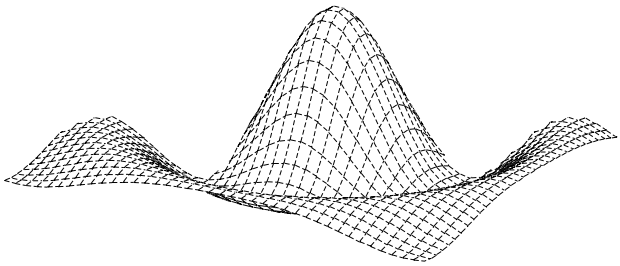
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Einführung in

MATLAB



FB Physik, Mathematik & Informatik · Johannes Gutenberg–Universität Mainz
Staudingerweg 9 · D-55099 Mainz

- erstellt für das Sommersemester 2004 von B. Düring, B. Gebauer, M. P. Gualdani, S. Holst und B. Schappel (Institut für Mathematik, Joh. Gutenberg Universität Mainz, 55099 Mainz)
- aktualisiert für das Sommersemester 2007 von S. Krause, P. Milisic und C. Schneider (Institut für Mathematik, Joh. Gutenberg Universität Mainz, 55099 Mainz)
- aktualisiert für das Sommersemester 2011 von H. Schier (Institut für Mathematik, Joh.Gutenberg Universität Mainz, 55099 Mainz)

Was ist MATLAB?

MATLAB (Matrix Laboratory) ist eine interaktive Interpreter-Sprache, die einen einfachen Zugang zu grundlegenden numerischen Verfahren – wie beispielsweise der Lösung linearer Gleichungssysteme oder ähnlichem – bietet.

Wie startet man MATLAB?

In den Rechner-Pools im 3., 4. oder 5. Stock des Mathematik-Gebäudes loggt man sich mit seinem ZDV-Account an einem Rechner ein. Auf diesen Rechnern selbst ist MATLAB im Regelfall bereits lokal installiert.
Sollte das nicht der Fall sein, muss man sich zusätzlich durch

Start → Alle Programme → Zubehör → Kommunikation → Remotedesktopverbindung

auf einem ATS-Server des ZDV anmelden.

Dazu gibt man im sich öffnenden Dialog-Fenster seinen ZDV-Benutzernamen und sein Passwort an. Ferner muss man im Feld „Anmelden an:“ noch „Uni Mainz“ auswählen.

Hilfe?!

Mittels

Help → Product Help (F1)

kann man sich die MATLAB-Hilfe ansehen. Im Navigationsbaum im linken Teil des Fensters findet man unter

MATLAB → Functions

eine Liste aller MATLAB-Befehle. Über dem Navigationsbaum befindet sich das Suchfeld, mit dem man auch explizit nach Befehlen suchen kann.
Eine weitere Alternative stellt der „Function Browser“ dar, den man über

Help → Function Browser (Umschalt + F1)

erreicht. Hier werden allerdings Pop-Ups über die MATLAB-Konsole gelegt.
Zum Beispiel liefert die Eingabe des Begriffs „eigenvalue“ im Suchfeld schnell einen Überblick, wie man mit MATLAB die Eigenwerte einer Matrix berechnen kann.
Außerdem kann man auch direkt in MATLAB die Befehle `help` oder `help name` eingeben:

- `help` listet alle Befehle und internen Variablen auf.
- `help name` gibt Hilfetext zur Variable oder Funktion „name“ aus.

Beispiel: Beschreibung der Methode zur Eigenwertberechnung einer Matrix

```
>> help eig
```

Ein ausführliche Beschreibung eines MATLAB Befehles erhält man durch den Befehl `doc name` eingeben:
Beispiel: Ausführliche Beschreibung der Methode zur Eigenwertberechnung einer Matrix

```
>> doc eig
```

Variablen

Der Begriff steht für einen Bezeichner unter dem beliebige Daten, z.B. Zahlen, sonstige Zeichen, sowie Felder von Zahlen oder Zeichenketten, abgelegt sein können. Die Zuordnung von Daten erfolgt durch den Befehl:

`[ varname = ]expression[;]` ([...] kann entfallen)

Das Gleichheitszeichen steht hier für eine Zuweisung, es ist nicht wie eine mathematische Gleichung zu lesen. Im Detail heißt das, daß zuerst der Wert des Ausdrucks „expression“ bestimmt wird, und dann dieser Wert unter dem Bezeichner „varname“ abgelegt wird. Wird varname nicht angegeben, wird der Wert dem Bezeichner „ans“ (für answer) zugewiesen.

- `varname = expression` weist der Variablen „varname“ den Wert von „expression“ zu.
- Strings (Zeichenketten) werden durch einfache Hochkommata „`'`“ umschlossen und so von Variablennamen unterschieden.

Es können mehrere Befehle in einer Zeile zusammengefaßt stehen:

- `;` trennt Befehle und unterdrückt eine Ausgabe.
- `,` trennt Befehle, gibt aber Werte aus.

Beispiel:

```
>> x12 = 1/8, long_name = 'Ein String'
x12 = 0.12500
long_name = Ein String
>> sqrt(-1)-i
ans = 0
```

Vorsicht: MATLAB unterscheidet zwischen Groß- und Kleinbuchstaben.

Es gibt die „üblichen“ Funktionen

`+, -, *, /, ^, sin, cos, exp, acos, abs, log, ...`

Beispiel:

```
>> x = sqrt(2); sin(x)/x
ans = 0.69846
```

Vektoren

Matrizen und Vektoren sind die wichtigsten Grundbausteine zur Programmierung in MATLAB.

Zeilenvektoren:
`v = [ 1 2 3 ]` oder `v = [ 1, 2, 3 ]` für $v = (1, 2, 3)$.

Spaltenvektoren:

`v = [ 1; 2; 3 ]` für $v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

Automatische Erzeugung einiger wichtiger Vektoren:

- `Anfang [: Inkrement] : Ende`

Beispiel:

```
>> x = 3:6
x =
  3  4  5  6
>> y = 0:.15:.7
y =
0.00000  0.15000  0.30000  0.45000  0.60000
>> z = [4:-1:0]*pi/4
z =
3.14159  2.35619  1.57080  0.78540  0.00000
```

Matrizen

```
>> A = [ 1 2; 3 4]
A =
  1  2
  3  4
```

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Für einige wichtige $(m \times n)$-Matrizen existieren eigene Befehle. Falls m gleich n ist, muss nur ein Argument angegeben werden.

- `eye(m,n)` erzeugt eine Matrix mit 1-ern auf der Hauptdiagonale. Im Spezialfall $m = n$ ergibt sich die Einheitsmatrix.
- `zeros(m,n)` erzeugt eine Nullmatrix der Dimension $(m \times n)$.
- `ones(m,n)` erzeugt eine Matrix der Dimension $(m \times n)$ mit $a_{ij} = 1 \; \forall i, j$.
- `rand(m,n)` erzeugt eine Zufallsmatrix der Dimension $(m \times n)$ mit gleichverteilten Einträgen aus $(0, 1)$. Andere Verteilungen siehe `help randn`.

Bestimmung der Dimension einer Variablen:

- `length(v)` gibt die Länge des Vektors v zurück.
- `[Zeilen,Spalten] = size(A)` gibt die Anzahl der Zeilen und Spalten von A zurück.

Grundlegende Operationen

Addition und Subtraktion bzw. Multiplikation erfolgen nun einfach durch $+$, $-$, $*$.

```
>> A = [1 2; 3 4]; B = 2*ones(2,2);
>> A+B, A-B, A*B
ans =
  3  4
  5  6
ans =
  -1   0
   1   2
ans =
   6   6
  14  14
```

- `A'` transponiert und konjugiert A.
- `A.'` transponiert A.

Elementweise Operationen

`*, /, ^` gibt es auch noch mit einem `.` vorgestellt, dann wird die Verknüpfung elementweise durchgeführt.
Beispiele:

```
>> x = 1:2; A = [1 2; 3 4];
>> [A.^2 A^2]
ans =
   1   4   7  10
   9  16  15  22
>> x.^2-x.*x
ans =
  0  0
```

Matrixmanipulationen

- `v(index)` wählt die durch „index“ spezifizierten Elemente aus dem Zeilen- oder Spaltenvektor „v“ aus.
- `A(index1,index2)` wählt die durch „index1“ und „index2“ spezifizierten Elemente aus der Matrix „A“ aus.
- `reshape(A,m,n)` formt A in eine $(m \times n)$-Matrix um. Dabei werden die Einträge von A spaltenweise gelesen und auch spaltenweise in die umgeformte Matrix geschrieben.
- `A(:)` gibt die Spalten der Matrix A hintereinander als Spaltenvektor aus.
- `diag(A[,k])` liefert die k-te Diagonale der Matrix A in einem Spaltenvektor.
- `diag(v[,k])` erzeugt eine Matrix mit dem Vektor v in der k-ten Diagonalen.
- `A(k,:) = []` löscht die k-te Zeile aus A.
- `A(:,k) = []` löscht die k-te Spalte aus A.

Beispiele: $A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$, $v = \begin{pmatrix} 7 \\ 8 \end{pmatrix}$.

```
>> A = [1 3 5; 2 4 6]; v = [7; 8];
>> v(2)
ans = 8
>> A(2,3)
ans = 6
```

```
>> A(2,2:3)
ans =
  4  6
>> A(1,:)
ans =
  1  3  5
>> reshape(A,3,2)
ans =
  1  4
  2  5
  3  6
>> A(:)'
ans =
  1  2  3  4  5  6
>> A([2 1],:)
ans =
  2  4  6
  1  3  5
```

Was ist `diag(ones(3,1))`?

Logische Operatoren

`<`	kleiner	`<=`	kleiner oder gleich
`>`	größer	`>=`	größer oder gleich
`==`	gleich	`~=`	ungleich
`&`	und	`|`	oder
`~`	nicht		

Werte logischer Ausdrücke: 0 entspricht `false`, alle anderen Werte entsprechen `true`.

Funktionen und Skripte

- `whos` zeigt alle selbstdefinierten Variablen und Funktionen an.
- `clear name` löscht „name“ aus dem Speicher; falls „name“ nicht angegeben wird, werden *alle* Variablen und Funktionen gelöscht.
- `type name` gibt die Funktion „name“ am Bildschirm aus.

MATLAB läßt sich durch eigene Skripte und Funktionen leicht erweitern. Sowohl Skripte als auch Funktionen sind einfache *Textdateien*, die den Suffix `.m` haben.
Hierzu können folgende Befehle benutzt werden:

- `edit` öffnet ein Fenster eines Texteditors.
- `edit datei` öffnet die Datei „datei.m“ im Texteditor.

Funktionen können mit Argumenten aufgerufen werden und liefern am Ende einen oder mehrere Rückgabewerte. Alle innerhalb der Funktion verwendeten Variablen sind *lokal*, d.h. sie beeinflussen die vorher definierten Variablen nicht.

Skripte hingegen verhalten sich so, als ob deren Inhalt Befehl für Befehl am Prompt eingegeben würde.

Eine Funktion `dolittle`, die in der Datei `dolittle.m` gespeichert ist, könnte etwa folgendermaßen aussehen:

```
function [out1,out2] = dolittle(x)
% Das ist ein Kommentar zu dolittle
out1 = x^2;
out2 = out1*x;
```

Aufruf der Funktion:

```
>> [x1,x2]=dolittle(2)
x1 = 4
x2 = 8
```

Die beiden Variablen `out1`, `out2` in `dolittle` sind also nur lokal.
Wichtig: Eine vorher definierte Variable `out1` oder `out2` wird durch den Aufruf der Funktion *nicht* beeinflußt.
Ein Skript `doless`, das in der Datei `doless.m` gespeichert ist, könnte etwa folgendermaßen aussehen:

```
eins = 1;
zwei = 2;
drei = eins + zwei
```

Aufruf dieses Skriptes:

```
>> doless
drei = 3
```

Schleifen und bedingte Anweisungen

Syntax	Beispiel

`for name = expr`
`  ...`
`end`
	`for n = 1:10`
	`  [x(n),y(n)]=dolittle(n);`
	`end`

`if condition`
`  ...`
`[else`
`  ...]`
`end`
	`if x==0`
	`  error('x ist 0!');`
	`else`
	`  y = 1/x;`
	`end`

`if condition`
`  ...`
`[elseif`
`  ...]`
`[else`
`  ...]`
`end`
	`if a<b`
	`  erg = 0;`
	`elseif a==b`
	`  erg = 1;`
	`else`
	`  erg = 2;`
	`end`

`while condition`
`  ...`
`end`
	`while t<T`
	`  t = t+h;`
	`end`

Achtung: Schleifen und Funktionsaufrufe sind sehr teuer, deswegen möglichst alle Operationen „vektorisieren“!

Vergleich von Programmen oder Funktionen

- `tic`, *Befehle*, `toc` gibt nach dem Ausführen von *Befehle* die hierzu benötigte Zeit an. Der Befehl `cputime` liefert die von Matlab seit dem Start verwendete Prozessorzeit.

Ein- und Ausgabe

- `save datname [var1 [var2 ...]]` speichert alle bisher verwendeten Variablen bzw. die angegebenen Variablen „var1“ usw. in der Datei „datname.mat“ ab. Die Endung „.mat“ wird automatisch an den Dateinamen angehängt.
- `load datname` liest die Datei „datname.mat“.
- `format [long|short|long e|short e]` vergrößert oder verkleinert die Anzahl der ausgegebenen Kommastellen bei Festkommazahlen bzw. Gleitkommazahlen. Nur `format` stellt das Standardverhalten wieder her.
- `disp(string)` gibt den String `string` auf dem Bildschirm aus.

Beispiel:

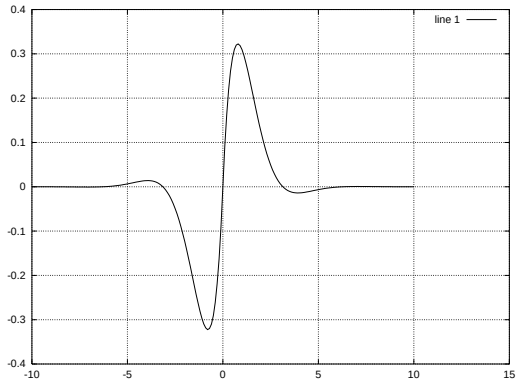
```
>> x = 10.5;
>> disp(['Der Wert von x ist ', num2str(x)]);
Der Wert von x ist 10.5
```

2D-Grafik

- `plot(x,y[,fmt])` zeichnet eine Linie durch die Punkte (x_j, y_j). Mit dem String „fmt“ können Linienart und Farbe gewählt werden (siehe `help plot`).
- `semilogx(x,y[,fmt])` Syntax wie `plot`, die x-Achse wird logarithmisch gezeichnet.
- `semilogy(x,y[,fmt])` Syntax wie `plot`, die y-Achse wird logarithmisch gezeichnet.
- `loglog(x,y[,fmt])` Syntax wie `plot`, beide Achsen werden logarithmisch gezeichnet.

Beispiel:

```
>> x = -10:.1:10;
>> y = sin(x).*exp(-abs(x));
>> plot(x,y); grid;
```



Befehle für 2D-Grafiken:

- `title(string)` schreibt „string“ als Titelzeile in die Grafik.
- `xlabel(string)` beschriftet die x-Achse mit „string“.
- `ylabel(string)` beschriftet die y-Achse mit „string“.
- `axis(v)` legt den Ausschnitt fest. v ist ein Vektor der Form v = (xmin, xmax, ymin, ymax).
- `hold [on|off]` legt fest, ob die nächste Grafikausgabe die alte Grafik löschen soll.
- `close` schließt das Fenstr mit der letzten Grafikausgabe.

3D-Grafik

- `[X,Y] = meshgrid(x,y)` erzeugt aus zwei Vektoren x und y zwei Matrizen X und Y. Die Matrix X hat `length(y)`-viele Zeilen, die alle gerade der Vektor x sind. Die Matrix Y hat `length(x)`-viele Spalten, die alle gerade der Vektor y sind. Beispiel:

```
>> x = [1 2]; y=[3 4 5]; [X,Y]=meshgrid(x,y)
X = 1  2
    1  2
    1  2
Y = 3  3
    4  4
    5  5
```

- Enthält Z die Werte von $f(X,Y)$, so kann man die Funktion mittels der Befehle `mesh(X,Y,Z)`, `surf(X,Y,Z)` oder `surf(x,y,Z)` plotten.

Mit Hilfe der folgenden Programmzeilen kann man den Graphen der Funktion auf dem Titelblatt dieses Skripts zeichnen:

```
>>[X,Y]=meshgrid(-8:0.05:8);
>>R=sqrt(X.^2+Y.^2)+eps;
>>Z=sin(R)./R;
>>surf(X,Y,Z,'EdgeColor','none');
>>camlight left;
```

Logische Matrizen

In logische Ausdrücke können Matrizen eingesetzt werden. Das Ergebnis ist eine *logische Matrix* aus 1-en und 0-en, wobei an der (i,j)-ten Stelle eine 1 steht, wenn der entsprechende Matrixeintrag den logischen Ausdruck erfüllt.

```
>> A = [ 1, 2; 3, 4];
A =   1     2
      3     4
>> A>2
ans = 0     0
      1     1
```

Eine logische Matrix kann direkt als Index verwendet werden

```
>> A(A>2)=7
A =   1     2
      7     7
```

oder mit `find` in Zeilen- und Spaltennummern umgewandelt werden.

```
>> [Zeile,Spalte]=find(A>2)
Zeile =
     2
     2
Spalte =
     1
     2
```

Mit Hilfe der Befehle `any(...)` und `all(...)` kann man überprüfen, ob irgendein bzw. alle Einträge eines Vektors ungleich Null sind, siehe `help any` und `help all`. Auch hier können logische Matrizen eingesetzt werden.

```
if any(any(X==0))
  error('In der Matrix ist mindestens ein Eintrag gleich Null!');
else
  Y = 1./X;
end;
```

Weiterführende Literatur

- MATLAB Primer (Third Edition):
  `http://www.math.toronto.edu/mpugh/primer.pdf`
- MATLAB Einführung der Universität Hamburg:
  `http://www.rrz.uni-hamburg.de/RRZ/W.Wiedl/Skripte/Matlab`
- MATLAB-Kurs der Uni-Karlsruhe:
  `http://www.rz.uni-karlsruhe.de/rd/1747.php`
- B. Hunt: A guide to MATLAB for beginners and experienced users. Dieses Buch kann man sich der Bereichsbibliothek Physik, Mathematik und Chemie ansehen.

Übungen

1. Andreas, Burkhard und Christian haben am 30. März Geburtstag. Zusammen sind sie jetzt 200 Jahre alt. Vor 60 Jahren war Christian ebenso alt wie Andreas und Burkhard zusammen. Der Altersunterschied zwischen Christian und Burkhard ist doppelt so groß wie der zwischen Burkhard und Andreas. Beantworte mit Hilfe von MATLAB die Frage, wie alt die drei heute sind.

   Hinweis: `x=A\b` berechnet die Lösung von $Ax = b$.

2. a) Schreibe eine MATLAB-Funktion

   `function y = signum(x)`,

   die das Signum einer gegebenen Zahl x berechnet und in y zurückgibt.

   b) Ändere die Funktion aus a) so ab, dass sie auch eine Matrix als Eingabe akzeptiert und eintragsweise das Signum berechnet.

3. Schreibe eine MATLAB-Funktion, die zu einem gegebenen Vektor x den Mittelwert und die Standardabweichung berechnet. Zur Erinnerung:

   - Mittelwert: $\overline{x} = \frac{1}{n} \sum_{j=1}^{n} x_j$
   - Standardabweichung: $s = \sqrt{\frac{1}{n-1} \sum_{j=1}^{n} (x_j - \overline{x})^2}$.

   Versuche die Funktion einmal mit und einmal ohne `for`-Schleifen zu implementieren.

4. Berechne ein Matrix-Vektor-Produkt einer (100×100)-Zufallsmatrix mit einem Zufallsvektor, indem einerseits die eingebaute Funktion `*`, andererseits `for`-Schleifen, verwendet werden. Welche Implementierung ist die schnellere?

5. Plotte die Funktion $f(x) = x \sin(1/x)$ auf dem Intervall $[-1, 1]$.

6. Die Zahl e läßt sich bekanntlich durch den Grenzwert

   $$e = \lim_{n \to \infty} (1 + \frac{1}{n})^n$$

   darstellen. Zur Berechnung von e soll nun dieser Grenzprozeß numerisch simuliert werden. Schreibe ein Programm, das für $n = 10, 10^2, \ldots, 10^{15}$ die Näherungen

   $$e_n := (1 + \frac{1}{n})^n$$

   und den absoluten Fehler

   $$|e - e_n|$$

   berechnet. Gib die Ergebnisse in einer Art Tabelle aus und vergleiche die Werte (gegebenenfalls muss man mit `format` ... das Ausgabeformat ändern). Was fällt auf?

7. Es soll ein Polynom $p(x) = \sum_{i=0}^{n} a_i x^i$ an einer Stelle x ausgewertet werden. Um Multiplikationen zu sparen, spaltet man $p(x)$ durch Klammerung auf in

$$p(x) = ((\ldots((a_n x + a_{n-1})x + a_{n-2})x + \\ \ldots)x + a_1)x + a_0$$

und berechnet die Klammern sukzessive von innen nach außen. Diese Vorgehensweise wird als Hornerschema bezeichnet.

a) Schreibe eine Funktion
`function p = horner(x,a)`,
die ein Polynom mit dem Hornerschema an der Stelle `x` auswertet. Die Koeffizienten a_i des Polynoms werden dabei im Eingabevektor `a` übergeben.

b) Modifiziere gegebenenfalls das Programm, damit es mehrere Funktionswerte gleichzeitig berechnen kann, falls `x` ein Vektor ist.