Technische Universität München
Zentrum Mathematik
Lehrstuhl für Numerische Mathematik

Einführung in MATLAB

Bernd Simeon, Meike Schaub, Michael Günther

Vorwort

Moderne mathematische Softwarewerkzeuge gewinnen immer mehr an Bedeutung. Auch und gerade in der Lehre entwickelt sich das Programmpaket MATLAB zu einem Standard. Diese Kurzeinführung ist als erster Einstieg und als Orientierung für Studierende und Mitarbeiter gedacht. Im ersten Teil werden die Grundlagen vermittelt und die Syntax an verschiedenen Beispielen vorgestellt. Der zweite Teil beschäftigt sich mit Laufzeitoptimierung, Grafikprogrammierung sowie einem Überblick über die Toolboxen.

1 MATLAB kennenlernen

1.1 Was ist MATLAB?

MATLAB (*MAT*rix *LAB*oratory) ist ein Softwarepaket für numerische Berechnungen und Visualisierung, das seit Ende der siebziger Jahre von C. Moler entwickelt wird. Seinen Ursprung hat es in den Lineare-Algebra Paketen LINPACK und EISPACK. Im Gegensatz zu MAPLE und MATHEMATICA können mit MATLAB keine symbolischen Berechnungen durchgeführt werden, es sei denn, die Symbolic Toolbox ist zusätzlich installiert (siehe den Abschnitt über Toolboxen).

Nach dem Aufruf von `matlab` erscheint die grafische Oberfläche. Im Command Window werden Eingaben direkt ausgewertet oder ausgeführt (Ausnahme: M-files, siehe Abschnitt 1.5).

`<command> <CR>`	Auswertung des Befehls `command`
`result = <expression> <CR>`	Zuweisung mit Anzeige des Ergebnisses
`result = <expression>; <CR>`	Zuweisung ohne Anzeige des Ergebnisses

Variablen- und Funktionsnamen in MATLAB sind case-sensitiv.

Im `Workspace` werden die bisher definierten Variablen und ihr Speicherbedarf angezeigt. Die Variablen können dort auch direkt betrachtet und verändert werden. Zusätzlich wird das aktuelle Verzeichnis eingeblendet sowie die `Command History`, die ein schnelles Wiederholen von Befehlen erlaubt.

Über die grafische Oberfläche stehen Hilfeseiten zur Verfügung. Informationen zum Befehl `<command>` erhält man über `help <command>`.

1.2 Zahldarstellung in MATLAB

Zahlen werden ausschließlich als doppelt genaue reelle (bzw. komplexe) Zahlen gemäß dem *IEEE Standard Binary Floating Point Arithmetic* dargestellt: 8 Byte (64 Bit) stehen zur Verfügung, und zwar 1 Bit für das Vorzeichen, 11 Bit für den Exponenten und 52 Bit für die Mantisse (normalisiert).

Kleinste darstellbare Zahl: `realmin`$\approx 2.2 \cdot 10^{-308}$, größte darstellbare Zahl: `realmax`$\approx 1.8 \cdot 10^{308}$.

Die Genauigkeit der Rundung $rd(x)$ einer reellen Zahl x ist durch die Konstante `eps` gegeben:

$$\left|\frac{x - rd(x)}{x}\right| \leq \texttt{eps} = 2^{-52} \approx 2.2 \cdot 10^{-16}.$$

Damit ergibt sich als Faustregel eine Genauigkeit von 16 Dezimalstellen. Exponentenunterlauf bzw. -überlauf erfolgt etwa ab 10^{-308} bzw. 10^{308}.

Arithmetische Katastrophen: Eine Division durch 0 führt auf das Ergebnis `Inf` (für *infinity*), und eine Division von 0 durch 0 auf `NaN` (für *Not a Number*).

1.3 Matrizen in MATLAB

Matrizen stellen die zentrale Datenstruktur in `MATLAB` dar mit den Sonderfällen Skalar und (Zeilen/Spalten)-Vektor. Die Dimensionen werden automatisch überwacht.

1.3.1 Eingabe und Erzeugung von Matrizen

Skalare werden durch direkte Zuweisung definiert. Die Anweisungen

```
s=5;
z=10+4i;
```

weisen s als Skalar den Wert 5 zu und z die komplexe Zahl $10 + 4i$.

Die Anweisungen

```
x=[1,2,3];
xt=[1;2;3];
y=1:0.1:2;
```

erzeugen den Zeilenvektor x mit den Einträgen $1, 2, 3$. Dessen Spaltenvektor xt kann auch einfacher durch die Zuweisung `xt=x'` mit dem Transponierungsoperator `'` erzeugt werden. Die letzte Anweisung definiert den Gittervektor
$y = (1, 1.1, 1.2, \ldots, 1.9, 2)$ (als Zeilenvektor).

Matrizen können direkt eingegeben werden:

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

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

wobei die Elemente durch ␣ oder `,` separiert werden. Ein `<CR>` ist mit dem Semikolon `;` äquivalent. Matrizen können auch durch Blöcke zusammengesetzt werden:

```
B = [A,[5;6]];
```

$$B = \begin{pmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{pmatrix}.$$

Elemente von Matrizen und Vektoren werden durch Indizierung ab 1 spezifiziert: `A(2,1)` bezeichnet das Element a_{21} der Matrix A und `x(5)` das fünfte Element des Vektors x.

Mit dem Indexoperator `:` können einfach Untermatrizen gebildet werden:

`A(:,1)`	erste Spalte von A
`A(2,:)`	zweite Zeile von A
`A(2:end,1:2)`	Untermatrix $\begin{pmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$

Dabei bezeichnet `end` den letzten möglichen Index.

1.3.2 Operationen

Die Operatoren `+ - * ^` haben die üblichen Bedeutungen. Damit sind für die Matrizen A und B und den Skalar s definiert:

`s*A`, `A*B`, `A+B`, `A^2` und `s+A` $= (s \cdot \delta_{ij} + a_{ij})$.

Die Division mit `/` und `\` spielt eine besondere Rolle:

`x=b/A`	löst das lineare Gleichungssystem	$x \cdot A = b$,
`x=A\b`	löst das lineare Gleichungssystem	$A \cdot x = b$.

Unter- bzw. überbestimmte Systeme sind zugelassen, solange die Dimensionen von x, b und A zueinander passen (vgl. hierzu auch den nächsten Abschnitt).

Zusätzlich existieren elementweise Operationen: `A.*B`$= (a_{ij} \cdot b_{ij})$, `A.^B`$= (a_{ij}^{b_{ij}})$, und analog für `./` und `.\`, sowie Operatoren für die Relationen `<`, `==`, `>=` und `<=`. Die Anweisungen

```
x=[1,2,3]; x<=1;
```

liefern als Antwort den booleschen Vektor `ans=[1,0,0]`. Als logische Operatoren stehen zur Verfügung: `&` für $\wedge$, `|` für $\vee$, `~` für $\neg$ und `xor` für $\dot{\vee}$.

1.3.3 Standardalgorithmen der Linearen Algebra

Als Erbe von `LINPACK` und `EISPACK` enthält `MATLAB` eine Fülle von Standardalgorithmen der Linearen Algebra (wobei die Implementierung inzwischen auf `LAPACK` basiert):

1. *LU-Zerlegung*

   ```
   [L,U]=lu(A);
   y=L\b; x=U\y;
   ```

   Durch den ersten Befehl wird eine LU-Zerlegung der Matrix A mit Spaltenpivotsuche durchgeführt: $L \cdot U = P \cdot A$; anschließend durch Vorwärts- und Rückwärtssubstitution das Gleichungssystem $A \cdot x = b$ gelöst (oben enthält `L` aus `[L,U]=lu(A);` zusätzlich die Information von P^T).

   Der Befehl `x=A\b` liefert für reguläre Matrizen A aus $\mathbb{R}^{n \times n}$ das gleiche Ergebnis. Ansonsten wird eine Least-Square-Lösung mit dem QR-Verfahren berechnet (siehe unten).

2. *QR-Zerlegung*

   ```
   [Q,R]=qr(A);
   y=Q'*b; x=R\y;
   ```

   Es wird eine QR-Zerlegung der Matrix A ohne Pivotsuche durchgeführt: $A = Q \cdot R$. Anschließend wird das Gleichungssystem $A \cdot x = b$ gelöst. Eine QR-Zerlegung der Form $P \cdot A = Q \cdot R$ mit Pivotsuche erhält man durch `[Q,R,P] = qr(A);`.

Weitere wichtige Befehle sind:

`[U,S,V]=svd(A);`	Singulärwertezerlegung von A: $A = U \cdot S \cdot V^T$ mit Singulärwerten $S = \text{diag}\,(s_1, \ldots, s_n)$
`cond(A);`	Kondition von A, also $\max \lvert s_i \rvert / \min \lvert s_i \rvert$
`[X,D] = eig(A);`	Eigenwerte D (als Diagonalmatrix) und Eigenvektoren X (spaltenweise) der Matrix A

1.3.4 Funktionen

`MATLAB` stellt die üblichen Standardfunktionen `sin`, `cos` usw. auch für Matrizen zur Verfügung:

$$\texttt{sin(A)} = (sin(a_{ij})) \qquad \text{(elementweise Auswertung).}$$

Zusätzlich kann man Funktionen über M-files oder als `inline`-Funktion definieren.

1.4 Strukturvektor

Neben den Matrizen gibt es noch eine weitere Datenstruktur in `MATLAB`, den Strukturvektor, der Felder unterschiedlichen Typs in einem Vektor zusammenfasst. Jedes Feld kann dabei vom Typ Matrix, String oder einem weiteren Strukturvektor sein. Bezeichnet werden die einzelnen Felder dann mit

```
s.feldname
```

mit dem Strukturvektor `s` und dem Feld mit Namen `feldname`. Sowohl der Strukturvektor als auch die einzelnen Felder können zusätzlich indiziert sein, so dass auch z.B.

```
s(k).A(i,j)=5
```

als Kommando zugelassen ist.

Beispiel:

```
s(1).name = 'Student';
s(1).name = 'Karl';
s(1).jahr = '2004';
s(1).note = '2.3';
s(2).name = ...
```

1.5 M-files

M-files sind Dateien mit der Endung `.m`, die Skripten (Sequenzen häufiger gebrauchter Befehle) oder `MATLAB`-Unterprogramme enthalten.

Der `Search Path` legt dabei fest, in welchen Unterverzeichnissen außer dem aktuellen Verzeichnis nach M-Files gesucht wird. Er kann über den Menupunkt `File`, → `Set Path` editiert werden.

Beispiele:

- Skript `solveqr.m`

```
[Q,R]=qr(A)
y=Q'*b; x=R\y;
```

  Es erfolgt keine Parameterübergabe, alle Variablen sind global. Aufruf des Skripts erfolgt durch `solveqr`.

- Funktion `sinquad.m`

```
function y = sinquad(x)
y=sin(x).*sin(x);
```

  Als Parameter wird die Matrix x übergeben. Aufruf der Funktion erfolgt durch `sinquad(x)`.

Die Verwendung von lokalen Variablen ist möglich. Globale Variablen werden vor dem ersten Auftreten und in jedem Unterprogramm durch `global` definiert. M-files sind das zentrale Konstrukt in `MATLAB`, um Funktionen oder Prozeduren mit Variablen einzuführen. Die M-Files zu den `MATLAB`-internen Funktionen befinden sich im Unterverzeichnis `toolbox/matlab/` ausgehend vom `MATLAB`-Installationsverzeichnis.

1.6 Bedingte Verzweigungen

Um bedingte Verzweigungen zu realisieren, stehen die Befehle `if` und `switch` zur Verfügung. Zum Beispiel kann die Signum-Funktion über

```
function s = sign(x)

if x<0
    s = -1;
```

```
elseif x==0
    s = 0;
else
    s=1;
end
```

berechnet werden. Treten mehrere Fälle auf, kann die `switch`-Anweisung verwendet werden.

```
switch sign(x)
  case -1
    disp('x ist negativ')
  case 0
    disp('x ist null')
  otherwise
    disp('x ist positiv')
end
```

1.7 Schleifen

Prinzipiell gibt es zwei unterschiedliche Schleifen, die `for`- und die `while`-Schleife. Die `for`-Schleife arbeitet mit einem Schleifenzähler und sieht zur Berechnung der Fibonacci-Folge so aus:

```
x(1) = 1; x(2) = 1;
for i = 3:100
    x(i) = x(i-1) + x(i-2);
end
```

Die `while`-Schleife bricht ab, sobald eine Bedingung erfüllt ist. Bei iterativen Lösern erhält man zum Beispiel

```
err=1; tol=1e-5;
while norm(err)>=tol
    ...
    err = ...
end
```

1.8 Grafik in `MATLAB`

Zentrales Kommando für zweidimensionale Grafiken in `MATLAB` ist `plot`:

```
plot(x,y)
```

plottet den Vektor y gegen den Vektor x. Die Anweisung

```
plot(x1,y1,'o',x2,y2,'--g')
```

plottet den Vektor $y1$ gegen den Vektor $x1$ mit der Markierung "o". Der Vektor $y2$ wird gegen den Vektor $x2$ in grün mit "- -" (gestrichelte Linie) geplottet.

Für dreidimensionale Grafiken muss zuerst mit der Anweisung `meshgrid` ein Gitter generiert werden:

```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
```

erzeugt ein Gitter im Bereich $[-2, 2] \times [-2, 2]$ mit Gitterweite 0.2. Die Anweisung (beachte die Verwendung elementweiser Operatoren)

```
Z = X .* exp(-X.*X - Y.*Y);
```

wertet nun die Funktion $x \cdot \exp(-x^2 - y^2)$ auf diesem Gitter aus. Mit

```
mesh(X,Y,Z)
```

oder `surf(X,Y,Z)` kann das Ergebnis als 3-d Grafik geplottet werden.

Weitere Details zur Grafik im Abschnitt 2.2.

1.9 Daten einlesen und speichern

Zum Einlesen und Speichern stehen die Befehle `load` und `save` zur Verfügung. Die Anweisungen

```
load results.dat
x=results(1,:);
y=results(2,:);
plot(x,y)
```

lesen die Datei `results.dat` ein und speichern die erste Zeile der Daten im Zeilenvektor x, und die zweite Zeile im Zeilenvektor y ab. Anschließend wird der Vektor y gegen den Vektor x geplottet.

Das Kommando

```
save amat.dat A -ascii
```

speichert die Matrix A in `amat.dat` als ASCII-Datei ab.

Ohne den Zusatz `-ascii` speichert `MATLAB` in einem eigenen binären Format ab, was im Gegensatz zum `ascii`-Format größere Datenmengen erlaubt. Ohne Namenszusatz erzeugt dies ein `.mat`-File.

1.10 Sonstige wichtige Kommandos

`clear`	löscht alles
`clf`	löscht Grafikfenster
`clc`	putzt Eingabefenster
`tic; <command>; toc`	Zeitmessung
`diary <filename>`	Protokoll in `<filename>`
`diary off`	Ende des Protokolls

2 Spezielle Themen

Der zweite Teil beschäftigt sich mit der Laufzeitoptimierung, Grafikprogrammierung und den Toolboxen. In dieser Einführung kann nur auf die wesentlichen Punkte eingegangen werden, bezüglich der Details muss auf die entsprechenden Handbücher verwiesen werden.

2.1 Laufzeitoptimierung

M-Files sind wegen dem Interpretermodus im Vergleich zu C- oder Fortran-Code eventuell langsam. Oft lassen sich die M-Files aber optimieren.

Beispiel:

```
n=5e5, x=randn(n,1);
```

a) `tic, s=0; for i=1:n, s=s+x(i)*x(i); end, toc`

b) `tic, s=sum(x.*x); toc`

Als Rechenzeiten ergeben sich in etwa $1,252$ sec. für a) und $0,041$ sec. für b), da Vektor/Matrixoperationen in `MATLAB` günstiger sind als skalare Operationen. Für große Vektoren und Matrizen macht sich dies deutlich bemerkbar, bei kleineren Dimensionen spielt es keine große Rolle.

Beispiel: Gauß-Elimination

a)
```
for i=k+1:n
  for j=k+1:n
    A(i,j) = A(i,j) - A(i,k)*A(k,j)/A(k,k);
  end
end
```

b)
```
i=k+1:n; j=k+1:n;
A(i,j) = A(i,j) - A(i,k)*A(k,j)/A(k,k);
```

In diesem Fall ergeben sich ungefähr gleiche Laufzeiten für Dimension $n = 50$, für $n = 500$ erhält man mit einer Zufallsmatrix ca. $1,533$ sec. im Fall a) und 0.05 sec. im Fall b). Weitere Funktionen, die auf Matrizen basieren und daher sehr schnell sind, sind `max`, `cumsum`, `prod` etc.

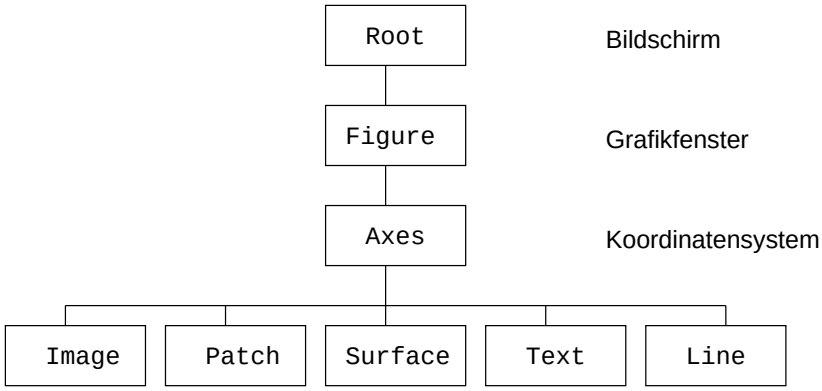
Ebenfalls zeitaufwendig ist die Speicherzuweisung. So ist das Kommando

```
x(1:2)=1;
for i=3:n, x(i)=0.25*x(i-1)^2-x(i-2); end
```

ungünstig, da in jedem Schritt neuer Speicher angefordert werden muss. Besser ist dagegen eine Vorabreservierung mit `x=ones(n,1)`.

2.2 Handle Graphics

Neben den high-level Grafikbefehlen wie `plot` oder `surf` bietet `MATLAB` auch low-level Befehle zur Manipulation elementarer Grafikobjekte, die unter dem Begriff *Handle Graphics* zusammengefaßt werden. Jede grafische Darstellung wird in `MATLAB` über eine hierarchische Struktur aufgebaut, in der einzelne Objekte durch Zeigervariable ('handles') identifiziert werden:



Elementare Grafikobjekte

Zu jedem Objekt existiert damit ein handle, über den es manipuliert werden kann. Generell wird ein handle über die Zuweisung

```
<identifier> = <Grafik-Befehl>
```

mit einer Zahl belegt, die auf das entsprechende Objekt verweist. Bsp.:

```
x = 0:2*pi/40:2*pi;
ha = plot(x,sin(x));          definiert handle ha
set(ha,'LineStyle','--')      ändert Linienstil
```

Die wichtigsten Befehle zum Einsatz von handles:

`delete(h)`	löscht Objekt mit handle `h`
`set(h,<property>,<value>)`	ändert Eigenschaft `<property>`
`set(h)`	zeigt alle Eigenschaften
`get(h)`	zeigt aktuelle Belegung der Eigenschaften
`gcf`	handle auf das aktuelle Grafikfenster
`gca`	handle auf das aktuelle Koordinatensystem
`drawnow`	aktualisiere Grafikfenster (für M-files)

Daneben gibt es noch primitive Grafikobjekte, mit denen sich eigene Darstellungen aufbauen lassen (Optionen siehe Handbuch oder `help`):

`axes`	Vorgabe des Koordinatensystems
`line`	Streckenzug
`text`	Text in der Grafik
`surface`	Oberfläche
`patch`	farbiges Polygon
`image`	Pixel-Bild

Zum Ändern/Editieren eines Plots muss man aber nicht unbedingt die elementaren Befehle von *Handle Graphics* verwenden. Die Alternative dazu: Über das aktuelle Grafikfenster kann durch die Schaltfläche `File` die Option `Properties` ausgewählt werden. welche einen dialogbasierten Editor aufruft.

2.3 Debugging

Neben den Befehlen

`keyboard`	Unterbrechung des Laufs eines M-Files
`return`	Fortführung des Laufs eines M-Files

zum Unterbrechen und Fortführen eines M-Files stellt `MATLAB` einen eigenen Debugger zur Verfügung, der über die Kopfleiste des Editorfensters oder über Kommandos gesteuert werden kann. Dazu werden Breakpoints durch Mausklick rechts neben die Zeilennummer festgelegt und das Programm Schritt für Schritt durchgeführt. An den Breakpoints ist es dann möglich, Variableneinträge zu überprüfen und abzuändern, indem man den Cursor über den Variablennamen bewegt oder den lokalen Workspace verwendet. Hinabsteigen und Hinaufsteifen in Unterprogramme wird durch die Befehle `Step In`, `Step Out` durchgeführt.

Die alternativen Kommandos lauten z.B.

`dbstep`	Führt einen Schritt des M-files aus
`dbstop`	Verlasse Debugging-Mode
`dbcont`	Fortsetzung bis zum nächsten Breakpoint
`dbclear`	Lösche Breakpoints

.

2.4 Toolboxen

Für besondere Anwendungen bietet `MATLAB` Toolboxen an, die eine Sammlung von M-Files zu bestimmten Themengebieten beinhalten. Einen Überblick gibt die folgende Tabelle.

Name	Beschreibung
PDE-Toolbox	Graphische Oberfläche zur Lösung von partiellen Differentialgleichungen
Neural Network Toolbox	Tools für Entwurf, Implementierung, Visualisierung und Simulation neuronaler Netze
Optimization Toolbox	Numerische Optimierung (Lineare Programmierung, beschränkte oder unbeschränkte nichtlineare Probleme, nichtlineare Least Squares).
Signal Processing Toolbox	Signalverarbeitung, Filtertechniken, FFT und verwandte Transformationen
Wavelet Toolbox	Tools zur Analyse, Synthese, Rauschunterdrückung und Komprimierung von Signalen und Bildern
Symbolic Math Toolbox	Einbindung von `Maple`-Befehlen in `MATLAB`, symbolisches Rechnen
Statistics Toolbox	Analyse von Wahrscheinlichkeitsverteilungen, Beschreibung von Daten, Testen von Hypothesen
Financial Toolbox	Umgang mit Finanzdaten, Bewertung von Finanzprodukten

2.5 Literatur, FTP-Server

Handbücher zu Matlab sind in der Bibliothek aufgestellt; empfohlen sei der

MATLAB Guide, Desmond J. Higham and Nicholas J. Higham, SIAM, 2000

Weitere Literatur (über 300 Bücher gibt es bereits rund um `MATLAB`):

http://www.mathworks.com/support/books/

Mit `MATLAB` entwickelte public domain software ist über `ftp` und `WWW` abrufbar:

ftp://ftp.mathworks.com