EINFÜHRUNG IN MATLAB

S. BARTELS, 11.12.2013

I.A. **Aufbau.** MATLAB steht für *Matrix Laboratory* und ist ein kommerzielles Programmpaket, welches Implementierungen einer Vielzahl numerischer Verfahren bereitstellt und es erlaubt, eigene Programme zu erstellen. Es ist eine Interpreter-Sprache, das heißt Programme sind Folgen von Kommandos, die ohne Kompilierung abgearbeitet werden. Die Benutzeroberfläche besteht im Wesentlichen aus dem *Command Window*, in dem Befehle eingegeben, und einem Editor, in dem Programme erstellt werden können. Diese werden im Format `prog.m` abgespeichert, und können dann in einer Kommandozeile oder von anderen Programmen aus über den Befehl `prog` gestartet werden. Ein Kommando wird mit einem Semikolon abgeschlossen. Geschieht dies nicht, so wird das Resultat der Operation angezeigt. Variablen werden standardmäßig als Typ *double* definiert, sie können jedoch problemlos wie Variablen vom Typ *integer* verwendet werden beispielsweise bei der Indizierung von Arrays. In der Regel werden Variablen von MATLAB als Matrizen behandelt.

I.B. **Listen und Arrays.** Zentrale Objekte in MATLAB sind Matrizen beziehungsweise Arrays und Listen. Diese werden mit Hilfe eckiger Klammern definiert. Einträge einer Zeile werden durch Kommata und verschiedene Zeilen durch Semikolons getrennt. Auf die Einträge eines Arrays wird beginnend mit dem Index 1 zugegriffen. Über Indexlisten können Teilmatrizen wie $A_{IJ} = (a_{ij})_{i\in I, j\in J}$ extrahiert werden; statt Indexlisten können dabei auch boolesche Listen verwendet werden. Tabelle 1 zeigt einige wichtige Operationen.

`[a,b,...;x,y,...]`	Definition eines Arrays
`[a,b,...],[x;y;...]`	Definition eines Zeilen- oder Spaltenvektors
`A(i,j)`, `I(j)`	Zugriff auf die Einträge eines Arrays
`a:b`, `a:step:b`	Liste von a bis b mit Schrittweite 1 oder *step*
`A(i,:)`, `A(:,j)`	i-te Zeile und j-te Spalte von A
`A(I,J)`	Teilmatrix definiert durch Listen I und J
`ones(n,m)`	Array mit Einträgen 1
`zeros(n,m)`	Array mit Einträgen 0

TABELLE 1. Erstellung von Listen und Arrays.

I.C. **Matrixoperationen.** Die grundlegenden Matrixoperationen sind in MATLAB definiert und lassen sich in kanonischer Weise verwenden, wobei die Wohlgestelltheit der Operation sichergestellt werden sollte. Matrixfaktorisierungen und Approximationen von Eigenvektoren und -werten stehen ebenfalls zur Verfügung. Einige Standardroutinen sind in Tabelle 2 aufgeführt.

`A'`	transponierte Matrix
`A+B`, `A-B`, `A*B`	Addition, Subtraktion und Produkt von Matrizen
`inv(A)`, `det(A)`	Inverse und Determinante einer Matrix
`x = A\b`	Lösung des linearen Gleichungssystems $Ax = b$
`eye(n)`	Einheitsmatrix der Dimension n
`A.*B`, `A./B`	komponentenweise Multiplikation und Division
`diag(A)`	Extraktion der Diagonalelemente
`[L,U] = lu(A)`	LU-Faktorisierung einer Matrix
`L = chol(A)`	Cholesky-Faktorisierung einer Matrix
`[Q,R] = qr(A)`	QR-Faktorisierung einer Matrix
`[V,D] = eig(A)`	Approximation von Eigenvektoren und -werten

TABELLE 2. Elementare Matrixoperationen.

I.D. **Manipulation von Arrays.** Verschiedene mengentheoretische Operationen und Umsortierungen von Arrays sind in Routinen verfügbar. Diese erlauben meist weitere Argumente und Ausgabewerte, mit denen die Ausführung präzisiert werden kann wie beispielsweise die Bildung des zeilen- oder spaltenweisen Maximums. Tabelle 3 zeigt einige nützliche Befehle.

`A(:)`	Umordnung eines Arrays in einen Spaltenvektor
`reshape(A,m,n)`	Umordnung eines Arrays als $m \times n$ Array
`repmat(A,m,n)`	wiederholte Anordnung eines Arrays
`unique(A)`	Extraktion der Elemente eines Arrays
`setdiff(A,B)`	Komplement von A und B
`sort(A)`	Sortierung der Einträge eines Arrays
`sum(A,1)`, `sum(A,2)`	spalten- und zeilenweise Summenbildung
`max(A)`, `min(A)`	spaltenweise Extremwerte eines Arrays
`size(A)`, `length(I)`	Dimensionen eines Arrays und Länge einer Liste

TABELLE 3. Manipulation von Arrays.

I.E. **Elementare Funktionen.** Numerische Realisierungen einiger Funktionen sind unter ihren jeweiligen Namen verfügbar. Sie können auf Arrays angewendet werden, was in der Regel die komponentenweise Ausführung realisiert. Bei Ausnahmen wie `A^n` wird die kompontenweise Ausführung durch `A.^n` erzeugt. Eine kurze Übersicht findet sich in Tabelle 4.

`sqrt(x)`, `x^y`	Quadratwurzel und Potenzen
`exp(x)`, `ln(x)`	Exponentialfunktion und Logarithmus
`sin(x)`, `cos(x)`, `pi`	trigonometrische Funktionen und Konstante π
`norm(x,p)`	p-Norm eines Vektors

TABELLE 4. Elementare analytische Funktionen.

I.F. **Schleifen und Kontrollanweisungen.** Schleifen lassen sich über Listen oder Kontrollanweisungen in naheliegender Weise realisieren. Der Vergleich von Variablen kann auf Arrays angewendet werden. Tabelle 5 zeigt einige wichtige Kommandos.

`a==b`, `a~=b`	logischer Test auf Gleich- oder Ungleichheit
`a<b`, `a<=b`	logischer Vergleich zweier Zahlen
`E&&F`, `E||F`	logisches *und* beziehungsweise *oder*
`while E ... end`	*while*-Schleife mit booleschem Ausdruck E
`for i = I ... end`	*for*-Schleife über Einträge der Liste I
`if E ... end`	Fallunterscheidung
`tic ... toc`	Messen der CPU-Zeit

TABELLE 5. Logische Operationen und Schleifen.

I.G. **Text- und Grafikausgabe.** Wird ein Programm über eine Kommandozeile gestartet, so können Zwischenergebnisse im Kommandofenster ausgegeben werden. Funktionen oder andere Objekte können in Grafikfenstern sogenannten *figures* dargestellt werden. Eine Auswahl entsprechender MATLAB-Kommandos findet sich in Tabelle 6.

I.H. **Erstellung neuer Funktionen.** Neue Funktionen mit mehreren Ein- und Ausgabewerten lassen sich folgendermaßen definieren:

```
function [y1,y2,...] = new_function(x1,x2,...)
...
end
```

Dabei ist das abschließende `end` optional. Funktionen sollten als Datei mit dem Namen der Funktion also beispielsweise `new_function.m` abgespeichert werden. Eine Datei kann mehrere Funktionsdefinitionen enthalten, jedoch kann nur die erste von außen über den Dateinamen aufgerufen werden. Dabei muss man sich im Verzeichnis der Datei befinden oder der Pfad muss als Suchpfad eingerichtet worden sein.

I.I. **Verschiedene Befehle.** Neben einigen Unix-Befehlen wie `cd` und `ls` sind verschiedene Befehle zur Verwaltung der verwendeten Dateien und Verzeichnisse sowie Variablen verfügbar, die in Tabelle 7 dargestellt sind.

`disp(A)`	Anzeigen der Variablen A
`plot(X,Y,'-*')`	Polygonzug durch Punkte $(X(k), Y(k))$ in $\mathbb{R}^2$
`hold on, hold off`	Darstellung mehrerer Objekte in einer Grafik
`mesh(X,Y,Z)`	Darstellung eines zweidimensionalen Graphen
`meshgrid`	Erzeugung eines Gitters
`axis([x1,x2,...])`	Begrenzung des dargestellten Bereichs
`xlabel`, `ylabel`	Beschriftung der Achsen
`legend`	Einfügen einer Legende
`figure(k)`	Wahl eines Grafikfensters
`subplot(n,m,j)`	Darstellung mehrerer Plots in einem Fenster
`quiver`, `quiver3`	Visualisierung von Vektorfeldern
`trisurf`	Graph einer Funktion auf einem Dreiecksgitter
`tetramesh`	Darstellung einer Zerlegung in Tetraeder

TABELLE 6. Darstellung von Objekten.

`whos`, `clear`	Anzeigen und Löschen aller Variablen
`clc`, `clf`	Löschen des Kommando- oder Grafikfensters
`addpath`	Hinzufügen eines Suchpfads für Funktionen
`save`, `load`	Laden und Speichern von Variablen
`Ctrl-C`	Abbruch eines Programms

TABELLE 7. Verschiedene Befehle.

I.J. **Dünnbesetzte Matrizen.** Bei Matrizen mit vielen verschwindenden Einträgen lässt sich der Aufwand der Lösung zugehöriger linearer Gleichungssysteme reduzieren, sofern die Matrizen über den Matrixtyp *sparse* definiert werden. Für Indexlisten $I \subset \{1, 2, \dots, m\}$ und $J \subset \{1, 2, \dots, n\}$ sowie einen Vektor X gleicher Länge wird eine Matrix $A \in \mathbb{R}^{m\times n}$ definiert durch

$$a_{ij} = \sum_{k\,:\,I(k)=i, J(k)=j} X(k),$$

das heißt bei mehrfach auftretenden Indexpaaren werden die zugehörigen Einträge summiert. Der Zugriff auf einzelne Einträge einer dünnbesetzten Matrix ist im Allgmeinen ineffizient.

`sparse(I,J,X,m,n)`	Zusammensetzung einer dünnbesetzten Matrix
`speye(n)`	Einheitsmatrix als dünnbesetzte Matrix

TABELLE 8. Erzeugung dünnbesetzter Matrizen.

I.K. **Beispiele.** In Abbildung 1 ist die Eingabe verschiedener Befehle im Kommandofenster von MATLAB dargestellt. Die Berechnung der Determinante einer Matrix nach dem Laplaceschen Entwicklungssatz führt auf eine Rekursion, deren MATLAB-Realisierung in Abbildung 2 gezeigt ist. Eine Implementation des Bisektionsverfahrens und dessen Anwendung auf eine Funktion $f(x)$ ist ebenfalls in Abbildung 2 gezeigt.

```
>> A = [2,1;1,2]; b = [1;1];
>> x = A\b

x =

    0.3333
    0.3333

>> x'

ans =

    0.3333    0.3333

>>
```

```
>> x = [pi/2,0,1];
>> sin(x)

ans =

    1.0000         0    0.8415

>> sqrt(-1)

ans =

   0.0000 + 1.0000i

>>
>>
```

ABBILDUNG 1. Ausführung von Befehlen im Kommandofenster.

```
function val = laplace(A)
n = size(A,1);
val = 0;
if n == 1
    val = A(1,1);
else
    for j = 1:n
        I = 2:n;
        J = [1:j-1,j+1:n];
        val = val+(-1)^(1+j)...
            *A(1,j)...
            *laplace(A(I,J));
    end
end
```

```
function x = bisect(a,b)
x = a; z = b;
tol = 1e-4;
while z-x > tol
    c = (x+z)/2;
    if f(x)*f(c)<0
        z = c;
    else
        x = c;
    end
end

function y = f(x)
y = x^3+cos((pi/2)*x);
```

ABBILDUNG 2. Berechnung der Determinante nach Laplace (links) und Realisierung des Bisektionsverfahrens (rechts).

Die grafische Darstellung verschiedener Funktionen in einem Grafikfenster wird durch das in Abbildung 3 gezeigte Programm `several_plots.m` illustriert. Die daneben gezeigte Funktion `plot_bubble.m` wertet eine auf $\mathbb{R}^2$ definierte Funktion $f(x)$ aus und stellt sie grafisch dar. Die durch die Funktionen erzeugten Grafiken sind in Abbildung 4 gezeigt.

```
function several_plots
dx = .1;
X = 0:dx:pi;
Y1 = sin(X); plot(X,Y1,'-r');
hold on;
Y2 = cos(X); plot(X,Y2,':k');
hold off;
legend('sin','cos');
disp('press key'); pause; clf
Z1 = exp(X); plot(X,Z1,'-+');
hold on;
Z2 = log(X); plot(X,Z2,'-*');
hold off;
legend('exp','log');
```

```
function plot_bubble
dx = .1;
dy = .1;
[X,Y] = ...
    meshgrid(-2:dx:2,-2:dy:2);
W = f([X(:),Y(:)]);
Z = reshape(W,size(X));
mesh(X,Y,Z);

function y = f(x)
y = zeros(size(x,1),1);
r = sum(x.^2,2).^(1/2);
I = r<1;
y(I) = exp(-1./(1-r(I).^2));
```

ABBILDUNG 3. Darstellung eindimensionaler Funktionen (links) und einer auf $\mathbb{R}^2$ definierten Funktion (rechts).

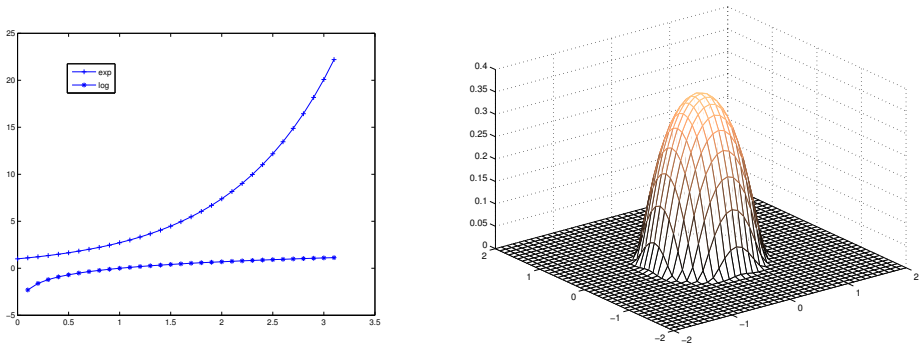

ABBILDUNG 4. Grafische Ausgaben der Funktionen `several_plots.m` (links) und `plot_bubble.m` (rechts).

I.L. **Freie Alternativen.** OCTAVE und SCILAB sind frei verfügbare Programmpakete, die mit MATLAB weitestgehend kompatibel sind. SCILAB ist sehr einfach zu installieren, jedoch sind die grafischen Möglichkeiten eingeschränkt und aufwendige Programme sind in der Regel recht langsam. Einige in der Syntax von MATLAB abweichende Befehle sind

`eye(n,n), sparse([I,J],X), function ... endfunction`

und Funktionen müssen mittels `execute` eingebunden werden. OCTAVE ist nahezu vollständig kompatibel zu MATLAB. Die Installation ist jedoch etwas aufwendiger.