Rheinisch Westfälische Technische Hochschule

Institut für Geometrie und Praktische Mathematik

Numerisches Rechnen — WS 2015 / 2016

Prof. Dr. Martin Grepl — Dipl.-Math. Jens Berger — M.Sc. Robert O'Connor

0. Übung

Diese Übung ist lediglich als Vorbereitung gedacht, sie muss noch nicht abgegeben werden. Sie wird am 27.10. in der Großübung besprochen.

Matlab ist eine kommerzielle Software für die Lösung mathematischer Probleme. Sie bietet viele Möglichkeiten für die Arbeit mit Matrizen und für grafische Darstellungen.

Vorteile:

- einfach zu bedienen
- Visualisierung
- umfangreiche Funktionsbibliotheken

Nachteile:

- Lizenzkosten
- kann langsam sein

In diesem Dokument werden die Matlab Eingaben grün und die Ausgaben blau dargestellt.

1 Erste Schritte

Hier sind einige oft benutzte Symbole:

Symbol	Bedeutung
>>	die Eingabeaufforderung
%	Kommentar
;	schließt einen Befehl, dessen Ergebnis nicht angezeigt werden soll
=	der Zuweisungsoperator

Um der Variablen A den Wert 5 zuzuweisen, schreibt man $A = 5$ nach dem Eingabeprompt.

```
>> A=5
A =
     5
```

Um die Ergebnisausgabe zu unterdrücken, benutzt man ein Semikolon.

```
>> A=5;
```

1.1 Arithmetische Operationen

Folgende Tabelle enthält einige häufig benutzte arithmetische Operationen:

Operator	Definition
+	Addition
−	Subtraktion
*	Multiplikation
/	Division
^	Potenzierung

```
>> A=(5 + 13) * 2 - 1
A =
     35
```

1.2 Eingebaute Funktionen

Funktion	Wirkung
abs	Absolutbetrag
sqrt	Quadratwurzel
exp	Exponentialfunktion
log	Natürlicher Logarithmus
log10	Zehnerlogarithmus
sin	Sinus
cos	Kosinus
tan	Tangens

```
>> abs(-123)
ans =
     123
```

```
>> cos(pi)
ans =
    -1
```

Um weitere Infos zu erhalten, benutzt man **help** ⟨**ANWEISUNGSNAME**⟩

```
>> help abs
 ABS    Absolute value.
ABS(X) is the absolute value of the elements of X. When
X is complex, ABS(X) is the complex modulus (magnitude) of
the elements of X.
```

See also SIGN, ANGLE, UNWRAP, HYPOT.

Overloaded methods:
codistributed/abs
gpuArray/abs
iddata/abs

Reference page in Help browser
doc abs

Es ist oft hilfreich, **help** mit einem bekannten Befehl zu benutzen, um ähnliche Befehle zu finden. In dieser Einführung werden viele Funktionen nur erwähnt. Mit **help** bekommen Sie mehr Informationen dazu.

1.3 Funktionen 'Inline' definieren

Mit Hilfe des At-Zeichens @ kann man Funktionen einfach 'inline' definieren.

```
>> f=@(x,y) x+y^2;

>> f(5,3)
ans =
     14
```

2 Vektoren und Matrizen

Der Grundstein Matlabs sind Operationen mit Vektoren und Matrizen. Vektoren werden dabei als eindimensionale Matrizen betrachtet. Matrizen werden zeilenweise eingegeben mit Kommas zwischen den Elementen und Semikolons zwischen den Zeilen.

```
>> M=[1,2;4,5]
M =
     1   2
     4   5
```

Mit der gleichen Schreibweise kann man auch Matrizen aus anderen Matrizen zusammensetzen.

```
>> B=[0*M,M;2*M,3*M]
B =
     0   0   1   2
     0   0   4   5
     2   4   3   6
     8  10  12  15
```

Vektoren können auch als Sequenzen definiert werden:

```
>> v=0:.3:1.5
v =
    0    .3    .6    .9   1.2   1.5
```

Die Eingabe 0 : .3 : 1.5 ergibt also einen Vektor mit 6 Elementen. Das erste Element ist 0. Die Schrittweite ist .3 und das letzte Element ist so groß wie möglich, ohne dabei 1.5 zu übersteigen. Die Schrittweite wird auf 1 gesetzt, wenn sie nicht explizit erwähnt wird:

```
>> v=0:5
v =
    0    1    2    3    4    5
```

Für folgende Beispiele seien A und B so definiert:

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

```
>> B=[2,0;0,1];
```

Wenn man mit Matrizen arbeitet, gibt es zwei Arten von Operationen: Matrixoperationen und elementweise Operationen. Die Operatoren $*$, /, \ und ^ sind Matrixoperatoren. Hierbei ergibt A/B (bzw. $A\backslash B$) die gleiche Ausgabe wie $A * B\hat{}(-1)$ (bzw. $A\hat{}(-1) * B$). Die Operatoren $.*$, ./ und .^ sind elementweise Operatoren. Die Operatoren $+$ und $-$ sind gleichzeitig Matrixoperatoren und elementweise Operatoren.

```
>> A*[2;3]
ans =
     8
    18
```

```
>> A*B
ans =
     2    2
     6    4
```

```
>> A.*B
ans =
     2    0
     0    4
```

Man kann auch bestimmte Matrixelemente mit Hilfe von Indizes manipulieren. Dabei können auch Sequenzen benutzt werden. Will man z.B. eine komplette Spalte oder Zeile, so benutzt man einen Doppelpunkt. Der größte mögliche Index wird durch **end** repräsentiert. Zuweisungen können auch für Matrixeinträge durchgeführt werden.

```
>> A(1,2)
```

```
ans =
     2
```

```
>> B(2,:)
ans =
     0     1
```

```
>> M=[1,2,3,4;5,6,7,8];
```

```
>> M(:,3:end)
ans =
     3     4
     7     8
```

```
>> M(1,3:4)=[20,22]
M =
     1     2    20    22
     5     6     7     8
```

Mittels des Hochkommas bekommt man die transponierte Matrix.

```
>> A'
ans =
     1     3
     2     4
```

Um das Gleichungssystem $Ax = f$ zu lösen, kann man $\mathbf{A\backslash f}$ eingeben. Die folgenden Funktionen, die beim Arbeiten mit Matrizen nützlich sein können, sind bereits in Matlab implementiert:

Funktion	Wirkung
ones	Matrix mit nur Einsen
zeros	Nullmatrix
eye	Einheitmatrix
diag	Diagonalmatrix
rand	Zufallsmatrix mit gleichverteilten Einträgen
randn	Zufallsmatrix mit normalverteilten Einträgen
det	Determinante
eig	Eigenwerte und Eigenvektoren
inv	Inverse
length	Länger eines Vektors
size	Dimensionen einer Matrix

```
>> M=eye(3)
M =
```

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


>> det(M)
ans =
     1
```

Siehe auch **help** für mehr Infos.

3 Das M-File

Es ist oft nützlich, bestimmte Befehle in einer Datei zu speichern. Über **File New Script** wird der Texteditor geöffnet. Damit könnte man folgende Datei als **mathematik.m** speichern:

Datei 1: mathematik.m

```
M=[1,2;3,4];      % keine Ausgabe wegen ';'
v=[1;1];

M*v

45^2
```

Sofern **mathematik.m** im aktuellen Arbeitsverzeichnis gespeichert ist, kann man die Datei wie folgt ausführen:

```
>> mathematik
ans =
     3
     7

ans =
   2025
```

Die Variablen M und v behalten auch danach die in **mathematik.m** bestimmten Werte.

3.1 Funktionen (noch einmal)

Funktionen können auch in M-Files definiert werden. Dazu benutzt man den Befehl **function** gefolgt von der Signatur der Funktion. Die Datei **addieren.m** ist ein Beispiel dafür.

Datei 2: addieren.m

```
function [summe]=addieren(x,y)
% addieren(x,y) addiert die Zahlen x und y

  summe=x+y;    % ';' wird benutzt um eine Ausgabe zu verhindern
```

Wenn sich **addieren.m** im aktuellen Arbeitsverzeichnis befindet, kann die Funktion wie folgt aufgerufen werden:

```
>> addieren(7,12)
ans =
     19
```

4 Bedingte Anweisung, Verzweigung und Schleifen

In Matlab werden die booleschen Werte wahr und falsch als 0 und 1 dargestellt.

Operator	Bedeutung
==	gleich
<	kleiner als
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
~=	ungleich
&	und
|	oder
~	nicht

```
>> false & true
ans =
     0
```

```
>> 1 == 0
ans =
     0
```

```
>> ~(1 == 0)
ans =
     1
```

4.1 Anweisung und Verzweigung

Manchmal sollen bestimmte Anweisungen nur unter bestimmten Bedingungen durchgeführt werden. In solchen Fällen benutzt man die '**if**' Anweisung. Ein bedingter Anweisungsblock kann folgende Form haben:

if ⟨*Bedingung*⟩

⟨*bedingterAnweisungsblock*⟩

end

Der bedingte Anweisungsblock wird nur durchgeführt, wenn die Bedingung wahr ist. Die '**else**' Anweisung ist nützlich, wenn ein zweiter bedingter Anweisungsblock durchgeführt werden soll, falls die Bedingung falsch ist. Dann hat man

if ⟨*Bedingung*⟩

⟨*bedingterAnweisungsblock*1⟩

else

⟨*bedingterAnweisungsblock*2⟩

end

Datei 3: anweisungen.m

```
function anweisungen(n)

  if mod(n,2)==0
    disp([num2str(n),' ist eine gerade Zahl'])
  else
    disp([num2str(n),' ist eine ungerade Zahl'])
  end

  if n==3
    disp('n ist 3')
  end
```

Es existiert auch eine '**elseif**'-Anweisung.

```
>> anweisungen(18)
18 ist eine gerade Zahl

>> anweisungen(3)
3 ist eine ungerade Zahl
n ist 3
```

4.2 Schleife

Die for- und while-Schleifen bieten einfache Möglichkeiten, Berechnungen wiederholt durchzuführen. Leider sind sie oft nicht so effizient wie geeignete Vektoroperationen. In der folgenden Datei wird eine for-Schleife benutzt, um die Fakultät zu berechnen.

Datei 4: forfakultaet.m

```
function z=forfakultaet(n)
% Berechnet die Fakultät der positiven Ganzzahl n mittels einer
% for-Schleife
```

```
z=1;
for i=1:n
  z=z*i;
end
```

Man kann dass gleiche mit einer while-Schleife schreiben.

Datei 5: whilefakultaet.m

```
function z=whilefakultaet(n)
% Berechnet die Fakultaet der positiven Ganzzahl n mittels einer
% while-Schleife
  z=1;
  while n > 1
    z=z*n;
    n=n-1;
  end
```

5 Plotten

Die einfachste Möglichkeit, eine Funktion zu plotten, bietet die **plot**-Funktion.

Datei 6: plot1.m

```
x=0:100;
y=x.^2;
plot(x,y)   % plotten
```

Damit ist es auch möglich, mehrere Funktionen zu plotten. Die folgende Datei zeigt auch, wie man einen Plot schöner gestalten kann.

Datei 7: plot2.m

```
t=0:.01:1;

f1=sin(2*pi*t);
f2=cos(2*pi*t);

plot(t,f1,'b',t,f2,'r*')    % plotten

% Titel definieren und Variablen benennen (Handles in h speichern)
h=title('Sinus und Kosinus');
h(2)=xlabel('Zeitschritt');
h(3)=ylabel('Wert');

% Zeichenerklaerung
h(4)=legend('Sinus','Kosinus')

% Schriftart modifizieren
```

```
set(h(1),'FontSize',16)
set(h(2:3),'FontSize',13)
set(h(4),'FontSize',13)
set(h,'FontWeight','bold')
```

Zusammen mit **set** kann man auch **get** benutzen. Es gibt noch weitere Möglichkeiten, mehrere Funktionen gleichzeitig zu plotten. Mit **hold on** und **hold off** können neue Funktionen zu einem existierenden Plot hinzugefügt werden. Mit dem Befehl **figure** kann ein neues Fenster für einen weitere Plot geöffnet werden. Will man verschiedene Plots im gleichen Fenster haben, so ist **subplot** nützlich.

Aufgabe 1: (Matrizen)

Seien:

$$A = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix},$$

a) Geben Sie die Matrizen in Matlab ein. Benutzen Sie die Befehle **diag** und **eye**, um B und C einzugeben.

b) Berechnen Sie $2A + 3B^2 + C^2$.

c) Lösen Sie $Ax = v$ für $v = [1, 2, 3]^T$.

d) Berechnen Sie A^{-1}.

e) Erzeugen Sie die Matrix $D = \begin{bmatrix} A & B \\ C & A \end{bmatrix}$ aus den Submatrizen A, B und C.

Aufgabe 2: (Fibonacci)

Die Fibonacci-Folge f_1, f_2, f_3, ... ist so definiert, dass

$$f_n = f_{n-1} + f_{n-2}, \qquad \text{für } n > 2,$$

mit $f_1 = f_2 = 1$. Schreiben Sie eine Funktion **f=fibonacci(n)**, die für jedes eingegebene $n \in \mathbb{N}$ den Wert von f_n zurückgibt.

Aufgabe 3: (Plotten)

Plotten Sie die Funktion $log_{10}(x)$ für $x \in [.1, 10]$. Benutzen Sie auch **title**, **xlabel** und **ylabel**.

Aufgabe 4: (Erzeugung von Matrizen)

Betrachten Sie die folgende Matrix:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 \\ 4 & 3 & 2 & 1 & 0 & 1 \\ 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

a) Erzeugen Sie die Matrix mittels geschachtelter for-Schleifen.

b) Erzeugen Sie die Matrix mit Hilfe einer for-Schleife und dem Befehl **diag**.

Aufgabe 5: (Matrixmultiplikation)

Schreiben Sie eine Funktion, die zwei Matrizen mit Hilfe von for-Schleifen multipliziert.

Aufgabe 6: (Mehr Plotten)

Erstellen Sie mit Hilfe des Befehls **figure** ein neues Plotfenster mit der Nummer drei. In diesem Plotfenster sollen Sie nun zwei Plots erzeugen. Links plotten Sie x^2 und rechts x^3 mit $x \in [-2, 2]$. Benutzen Sie auch **title**, **xlabel** und **ylabel**.