# Einführung in MATLAB zur Veranstaltung "Einführung in die Numerik"

Christian Stohrer

Mathematisches Institut der Universität Basel

FS 2011

# MATLAB Einführung

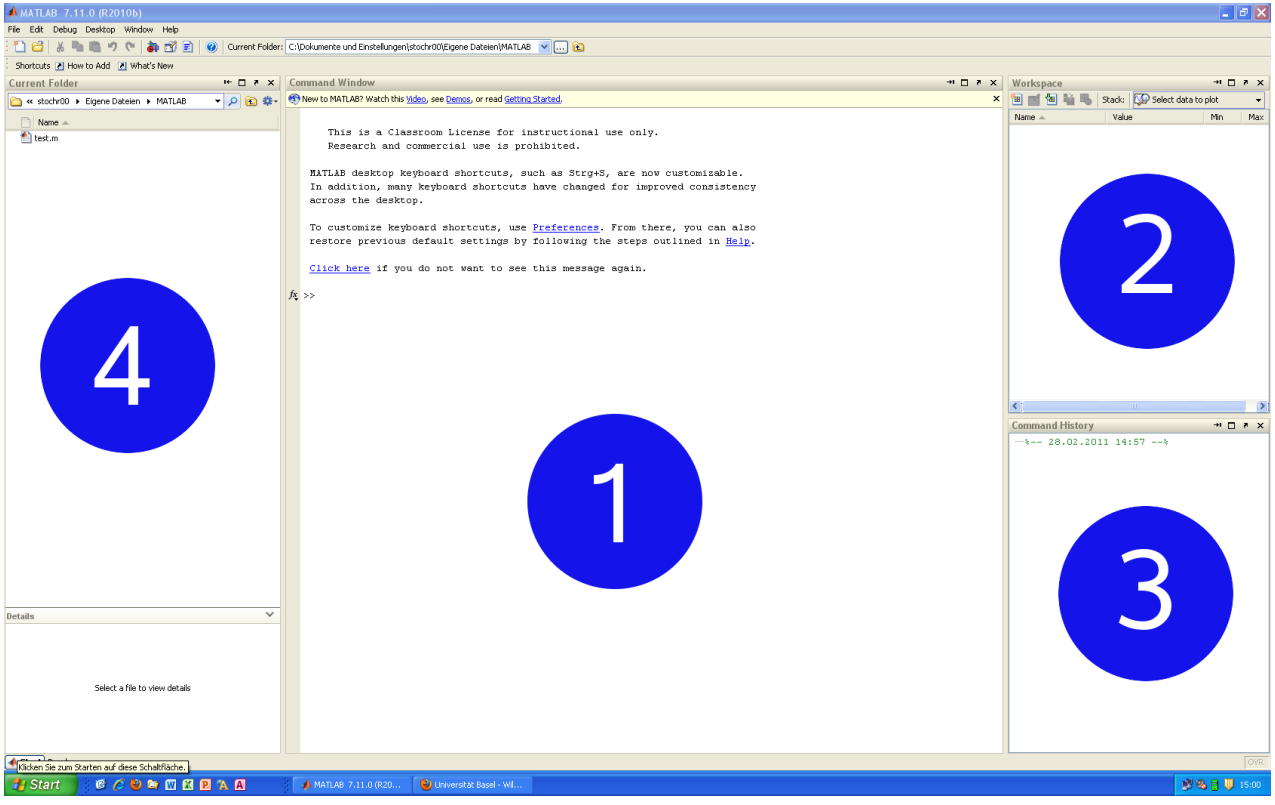
zur Veranstaltung "Einführung in die Numerik"

- Bitte logge dich mit deinem Uni-Acoount ein.
  (Egal ob unter Linux oder Windows)
- Starte MATLAB

  **Linux** Terminal aufrufen und `matlab` eingeben.
  Oder:
  Applications → Unibas applications → MATLAB

  **Windows** Doppelklick auf MATLAB-Icon.
  Oder:
  Start → Programme → MATLAB
- (optional) Standard Desktop von MATLAB auswählen.
  Desktop → Desktop Layout → Default

# MATLAB Desktop




1. Command Window
2. Workspace
3. Command History
4. Current Folder

Q: Ich habe kein MATLAB, wie bekomme ich eines?

A: Unter anderem hast du folgende Möglichkeiten:

- MATLAB ist auf den Uni-Rechnern installiert. (z.B. Computerräume URZ oder des Mathematischen Instituts)
- MATLAB erwerben (via URZ möglich unter `https://asknet.unibas.ch` )
- MATLAB-Clone verwenden: Octave, Scilab, ...

Q: Ich habe noch nie mit MATLAB gearbeitet, was nun?

A: Neben "Learning-by-doing" kannst du

- Praktikum II besuchen.
- Tutorials und Einführungen anschauen. Du findest solche unter Material auf: `http://www.math.unibas.ch/~grote/institut/vorlesungen/11fs/PraktikumII/`
- Die MATLAB Hilfe verwenden. Als Einstieg ist der Abschnitt "Getting Started" geeignet.

Einfache Rechnungen kann man direkt in das `Command Window` eintippen. Folgendes gilt es zu beachten:

- Dezimaldarstellung: richtig: `3.5`; falsch: `3,5`
- Zehner-Potenz-Faktor: für $1.25 \cdot 10^8$ tippe `1.25e8`
- Es gelten die üblichen Rechenregeln.
- Mathematische Standardfunktionen z.B.:
  `sqrt`, `sin`, `cos`, `tan`, `exp`, `log`, `abs`, `round`
- Mathematische Konstanten: `pi`, `i`, `eps`
- Mit einem Semikolon unterdrückt man die Ausgabe des Resultats.
  Das letzte Resultat ist unter `ans` gespeichert.
- `format long` zeigt Resultate auf 15 Stellen genau.
  Default: 5 Stellen, einstellbar mit `format short`.

# Hilfe zur Selbsthilfe

Viele Fragen kann man mit der MATLAB-Hilfe lösen.

- `help cmd` zeigt den Hilfstext des Befehls `cmd` an.
- `doc cmd` öffnet die Dokumentation zum Befehl `cmd`.

Der Umgang mit Variabeln ist intuitiv:
`x = 2;` speichert den Wert 2 in der Variablen `x`. Nun kann man überall wo man 2 eingeben kann, einfach `x` hinschreiben. Eine Auflistung der aktuellen Variablen findet man im Workspace.

**ACHTUNG!**

Als Variabelname sind z.B. auch Funktionsnamen (wie `sin`) oder Namen von Konstanten (wie `pi`) zulässig. Die sollte man vermeiden!

Einige wichtige Befehle für den Umgang mit Variabeln:

- `clear` und `clear all` (Löschen von Variabeln)
- `who` und `whos` (Auflistung der Variabeln)
- `save` und `load` (Speichern und laden)

# Anlegen von Matrizen

| MATLAB-Syntax | Mathematische Bedeutung |
|---|---|
| `v = [1, 2, 3, 4];` | $v := \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$ |
| `w = [1; 2; 3; 4];` | $w := \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$ |
| `A = [1, 2; 3, 4];` | $A := \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ |

Beachte auch die Befehle: `eye`, `zeros`, `ones` und `linspace`.

# Zugriff auf Elemente

Wir betrachten folgende Matrix

$$A = \begin{pmatrix} -7 & -6 & -5 & -4 \\ -3 & -2 & -1 & 0 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

- `A(1,1)`
- `A(2,1)`
- `A(2,4)`
- `A(2,end)`
- `A(end,end)`

- `A(:,2)`
- `A(3,:)`

- `A(1:2,2:4)`

# Matrixoperationen

Matrixoperationen

- Transponieren `'`
- Inverse `inv`
- Determinante `det`
- Grösse Bestimmen `length` und `size`
- Matrix-Matrix-Opertationen `+ - * ^`

Elementweise und skalare Operationen

- `.* ./ .^`

**ACHTUNG!**

Diese beide Operations-Typen muss man gut auseinander halten!

Um die lineare Gleichung $Ax = b$ lösen, tippt man `x = A\b`.

# Ein Beispiel zum plotten

```
>> clear all                                % Loescht Workspace
>> close all                                % Schliesst alle offenen Figures
>> x = -3:0.01:3;                           % Zahlen von -3 bis 3, Schrittweite 0.01
>> y1 = cos(x);                             % Berechnung von y1
>> y2 = sin(x);                             % Berechnung von y2
>> plot(x,y1)                               % Zeichnet y1 gegen x
>> plot(x,y2,'--')                          % Zeichnet y2 gegen x
>> figure                                   % Neue Figure erstellen
>> plot(x,x.^2)                             % Zeichnet x*x gegen x
>> hold on                                  % Verhindert das Ueberschreiben
>> plot(x,x.^3)                             % Zeichnet x.*x.*x gegen x in dieselbe Figure
>> hold off                                 % Laesst das Ueberschreiben wieder zu
>> figure                                   % Neue Figure erstellen
>> plot(x,x.^2,x,x.^3)                      % Zeichnet x.*x gegen x und x.*x.*x gegen x
>> plot(x,x.^2,'-.',x,x.^3,':')             % Wie oben, mit geaenderter Darstellung
>> axis([-2,2,-10,10])                      % Legt den zu zeichnenden Bereich fest
>> xlabel('x')                              % x-Achsen-Beschriftung
>> ylabel('y')                              % y-Achsen-Beschriftung
>> title('Zwei Funktionen in einem Plot')   % Titel setzen
>> legend('y = x^2','y = x^3')              % Legende erzeugen
```

# Anlegen und Ausführen eines Skriptes

Anstatt alles in der `Command Window` einztippen, kann man auch sogenannte Skripts anlegen.Ein neues Skript öffnet man mit `File` → New → Script oder mit einem Klick auf das entsprechende Icon.

Man zum Beispiel alle Befehle des Beispiels zum Plotten in ein Skript einfügen. Man muss nur die >> weglassen. Das ganze speichern wir unter `myPlots.m`.

Um das Skript auszuführen gibt man nun einfach `myPlots` im `Command Window` ein.

# Auf was man bei Skripts achten soll

**ACHTUNG!**

MATLAB führt ein Skript so aus, wie wenn alle Befehle im `Command Window` eingegeben worden wären. Dies bedeutet, dass ein Skript die Variabeln des `Workspace` gebrauchen oder aber auch verändern kann. Um sicher zu sein, dass ein Skript nicht auf alte Variabeln zurückgreift, sollte man zu Beginn jeweils den `Workspace` löschen (mit `clear all`).

# Bedingungen – `if else`

Bisher wurde jede Zeile eines Skripts genau einmal ausgeführt. Dies will und kann man manchmal ändern. Wir stellen uns dazu folgende Aufgabe:

Es soll geprüft werden, ob eine $x$ grösser oder gleich 4 ist. Falls ja, so runden wir die Zahl auf halbe Zahlen, speichern das Ergebnis $y := x$ und schreiben "bestanden, mit Note $y$". Falls nicht, so setzen wir $y = 0$ und schreiben "durchgefallen".

Folgendes Skript löst dieses Problem.

```
clear all;
x = 4.2563;
if x>=4
y = round(2*x)/2;
disp('bestanden, mit Note')
disp(y)
else
y = 0;
disp('durchgefallen')
end
```

# Bedingungen – `if else`

Einige Bemerkungen dazu:

- Ein Bedingung beginnt immer mit `if` und einem anschliessenden Test.
- Für den Test stehen unter anderem Vergleichsoperatoren (wie `>, <, >=, <=, ==, ~=`) und Verknüpfungen (wie `&&, ||`) zur Verfügung.
- Die Zeilen nach dem `if` werden nur ausgeführt, falls der Test positiv ist.
- Man kann ein `else` einfügen. Die Zeilen nach dem else werden nur ausgeführt, falls der Test negativ war.
- Man muss das Ende der Bedingung mit `end` kennzeichnen.

# Wiederholte Ausführung – `for`

Bisher wurde jede Zeile eines Skripts einmal oder keinmal ausgeführt. Eine mehrfache Auswertung ist manchmal gewünscht. Wir stellen uns folgende Aufgabe:

Wir möchten einen Countdown von einer Zahl $n$ auf dem Bildschirm ausgeben.

Folgendes Skript löst dieses Problem.

```
clear all;
n = 10;
for step = n:-1:0
disp(step)
pause(1)
end
```

# Wiederholte Ausführung – `for`

Einige Beremkungen dazu:

- Eine `for`-Schleife beginnt mit dem Wort `for`.
- Direkt danach kommt ein Ausdruck der Form `var = vec`, wobei `var` eine Variable und `vec` ein Zeilenvektor ist.
- Beim ersten Durchlauf gilt `var = vec(1)`, beim zweiten `var = vec(2)`, usw.
- Häufig ist der Zeilenvektor in der Form `1:10` gegeben.

# Eigene Funktionen

Wir haben schon einige Funktionen kennengelernt. Zum Beispiel die Funktion `sin`. Wir können `sin` einen Wert übergeben und MATLAB berechnet dann (ohne dass wir etwas davon mitbekommen) das Resultat.

In MATLAB ist es möglich selbst solche Funktionen zu programmieren. Dies hat einige Gemeinsamkeiten mit dem Programmieren eines Skripts, doch gibt es auch wichtige Unterschiede.

# Vergleich mit Skript

Wir bauen unser erstes Skript zu einer Funktion um.

## Skript

```
clear all;
x = 4.2563;
if x>=4
y = round(2*x)/2;
disp('bestanden, mit Note')
disp(y)
else
y = 0;
disp('durchgefallen')
end
```

## Funktion

```
function [y] = note(x)


if x>=4
y = round(2*x)/2;
disp('bestanden, mit Note')
disp(y)
else
y = 0;
disp('durchgefallen')
end
end
```

# Bemerkungen zu Funktionen

**ACHTUNG!**

- Die erste Zeile welche ausgeführt wird muss folgende Form haben:
  `function [out1, ..., outN] = name(in1, ..., inM)`
  Dabei bezeichnen `outI` die Rückgabeelemente und `inI` die Eingabeelemente.
- Es ist sehr empfehlenswert eine Funktion unter demsalben Namen zu speichern, wie in der ersten Zeile angegeben.
- Im Gegensatz zu einem Skript hat eine Funktion ihren eigenen `Workspace`!

# Beispiel: Arithmetisches und geometrisches Mittel

```
% agMean berechnet das arithmetische und
%     das geometrische Mittel zweier Zahlen

function [a, g] = agMean(x1,x2)

a = (x1 + x2)/2; % Berechnung des arithmetischen Mittels
g = sqrt(x1*x2); % Berechnung des geometrischen Mittels

end
```

# Debuggen eines Programms

Damit man eine Funktionen Zeile für Zeile berechnen lassen kann, gibt es den so genannten `Debug Modus`.

Die Verwendung wird direkt in MATLAB demonstriert.

Beim Verfassen dieser Einführung habe ich mit von folgenden Einführungen inspirieren lassen:

- *MATLAB Tutorial*; M. Grote, M. Mehlin, C. Staub (`http://jones.math.unibas.ch/~grote/institut/vorlesungen/11fs/PraktikumII/Material/matlab/MatlabTutorial.pdf`)
- *Matlab: eine kurze Einführung*; M. Grote, C. Kirsch, I. Sim (`http://jones.math.unibas.ch/~cohen/Teach/Spektral08/einmatlab.pdf`)
- *Einführung in die Programmierung mit MATLAB*; K. Smetana (`http://wwwmath.uni-muenster.de/num/Vorlesungen/MATLAB-Kurs_WS10/Script/matlab-einfuehrung.pdf`)

Hinweis: Die Links wurden am 1. März 2011 zum letzten Mal überprüft.