stelle plaziert und dann der Untermenüpunkt

```
Paste      (C-_)
```

angeklickt.

2.12 Suchen und Ersetzen von Text

Emacs bietet die Möglichkeit, in der geladenen Datei einen bestimmten Text zu suchen und ihn eventuell durch einen anderen Text zu ersetzen. Die Suche beginnt immer ab der Kursorposition und kann vorwärts oder rückwärts erfolgen. Dazu dient in der Kopfzeile das Feld `Search`, welches wieder ein Untermenü öffnet. Auf Einzelheiten gehen wir hier nicht ein.

2.13 Emacs-Kommandos mittels Tastenkombinationen

Alle Emacs-Kommandos können auch über die Tastatur eingegeben werden. Hierfür sind allerdings gewisse Tastenkombinationen erforderlich, um das Emacs-Kommando von der normalen Texteingabe zu unterscheiden. Dazu dienen die beiden Tasten Strg (*Control-Taste*) und Esc (*Escape-Taste*). In der Handhabung der beiden Tasten besteht ein fundamentaler Unterschied: Die Contral-Taste bleibt gedrückt, während eine weitere Taste betätigt wird (analog der Eingabe von Großbuchstaben). So bedeutet zum Beispiel die Tastenkombination `C-y`, daß bei gedrückter Strg - Taste zusätzlich noch die y - Taste betätigt wird.

Dagegen bedeutet die Tastenkombination `M-b`, daß zuerst kurz die Esc - Taste und danach die b - Taste gedrückt wird.

Einige wichtige Tastenkombinationen sind:

`C-x C-c`	Emacs beenden.
`C-x C-s`	speichert den Inhalt des aktuellen Puffers in die Datei gleichen Namens.
`C-x C-w`	speichert den Inhalt des aktuellen Puffers in eine Datei, deren Namen abgefragt wird.
`C-x C-f`	in einen neuen Puffer wird eine Datei geladen.
`C-k`	löscht ab Kursorposition bis zum Zeilenende (gelöschter Text wird in den Kill-Puffer geschrieben).
`C-k C-k`	löscht ab Kursorposition bis zum Zeilenende einschließlich (unsichtbarer) Zeilenende-Marke. Der folgede Text wird um eine Zeile noch oben gerückt.
`C-y`	fügt den Inhalt des Kill-Puffers an der Kursorposition ein.
`C-g`	bricht das aktuelle Emacs-Kommando ab.
`M-g`	plaziert den Kurser am Anfang der abgefragten Zeile.

2.14 Emacs-Kommando abbrechen

Jedes Emacs-Kommando (selbst wenn es mit der Maus angeklickt wurde) kann durch die Tastenkombination `C-g` abgebrochen werden. Dieses Kommando sollten Sie immer dann versuchen, wenn das Emacs-Fenster nicht mehr richtig arbeitet bzw. überhaupt nicht mehr reagiert.

2.15 Die Datei `.gnu-emacs`

Beim Aufruf von Emacs wird in Ihrem Home-Verzeichnis die Datei

```
.gnu-emacs
```

gelesen (falls vorhanden). In dieser Datei können gewisse Voreinstellungen angegeben werden. Zum Beispiel können die Funktions-Tasten F1, F2, usw. nach eigenem Geschmack mit Emacs-Kommandos belegt werden. Dazu sind allerdings fundierte Kenntnisse in Emacs und eventuell Grundkenntnisse in der Programmiersprache Emacs-Lisp erforderlich. Der Anfänger sollte diese Datei nicht verändern.

2.16 Weitergehende Informationen zu Emacs

Weitere Informationen (in englischer Sprache) zu Emacs erhalten Sie über den Menüpunkt `Help` in der Kopfzeile des Emacs-Fensters. Aus dem Untermenü klicken Sie dann

```
Emacs Tutorial      (C-h t)
```

an. Im Emacs-Fenster erhalten Sie dann eine Beschreibung der Emacs-Kommandos.

3 Grundlagen von Matlab

3.1 Vorbereitungen

Vor dem erstmaligen Aufruf von Matlab sollten Sie sich ein Verzeichnis mit dem Namen `matlab` anlegen. Dazu geben Sie nach dem Anmelden am Rechner das Kommando

```
mkdir matlab
```

ein.

Vor jedem Aufruf von Matlab wechseln Sie mit

```
cd matlab
```

in dieses Verzeichnis.

3.2 Matlab starten

Durch Eingabe des Kommandos

```
matlab
```

wird Matlab gestartet. Nach einigen Sekunden erscheint auf dem Bildschirm das Matlab-Fenster, welches wieder in drei Unterfenster aufgeteilt ist. Das momentan aktive Unterfenster ist mit einem blauen Balken am oberen Rand gekennzeichnet. Die Größe der Unterfenster lassen sich den individuellen Wünschen anpassen. Dazu klicken Sie mit der linken Maustaste auf den Rand zwischen den Fenstern und verschieben diesen mit der Maus (bei gedrückter linker Taste).

Für uns interessant ist zunächst nur das Unterfenster mit der Überschrift

Command Window .

Hier geben wir die Matlab-Befehle ein.

3.3 Matlab beenden

Um Matlab zu beenden, klicken Sie mit der linken Maustaste im MATLAB-Fenster den Menüpunkt **File** an und wählen dann (mit der linken Maustaste) in dem erscheinenden Untermenü den Punkt **Exit MATLAB** aus.

Alternativ können Sie im Command Window den Befehl

```
exit
```

eingeben.

Beim Verlassen wird die aktuelle Matlab-Konfiguration (Größe und Anzahl der Unterfenster) gespeichert und erscheint so beim nächsten MATLAB-Aufruf wieder.

3.4 Online-Hilfe

Eine komplette Dokumentation von MATLAB (in englischer Sprache) befindet sich auf dem Rechner. Diese wird gestartet durch Eingabe von

```
helpdesk
```

im Command Window. Es wird dann ein neues Fenster **Help** eröffnet. Um dieses Fenster wieder zu schließen, klicken wir mit der linken Maustaste den Menüpunkt **File** und dann den Unterpunkt **Close Help** an.

Ist man nur an Informationen zu einem bestimmten Matlab-Befehl interessiert, so gibt es zwei Möglichkeiten:

(i) Die Eingabe von

`doc` <*kommando*>

im Command Window liefert ausführliche Information in einem neuen Help-Window. Dabei bedeutet <*kommando*> ein Matlab-Kommando. Z.B. erhalten Sie durch

```
doc inv
```

Information über das Matlab-Kommando `inv`, welches die inverse Matrix berechnet.

(ii) Das Kommando

`help` <*kommando*>

gibt die Information im Command Window aus. Z. B. erhalten wir mittels

```
help inv
```

die folgende Ausgabe:

```
>> help inv

 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or
    nearly singular.

    See also SLASH, PINV, COND, CONDEST, LSQNONNEG, LSCOV.
```

Mit dem Befehl

`lookfor` <*schlüsselwort*>

wird nach (englischen) Schlüsselwörtern gesucht. So liefert beispeilsweise

```
lookfor sine
```

die folgenden Informationen:

```
COS    Inverse cosine.
ACOSH  Inverse hyperbolic cosine.
ASIN   Inverse sine.
ASINH  Inverse hyperbolic sine.
COS    Cosine.
COSH   Hyperbolic cosine.
SIN    Sine.
SINH   Hyperbolic sine.
TFFUNC time and frequency domain versions of a cosine modulated Gaussian pulse.
DST    Discrete sine transform.
IDST   Inverse discrete sine transform.
```

3.5 Eingabe von Matlab-Kommandos

Die Eingabe von Matlab-Kommandos erfolgt im Command Window. Matlab arbeitet interaktiv, d.h. jeder Matlab-Befehl wird nach Betätigen der Eingabetaste sofort ausgeführt (im Gegensatz zu den Programmiersprachen wie Fortran, Pascal oder C, bei denen das komplette Programm zuerst übersetzt werden muß).

Dabei ist zu beachten:

- Es wird zwischen Groß- und Kleinbuchstaben unterschieden.
- Das Zeichen % dient als Kommentarzeichen: der Text rechts davon bis zum Zeilenende wird als Kommentar betrachtet (und somit nicht als Matlab-Befehl interpretiert).
- Ein Matlab-Befehl kann sich über mehrere Zeilen erstrecken, muß dann jedoch am Zeilenende mit dem Zeichen ... markiert werden.
- In einer Zeile dürfen mehrere Matlab-Befehle stehen. Diese müssen jedoch durch einen Strichpunkt (bzw. ein Komma) getrennt werden.
- Ein Matlab-Kommando kann mit einem Strichpunkt abgeschlossen werden. Dann wird das Ergebnis dieses Befehls nicht im Command Window angezeigt. Ohne Strichpunkt am Ende erscheint das Ergebnis im Command Window.

Hier sind einige Beispiele für Matlab-Kommandos:

```
x = 1.5
X = 1e-7;
a = 1.5; b = 4.5; c = 3.8; s = (a+b+c)/3; % Mittelwert
u = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
      -1/8 + 1/9 - 1/10 + 1/11 - 1/12;
u
```

`X` und `x` sind verschiedene Variablen. Die letzte Zeile bedeutet, daß der Wert der Variablen `u` im Command Window ausgegeben wird.

In Matlab können auch Linux-Kommandos eingegeben werden. Diese werden mit einem vorangestellten ! gekennzeichnet. Zum Beispiel wird durch

```
!ls -al
```

der Inhalt des aktuellen Verzeichnisses im Command Window aufgelistet.

3.6 Matlab-Programme

Es besteht die Möglichkeit, mehrere Matlab-Kommandos in eine Datei zu schreiben (mit Hilfe eines Editors). Diese muß den Zusatz `.m` haben und wird als *Matlab-Skript* bezeichnet. Es gibt zwei Möglichkeiten:

(i) **Verwendung eines Linux-Editors (z.B. emacs)**
Dies hat den Vorteil, daß ein Matlab-Skript erstellt werden kann, ohne Matlab selbst gestartet zu haben. Sie können Ihr Programm zu Hause eingeben und im CIP-Pool dann unter Matlab ausführen.

(ii) **Verwendung des Matlab-Editors**
Matlab besitzt einen eigenen Editor. Dieser wird aktiviert durch Anklicken (mit der linken Maustaste) von **File**. In dem erscheinenden Untermenü wird zuerst **New** und dann **M-file** angeklickt. Dadurch wird eine neues Fenster eröffnet, und wir können nun das Programm eintippen. Soll ein bereits existierendes Matlab-Skript geändert werden, so klicken wir statt **New** den Menüpunkt **Open...** an und geben dann den Dateinamen ein.

Als Beispiel legen wir eine Datei namens `beispiel1.m` an, welche einige Matlab-Kommandos enthält:

```
% Es werden Messergebnisse gezeichnet.
x = [0.333 0.167 0.0833 0.0416 0.0208 0.0104 0.0052];
y = [3.636 3.636 3.236 2.666 2.114 1.466 0.866];
plot(x,y,'+');
xlabel(' Konzentration c','Fontsize',15);
ylabel(' Geschwindigkeit \nu','Fontsize',15);
title('Messergebnisse','Fontsize',15);
```

Dieses Matlab-Programm wird durch Eingabe von

```
beispiel1
```

gestartet (im Command Window). Die einzelnen Matlab-Kommandos werden später erläutert.

3.7 Matlab Workspace

Matlab besitzt einen Arbeitsspeicher (*Matlab Workspace*), in dem alle während einer Sitzung erzeugten Variablen mit ihrem zuletzt zugewiesenen Wert abgelegt werden. Den Workspace kann man sich im linken oberen Matlab-Unterfenster anzeigen lassen. Dazu klicken Sie mit der linken Maustaste auf das Feld **Workspace**.

Durch die Matlab-Kommandos `who` bzw. `whos` erhält man die Informationen auch im Command Window. Nachdem wir das obige Programm `beispiel1` ausgeführt haben, liefert das Kommando

```
whos
```

die Informationen

```
>> whos
   Name      Size          Bytes  Class

   x         1x7              56  double array
   y         1x7              56  double array

   Grand total is 14 elements using 112 bytes

>>
```

Inhalt des Workspaces speichern

In dem Menüpunkt **File** wird der Unterpunkt **Save Workspace As...** angeklickt und danach der Dateiname eingegeben. Die Datei erhält automatisch den Zusatz `.mat`.

Alternativ kann im Command Window der Befehl `save` verwendet werden. So wird zum Beispiel durch das Kommando

```
save('april25')
```

der Inhalt des Arbeitsspeichers in die Datei namens `april25.mat` kopiert. Beachten Sie, daß im save-Kommando der Name in Hochkommata eingeschlossen werden muß.

Laden in den Workspace

In dem Menüpunkt **File** wird der Unterpunkt **Open...** angeklickt und danach der Dateiname eingegeben. Dies muß eine Datei mit dem Zusatz `.mat` sein.

Alternativ kann im Command Window der Befehl `load` verwendet werden. So wird zum Beispiel durch das Kommando

```
load('april25')
```

die Datei `april25.mat` in den Arbeitsspeicher geladen.

Workspace löschen
Der gesamte Workspace wird gelöscht durch Anklicken von **Edit** und danach **Clear Workspace**. Matlab fragt dann nochmals nach: **Are you sure you want to clear your workspace?** Erst wenn Sie dann **yes** anklicken, wird der Arbeitsspeicher gelöscht.

Alternativ können Sie im Command Window den Befehl

```
clear
```

eingeben. Dabei wird ohne Nachfragen gelöscht. Mit diesem Befehl können auch einzelne Variablen aus dem Arbeitsspeicher entfernt werden. Zum Beispiel wird durch

```
clear x
```

nur die Variable `x` gelöscht.

4 Umgang mit Matrizen

Matlab arbeitet stets mit dem Datentyp Matrix (engl. *array*). Daher stammt auch der Name Matlab: **Mat**rix **lab**oratory. Eine reelle Zahl ist dann eine 1×1 - Matrix, ein Zeilenvektor eine $1 \times n$ - Matrix und ein Spaltenvektor eine $n \times 1$ - Matrix. Intern werden die Matrixelemente als reelle Zahlen mit doppelter Genauigkeit dargestellt.

4.1 Matrix belegen

Eine Matrix wird zeilenweise eingegeben; jede Zeile wird mit einem Strichpunkt abgeschlossen, die einzelnen Elemente der Zeile werden durch Leerzeichen getrennt. So wird zum Beispiel durch

```
A = [1 2 3 4; 5 6 7 8; 9 10 12 12; 13 14 15 16]
```

die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \tag{1}$$

definiert. Beachten Sie die eckigen Klammern. Durch

```
x = [1 2 3 4]
y = [10; 20; 30]
```

werden die Vektoren

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \quad \text{und} \quad y = \begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix}$$

erzeugt. Auf ein Matrix-Element wird durch Angabe seiner Position (Zeilenindex, Spaltenindex) zu zugegriffen. Zum Beispiel erhalten wir durch Eingabe von

```
A(3,2) = 0
```

die Matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad .$$

Eine Matrix kann mit Nullen bzw. Einsen vorbelegt werden. Durch

```
A = zeros(4,3)
B = ones(2,5)
```

erhalten wir die Matrizen

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{und} \quad B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad .$$

Die $n \times n$ - Einheitsmatrix wird in Matlab durch das Kommando `eye(n,n)` erzeugt. Dabei muß `n` schon mit einem Wert belegt sein. So liefert

```
n = 3
I = eye(n,n)
```

die Matrix

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

Das Matlab-Kommando

```
x = 0:0.1:2
```

erzeugt den Zeilenvektor $x = (0, 0.1, 0.2, \ldots, 1.9, 2.0)$.

4.2 Teilbereiche einer Matrix

Von einer bereits definierten Matrix kann man Teilbereiche (Zeilen, Spalten, Untermatrizen) ansprechen. Betrachten wir die Matrix aus (1), so liefern die Kommandos

```
u = A(2,:)
v = A(:,3)
B = A(1:3,1:2)
C = A(3:4,3:4)
```

die folgenden Vektoren bzw. Matrizen:

$$u = \begin{pmatrix} 5 & 6 & 7 & 8 \end{pmatrix}, \quad v = \begin{pmatrix} 3 \\ 7 \\ 11 \\ 15 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix}, \quad C = \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix} \quad .$$

Eine Matrix kann auch mit Hilfe von Teilmatrizen definiert werden. So liefert zum Beispiel die Sequenz

```
I = eye(2,2);
Z = zeros(2,2);
E = [2 -1; -1 2];
D = [E,Z,I; Z,E,Z; I,Z,E]
```

die folgende Matrix:

$$D = \begin{pmatrix} 2 & -1 & 0 & 0 & 1 & 0 \\ -1 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & -1 \\ 0 & 1 & 0 & 0 & -1 & 2 \end{pmatrix} \quad .$$

Aus einer Matrix können Teilbereiche gestrichen werden. Dadurch ändert sich die Größe der Matrix. Betrachten wir die Matrix A aus (1) und geben dann

```
A(2,:) = []
```

ein, so wird aus A die zweite Zeile gestrichen:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad .$$

4.3 Operationen mit Matrizen

Zwei Matrizen A und B derselben Größe werden addiert bzw. subtrahiert durch

```
C = A + B
D = A - B
```

Diese Operationen werden (wie aus der Mathematik bekannt) elementweise durchgeführt. Vorsicht ist dagegen bei der Multiplikation geboten, denn hier kennt die Mathematik zwei Möglichkeiten: elementweise Multiplikation und das Matrixprodukt. Dementsprechend gibt es auch in Matlab die beiden Möglichkeiten

```
C = A * B      Matrixprodukt
D = A .* B     elementweise Multiplikation
```

Das Matrixprodukt ist nur definiert, falls A eine $m \times n$-Matrix und B eine $n \times r$-Matrix ist und ergibt dann eine $m \times r$-Matrix.

Entsprechendes gilt bei der Potenz:

```
C = A^3      entspricht C = A*A*A
D = A.^3     elementweise Potenzierung
```

Die Matrixpotenz A^3 ist nur für eine quadratische Matrix (d.h. eine $n \times n$-Matrix) erklärt.

Die elementweise Division zweier Matrizen erfolgt mittels

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

,

was $c_{ij} = a_{ij}/b_{ij}$ bedeutet. Für zwei quadratische Matrizen kennt Matlab auch die Operation

```
D = A/B
```

.

Hier ist D die Matrix, welche $A = D \cdot B$ erfüllt.

Möglich sind auch die Anweisungen

```
E1 = A .* 2;
E2 = 2 ./ A;
E3 = A + 2;
E4 = A - 2;
```

Gedanklich wird dabei die reelle Zahl 2 zu einer Matrix, welche dieselbe Größe wie A hat, und deren Elemente alle den Wert 2 haben.

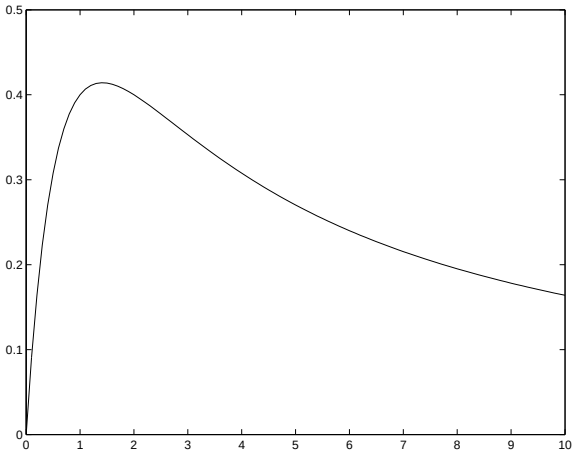
Als Beispiel erstellen wir von der Funktion

$$f(x) := \frac{x}{1 + ax + rx^2}$$

für $a = 1$, $r = 0.5$ eine Wertetabelle und zeichnen dann diese Funktion. Die entsprechenden Matlab-Befehle lauten

```
a = 1;
r = 0.5;
x = 0:0.1:10;
y = x ./(1 + a.*x +r.*x.^2);
plot(x,y);
```

Als Ergebnis erhalten wir die folgende Grafik:



Das Matlab-Kommando `plot` zeichnet die Wertetabelle. Auf diesen Befehl gehen wir später noch ausführlich ein.

4.4 Funktionen mit Matrizen

Ist A eine bereits definierte Matrix, so liefert der Matlab-Befehl

```
size(A)
```

die Größe der Matrix. Möglich ist auch

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

Hier wird in m die Zeilenzahl und in n die Spaltenzahl abgespeichert. Durch

```
B = zeros(size(A))
```

wird eine mit Nullen vorbelegte Matrix B erzeugt, die dieselbe Größe wie A hat.

Für eine Matrix A liefert das Kommando

```
diag(A)
```

die Diagonale (als Spaltenvektor). Umgekehrt ergibt für einen Spaltenvektor v (mit n Komponenten) der Matlab-Befehl

```
B = diag(v)
```

eine Diagonalmatrix mit den Elementen von v in der Diagonalen. Weiter liefert

```
C = diag(v,1)
```

eine $(n+1) \times (n+1)$-Matix C mit dem Vektor v in der ersten oberen Nebendiagonalen. Zum Beispiel erhalten wir durch

```
e = ones(4,1);
S = diag(11:11:55) + diag(e,1) + diag(e,-1);
```

die Matrix

$$\begin{pmatrix} 11 & 1 & 0 & 0 & 0 \\ 1 & 22 & 1 & 0 & 0 \\ 0 & 1 & 33 & 1 & 0 \\ 0 & 0 & 1 & 44 & 1 \\ 0 & 0 & 0 & 1 & 55 \end{pmatrix} .$$

Durch

```
sum(A)
```

werden die Elemente in den Spalten aufsummiert (ergibt als Ergebnis einen Zeilenvektor).

Schließlich erhält man durch

```
B = A'
```

die zu A transponierte Matrix. Möglich sind auch

`u = sum(diag(A));`	(summiert die Diagonalelemente)
`y = sum(A');`	(Elemente in den Zeilen werden summiert)
`z = sum(A(:,3);`	(summiert die Elemente der 3. Spalte)
`v = sum(sum(A));`	(summiert über alle Elemente der Matrix A)
`D = diag(diag(A));`	(Diagonalmatrix)

Matlab kennt alle wichtigen (aus der Mathematik bekannten) Funktionen, welche durch das Kommando

```
help elfun
```

aufgelistet werden. Die wichtigsten davon sind:

`exp(a)`	Exponentialfunktion
`log(a)`	natürlicher Logarithmus
`sqrt(a)`	Quadratwurzel
`sin(a)`	Sinus
`cos(a)`	Kosinus
`tan(a)`	Tangens
`ceil(a)`	nächste ganze Zahl $\geq a$
`floor(a)`	nächste ganze Zahl $\leq a$
`round(a)`	rundet zur nächsten ganzen Zahl
`fix(a)`	$\begin{cases} \texttt{ceil(a)} & \text{falls } a \geq 0 \\ \texttt{floor(a)} & \text{falls } a \leq 0 \end{cases}$

Die Variable darf natürlich auch eine Matrix sein. Dann liefern diese Funktionen als Ergebnis eine Matrix derselben Größe. Die Funktionsauswertung erfolgt elementweise.

Als Beispiel erstellen wir von der Funktion

$$f(x) := \frac{a}{1+\exp(b-cx)} \qquad a,b,c \in \mathbb{R}$$

eine Wertetabelle für $x = 0, 0.1, 0.2, \ldots, 9.9, 10$ und zeichnen diese anschließend.

```
a=100;
b=5;
c=1
x=0:0.1:10;
f = a./(1+exp(b - c.*x));
plot(x,f);
```

Hier ist `f` ein Zeilenvektor, der dieselbe Größe wie `x` hat. Seine Komponenten sind die Funktionswerte $f(x_i)$.

Desweiteren enthält die Variable `pi` als Voreinstellung den Wert π.

5 Ein- und Ausgabe

5.1 Einlesen aus einer Datei

Zum Einlesen von Größen aus einer Datei bietet Matlab mehrere Möglichkeiten. Diese sind von der Form der Eingabedatei abhängig: nur Zahlen (numerische Größen), nur Text oder Text und Zahlen gemischt.

Numerische Größen

Enthält die einzulesende Datei nur numerische Gößen, und zwar in jeder Zeile die gleiche Anzahl von Zahlen (durch Leerzeichen getrennt), so steht uns der Matlab-Befehl

`A = load('<name>')`

zur Verfügung. Dabei ist dann `A` eine Matrix. Befinden sich zum Beispiel in der Datei `daten1.ein` die Zahlen

```
1.1 1.2 1.3 1.4 1.5
2.1 2.2 2.3 2.4 2.5
3.1 3.2 3.3 3.4 3.5
4.1 4.2 4.3 4.4 4.5
```

so liefert das Kommanodo

```
A = load('daten1.ein')
```

eine 4×5 - Matrix.

Text und numerische Größen

Im Normalfall hat man eine Eingabedatei, die sowohl Text als auch Zahlen enthält. Häufig sind auch die einzelnen Zeilen unterschiedlich lang. Dann verwendet man zum Einlesen die Matlab-Kommandos `fgetl` bzw. `fscanf`.

Dazu muß aber zuerst die Datei mittels

`fid = fopen('dateiname')`

zum Lesen angemeldet werden. Durch diesen Befehl wird die Variable `fid` (die natürlich frei wählbar ist) der entsprechenden Datei zugeordnet.

Das Kommando

```
zeile1 = fgetl(fid)
```

liest nun eine Text-Zeile aus der angemeldeten Datei ein und speichert sie in der Variablen `zeile1`. Numerische Größen werden mit dem Matlab-Befehl `fscanf` eingelesen. Zum Beispiel bewirkt das Kommando

```
B = fscanf(fid,'%g',[3,2])   ,
```

daß aus der Datei die nächsten 6 Zahlen eingelesen und in einer 3×2 - Matrix gespeichert werden. Dabei ist zu beachten, daß die Datei **zeilenweise** gelesen, die Matrix aber **spaltenweise** belegt wird.

Nach dem Einlesen wird die Datei mit dem Befehl

```
fclose(fid)
```

wieder abgemeldet.

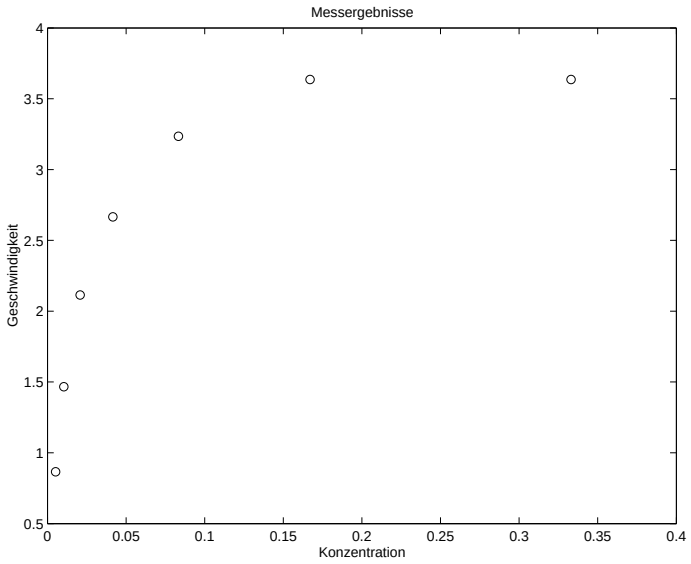
Als Beispiel nehmen wir an, daß die Datei `ergebnisse.dat` folgende Gestalt hat:

```
Messergebnisse
Konzentration
Geschwindigkeit
0.333 3.636
0.167 3.636
0.0833 3.235
0.0416 2.666
0.0208 2.114
0.0104 1.466
0.0052 0.866
```

Das folgende Matlab-Programm liest diese Datei ein und erstellt dann eine Zeichnung. Dabei wird der Text für die Überschrift bzw. die Achsenbeschriftung verwendet.

```
fid = fopen('ergebnisse.dat');
zeile1 = fgetl(fid);
zeile2 = fgetl(fid);
zeile3 = fgetl(fid);
Z = fscanf(fid,'%g',[2,7]);
plot(Z(1,:),Z(2,:),'o');
title(zeile1);
xlabel(zeile2);
ylabel(zeile3);
fclose(fid);
```

Als Ergebnis erhalten wir die folgende Grafik:



Manchmal besteht der Wunsch, alle Werte aus einer Datei einzulesen, wobei jedoch die Anzahl der Daten nicht bekannt ist. Nehmen wir mal an, daß die Datei `test.dat` folgende Gestalt hat:

```
1 2 3 4 5 6 7
8 9 10 11 12
13 14 15 16
17 18
19
20 21
```

Die Matlab-Befehle

```
fid = fopen('test.dat');
[x,count] = fscanf(fid,'%g',[1,inf])
fclose(fid);
```

liefern dann als Ergebnisse

```
x =
  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21

count =
      21
```

Die Angabe von `inf` beim Aufruf von `fscanf` bewirkt, daß alle Zahlen gelesen werden. Die Anzahl der Werte wird in der Variablen `count` gespeichert.

5.2 Ausgabe in eine Datei

Auch zur Ausgabe von Matlab-Größen in eine Datei gibt es mehrere Möglichkeiten.

Numerische Größen

Zur Ausgabe von rein numerischen Größen in eine Datei steht uns das `save` - Kommando zur Verfügung. Haben wir zum Beispiel eine Matrix

```
A = [1 2 3 4; 5 6 7 8];
```

definiert, so wird diese mittels

```
save('ergebnis.dat','-ASCII','A');
```

in eine Datei namens `ergebnis.dat` gespeichert. Der Zusatz `'-ASCII'` bewirkt die Abspeicherung im Standartformat (Ascii-Format); die Datei kann dann mit jedem gängigen Texteditor gelesen bzw. mit dem `a2ps` - Kommando auf dem Drucker ausgegeben werden.

Im obigen Fall erhalten wir durch `emacs ergebnis.dat` die Ausgabe

```
1.0000000e+00   2.0000000e+00   3.0000000e+00   4.0000000e+00
5.0000000e+00   6.0000000e+00   7.0000000e+00   8.0000000e+00
```

Text und numerische Größen

Hierfür steht uns das Matlab-Kommando `fprintf` zur Verfügung, welches folgenden Aufbau besitzt:

`fprintf(fid,` <*format*> `,` <*variable*> `);` .

Dabei muß zuerst mit dem `fopen` - Befehl der Variablen `fid` eine Datei zugeordnet werden. Für <*format*> müssen wir eine Formatangabe einsetzen; hier wird angegeben, wie die reellen Zahlen in der Datei dargestellt werden sollen. Möglich ist unter anderem:

%e	Gleitpunktdarstellung, z.B 1.105171e+01
%f	Festkommadarstellung, z.B. 11.05171
%g	wählt automatisch die günstigste Darstellung
%12.6f	Festkommadarstellung mit insgesamt 12 Stellen, davon 6 Nachkommastellen
%14.8e	Gleitkommadarstellung mit insgesamt 14 Stellen, davon 8 Nachkommastellen
\n	neue Zeile

Außerdem kann die Formatangabe noch Text enthalten, der dann ebenfalls in die Datei geschrieben wird. Die Formatangabe wird in Hochkommata eingeschlossen.

Im folgenden Beispiel wird eine Wertetabelle von der Exponentialfunktion erstellt und in die Datei `wertetab.txt` gespeichert.

```
x = 0:0.1:1;
y = [x; exp(x)];
fid = fopen('wertetab.txt','w');
fprintf(fid,'Exponentialfunktion\n\n');
fprintf(fid,'%5.2f  %12.8f\n',y);
fclose(fid);
```

Danach hat die Datei `wertetab.txt` den folgenden Inhalt:

```
Exponentialfunktion

0.00    1.00000000
0.10    1.10517092
0.20    1.22140276
0.30    1.34985881
0.40    1.49182470
0.50    1.64872127
0.60    1.82211880
0.70    2.01375271
0.80    2.22554093
0.90    2.45960311
1.00    2.71828183
```

Falls beim Aufruf von `fid = fopen('wertetab.txt','w')` die Datei `wertetab.txt` schon existiert, so wird deren Inhalt zunächst gelöscht. Möchte man die Ergebnisse an die bereits bestehende Datei `wertetab.txt` anhängen, so wird

```
fid = fopen('wertetab.txt','a');
```

verwendet.

5.3 Ausgabe auf dem Bildschirm

Falls man bei einem Matlab-Befehl das Semicolon wegläßt, so erscheint das Ergebnis sofort auf dem Bildschirm. Mit Hilfe des `format` - Befehls kann man die Zahlendarstellung verändern. Es gibt folgende Möglichkeiten:

`format short`	Festkommadarstellung mit 5 Stellen
`format long`	Festkommadarstellung mit 15 Stellen
`format short e`	Gleitkommadarstellung mit 5 Stellen
`format long e`	Gleitkommadarstellung mit 15
`format compact`	unterdrückt Leerzeilen

Zwei weitere Möglichkeiten bilden die Matlab-Kommandos `disp` und `sprintf`. Dabei entspricht `sprintf` dem in Abschnitt 5.2 behandelten Matlab-Befehl `fprintf`, das Ergebnis wird jedoch in eine String-Variable (Text-Variable) gespeichert. Mit dem `disp` - Befehl kann sowohl eine Variable als auch Text ausgegeben werden.

Das folgende Matlab-Programm demonstriert die Verwendung der beiden Kommandos:

```
A = [1 2 3 4; 4 5 6 7];
disp('Eingegeben wurde die Matrix A =');
disp(A);
text = sprintf('   A(2,3) = %6.2f',A(2,3));
disp(text);
```

Als Ergebnis erhalten wir auf dem Bildschirm die Ausgabe

```
Eingegeben wurde die Matrix A =
     1     2     3     4
     4     5     6     7

    A(2,3) =   6.00
```

5.4 Eingabe über den Bildschirm

Mit dem Befehl `input` können innerhalb eines Matlab-Programmes Daten über den Bildschirm eingelesen werden. Er hat die Form

```
x = input('Text');    ,
c = input('Text','s');    .
```

Matlab gibt dann auf dem Bildschirm den Text aus und wartet auf die Eingabe. Im ersten Fall wird eine numerische Eingabe erwartet, welche in der Variablen `x` gespeichert wird; im zweiten Fall wird die Eingabe als String (Text) behandelt und in `c` abgelegt.

Das folgende Matlab-Programm erstellt eine Wertetabelle der logistischen Kurve

$$L(t) := \frac{a}{1+\exp(b-ct)} \quad ,$$

wobei die Parameter a, b, c über den Bildschirm eingelesen werden.

```
disp(' Es wird eine Wertetabelle der log. Kurve erstellt');
disp(' Geben Sie die folgenden Parameter ein:');
a = input(' a = ');
b = input(' b = ');
c = input(' c = ');
x=0:0.5:50;
y=[x; a./(1.0 + exp(b -c.*x))];
fid = fopen('wertetab.txt','w');
fprintf(fid,' logistische Kurve mit den Parametern\n');
fprintf(fid,'    a = %g\n',a);
fprintf(fid,'    b = %g\n',b);
fprintf(fid,'    c = %g\n\n',c);
fprintf(fid,'   %6.2f   %9.4f \n',y);
fclose(fid);
```

6 Graphik

Matlab besitzt ein umfangreiches Graphik-Paket. Durch die Eingabe des ersten Graphik-Befehls wird ein neues Matlab-Fenster mit der Überschrift `Figure` geöffnet. Eine erstellte Graphik kann als Postskript-Datei abgespeichert und so in andere Texte (z.B. Latex-Texte) eingebunden werden.

6.1 2D-Graphik

Zum Zeichnen von reellen Funktionen (oder Meßergebnissen) in der Ebenen verwendet man das Matlab-Kommando `plot`. Dabei wird zunächst eine Wertetabelle von der Funktion erstellt, welche als Parameter an das `plot` - Kommando übergeben werden. Das folgende Beispiel zeichnet die Funktion $\sin(x)$ im Intervall $[0, 10]$:

```
x = 0:0.1:10;
y = sin(x);
plot(x,y);
```

Jeder Aufruf von `plot` löscht standardmäßig eine bereits vorhandene Kurve. Um dies zu vermeiden, gibt man vor dem `plot` - Aufruf das Kommando

```
hold on
```

ein. Dadurch wird die Kurve in die vorhandene Graphik eingefügt. Mit einem `plot` - Kommando können auch mehrere Kurven gleichzeitig gezeichnet werden. Es gibt also zwei Möglichkeiten, um zwei (bzw. mehrere) Kurven in ein Schaubild zu zeichnen:

1. Möglichkeit

```
x = 0:0.1:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y);
hold on;
plot(x,z);
```

2. Möglichkeit

```
x = 0:0.1:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y,x,z);
```

Desweiteren können in der Zeichnung Linientyp, Farbe und Markierungstyp gewählt werden. Dazu wird im `plot` - Befehl ein weiterer Parameter übergeben.

Linientyp
Für den Linientyp gibt es die folgenden Möglichkeiten:

`-`	durchgezogene Linie
`:`	gepunktete Linie
`-.`	Punkt-Strich-Linie
`--`	gestrichelte Linie

Markierungstyp
Weiter besteht die Möglichkeit, die Punkte der Wertetabelle zu markieren bzw. nur die Punkte zu zeichnen (ohne die Verbindungsstrecke). Matlab hat die folgenden Markierungstypen:

`.`	Punkt
`o`	Kreis
`x`	Kreuz
`+`	Plus
`*`	Stern
`s`	Quadrat
`d`	Raute
`v`	Dreieck, Spitze nach unten
`^`	Dreieck, Spitze nach oben
`<`	Dreieck, Spitze nach links
`>`	Dreieck, Spitze nach rechts
`p`	Pentagramm
`h`	Hexagramm

Farben

Die Auswahl der Farbe erfolgt durch:

y	gelb
m	magenta
c	cyan
r	rot
g	grün
b	blau
w	weiß
k	schwarz

Im folgenden Beispiel wird die erste Kurve rot gezeichnet, wobei die berechneten Punkte mit einem Quadrat markiert und durch eine Gerade verbunden werden. Bei der zweiten Kurve werden die Punkte nur durch einen grünen Kreis markiert.

```
x = 0:0.25:10;
y = sin(x);
z = sin(x+0.5);
plot(x,y,'rs-');
hold on;
plot(x,z,'go');
```

Desweiteren besteht die Möglichkeit, die Größe und die Farbe der Markierungen zu ändern sowie das Innere der Markierungszeichen durch eine Farbe auszufüllen. Dazu werden weitere Parameter in das `plot`-Kommando aufgenommen. Testen Sie mal das folgende Matlab-Programm:

```
x = 0:0.5:10;
y = sin(x);
plot(x,y,'rs--','MarkerEdgeColor','k',...
               'MarkerFaceColor','g',...
               'MarkerSize',15)
```

6.2 3D-Graphik

Matlab besitzt eine ausgezeichnete 3D-Graphik. Wir stellen hier nur einige Elemente vor.

Zeichnen einer Kurve im Raum

In der Mathematik ist eine Raumkurve in der Regel durch eine Parametrisierung gegeben:

$$(x(t), y(t), z(t)) \quad \text{mit } t \in [a, b].$$

Dabei sind $x(t)$, $y(t)$, $z(t)$ reelle differenzierbare Funktionen.

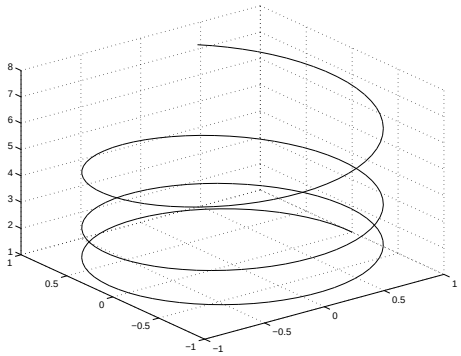
Im Matlab verwendet man zum Zeichnen von Raumkurven das Kommando

```
plot3(x,y,z)
```

Dabei sind x, y, z Vektoren, welche die Werte $x(t_i)$, $y(t_i)$, $z(t_i)$, $i = 1, \ldots, n$ enthalten.

Dazu betrachten wir das folgenden Beispiel

```
t=0:0.01:2;
x=cos(10.*t);
y=sin(10.*t);
z=exp(t);
plot3(x,y,z);
grid on;
```



Der Matlab-Befehl `grid on` bewirkt, daß ein Gitter gezeichnet wird. Dadurch ergibt sich eine bessere räumliche Darstellung der Kurve. Mit dem Kommando `grid off` wird dieses Gitter wieder entfernt.

Zeichnen des Graphen einer Funktion $f : D \subset \mathbb{R}^2 \to \mathbb{R}$

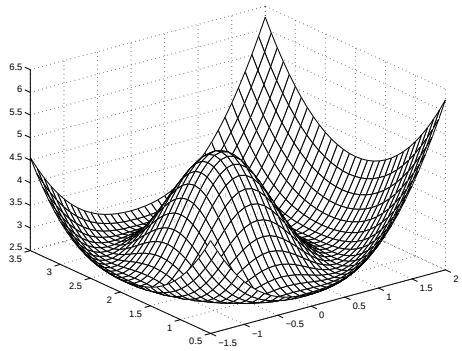
Eine Funktion $f : D \subset \mathbb{R}^2 \to \mathbb{R}$ kann als "Gebirge" über dem Definitionsbereich oder als Höhenkarte dargestellt werden. Für die Darstellung als Gebirge gibt es mehrere Varianten. Wir veranschaulichen dies exemplarisch für die Funktion

$$f(x, y) := 5 \exp(-x^2 - (y-2)^2) + x^2 + (y-2)^2$$

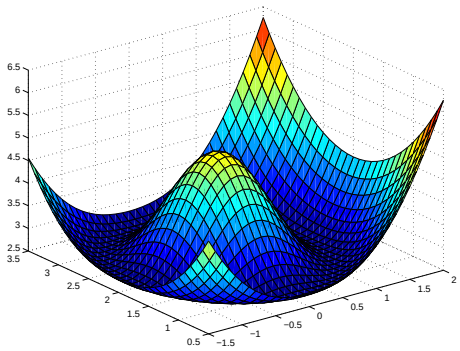
mit Definitionsbereich $D = [-1.5, 2] \times [0.5, 3.5]$.

Der Matlab-Befehl `meshgrid` wählt aus dem Definitionsbereich Gitterpunkte aus, in welchen die Funktion ausgewertet wird.

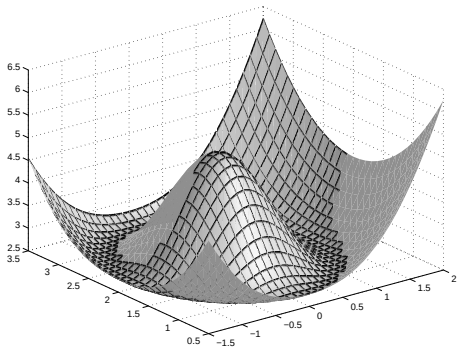
```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
mesh(X,Y,Z);
```



```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
surf(X,Y,Z);
colormap(jet)
```



```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
surfl(X,Y,Z);
shading interp;
colormap(pink);
```



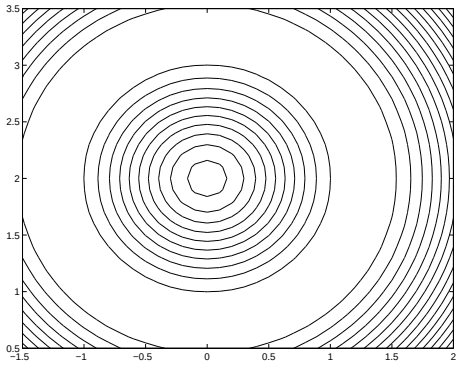
Das Matlab-Kommando

```
help graph3d
```

liefert weitere Information über die vielfältigen Möglichkeiten (etwa hinsichtlich Farbenwahl oder Schattierungen) bei 3D-Zeichnungen.

Das folgende Matlab-Programm erstellt von der obigen Funktion eine Höhenkarte.

```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
contour(X,Y,Z,15);
```



Selbstverständlich kann man die einzelnen Höhenlinien auch mit dem entsprechenden Wert beschriften. Testen Sie mal das folgende Programm:

```
[X,Y] = meshgrid(-1.5:0.1:2,0.5:0.1:3.5);
Z = 5.*exp(-X.^2 - (Y-2).^2) + X.^2 + (Y-2).^2;
[c,h] = contour(X,Y,Z,15);
clabel(c,h,'manual');
```

Nach dem Start dieses Programms können Sie mit der Maus die Höhenlinien auswählen, die beschriftet werden sollen.

6.3 Graphik beschriften

Es gibt eine ganze Reihe von Möglichkeiten, um eine erstellte Graphik zu beschriften. Dabei können einige Elemente aus dem Paket TEX bzw. LATEX verwendet werden.

Achsen-Handling

Normalerweise wählt Matlab die Achsen und deren Skalierung automatisch so, daß die Graphik das Figure-Fenster gerade ausfüllt. Mit dem Befehl `axis` wird das Erscheinungsbild der Achsen beeinflußt:

`axis off`	Achsen werden entfernt
`axis on`	Achsen werden gezeichnet
`axis equal`	gleicher Maßstab für alle Achsen
`axis normal`	Maßstab so, daß Graphik raumfüllend
`axis square`	liefert ein quadratisches Bild

Der Befehl

```
axis([xmin xmax ymin ymax])
```

bewirkt, daß die X-Achse von *xmin* bis *xmax* und die Y-Achse von *ymin* bis *ymax* gezeichnet wird.

Achsenbeschriftung

Die Achsen werden mit den Kommandos `xlabel, ylabel` bzw. `zlabel` beschriftet. Beispiele dazu sind

```
xlabel('Konzentration c [\mu Mol]');
ylabel('Geschwindigkeit \nu(c)');
```

Dabei sind auch einige Sonderzeichen aus der Mathematik möglich, wie etwa griechische Buchstaben, welche wie in TEX eingegeben werden.

Überschrift

Durch den Matlab-Befehl `title` wird das Bild mit einer Überschrift versehen, z. B.

```
title('Logistische Kurve')
```

Text in der Graphik

An jeder beliebigen Stelle der Zeichnung kann Text plaziert werden. Dazu gibt es zwei Möglichkeiten:

`gtext('a = 100')`	Text wird mit der Maus plaziert
`text(10,20,'c = 1.5')`	Text wird an der Stelle (10,20) plaziert

Schriftarten

Bei den obigen Kommandos können für den Text verschiedene Schriftarten gewählt werden. So wird zum Beispiel durch

```
title('{\bf Schaubild1: }{\it Kaefer} ueber der Zeit');}')
```

das Wort **Schaubild1:** fett gedruckt und das Wort *Kaefer* in der Schriftart Italics. Eine weitere Schriftart ist *slantet*, welche mit `\sl` eingeleitet wird.

Schriftgröße

Die Schriftgöße wird durch weitere Parameter angegeben:

```
title('logistische Kurve','FontSize',15);
gtext('a = 300','FontSize',5);
xlabel(' Zeit \tau','FontSize',12);
```

6.4 Graphik abspeichern

Eine erstellte Graphik kann in verschiedenen Formaten abgespeichert und so in andere Texte (z.B. TEX - Programme) eingebunden werden.

Dazu klicken Sie in dem Fenster `Figure` zunächst den Menüpunkt `File` und dann `Export...` an. Danach erscheint ein neues Fenster mit der Überschrift `Export Figure`. Hier geben Sie zuerst einen Dateinamen ein, unter dem die Graphik gespeichert werden soll, zum Beispiel `Bild1.eps`. Dann wählen Sie in dem Feld `Save as type` das Format aus und klicken dann das Feld `Save` an. Für unsere Zwecke ist dies

```
EPS Level 2 Color file
```

,

wodurch die Graphik im Postskript-Format (encapsulated postscript) abgespeichert wird. Mit dem Kommando

```
lpr Bild1.eps
```

wird die Graphik auf dem Laserdrucker in V203 ausgegeben.

6.5 Balken- und Kuchendiagramme

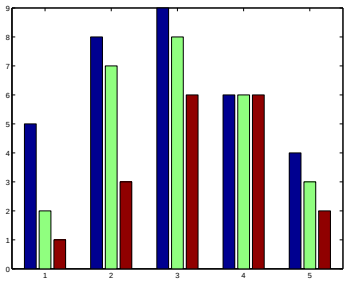
Für Balkendiagramme müssen die zu zeichnenden Werte zunächst in einer Matrix abgelegt werden, z. B.

```
Y = [5 2 1; 8 7 3; 9 8 6; 6 6 6; 4 3 2];
```

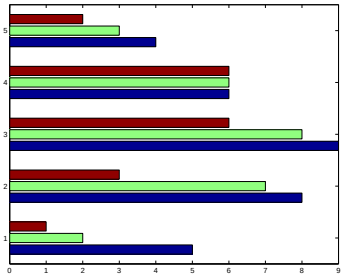
.

Danach liefern die Matlab-Kommandos `bar, barh` bzw. `bar3` die folgenden Diagramme:

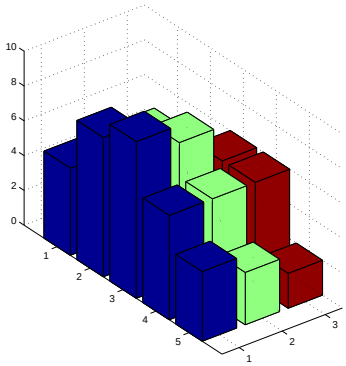
```
bar(Y);
```



```
barh(Y);
```

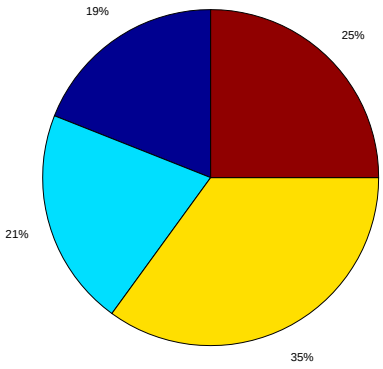


```
bar3(Y);
```

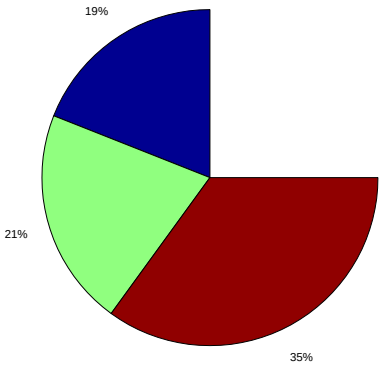


Für Kuchendiagramme werden die Anteile in einem Vektor zusammengefaßt und dann mit dem Matlab-Befehl `pie` gezeichnet, z. B.

```
x = [0.19 0.21,0.35,0.25];
pie(x);
```




```
u = [0.19 0.21,0.35];
pie(u);
```



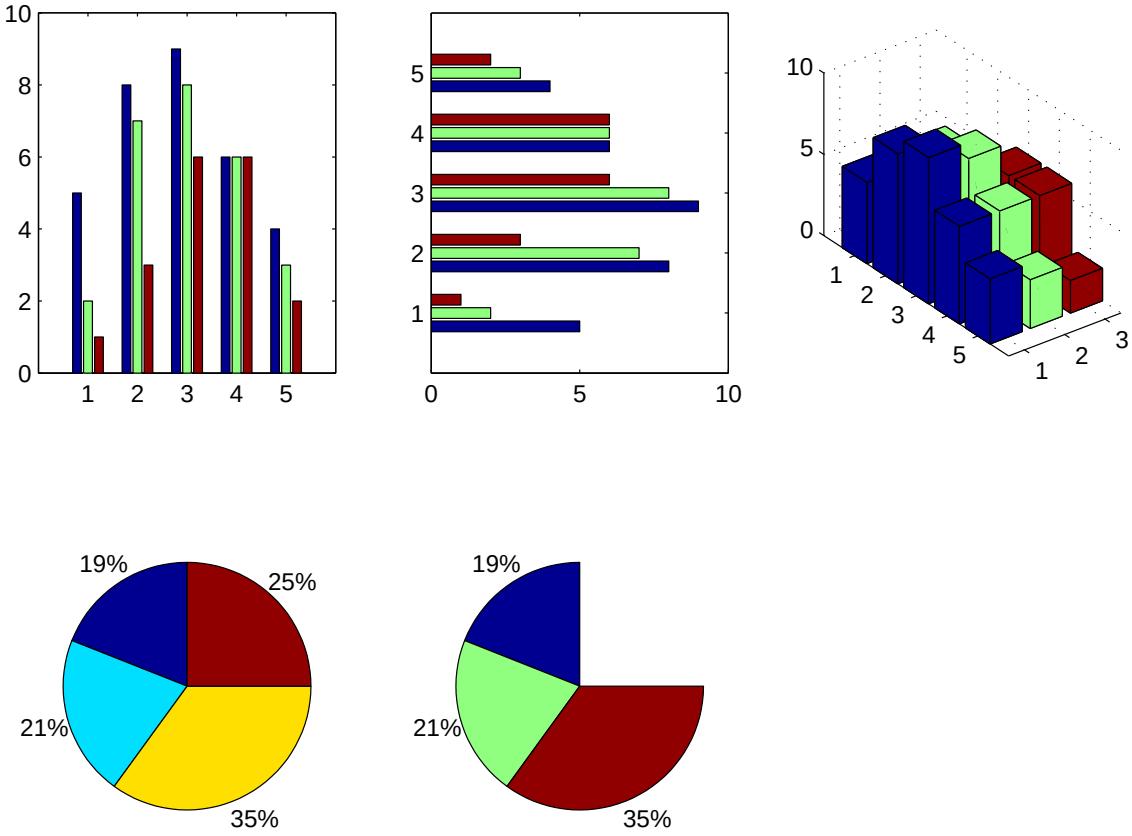
Selbstverständlich können zur Beschriftung der Diagramme die in Abschnitt 6.3 genannten Kommandos verwendet werden.

6.6 Unterbilder

Man kann mehrere Bilder in einem Graphik-Fenster darstellen. Dazu wird das Matlab-Kommando `subplot(m,n,r)` verwendet. Dies bedeutet, daß das Graphik-Fenster in insgesamt mn Unterbilder (mit m Zeilen und n Spalten) zerlegt wird und aktuell das r-te Bild angesprochen wird. Dazu betrachten wir das folgende Beispiel:

```
Y = [5 2 1; 8 7 3; 9 8 6; 6 6 6; 4 3 2];
subplot(2,3,1);
bar(Y);
subplot(2,3,2);
barh(Y);
subplot(2,3,3);
bar3(Y);
x = [0.19 0.21,0.35,0.25];
subplot(2,3,4);
pie(x);
subplot(2,3,5);
u = [0.19 0.21,0.35];
pie(u);
```

Als Ergebnis erhalten wir das folgende Bild:



7 Programmsteuerung

7.1 Logische Ausdrücke

Beim Programmieren besteht häufig der Wunsch, eine bestimmte Anweisung von einer Bedingung abhängig zu machen. Zum Beispiel kann man die Quadratwurzel einer reellen Zahl a nur für $a \geq 0$ berechnen. Eine solche Bedingung nennt man einen *logischen Ausdruck*; er kann nur die Werte *wahr* (engl. *true*) oder *falsch* (engl. *false*) annehmen. In Matlab wird der Wert *wahr* durch 1 und der Wert *falsch* durch 0 dargestellt.

Logische Ausdrücke in der Mathematik sind zum Beispiel

$$\begin{aligned} x &< 10 \\ a+b &\geq y+5 \\ x^2+y^2 &= 5 \\ a &\neq \sqrt{x-y} \end{aligned}$$

Die Zeichen $< > \leq \geq \neq =$ nennt man *Vergleichsoperatoren.*

In Matlab gibt es diese Vergleichsoperatoren ebenfalls:

Matlab	math. Bedeutung
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
==	$=$
~=	$\neq$

In Matlab-Notation lauten die obigen Beispiele:

```
...
x < 10
(a + b) >= (y + 5)
(x.^2 + y.^2) == 5
a ~= sqrt(x-y)
...
```

Im allgemeinen wirken die Vergleichsoperatoren auf Matrizen. Haben wir zum Beispiel

$$A = \begin{pmatrix} 2 & 7 & 5 & 3 \\ 4 & 6 & 8 & 0 \end{pmatrix} \quad , \quad B = \begin{pmatrix} 5 & 7 & 3 & 2 \\ 5 & 5 & 7 & 0 \end{pmatrix} \quad ,$$

so liefert der Matlab-Befehl

```
A == B
```

das Ergebnis

```
0     1     0     0
0     0     0     1
```

Logische Ausdrücke kann man kombinieren durch die logischen Operatoren *und, oder* bzw. *nicht*; zum Beispiel

$$a \neq \sqrt{x-y} \qquad \text{und} \qquad x < 10 \quad .$$

In Matlab gibt es dafür die Operatoren

Matlab	math. Bedeutung
&	und
|	oder
~	nicht

Der logische Ausdruck aus obigem Beispiel lautet in dann

```
(a ~= sqrt(x-y)) & (x < 10)
```

An dieser Stelle sind einige Worte zur Klammersetzung angebracht. Matlab hat gewisse Voreinstellungen, wenn runde Klammern fehlen. Dies kann dazu führen, daß nicht die von uns gewünschte Größe berechnet wird. Da zusätzliche (überflüssige) Klammerung (mit runden Klammern) nicht schadet, lautet die Empfehlung im Zusammenhang mit den logischen Ausdrücken: verwende generell eine Klammerung. Dies erhöht zum einen die Lesbarkeit des Matlabprogrammes, zum anderen braucht man sich dann die Matlab-Voreinstellungen nicht zu merken.

7.2 Verzweigungen

Verzweigungen dienen dazu, in Abhängigkeit vom Wert eines logischen Ausdruckes verschiedene Programmteile auszuführen. Dabei unterscheidet man

Einseitige Verzweigung

Diese haben in Matlab die Form

```
if  logischer Ausdruck
   Anweisungen ("if-Block")
end
```

Bei der Programmausführung wird zunächst der logische Ausdruck ausgewertet. Hat dieser den Wert 1 (*wahr*), so werden die Anweisungen des if-Blockes ausgeführt, andernfalls wird in die Zeile nach `end` gesprungen.

Als Beispiel berechnen wir die Quadratwurzel einer nichtnegativen reellen Zahl:

```
...
if a >= 0
  b = sqrt(a)
end
...
```

Zweiseitige Verzweigung

Hier hat man zusätzlich noch die Möglichkeit, einen Programmteil auszuführen, falls der *logische Ausdruck* den Wert 0 (*falsch*) hat. Die Form ist

```
if      logischer Ausdruck
   Anweisungen ("if-Block")
else
   Anweisungen ("else-Block")
end
```

Hat bei der Programmausführung der *logische Ausdruck* den Wert 1 (*wahr*), so wird der if-Block ausgeführt; hat er den Wert 0 (*falsch*), so wird der else-Block ausgeführt.

Als Beispiel berechnen wir wieder die Quadratwurzel einer reellen Zahl. Ist diese Zahl negativ, so soll nun eine Fehlermeldung ausgedruckt werden.

```
...
if a >= 0
   b = sqrt(a);
   text=sprintf('Die Quadratwurzel ist: %8.4f',b);
   disp(text);
else
   disp('Fehler: eingegebener Wert ist negativ');
end
...
```

In einem if-Block bzw. else-Block dürfen mehrere Anweisungen stehen, darunter selbst wieder if-Anweisungen.

Mehrfachverzweigungen

Auch Mehrfachverzweigungen können wir in Matlab realisieren. Diese haben die Struktur

`if` *logischer Ausdruck 1*

Anweisungen ("if-Block")

`elseif` *logischer Ausdruck 2*

Anweisungen ("elseif - Block")

`else`

Anweisungen ("else-Block")

`end`

Bei der Programmausführung wird zuerst der *logische Ausdruck 1* ausgewertet. Hat dieser den Wert 1 (*wahr*), so wird der if-Block ausgeführt. Hat der *logische Ausdruck 1* den Wert 0 (*falsch*), dann wird als nächstes der *logische Ausdruck 2* berechnet; hat er den Wert 1 (*wahr*), so wird der elseif-Block ausgeführt, andernfalls der else-Block. In den else-Block gelangt man also nur dann, wenn sowohl der *logische Ausdruck 1* als auch der *logische Ausdruck 2* den Wert 0 (*falsch*) haben.

Als Beispiel berechnen wir die sogenannte Signumsfunktion, welche definiert ist durch

$$\text{sign}(x) := \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x = 0 \\ -1 & \text{für } x < 0 \end{cases} .$$

In Matlab wird dies realisiert durch folgendes Programm:

```
x = input(' x = ');
if x > 0
  sigma = 1;
elseif x == 0
  sigma = 0;
else
  sigma = -1;
end
text=sprintf(' sigma(x) = %g',sigma);
disp(text);
```

Wir erwähnen noch, daß auch mehrere elseif -Teile möglich sind.

Als Alternative zu solchen geschachtelten if-Anweisungen bei Mehrfachverzweigungen gibt es die switch-Anweisung, die folgenden formalen Aufbau hat:

```
switch  ausdruck
case wert1
     1. case-Block
case wert2
     2. case-Block
  ⋮
case wertn
     n. case-Block
otherwise
     sonst-Block
END
```

Hat der *ausdruck* den Wert *wert1*, so werden die Anweisungen des 1. case-Blockes ausgeführt; hat er den Wert *wert2*, so werden die Anweisungen des 2. case-Blockes ausgeführt usw. Hat *ausdruck* einen Wert, welcher von *wert1* bis *wertn* verschieden ist, so wird der sonst-Block ausgeführt. Der otherwise-Teil darf auch weggelassen werden.

Testen Sie dazu das folgende Matlab-Programm:

```
n = input(' n = ');
switch n
  case 1
    disp('Januar');
  case 2
    disp('Februar');
  case 3
    disp('März');
  case {4,5,6}
    disp('Frühjahr');
  case {7,8,9,10,11,12}
    disp('2. Halbjahr');
  otherwise
    disp('falsche Eingabe');
end
```

7.3 Schleifenbildung

Beim Programmieren besteht häufig der Wunsch, gewisse Programmteile mehrfach zu durchlaufen (mit verschiedenen Daten). Dazu dient das Konzept der Schleifenbildung. Matlab kennt verschiedene Möglichkeiten:

Die for-Schleife

Ist im voraus bekannt, wie oft ein Programmteil (eine Schleife) durchlaufen werden soll, so verwendet man die sogenannte for-Schleife. Sie hat die Form

`for` *index = start:inkrement:ende*

Anweisungen

`end`

Ist das Inkrement 1, so darf es auch weggelassen werden. Das folgende Beispiel belegt einen Vektor.

```
for i = 1:10
   x(i) = 2*i
end
```

Selbstverständlich dürfen die Schleifen auch geschachtelt werden, zum Beispiel

```
for i = 1:5
  for j = 1:3
    A(i,j) = i+j-1
  end
end
```

Die while-Schleife

Hängt die Anzahl der Schleifendurchläufe von einer logischen Bedingung ab, so verwendet man die while-Schleife:

`while` *logischer Ausdruck*

Anweisungen

`end`

Dazu betrachten wir ein einfaches Beispiel:

```
disp('Gib eine positive Zahl ein ');
x = input('x = ');
while x <= 0
   disp('Fehler: x <= 0');
   disp('Gib eine positive Zahl ein');
   x = input('x = ');
end
y = log(x)
```

Hier wird so lange eine reelle Zahl über den Bildschirm eingelesen, bis eine positive Zahl eingegeben wird. Erst dann wird mit der Berechnung von `log(x)` fortgefahren.

Vorzeitiges Beenden einer Schleife

Durch den Matlab-Befehl

```
break
```

im Anweisungsteil einer Schleife wird die Schleife sofort beendet und zu der Zeile nach `end` gesprungen.

Durch das Kommando

```
continue
```

wird der aktuelle Schleifendurchlauf abgebrochen und die Schleifenabarbeitung mit dem nächsten Schleifendurchlauf fortgesetzt.

Anmerkung zur Schleifenbildung

Matlab ist eine Sprache, die für den Umgang mit Matrizen und Vektoren entwickelt wurde, d.h. Operationen mit Matrizen und Vektoren werden schnell ausgeführt. Schleifen erfordern dagegen wesentlich mehr Rechenzeit. Deshalb sollten Sie die Schleifenbildung möglichst vermeiden. Im folgenden wird eine Wertetabelle von $\sin(t)$ erstellt; dies kann in Matlab auf verschiedene Arten realisiert werden:

1. Möglichkeit

```
tic
i = 0;
for t = 0:0.01:10
  i = i +1;
  y(i) = sin(t);
end
toc
```

2. Möglichkeit

```
tic
t = 0:0.01:10;
y = sin(t);
toc
```

Das zweite Beispiel wird wesentlich schneller ausgeführt. Bei dieser Gelegenheit lernen wir auch noch die Matlab-Befehle `tic` und `toc` kennen, welche die benötigte Rechenzeit der zwischen tic und toc eingeschlossenen Kommandos bestimmen.

7.4 Umgang mit Funktionen

Funktionen

Selbstdefinierte Funktionen sind Matlab-Programme, welche Eingabe-Parameter akzeptieren und ein Ergebnis zurückliefern. Sie benutzen einen eigenen Workspace. Funktionen haben folgenden Aufbau:

```
function y = mittelwert(x)
% Kommentarzeilen
      Matlab-Anweisungen
y =  ....
```

Die erste Zeile beginnt immer mit dem Schlüsselwort `function`, danach kommt die Ausgabevariable (hier `y`), dann der Name der Funktion (hier `mittelwert`) und die Eingabeparameter. Das Programm ist in die Datei mit dem Namen *<Funktionsname>* `.m` (hier also `mittelwert.m`) zu speichern. In der zweiten Zeile folgt ein Kommentar, eingeleitet durch das Zeichen % (es sind auch mehrere Kommentarzeilen möglich). Danach folgen die Berechnungen (Matlab-Anweisungen). Am Ende erfolgt eine Wertzuweisung auf den Ausgabeparameter.

Als Beispiel erstellen wir eine Funktion, welche zunächst die Länge eines Vektors feststellt und dann denMittelwert berechnet.

```
function mw = mittelwert(u)
% Diese Funktion berechnet den Mittelwert eines Vektors
n = length(u);
mw = sum(u)/n;
```

Dieses Programm muß in eine Datei namens `mittelwert.m` gespeichert werden. Der Befehl `length(u)` bestimmt die Länge des Vektors u, `sum(u)` summiert die Komponenten auf.

Im folgenden Beispiel wird diese Funktion nun aufgerufen:

```
u = [1 2 3 4 5 6];
mittelwert(u)
z = [10 20 30 40 50];
mittelwert(z)
```

Mit dem Matlab-Kommando `help mittelwert` wird die Kommentarzeile auf dem Bildschirm ausgegeben.

Unterfunktionen

Jede Funktion, die Sie während Ihrer Matlab-Sitzung aufrufen wollen, muß sich in einer eigenen Datei befinden. Werden jedoch innerhalb der Funktion weitere Funktionen aufgerufen, so dürfen diese in derselben Datei stehen. Solche Funktionen werden als Unterfunktionen bezeichnet; sie können von anderen Matlab-Programmen jedoch nicht direkt aufgerufen werden. Dazu betrachten wir folgendes Beispiel, welches in die Datei `auswert.m` gespeichert wird.

```
function [mittel,fehler] = auswert(u)
mittel = mittelwert(u);
fehler = quadrat(u,mittel);

function mw = mittelwert(u)
% Diese Funktion berechnet den Mittelwert eines Vektors
n = length(u);
mw = sum(u)/n;

function error = quadrat(u,mw)
error = sum((u-mw).^2)
```

Hier werden zwei Unterfunktionen (`mittelwert` und `quadrat`) definiert, die von der eigentlichen Funktion (`auswert`) intern aufgerufen werden. In einer Matlab-Sitzung würde man die Funktion `auswert` etwa so aufrufen:

```
w = [1 2 3 4 5 6 7];
[a,b] = auswert(u)
```

Funktionen als Parameter in anderen Funktionen

Eine Funktion darf als Parameter auch eine andere Funktion aufrufen. Dabei sind jedoch einige Dinge zu beachten, was anhand eines Beispiels deutlich wird: In die Datei `funkzeichne.m` schreiben wir die folgenden Matlab-Funktion:

```
function x = funkzeichne(funk,data)
werte = feval(funk,data);
plot(data,werte);
```

welche eine andere Funktion zeichnet. Dabei ist `data` ein Vektor, der die Werte aus dem Definitionsbereich enthält, an denen die Funktion ausgewertet wird. Der Parameter `funk` ist die zu zeichnende Funktion. Das Matlab-Kommando `feval` wertet die Funktion `funk` an den Punkten `data` aus.
In einer Matlab-Sitzung wird dann die Funktion `funkzeichne` wie folgt aufgerufen:

```
t = -3:0.01:3;
funkzeichne(@sin,t);
```

Dadurch wird die Sinus-Funktion im Intervall $[-3, 3]$ gezeichnet. Beachten Sie, daß die Übergabe der Funktion `sin` als Parameter `@sin` lautet, also mit vorangestelltem @-Zeichen. Entsprechend geht man bei selbst definierten Funktionen vor.

7.5 Komplexe Zahlen

Matlab kann auch mit komplexen Zahlen arbeiten. Als imaginäre Einheit wird `i` oder `j` verwendet. Es sind dann die aus der Mathematik bekannten Operationen mit komplexen Zahlen möglich, zum Beispiel

```
z = 2 + 3i;
u = 5 - 6.3i;
v = z.*u;
r = abs(z);
```

Dabei ist `abs(z)` der Betrag einer komplexen Zahl. Desweiteren liefern die Funktionen

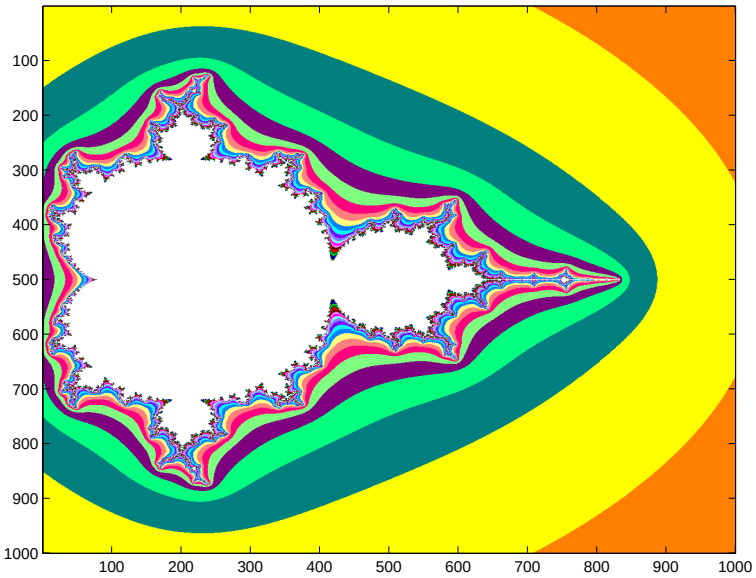
`real(z)` bzw. `imag(z)`

den Realteil bzw. den Imaginärteil einer komplexen Zahl. Sind `x` und `y` reelle Zahlen, so erzeugt

```
z = complex(x,y)
```

eine komplexe Zahl mit Realteil `x` und Imaginärteil `y`.

Das folgende Matlab-Programm zeichnet die bekannte Mandelbrotmenge.



```
% zeichnet die Mandelbrotmenge
N = 32;
L1 = 600;
L2 = 600;
C = zeros(L1,L2);
rmax = 15;
for j = 1:L1
  for k = 1:L2
    count = 0;
    z = complex(0,0);
    c = complex(-0.5 + 3.*j./L1, -1.5 + 3.*k./L2);
    r = 0;
    while ((r < rmax) & (count < N))
      z = z*z-c;
      r = abs(z);
      count = count+1;
    end
    C(j,k) = count;
  end
end
colormap(colorcube(N));
image(C');
```