

# Mathematische Simulationssoftware

---

Dr. David Willems  
31. Januar 2019  
Universität Koblenz-Landau

Notizen

---

---

---

---

---

---

---

---

## Inhaltsverzeichnis

1. Organisatorisches
2. Erste Schritte mit MATLAB
3. MATLAB als intelligenter Taschenrechner
4. Vektoren & Matrizen
5. Daten speichern und laden
6. Abbildungen
7. Skripte & Funktionen

Notizen

---

---

---

---

---

---

---

---

## Organisatorisches

---

Notizen

---

---

---

---

---

---

---

---

## Organisatorisches

### Kontakt

Dr. David Willems

- ▶ Email: davidwillems@uni-koblenz.de
- ▶ Büro: G 329
- ▶ Sprechstunde: Wenn die Bürotür offen ist

### Termine

Vorlesung mit Übung:

- ▶ Dienstag, 18.02. von 09:00 – 16:00 Uhr
- ▶ Mittwoch, 19.02. von 09:00 – 16:00 Uhr
- ▶ Donnerstag, 20.02. von 14:00 – 16:00 Uhr  
    ↪ Klausurtermin!

3

Notizen

---

---

---

---

---

---

---

---

## Organisatorisches

### Materialien

Kursmaterial (u. A. auch diese Präsentation) wird unter  
<http://uni-ko-ld.de/r4>  
zur Verfügung gestellt.

### Weiterführende Literatur

- ▶ Wolfgang Schweizer. *MATLAB kompakt*. Walter de Gruyter GmbH & Co KG, 2016
- ▶ Frank Haüßer und Yury Luchnikov. *Modellierung mit MATLAB: Eine praktische Einführung*. Springer-Verlag, 2010
- ▶ Folkmar Bornemann. *Modellierung mit MATLAB: eine konzise Einführung mit MATLAB*. Springer-Verlag, 2016
- ▶ Svein Linge und Hans Petter Langtangen. *Programming for Computations-MATLAB/Octave*. Springer, 2016

Die letzten drei Bücher sind als E-Book verfügbar!  
↪ Homepage

4

Notizen

---

---

---

---

---

---

---

---

## Erste Schritte mit MATLAB

Notizen

---

---

---

---

---

---

---

---

## Erste Schritte mit MATLAB

### Was ist MATLAB?

- ▶ Ein (bequemes) Tool für numerische Berechnungen und Simulationen
- ▶ Eine Hochsprache
- ▶ Eine interpretierte Sprache
- ▶ Proprietäre Software...

### Alternativen zu MATLAB

- ▶ FREEMAT: <http://freemat.sourceforge.net/>
- ▶ GNU OCTAVE: <https://www.gnu.org/software/octave/>
- ▶ SCILAB: <http://www.scilab.org/>

5

Notizen

---

---

---

---

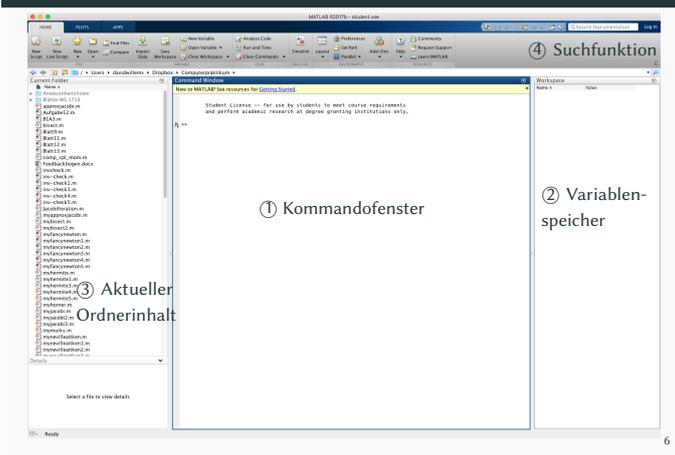
---

---

---

---

## Erste Schritte mit MATLAB



6

Notizen

---

---

---

---

---

---

---

---

## Erste Schritte mit MATLAB

### Erste Kommandos

Hinter der Eingabeaufforderung (>>) können einfache Befehle eingegeben und ausgeführt werden.

```
Command Window
>> 1 + 1
ans =
    2
```

Leerzeichen um Operatoren können entfallen!

7

Notizen

---

---

---

---

---

---

---

---

## Erste Schritte mit MATLAB

### Hilfefunktion und Dokumentation

MATLAB verfügt über eine umfassende Hilfe, eine Dokumentation und viele eingebaute Beispiele. Die Syntax und Dokumentation kann über `help funktionname` im Kommandofenster angezeigt werden.

#### Command Window

```
>> help exit
exit  Exit from MATLAB.
     exit terminates MATLAB after running finish.m, if
     finish.m exists.
     It is the same as QUIT and takes the same
     termination options.
     For more information, see the help for QUIT.

     See also quit.

fx Reference page for exit
```

8

Notizen

---

---

---

---

---

---

---

---

## Erste Schritte mit MATLAB

### Hilfefunktion und Dokumentation

- ▶ Ersetzt man den Befehl `help` durch `doc`, so öffnet sich die Hilfe in einem neuen Fenster.
- ▶ `doc` bzw. `help` setzen voraus, dass man den Namen der Funktion bereits kennt. Ist dies nicht der Fall, kann über `lookfor name` nach Funktionen gesucht werden, in denen `name` vorkommt.
- ▶ Mit dem Befehl `demo` lassen sich (optional) installierte Demonstrationen anzeigen und durcharbeiten.

9

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Variablen

Wie andere Programmiersprachen kann man in MATLAB Variablen anlegen und damit arbeiten.

Command Window

```
>> x = 1;
```

Ohne abschließendes Semikolon wird das Ergebnis zusätzlich im Kommandofenster angezeigt:

Command Window

```
>> x = 1
x =
1
```

11

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Variablen

- ▶ Variablenamen müssen zwingend mit einem Buchstaben beginnen; danach können weitere Buchstaben (keine Umlaute!), Ziffern oder Unterstriche folgen.
- ▶ Groß- und Kleinschreibung werden unterschieden.
- ▶ Variablenzuweisungen können mit Berechnungen verbunden sein.
- ▶ Mit einer weiteren Zuweisung wird der Inhalt einer Variablen ohne Nachfrage **überschrieben**.
- ▶ Mit den Befehlen

`who` bzw. `whos`

können die sich aktuell im Speicher befindlichen Variablen angezeigt werden.

12

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Command Window

```
>> who
Your variables are:
ans x
```

Command Window

```
>> whos
Name      Size      Bytes  Class  Attributes
ans       1x1         8  double
x         1x1         8  double
```

Mit `clear VarName` wird die Variable `VarName` gelöscht. `clear` löscht alle aktuellen Variablen.

13

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Datentypen

MATLAB rechnet i. A. mit Fließkommazahlen in doppelter Genauigkeit (Typ double). Dezimaltrennzeichen ist ein **Punkt**, Zehnerpotenzen können über die Buchstaben **e** oder **E** gekennzeichnet werden.

#### Command Window

```
>> 1
ans =
     1
>> -.123
ans =
-0.1230
>> 1.2e1
ans =
    12
>> 2E-3
ans =
  0.0020
```

15

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Komplexe Zahlen werden durch Imaginärteil **i** oder **j** gekennzeichnet:

#### Command Window

```
>> 1+2i
ans =
 1.0000 + 2.0000i
>> 3j
ans =
 0.0000 + 3.0000i
>> i
ans =
 0.0000 + 1.0000i
```

Daher beim Imaginärteil einer komplexen Zahl auf **+** verzichten!

Notizen

---

---

---

---

---

---

---

---

### Achtung!

Wird **i** zuvor als Variable verwendet, so zeigt der letzte Befehl den Inhalt der Variablen **i** an und nicht die erwartete imaginäre Einheit.

16

## MATLAB als intelligenter Taschenrechner

### AUFGABE 1 | Elementare arithmetische Operationen

Berechnen Sie die folgenden Ausdrücke. Überprüfen Sie ihre Ergebnisse mit MATLAB.

- ▶  $2/2 * 3$
- ▶  $6 - 2/5 + 7^2 - 1$
- ▶  $10/2 \setminus 5 - 3 + 2 * 4$
- ▶  $3^2/4$
- ▶  $3^2^2$
- ▶  $2 + \text{round}(6/9 + 3 * 2)/2 - 3$
- ▶  $2 + \text{floor}(6/9 + 3 * 2)/2 - 3$
- ▶  $2 + \text{ceil}(6/9 + 3 * 2)/2 - 3$

Was ist der Unterschied zwischen den Funktionen **round**, **floor** und **ceil**?

17

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Wir wollen kurz<sup>1</sup> das Thema Rechengenauigkeit in der numerischen Mathematik betrachten:

### Genauigkeit

- ▶ MATLAB kann Zahlen im Bereich von etwa  $10^{-308}$  bis  $10^{308}$  darstellen; die exakten Werte sind in den Konstanten `realmin` bzw. `realmax` hinterlegt.
- ▶ Zwischen einer darstellbaren Zahl und der nächstgrößeren ist eine kleine Differenz, die mit `eps` angezeigt werden kann.
- ▶ Rundungsfehler sind bei numerischen Rechnungen unvermeidlich und treten daher auch bei MATLAB auf.

<sup>1</sup>Für mehr Informationen sei auf „IEEE Standard for Floating-Point Arithmetic“. In: *IEEE Std 754-2008* (Aug. 2008), S. 1–70. doi: 10.1109/IEEESTD.2008.4610935 verwiesen.

18

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

```
Command Window
>> realmax
ans =
    1.7977e+308
>> eps(1)
ans =
    2.2204e-16
>> 10*(1-0.9)
ans =
    -2.2204e-16
```

Insbesondere gelten z. B. das Assoziativ- und das Distributivgesetz nicht!

19

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Unendlich

MATLAB verwendet die Symbole `Inf` bzw. `-Inf` für plus bzw. minus unendlich.

```
Command Window
>> -1/0
ans =
    -Inf
>> 2+realmax
ans =
    Inf
>> (realmax+1)-realmax
ans =
    0
```

Es gelten die üblichen „Rechenregeln“ für das Rechnen mit  $\infty$ .

21

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Not a Number

Ergebnisse, die nicht (sinnvoll) als Zahl darstellbar sind, werden als „Not a Number“ (NaN) bezeichnet.

```
Command Window
>> 0/0
ans =
    NaN

>> Inf/Inf
ans =
    NaN
```

Rechnungen mit NaN werden immer zu NaN ausgewertet.

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Vergleichsoperatoren

MATLAB verfügt über diverse Vergleichsoperatoren, die über „gewöhnliche“ Taschenrechner hinausgehen.

- == gleich
- ~= ungleich
- > größer
- < kleiner
- >= größer oder gleich
- <= kleiner oder gleich

Das Ergebnis eines Vergleichs ist 0 oder 1, aber jeder Wert ungleich 0 wird als wahr interpretiert!

Obige Bedingungen können mit komplexeren Ausdrücken verknüpft werden.

- & und
- | oder
- ~ nicht
- XOR entweder oder

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Das folgende Beispiel prüft, ob der Wert von x zwischen 2 und 6 liegt.

```
Command Window
>> x = 4

x =

     4

>> (x >= 2) & (x <= 6)

ans =

     1
```

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Hinweis

Wir wollen an dieser Stelle einige, häufig verwendete Funktionen betrachten. Für eine vollständige Liste sei auf die Hilfefunktion verwiesen.

### Trigonometrische Funktionen

Die Funktionen `sin`, `cos` und `tan` berechnen Sinus, Kosinus und Tangens des Arguments (im Bogenmaß).

Mit den Funktionen `sind`, `cosd` und `tand` kann das Argument in Grad anstatt im Bogenmaß angegeben werden.

Die zugehörigen Umkehrfunktionen lauten `asin`, `acos` und `atan` bzw. `asind`, `acosd` und `atand`.

### Hyperbolische Funktionen

Die hyperbolischen Funktionen werden über `sinh`, `cosh` und `tanh` aufgerufen.

27

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Command Window

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

>> sin(pi)
ans =
 1.2246e-16

>> asind(1)
ans =
    90
```

fx

28

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Die Exponentialfunktion

Die Exponentialfunktion wird in MATLAB mit `exp` angesprochen. `exp(1)` liefert die bekannte eulersche Zahl  $e$ .

Command Window

```
>> exp(1)
ans =
    2.7183
```

fx

Das Argument der  $e$ -Funktion kann auch komplex sein. Das Ergebnis wird dann gemäß

$$e^{\sigma + j\omega} = e^{\sigma} (\cos(\omega) + j \sin(\omega))$$

bestimmt.

29

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Die Logarithmusfunktion

Die Umkehrfunktion, der natürliche Logarithmus, wird über die Funktion `log` angesprochen. Der Logarithmus zur Basis 10 heißt `log10` und der Logarithmus zur Basis 2 entsprechend `log2`.

Command Window

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

>> log(exp(3))
ans =
    3

>> log10(0.01)
ans =
   -2
```

30

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Wurzeln

Mit dem Befehl `sqrt` wird die (Quadrat-)Wurzel einer Zahl bestimmt. Das Argument kann auch negativ oder komplex sein.

Command Window

```
>> sqrt(2)
ans =
    1.4142

>> sqrt(-1)
ans =
    0.0000 + 1.0000i

>> x = sqrt(3+4i)
x =
    2.0000 + 1.0000i
```

31

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

### Komplexe Rechnungen

Nun wollen wir einige spezielle Funktionen für komplexe Zahlen kennenlernen. Die Befehle `real` und `imag` liefern Real- und Imaginärteil einer komplexen Zahl; `abs` gibt deren Betrag an und `angle` die Phase.

Command Window

```
>> real(x)
ans =
    2

>> imag(x)
ans =
    1

>> abs(x)
ans =
    2.2361
```

Command Window

```
>> angle(x)
ans =
    0.4636

>> conj(x)
ans =
    2.0000 - 1.0000i
```

32

Notizen

---

---

---

---

---

---

---

---

**AUFGABE 2 | Elementare mathematische Funktionen**

1. Berechnen Sie die folgenden Ausdrücke:

- ▶  $j\sqrt{2}$
- ▶  $s \cos^2(\pi/4)$
- ▶  $|-2 + j|$
- ▶  $\lg(2,3 \cdot 10^{-4})$
- ▶  $1/e$

2. Es seien  $z_1 = 4 + 3i$  und  $z_2 = -2 - 2i$ . Berechnen Sie:

- ▶  $z_1 * \bar{z}_1$
- ▶ den Realteil von  $z_2$
- ▶ den Imaginärteil von  $z_1$
- ▶  $|z_1|^2$
- ▶  $\frac{1}{2}(z_2 + \bar{z}_2)$
- ▶  $\frac{1}{2j}(z_1 - \bar{z}_1)$

Was fällt Ihnen auf? Können Sie die Vermutung beweisen?

Notizen

---

---

---

---

---

---

---

---

**Funktionen**

Bisher haben wir bereits einige vorgefertigte Funktionen kennengelernt. Wie können wir aber eigene Funktionen nach unseren Bedürfnissen definieren?

Für umfangreiche Funktionen werden wir das im Kapitel „Skripte & Funktionen“ lernen. „Kleine“ Funktionen werden in MATLAB als anonyme Funktionen bezeichnet und bestehen aus einem MATLAB-Ausdruck und beliebig vielen Ein- und Ausgabeparametern.

Die Syntax für eine Funktion  $f$  lautet

$$f = @(Argumente) Expression$$

Notizen

---

---

---

---

---

---

---

---

Wir wollen uns eine anonyme Funktion `sqf` definieren, die das Quadrat einer Zahl `x` berechnet.

Dies geschieht mit

```
Command Window
>> sqf = @(x) x.^2
sqf =
function_handle with value:
@(x) x.^2
```

**Achtung!**

Aktuell wissen wir noch nicht, warum man diese seltsam anmutende Schreibweise `x.^2` benutzt. Das lernen wir im nächsten Kapitel.

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Einsetzen verschiedener Werte bestätigt jedoch die Richtigkeit der Funktion:

```
Command Window
>> sqrt(5)

ans =
    2.2361

>> sqrt(i)

ans =
   -1.0000i
```

37

Notizen

---

---

---

---

---

---

---

---

## MATLAB als intelligenter Taschenrechner

Anonyme Funktionen können von mehreren Variablen abhängen.

```
Command Window
>> myellipse = @(x,y) (x.^2 + y.^2 + x.*y);
>> x = 1;
>> y = 10;
>> z = myellipse(x, y)

z =
    111
```

38

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Vektoren

Wir wollen nun die Vorteile von MATLAB ausnutzen und lernen im Folgenden die Eigenschaften von Vektoren & Matrizen kennen.

Matrizen werden in MATLAB durch eckige Klammern gekennzeichnet; Elemente innerhalb einer Zeile werden durch Kommata oder Leerzeichen getrennt.

```
Command Window
>> x = [1, 2, 3, 4]
x =
     1     2     3     4
>> x = [1 2 3 4]
x =
     1     2     3     4
fx
```

40

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Analog erzeugen wir einen Spaltenvektor:

```
Command Window
>> x = [1; 2; 3; 4]
x =
     1
     2
     3
     4
fx
```

41

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Matrizen

Matrizen werden in MATLAB in Analogie zu Vektoren definiert. In der folgenden Matrix X mögen die Einträge auf die Zeilen- und Spaltenindizes hinweisen.

```
Command Window
>> X = [11, 12, 13, 14; 21, 22, 23, 24]
X =
    11    12    13    14
    21    22    23    24
fx
```

X ist also eine  $2 \times 4$ -Matrix. Mit eckigen Klammern können einzelne Matrizen bzw. Vektoren aneinander gehängt werden, sofern sie „passen“.

42

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Command Window

```
>> x = [1, 2, 3, 4]
x =
     1     2     3     4
>> Xx = [X; x]
Xx =
    11    12    13    14
    21    22    23    24
fx    1     2     3     4
```

Wir haben unsere  $2 \times 4$ -Matrix um eine Zeile zu einer  $3 \times 4$ -Matrix erweitert.

43

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Vektoren mit Zahlenfolgen

Vektoren mit speziellen Zahlenfolgen wie  $x = [1, 2, 3, 4]$  können deutlich schneller über einen Doppelpunkt gemäß

Anfangswert:Endwert

erzeugt werden.

Command Window

```
>> x = 1:4
x =
     1     2     3     4
fx
```

Die Schrittweite muss nicht 1 betragen, sondern kann gesondert über  
Anfangswert:Schrittweite:Endwert  
angegeben werden.

44

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Command Window

```
>> x = 1:2:4
x =
     1     3
>> x = 4:-1:1
x =
     4     3     2     1
fx
```

45

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Transponierte

Mit einem nachgestellten Hochkomma ' lassen sich Vektoren bzw. Matrizen transponieren.

#### Command Window

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

46

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Elementare Operationen

Viele Operationen, die wir bisher kennengelernt haben, lassen sich auch auf Matrizen anwenden. So können Matrizen addiert und subtrahiert werden, sofern die Dimensionen übereinstimmen.

#### Command Window

```
>> x=1:3; y = 4:6; x+y  
  
ans =  
  
     5     7     9  
  
>> A = [11, 12; 21, 22]; B = [1, 2; 3, 4]; A+B  
  
ans =  
  
    12    14  
    24    26
```

48

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Elementare Operationen

Ebenso sind die skalare Multiplikation, die Matrixmultiplikation und die Matrix-Vektor-Multiplikation definiert.

#### Command Window

```
>> A = [1 2; 0 1]; x = [-1; 1]; A*x  
  
ans =  
  
     1  
     1  
  
>> A*A  
  
ans =  
  
     1     4  
     0     1
```

Wie lassen sich jedoch Lineare Gleichungssysteme der Form  $Ax = b$  lösen?

49

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Lineare Gleichungssysteme

Lineare Gleichungssysteme der Form  $Ax = b$  können (in der Theorie) auf zwei Arten gelöst werden:

1. Multiplikation der Gleichung von links mit  $A^{-1}$ :  
 $x = A^{-1}b$

2. Gauß-Algorithmus, QR-Zerlegung

Die erste Variante ist jedoch in der Praxis oft problematisch. Wir beschränken uns daher auf die zweite Variante.

Dazu betrachten wir zwei Operatoren: `/` für Rechts- und `\` für Linksdivision.

Was geht schief?

50

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Lineare Gleichungssysteme

Betrachten wir zunächst ein einfaches Beispiel und berechnen  $1/2$  sowie  $1 \setminus 2$ .

Command Window

```
>> 1/2
```

```
ans =
```

```
0.5000
```

```
>> 1\2
```

```
ans =
```

```
2
```

Übertragen auf (quadratische) Matrizen bedeutet  $A/B$  also  $A$  multipliziert mit  $B^{-1}$  und  $A \setminus B$  entsprechend  $A^{-1}$  multipliziert mit  $B$ .

51

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Command Window

```
>> A = [1 2; 0 1]; b = [1;1]; x = A\b
```

```
x =
```

```
-1
```

```
1
```

52

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 3 | Lineare Gleichungssysteme

Lösen Sie das Lineare Gleichungssystem  $Ax = b$  für

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

2.  $A = \begin{pmatrix} 6 & 3 & 4 \\ 0 & 0 & -5 \end{pmatrix}$  und  $b = \begin{pmatrix} 1 \\ 10 \end{pmatrix}$ .

Was fällt Ihnen auf?

53

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Zur Vollständigkeit

Einige spezielle Matrixoperationen:

1. Die Inverse einer Matrix erhält man mit dem Befehl `inv`.
2. Die Determinante einer Matrix bestimmt man mit `det`.
3. `trace` berechnet die Spur einer Matrix.
4. `rank` berechnet den Rang einer Matrix.
5. Der Befehl `transpose` ist äquivalent zu einem nachgestellten `'`.

### Command Window

```
>> A, inv(A)
A =
     1     2
     0     1

ans =
     1    -2
     0     1
```

54

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Einige Besonderheiten von MATLAB

MATLAB beherrscht einige Operationen auf Matrizen, die Berechnungen signifikant vereinfachen können.

Alle elementaren Funktionen, die wir bisher kennengelernt haben, können auch auf Matrizen angewandt werden.

### Command Window

```
>> x = 1:5; sqrt(x)

ans =
     1.0000     1.4142     1.7321     2.0000     2.2361

>> X = [1 2; 0 1]; exp(X)

ans =
     2.7183     7.3891
     1.0000     2.7183
```

55

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 4 | Vektoren & Matrizen I

1. Sei  $A = \begin{bmatrix} 2 & 4 & 2 & 3 & 1 \end{bmatrix}$  und  $B = \begin{bmatrix} 2 & 5 & 8 & 3 & 7 \end{bmatrix}^T$ . Berechnen Sie  $AB$  und  $BA$ .
2. Gegeben Sei der Vektor  $v = \begin{bmatrix} 1 & 2 & 3 & \dots & 100 \end{bmatrix}^T$ . Berechnen Sie die Norm dieses Vektors, d. h.  $|v| := \sqrt{v^T v}$ .
3. Erstellen Sie einen Vektor, der alle geraden Zahlen zwischen 31 und 73 enthält.
4. Sei  $x = \begin{bmatrix} 0 & \pi/2 & \pi & 3\pi/2 & 2\pi \end{bmatrix}$ .
  - 4.1 Addieren Sie  $\pi$  zu jedem Element.
  - 4.2 Berechnen Sie die Wurzel jedes Elements.
  - 4.3 Berechnen Sie den Sinus und den Kosinus jedes Elements.

56

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Punktweise Operationen

Neben den „gewöhnlichen“ Matrixoperationen stellt MATLAB auch **element- bzw. punktweise Operationen** zur Verfügung. Diese werden durch einen vorgestellten Punkt vor dem Operatorzeichen unterschieden.

Command Window

```
>> A = [1 2; 3 4]; B = [1 2; 1 2]; A.*B, A./B  
  
ans =  
    1     4  
    3     8  
  
ans =  
    1     1  
    3     2  
fx
```

57

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 5 | Vektoren & Matrizen II

1. Berechnen Sie das Skalarprodukt der Vektoren  $a = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix}^T$  und  $b = \begin{bmatrix} 1 & -2 & 5 \end{bmatrix}^T$  ohne die Rechenregel  $a^T b$  zu verwenden.
2. Es seien  $x = \begin{bmatrix} 1 \\ 3 \\ 7 \\ 2 \end{bmatrix}$  und  $y = \begin{bmatrix} -1 \\ 2 \\ -8 \\ 5 \end{bmatrix}$ .
  - 2.1 Berechnen Sie die Summe der beiden Vektoren  $x$  und  $y$ .
  - 2.2 Dividieren Sie jedes Element von  $x$  durch das entsprechende Element von  $y$ .
  - 2.3 Multiplizieren Sie jedes Element von  $y$  mit dem korrespondierenden Element von  $x$ .
  - 2.4 Potenzieren Sie jedes Element von  $x$  mit dem korrespondierenden Element von  $y$ .

58

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Spezielle Matrizen

Für einige Matrizen, die man häufig benötigt, stehen eigenständige Befehle zur Verfügung.

**zeros**(*i*, *j*) erstellt eine Nullmatrix mit *i* Zeilen und *j* Spalten.

**ones**(*i*, *j*) erstellt eine  $i \times j$ -Matrix mit ausschließlich 1-Einträgen.

**eye**(*i*, *j*) erstellt eine  $i \times j$  Einheitsmatrix.

**rand**(*i*, *j*) erstellt eine  $i \times j$ -Matrix mit gleichverteilten Zufallszahlen aus dem Intervall (0,1).

#### Command Window

```
>> rand(3)
ans =
    0.8774    0.0969    0.9841
    0.9446    0.8006    0.3885
fx      0.3273    0.1513    0.8397
```

60

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Intervalle

Mit Hilfe der Doppelpunktsyntax haben wir Vektoren mit gleichmäßigen Zahlenfolgen erzeugt. Ähnlich funktioniert die Funktion `linspace`. Wir geben jedoch nicht die Schrittweite, sondern die Schrittzahl<sup>2</sup> gemäß

`linspace(Anfangswert, Endwert, Anzahl)`

an.

#### Command Window

```
>> x = linspace(0, 1, 5)
x =
fx      0    0.2500    0.5000    0.7500    1.0000
```

<sup>2</sup>Lässt man den Parameter Anzahl weg, so werden 100 Werte erzeugt.

61

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Größe und Dimension

Mit dem Befehl `length` wird die Länge eines Vektors ausgegeben; der Befehl `size` gibt einen Vektor zurück, dessen Komponenten die Zeilen- und Spaltenzahl einer Matrix sind.

#### Command Window

```
>> x = linspace(0, 1, 5); length(x)
ans =
    5
>> A = rand(3, 2); size(A)
ans =
fx      3      2
```

63

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 6 | Matrixmanipulation

Erstellen Sie eine  $(7 \times 8)$ -Matrix  $A$  mit verschiedenen Einträgen. Erklären Sie, was die folgenden Befehle bewirken:

1.  $A'$
2.  $A(:, [1 \ 4])$
3.  $A([2 \ 6], [6 \ 1])$
4.  $\text{reshape}(A, 2, 28)$
5.  $A(:)$
6.  $\text{flipud}(A)$
7.  $\text{fliplr}(A)$
8.  $[A; A(\text{end}, :)]$
9.  $A(1:6, :)$
10.  $[A; A(1:2, :)]$
11.  $\text{sum}(A)$
12.  $\text{sum}(A')$
13.  $\text{sum}(A, 2)$
14.  $[A \ \text{zeros}(\text{size}(A)); \text{ones}(\text{size}(A)) \ A]$

64

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Matrizen manipulieren

Bisher haben wir einige „straight forward“ Berechnungen mit Vektoren und Matrizen kennengelernt. Aber auf der anderen Seite erfordern einige Probleme „nicht-kanonische“ Rechnungen.

- ▶ Wie lautet der Eintrag  $a_{550,1234}$  einer  $1000 \times 2000$ -Matrix  $A$ ?
- ▶ Wie bilde ich die Summe der ersten zehn Spalten von  $A$ ?
- ▶ Wie kann ich Spalten meiner Matrix vertauschen?
- ▶ Was ist das Produkt aller Einträge von  $A$ ?

Dazu wollen wir im folgenden die Matrix  $A = [11, 12, 13; 21, 22, 23]$  betrachten. Wie früher stehen die Einträge der Matrix hier sinnbildlich für ihren jeweiligen Index.

66

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Auf (einzelne) Elemente zugreifen

Wie wir im Abschnitt zuvor gelernt haben, erhalten wir mit  $\text{size}(A)$  in unserem Fall den Vektor  $[2, 3]$ . Auf den Eintrag  $a_{i,j}$  der Matrix  $A$  können wir mit der Syntax  $A(i, j)$  zugreifen.

```
Command Window
>> A(2, 3)
ans =
    23
```

Durch eine Zuweisung kann das entsprechende Element geändert werden.

67

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Command Window

```
>> A(2, 3) = 233  
  
A =  
    11    12    13  
    21    22    233
```

Wie erfolgt jedoch der Zugriff auf eine Zeile bzw. Spalte? Die obige Syntax hat eine wildcard, den Doppelpunkt `:`. So liefert `A(:, j)` die  $j$ -te Spalte der Matrix  $A$ . Analog erhält man mit `A(i, :)` die  $i$ -te Zeile von  $A$ .

68

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Command Window

```
>> A(:, 1)  
  
ans =  
    11  
    21  
  
>> A(2, :)  
  
ans =  
    21    22    233
```

69

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Auf (mehrere) Elemente zugreifen

Möchte man nicht nur auf einen Eintrag / eine Zeile / Spalte zugreifen, sondern auf komplette Teilmatrizen, so ist dies ebenfalls mit dieser Syntax möglich.

Der Befehl

$$A(:, [1, 3])$$

greift beispielsweise auf die (komplette) erste und dritte Spalte der Matrix  $A$  zu.

Analog kann auch auf Zeilen und entsprechende Teilmengen zugegriffen werden.

70

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Command Window

```
>> A(:, [1, 3])

ans =
    11    13
    21    233

>> A([1, 2], 1)

ans =
    11
    21

>> A(1, [1, 2])

ans =
    11    12
```

71

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Spaltentausch

Manchmal haben o. B. d. A. die Spalten einer Matrix eine „anschauliche“ Bedeutung, z. B. Ergebnisse einer Rechnung oder Messung.

Möchte man die Reihenfolge der Spalten umsortieren, also z. B. die  $i$ -te und 1-te Spalte tauschen, so geschieht dies mit

$$A(:, [i, 1]) = A(:, [1, i]).$$

Analog verfährt man für Zeilen.

72

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Command Window

```
>> A = [11:14; 21:24; 31:34; 41:44]

A =
    11    12    13    14
    21    22    23    24
    31    32    33    34
    41    42    43    44

>> A(:, [1, 3]) = A(:, [3, 1])

A =
    13    12    11    14
    23    22    21    24
    33    32    31    34
    43    42    41    44
```

73

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Logische Indizierung

Im Abschnitt „Logische Operatoren“ haben wir logische Operatoren kennengelernt. Diese kann man auch zur praktischen Indizierung von Vektoren und Matrizen benutzen.

Angenommen der Vektor  $x$  enthält eine Messgröße. Uns interessieren nun die Indizes der Einträge dieser Zeitreihe, die größer oder kleiner (gleich) einem gewissen Schwellwert  $x\_thresh$  sind.

Diese erhält man mit der Abfrage

```
x >= x_thresh
```

74

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Command Window

```
>> A = rand(3)

A =

    0.3922    0.7060    0.0462
    0.6555    0.0318    0.0971
    0.1712    0.2769    0.8235

>> A>=0.5

ans =

    3x3 logical array

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

75

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Möchte man anstatt der Indizes die konkreten Werte, so können wir die Indizierungstechnik aus diesem Abschnitt mit der logischen Indizierung verknüpfen:

### Command Window

```
A =

    0.3922    0.7060    0.0462
    0.6555    0.0318    0.0971
    0.1712    0.2769    0.8235

>> A(A>=0.5)

ans =

    0.6555
    0.7060
    0.8235
```

76

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Von Summen und Produkten

Die Befehle `sum` und `prod` erlauben es, Summen und Produkte beliebiger Einträge zu berechnen, ohne diese explizit aufschreiben zu müssen.

`sum(A)` berechnet die Spaltensummen von  $A$ ; die Funktion liefert also einen Zeilenvektor, der so viele Spalten wie  $A$  hat.

`prod(A)` berechnet das Produkt der Spaltenelemente von  $A$ . Das Ergebnis hat die gleiche Gestalt wie das von `sum`.

77

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Command Window

```
>> A = [1, 2, 3; 4, 5, 6]
A =
     1     2     3
     4     5     6

>> sum(A)

ans =
     5     7     9

>> sum(A(:, 1))

ans =
     5

>> prod(A)

ans =
     4    10    18
```

78

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Ist  $v$  ein Vektor, so erhält man mit `sum` bzw. `prod` die Summe bzw. das Produkt der Einträge.

### Command Window

```
>> v = 1:6

v =
     1     2     3     4     5     6

>> sum(v)

ans =
    21

>> prod(v)

ans =
   720
```

79

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 7 | Logische Indizierung

- ▶ Erstellen Sie zufällige Matrizen mit Größe ihrer Wahl.
- ▶ Nutzen Sie die logische Indizierung zum Zählen wie oft der (von Ihnen gewählte) Schwellwert überschritten wird.
- ▶ Machen Sie sich (kurz) mit den Funktionen
  - ▶ `isnan`
  - ▶ `isinf`
  - ▶ `isfinite`
  - ▶ `isnumeric`
  - ▶ `ismatrix` und
  - ▶ `isvector`vertraut.

80

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Polynome

Das (reelle) Polynom  $n$ -ten Grades

$$p(x) := a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$$

wird in MATLAB durch seine  $n + 1$  Koeffizienten in absteigender Reihenfolge definiert

$$p = [a_n, a_{n-1}, \dots, a_1, a_0].$$

Das Kommando `p = [1 3 2]`; definiert also das quadratische Polynom

$$p(x) = x^2 + 3x + 2.$$

82

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Polynome

Die Berechnung des Funktionswerts an einer Stelle  $x$  erfolgt mit der Funktion `polyval`. Für  $x$  kann auch ein Vektor verwendet werden.

Command Window

```
>> p = [1, 3, 2];
>> polyval(p, 0), polyval(p, linspace(0, 1, 10))

ans =

     2

ans =

    2.0000    2.3457    2.7160    3.1111    3.5309
    3.9753    4.4444    4.9383    5.4568    6.0000
fx
```

83

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Nullstellen von Polynomen

Gemäß des Fundamentalsatzes der Algebra hat ein Polynom vom Grad  $n$  genau  $n$  ggf. komplexe Nullstellen. Diese können mit dem Befehl `roots` bestimmt werden.

```
Command Window
>> xN = roots(p)

xN =

    -2
    -1
fx
```

Damit wird das Polynom in seine Linearfaktoren zerlegt

$$p(x) = a_n(x - x_{N_1})(x - x_{N_2}) \cdots (x - x_{N_n}).$$

84

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Nullstellen von Polynomen

Umgekehrt lässt sich mit `poly` aus den Nullstellen  $x_{N_1}, \dots, x_{N_n}$  der Vektor mit den Koeffizienten  $a_i$  bestimmen.

```
Command Window
>> xN = [-2; -1]; p = poly(xN)

p =

    1     3     2
fx
```

O. B. d. A. wird der Leitkoeffizient  $a_n$  des Polynoms  $p$  auf 1 normiert.

85

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Produkt zweier Polynome (Faltung)

Das Produkt der beiden Polynome

$$p_1(x) = x^2 + 3x + 2$$

$$p_2(x) = x^2 + 1$$

mit den Koeffizienten  $p_1 = [1 \ 3 \ 2]$  und  $p_2 = [1 \ 0 \ 1]$  kann mit dem Befehl `conv` berechnet werden.

```
Command Window
>> p1 = [1 3 2]; p2 = [1 0 1]; p = conv(p1, p2)

p =

    1     3     3     3     2
fx
```

86

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

Von der Richtigkeit kann man sich leicht überzeugen:

$$p(x) = (x^2 + 3x + 2)(x^2 + 1) = x^4 + 3x^3 + 3x^2 + 3x + 2.$$

### Achtung!

Die Verkettung ist nicht mit der Multiplikation von Vektoren zu verwechseln. Die Verwendung des Multiplikationsoperators \* würde hier zu einem Fehler führen.

87

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Polynomdivision

Für die Polynomdivision verwenden wir die Funktion `deconv`. Aus einer unecht gebrochenrationalen Funktion wird so eine ganzrationale Funktion  $q(x)$  und eine echt gebrochenrationale Funktion  $r(x)$ .

$$p(x) = \frac{x^2 + 3x + 2}{x^2 + 1} = q(x) + \frac{r(x)}{x^2 + 1} = 1 + \frac{3x + 1}{x^2 + 1}$$

### Command Window

```
>> [q, r] = deconv(p1, p2)

q =
    1

r =
    0     3     1
fx
```

88

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Partialbruchzerlegung

Als letztes wollen wir die Partialbruchzerlegung der echt gebrochenrationalen Funktion

$$\frac{4x^2 + 4x + 1}{x^3 + x^2} = \frac{r_1}{x - x_{p_1}} + \frac{r_2}{x - x_{p_2}} + \frac{r_3}{(x - x_{p_3})^2}$$

bestimmen.

Mit dem Befehl `residue` erhalten wir die Residuen  $r$  und die Polstellen  $x_p$  (also die Nullstellen des Nennerpolynoms).

Man kann nachrechnen, dass in unserem Fall

$$\frac{4x^2 + 4x + 1}{x^3 + x^2} = \frac{1}{x + 1} + \frac{3}{x} + \frac{1}{x^2}$$

gilt.

89

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

```
Command Window
>> [r, xP] = residue([4, 4, 1], [1, 1, 0, 0])

r =

     1
     3
     1

xP =

    -1
     0
     0

fx
```

90

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 8 | Polynomfunktionen

- ▶ Definieren Sie das Zählerpolynom  $4x^4 - 4$  als Variable  $zP$ .
- ▶ Definieren Sie die beiden Linearfaktoren des Nenners  $x - 2$  und  $x + 2$  als Variablen  $np1$  und  $np2$ .
- ▶ Berechnen Sie das Nennerpolynom  $np$  über eine Polynommultiplikation.
- ▶ Bestimmen Sie die Nullstellen des Zählerpolynoms.
- ▶ Zerlegen Sie die unecht gebrochenrationale Funktion  $zP/np$  in eine ganzrationale Funktion  $qp$  und einen echt gebrochenrationalen Rest  $rp$ .
- ▶ Führen Sie eine Partialbruchzerlegung des Restes  $rp$  durch. Bezeichnen Sie dabei die Residuen mit  $r$  und die Polstellen mit  $xP$ .

91

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Die Länge eines Vektors

Im Abschnitt „Mathematische Operationen auf Vektoren und Matrizen“ haben wir bereits einige häufig verwendete Matrixbefehle<sup>3</sup> kennengelernt.

Mit dem Befehl `norm` kann die Norm eines Vektors  $x$  berechnet werden. Die „verschiedenen“  $p$ -Normen

$$\|x\|_p := \left( \sum_{i=1}^n |x_i|^p \right)^{1/p}$$

können über einen optionalen Parameter  $p$  gemäß der Syntax

$$\|x\| = \text{norm}(x, p)$$

berechnet werden. Standardmäßig ist  $p=2$ .

<sup>3</sup>det, inv, rank, trace und transpose

93

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Die Länge eines Vektors

Für den Vektor  $x = [3, -2, 6]$  gilt

$$\|x\|_1 = |3| + |2| + |-6| = 11$$

$$\|x\|_2 = \sqrt{|3|^2 + |2|^2 + |-6|^2} = \sqrt{49} = 7$$

$$\|x\|_\infty = \max\{|3|, |2|, |-6|\} = 6$$

#### Command Window

```
>> x = [3, -2, 6]; norm(x, 1), norm(x), norm(x, Inf)

ans =
    11

ans =
     7

ans =
     6
```

94

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Wurzeln von Matrizen

Ebenso haben wir bereits gelernt, dass der Befehl `sqrt` elementweise die Wurzeln von Matrizen berechnet.

#### Command Window

```
>> A = [5 4; 4 5]; sqrt(A)

ans =

    2.2361    2.0000
    2.0000    2.2361
```

Wie kann man aber ein Problem der Form  $X \cdot X = A$  für Matrizen  $A$  und  $X$  lösen? Dies geschieht mit dem Befehl `sqrtm`.

95

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

#### Command Window

```
>> A = [5 4; 4 5]; sqrt(A), sqrtm(A)

ans =

    2.2361    2.0000
    2.0000    2.2361

ans =

    2.0000    1.0000
    1.0000    2.0000
```

96

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Matrix-Exponentialfunktion

Bei der Behandlung von Differentialgleichungen<sup>4</sup> ist die Matrix-Exponentialfunktion `expm` ein nützliches Werkzeug. Auch sie arbeitet wie `sqrtm` nicht elementweise.

#### Command Window

```
>> A = [1 1 1; 1 0 1; -1 -1 -1], expm(A)

A =
     1     1     1
     1     0     1
    -1    -1    -1

ans =
    2.5000    1.0000    1.5000
    1.0000    1.0000    1.0000
   -1.5000   -1.0000   -0.5000
```

<sup>4</sup>Siehe Vorlesung „Applied Differential Equations“

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Eigenwerte und Eigenvektoren

Als letztes Beispiel der Linearen Algebra wollen wir die oft benötigte Berechnung der Eigenwerte  $\lambda$  einer Matrix  $A$  betrachten.

Die Eigenwerte stellen die nichttriviale Lösung der Gleichung

$$Av = \lambda v \quad (*)$$

dar. Die Vektoren  $v$  nennt man Eigenvektoren zum Eigenwert  $\lambda$ .

In der Vorlesung zur Linearen Algebra zeigt man, dass  $\lambda$  Bedingung genau dann (\*) erfüllt, wenn  $\lambda$  auch

$$p_A(\lambda) = \det(A - I\lambda) = 0$$

erfüllt.

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Eigenwerte und Eigenvektoren

Die Eigenwerte einer Matrix bestimmt man in MATLAB mit dem Befehl `eig`.

#### Command Window

```
>> A = [-1 1; 0 -2], eig(A)

A =
    -1     1
     0    -2

ans =
    -1
    -2
```

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### Eigenwerte und Eigenvektoren

Gibt man dem Befehl `eig` zwei Rückgabewerte `V` und `D` an, so erhält man eine Matrix `V`, deren Spalten die (auf 1 normierten) Eigenvektoren zu den Eigenwerten aus der Diagonalmatrix `D` enthalten.

```
Command Window
>> A = [-1 1; 0 -2]; [V, D] = eig(A)

V =
    1.0000    -0.7071
         0     0.7071

D =
   -1     0
    0    -2
```

100

Notizen

---

---

---

---

---

---

---

---

## Vektoren & Matrizen

### AUFGABE 9 | Eigenwerte und Eigenvektoren

Bestimmen Sie für die Matrix

$$A = \begin{pmatrix} 0 & 2 & -1 \\ 2 & -1 & 1 \\ 2 & -1 & 3 \end{pmatrix}$$

die Eigenwerte von Hand. Kontrollieren Sie ihr Ergebnis mit `MATLAB`. Bestimmen Sie außerdem die zugehörigen Eigenvektoren mit `MATLAB`.

101

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Save und load

Bei umfangreicheren Rechnungen oder Messungen kann es nützlich sein, Inhalte von Variablen abzuspeichern, um so später schnell auf die Ergebnisse zugreifen zu können. Dies geschieht mit dem Befehl `save`.

Die Syntax hierfür lautet

```
save Dateiname Variablenliste
```

Ohne die Variablenliste werden alle aktuellen Variablen gespeichert.

Entsprechend lädt der Befehl

```
load Dateiname
```

die Variablen aus der Datei wieder in den Variablenspeicher.

102

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Achtung!

Eine bereits existierende Variable wird durch eine gleich lautende Variable beim Laden überschrieben.

Der Befehl `save` setzt voraus, dass Schreibrechte im aktuellen Verzeichnis bestehen. Der Dateiname kann auch eine Pfadbezeichnung beinhalten, wenn nicht in das aktuelle Verzeichnis geschrieben werden soll oder darf.

103

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Save und load

Wir erstellen 4 Variablen. Mit dem ersten `save`-Befehl werden alle gespeichert, mit dem zweiten nur die, die mit `x` beginnen, und mit dem dritten Befehl die Matrix `A` und `x1`.

#### Command Window

```
>> x1=1; x2=2; x3=3; A=eye(3);  
>> save MeineDatei_alles  
>> save MeineDatei_x x*  
>> save MeineDatei_Ax1 A x1
```

Lässt man, wie im obigen Beispiel, die Dateierweiterung weg, so ergänzt MATLAB automatisch die Endung `.mat`.

104

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Save und load

Um zu erkennen, welche Variablen geladen werden, löschen wir zuvor alle aktuell existierenden Variablen mittels `clear` aus dem Speicher.

Mit `who` bzw. `whos` können wir die geladenen Variablen anzeigen lassen.

```
Command Window
>> clear; load MeineDatei_x, whos
Name      Size      Bytes  Class  Attributes
x1        1x1         8  double
x2        1x1         8  double
x3        1x1         8  double

>> clear; load MeineDatei_alles, who

Your variables are:

fx A  x1  x2  x3
```

105

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Save und load

Manchmal ist es ratsam, die Daten im ASCII-Format<sup>5</sup> zu speichern und zu laden. Der Befehl

```
save MeineDatei.asc x* -ascii
```

speichert die Variablen `x1`, `x2` und `x3` in der ASCII-Datei `MeineDatei.asc`.

Dabei handelt es sich um eine gewöhnliche Textdatei mit folgendem Inhalt:

```
1.0000000e+00
2.0000000e+00
3.0000000e+00
```

<sup>5</sup>American Standard Code for Information Interchange

106

Notizen

---

---

---

---

---

---

---

---

## Daten speichern und laden

### Save und load

Nachteil des ASCII-Formats ist, dass sämtliche Informationen über Variablennamen beim Schreiben verloren gehen. Das macht sich beim Laden bemerkbar.

```
Command Window
>> clear, load MeineDatei.asc, whos
Name      Size      Bytes  Class
Attributes
fx MeineDatei  3x1         24  double
```

Das ASCII-Format ist also eher für den Dateiaustausch mit anderen Programmen geeignet.

107

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

Notizen

---

---

---

---

---

---

---

## Abbildungen

### MATLAB als Funktionsplotter

MATLAB bietet leistungsfähige Funktionen, mit denen Daten aller Art sehr vielfältig visualisiert werden können. Wir werden uns auf zwei- und dreidimensionale Funktionsplots beschränken. Für weitere Informationen sei auf die Hilfefunktion verwiesen.

Grafiken werden in eigenen Fenstern, so genannten **Figures**, dargestellt. Der Befehl `figure` ohne Argument erzeugt ein neues, fortlaufend nummeriertes Grafikfenster. Mit `figure(Nummer)` kann die Nummer auch vorgegeben werden.

Mit dem Befehl `close` wird das aktuelle Grafikfenster geschlossen, mit `close(Nummer)` das Grafikfenster mit der entsprechenden Nummer.

Der Befehl `clf` löscht den Inhalt des aktuellen Grafikfenster, schließt es aber nicht.

109

Notizen

---

---

---

---

---

---

---

## Abbildungen

### Der erste Plot

Wir wollen mit einer (linearen)  $x$ - $y$ -Darstellung der Sinusfunktion im Bereich von 0 bis  $2\pi$  beginnen. Dazu erstellen wir zunächst geeignete Vektoren  $x$  und  $y$  und plotten diese anschließend mit dem Befehl `plot(x, y)`

Command Window

```
>> x = linspace(0, 2*pi, 10); y = sin(x); plot(x, y)
```

Falls zuvor kein Grafikfenster erzeugt wurde, wird automatisch das Fenster mit Nummer 1 erstellt.

Die Datenpunkte werden als blauer, linear interpolierter Streckenzug dargestellt. Mit mehr Punkten erhalten wir einen „glatteren“ Verlauf des Plots.

110

Notizen

---

---

---

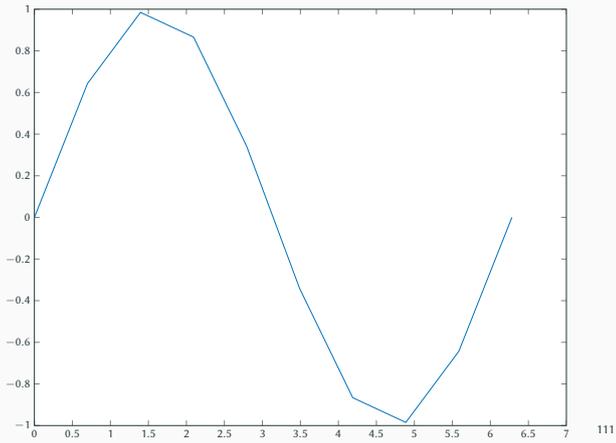
---

---

---

---

## Abbildungen



111

Notizen

---

---

---

---

---

---

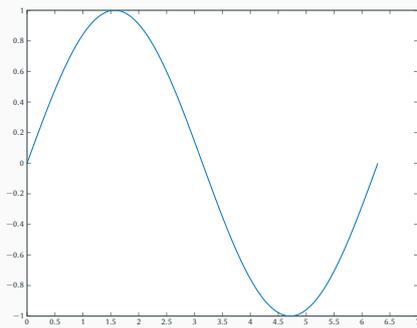
---

---

## Abbildungen

Command Window

```
>> x = linspace(0, 2*pi); y = sin(x); plot(x, y)
```



112

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Plotten mit Stil

Die Darstellungsart kann über Stiloptionen mit dem Befehl

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

verändert werden.

Die Stiloption besteht aus bis zu drei Stilparametern für Farbe, Markierung und Linienart.

Parameter	Farbe	Parameter	Markierung
r	rot	+	Pluszeichen
g	grün	o	Kreis
b	blau	*	Stern
c	cyan	.	Punkt
m	magenta	x	Kreuz
y	gelb	s	Quadrat
k	schwarz	d	Raute
w	weiß		

113

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

Parameter	Linienart
-	durchgezogen
--	gestrichelt
:	punktiert
-.	strichpunktiert

114

Notizen

---

---

---

---

---

---

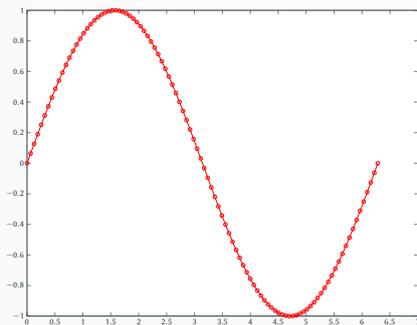
---

---

## Abbildungen

Command Window

```
>> x = linspace(0,2*pi); y = sin(x); plot(x, y, 'r-o')
```



115

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Plot-Optionen

Mit dem Schalter `grid` kann ein Raster im Grafikenfenster ein- und ausgeschaltet werden. Mit dem Parameter `on` bzw. `off` kann das Raster unabhängig vom aktuellen Zustand ein- bzw. ausgeschaltet werden.

Mit dem Befehl `axis` lässt sich die Achsenskalierung anpassen. Übergibt man der Funktion einen Vektor mit vier Zahlen gemäß

```
axis([xmin xmax ymin ymax])
```

so ändern sich die minimalen und maximalen Werte für Abszisse und Ordinate entsprechend. Mit

```
axis auto
```

wird die automatische Skalierung wieder aktiviert.

Möchte man nur die Skalierung der  $x$  bzw.  $y$ -Achse ändern, so sind die beiden folgenden Funktionen hilfreich: `xlim([xmin xmax])` bzw. `ylim([ymin ymax])`.

117

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Plots beschriften

Nun wollen wir unsere Abbildungen beschriften. Mit `title` kann eine Überschrift platziert werden, mit `xlabel` wird die x-Achse und mit `ylabel` die y-Achse beschriftet.

Command Window

```
>> title('Unser_erster_Plot')  
>> xlabel('Winkel_im_Bogenmaß')  
fx >> ylabel('Sinus')
```

118

Notizen

---

---

---

---

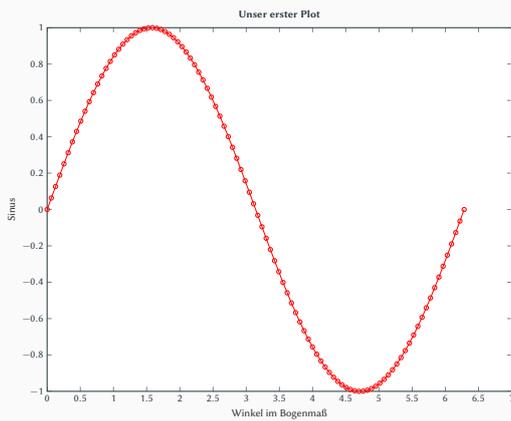
---

---

---

---

## Abbildungen



119

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Mehrere Plots auf einmal

Wir wollen nun eine zweite Kurve, den Kosinus, in dasselbe Diagramm zeichnen. Wir haben bereits gesehen, dass durch einen weiteren Aufruf der `plot`-Funktion der vorherige Plot gelöscht wird.

Es ist jedoch auch möglich der `plot`-Funktion mehrere Wertepaare auf einmal zu übergeben.

Command Window

```
>> x = linspace(0, 2*pi); ys = sin(x); yc = cos(x);  
fx >> plot(x, ys, x, yc)
```

Wir stellen fest, dass die beiden Kurven nicht in der selben Farbe gezeichnet wurden. Bei mehreren Argumenten im `plot`-Befehl durchläuft MATLAB die Farben automatisch.

121

Notizen

---

---

---

---

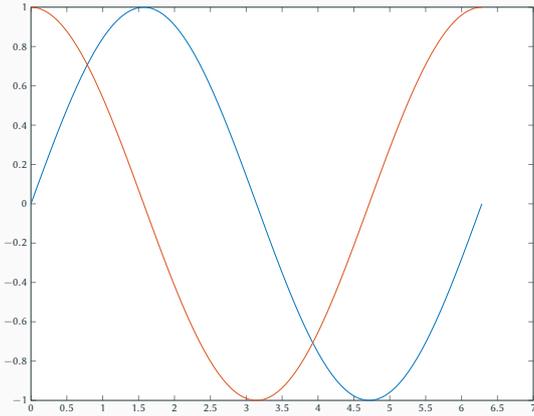
---

---

---

---

## Abbildungen



122

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Legenden

Um mehrere Kurven unterscheiden zu können, ist der Einsatz einer Legende hilfreich. Außerdem können, wie zuvor, auch verschiedene Zeichenstile an die `plot`-Funktion übergeben werden.

### Command Window

```
>> x = linspace(0, 2*pi); ys = sin(x); yc = cos(x);  
>> plot(x, ys, 'r-', x, yc, 'b+')  
fx >> legend('Sinus', 'Kosinus')
```

123

Notizen

---

---

---

---

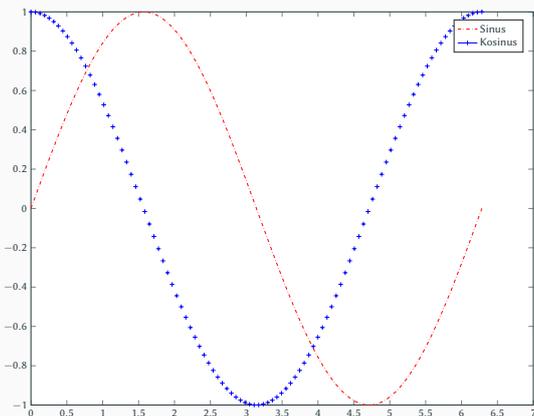
---

---

---

---

## Abbildungen



124

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Mehrere Plots auf einmal II

Eine Alternative zu der gerade erläuterten `plot`-Funktion bietet die Funktion `hold`.

Ähnlich wie die Funktion `grid` wechselt jeder Aufruf von `hold` zwischen der Möglichkeit, Kurven hinzufügen zu können und einem Neuzeichnen.

Mit `hold on` wird das aktuelle Diagramm festgehalten und jeder Plotbefehl fügt neue Kurven hinzu.

`hold off` stellt den ursprünglichen Zustand wieder her.

125

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

Command Window

```
>> clf
>> plot(x, ys, 'r-')
>> hold on
>> plot(x, yc, 'b+')
>> legend('Sinus', 'Kosinus')
>> hold off
```

fx

126

Notizen

---

---

---

---

---

---

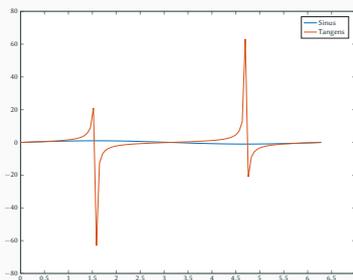
---

---

## Abbildungen

### Mehrere Plots auf einmal II

Wie wir gesehen haben, teilen sich die Plots die Achsen. Das kann nachteilig sein, wenn die Größenordnungen der beiden Kurven sehr verschieden sind. Wenn wir z. B. den Sinus und den Tangens in ein Grafikenfenster plotten, so ergibt sich das folgende Bild.



127

Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Subplots

Mit dem Befehl `subplot` wird das Grafikenfenster gemäß der Syntax

```
subplot(p, q, pos)6
```

in `p` Zeilen und `q` Spalten unterteilt.

Das Argument `pos` gibt an, auf welches Diagramm sich der nachfolgende `plot`-Befehl bezieht.

### Command Window

```
>> x = linspace(0, 2*pi);  
>> subplot(221), plot(x, sin(x), title('Sinus'))  
>> subplot(222), plot(x, cos(x), title('Kosinus'))  
>> subplot(223), plot(x, tan(x), title('Tangens'))  
fx >> subplot(224), plot(x, cot(x), title('Kotangens'))
```

<sup>6</sup>Die Kommata zwischen den Argumenten können auch weggelassen werden.

Notizen

---

---

---

---

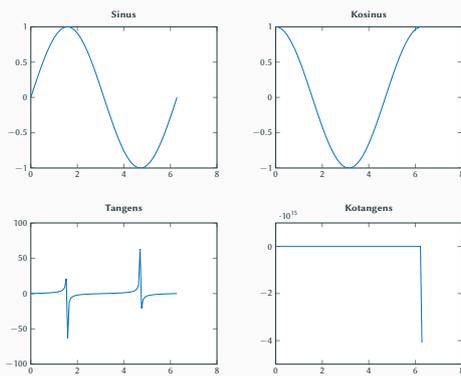
---

---

---

---

## Abbildungen



Notizen

---

---

---

---

---

---

---

---

## Abbildungen

### Grafiken drucken und exportieren

Die erzeugten Grafiken wollen wir auch ausdrucken oder für eine spätere Verwendung abspeichern. Am komfortabelsten geschieht dies direkt über das Datei-Menü des Grafikenfensters.

Möchte man Grafiken jedoch automatisiert abspeichern, so ist dies mit dem Befehl `print` möglich. Möchte man beispielsweise den Inhalt des Grafikenfensters mit der Nummer 2 abspeichern passiert dies über

```
print('-f2', 'Dateiname', '-dpng').
```

Als Dateiformate stehen gängige Bitmapformate (BMP, JPEG, PCX, PNG, TIFF) und Vektorformate (EMF, EPS, ILL, PDF) zur Verfügung.

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### MATLAB-Skripte

Bisher haben wir uns auf kurze „Programme“ beschränkt, die wir über das Kommandofenster geschrieben und ausgeführt haben. Wenn jedoch viele Kommandos hintereinander benötigt werden, ist dies jedoch sehr aufwändig.

Als Lösung bietet MATLAB sogenannte m-Files: Das sind Textdateien, in denen man MATLAB-Befehle speichern und ausführen kann. Als Editor kann hierfür jeder beliebige Texteditor genutzt werden, MATLAB bietet aber auch einen eigenen Editor, der einige Vorteile in diesem Zusammenhang hat.

Diesen öffnet man mit dem Befehl

```
edit
```

133

Notizen

---

---

---

---

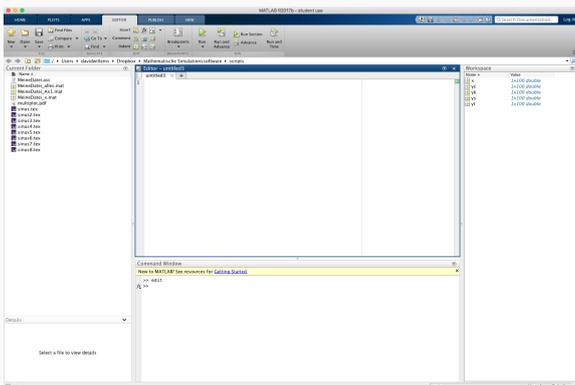
---

---

---

---

## Skripte & Funktionen



Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Der Editor

Die gewünschten Kommandos werden, wie zuvor im Kommandofenster, zeilenweise in die Datei geschrieben. Mehrere Kommandos in einer Zeile werden wieder über Kommata oder Semikolon getrennt.

Hilfreich sind **Kommentare**, die über ein % eingeleitet werden. Alles nach dem %-Zeichen wird später beim Aufruf ignoriert.<sup>7</sup>

```
Editor - /Users/davidwillems/Dropbox/Mathematische Simulationssoftware/scripts/plot_trigonometrisch.m
1 % Zerlege das Intervall [0,2pi] in 100 äquidistante Teile
2 x = linspace(0, 2*pi);
3
4 subplot(221), plot(x, sin(x)), title('Sinus') % Plote Sinus und co.
5 subplot(222), plot(x, cos(x)), title('Cosinus')
6 subplot(223), plot(x, tan(x)), title('Tangens')
7 subplot(224), plot(x, cot(x)), title('Kotangens')
```

### Achtung!

Alle Variablen, die in einer m-Datei angelegt werden, sind im globalen Variablenspeicher sichtbar. Sie überschreiben damit also auch eventuell bereits existierende Variablen. Auf die zuvor angelegten Variablen kann auch in der m-Datei zugegriffen werden.

135

<sup>7</sup> Kommentare können an einer beliebigen Stelle einer Zeile eingefügt werden.

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Kontrollstrukturen

Wie auch andere Hochsprachen kennt MATLAB verschiedene Kontrollstrukturen. Diese können wir nicht erschöpfend behandeln, sondern nur wesentliche Elemente kurz beschreiben.

Dazu starten wir mit...

### if-Statements

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

137

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

if expression1 ist eine logische Bedingung, bei deren Erfüllung<sup>8</sup> die unter

statements1 aufgeführten Kommandos ausgeführt werden.

elseif expression2 Wenn die Bedingung expression1 nicht erfüllt war, wird als nächstes die optionale Bedingung expression2 getestet. Ist sie erfüllt, so werden die unter

statements2 aufgeführten Kommandos ausgeführt. Weitere mit elseif eingeleiteten Bedingungen und Kommandos können folgen.

else Wenn bisher keine Bedingung erfüllt war, werden die Kommandos statements3 ausgeführt. Auch der Gebrauch von else ist optional.

end beendet das if-Statement.

<sup>8</sup> Alle Zahlenwerte ungleich Null werden als wahr (true) interpretiert.

138

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Command Window

```
if x > 10
    disp('x_ist_größer_als_10')
elseif x < 1
    disp('x_ist_kleiner_als_1')
else
    disp('x_liegt_zwischen_1_und_10')
end
```

139

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Deutlich seltener als eine `if`-Abfrage benötigt man eine `switch`-Anweisung, auch bedingte Verzweigung genannt.

### switch-Statements

```
switch switch_expr
    case case_expr1
        statements1
    case case_expr2
        statements2
    otherwise
        statements3
```

140

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

`switch switch_expr` wird zusammen mit `case case_expr` gemäß der logischen Bedingung `switch_expr == case_expr1` getestet und bei Erfüllung werden die unter `statements1` aufgeführten Kommandos ausgeführt.

Weitere mit `case` eingeleitete Bedingungen und Kommandos können folgen und werden der Reihe nach abgearbeitet.

`otherwise` Wenn bisher keine Bedingung erfüllt war, werden die Kommandos

`statements3` ausgeführt. Der Gebrauch von `otherwise` ist **optional**.

`end` beendet die Verzweigung.

141

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Command Window

```
switch x
  case 0
    disp('Der_Wert_von_x_ist_0')
    y = cos(x)
  case pi
    disp('Der_Wert_von_x_ist_pi')
    y = sin(x);
  otherwise
    disp('Der_Wert_von_x_ist_weder_0_noch_pi')
end
fx
```

142

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Auch wenn man sich in MATLAB durch die vektor-orientierte Arbeitsweise viele Schleifen sparen kann, so sind sie doch hin und wieder nützlich.

Die `for`-Schleife wiederholt Code gemäß der Vorgabe für den Schleifenzähler.

### for-Schleifen

```
for variable = initval:endval
  statements
end
```

for variable = initval:endval der Schleifenzähler variable wird von initval bis endval erhöht<sup>9</sup> und jedes mal werden die unter statements aufgeführten Kommandos ausgeführt.

end beendet die for-Schleife.

<sup>9</sup>Wie auch früher kann mit initval:schrittweite:endval eine Schrittweite angegeben werden.

143

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Command Window

```
for i=10:-2:0
  disp(['Countdown: ', num2str(i)])
end
fx
```

liefert die Ausgabe

```
Countdown: 10
Countdown: 8
Countdown: 6
Countdown: 4
Countdown: 2
Countdown: 0
```

144

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Die `while`-Schleife arbeitet ähnlich wie die `for`-Schleife; sie führt Kommandos solange aus, wie die Bedingung im Schleifenkopf erfüllt ist.

### while-Schleifen

```
while expression
    statements
end
```

`while expression` Solange die Bedingung `expression` erfüllt ist, werden die in `statements` aufgeführten Kommandos ausgeführt.  
`end` beendet die `while`-Schleife.

145

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Command Window

```
n = 10;
f = n;
while n > 1
    n = n-1;
    f = f*n;
end
fx disp(f)
```

Was gibt das obige Skript aus?

146

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Interaktion mit dem Benutzer

Wenn das Skript von Benutzereingaben abhängen soll, so ist die Funktion `input` das geeignete Instrument.

Mit `input` wird zur Eingabe einer Zahl während der Ausführung des Skripts aufgefordert.

### Command Window

```
>> x = input('Bitte geben Sie eine Zahl ein: \n')
Bitte geben Sie eine Zahl ein: 5

x =

    5
```

Benutzereingaben können eine nahezu unendlich große Fehlerquelle sein. Auf die Fehlerabfragen werden wir im Folgenden nur kurz eingehen.

147

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### AUFGABE 10 | Das erste Skript



- ▶ Erzeugen Sie eine leere Datei mit dem Namen `erstesskript.m`.
- ▶ Fordern Sie den Benutzer zur Eingabe einer Zahl zwischen  $\frac{\pi}{2}$  und  $2\pi$  auf.
- ▶ Begrenzen Sie die Eingabe mit den Funktionen `min` und `max` auf Werte zwischen  $\frac{\pi}{2}$  und  $2\pi$ ; verwenden Sie  $\frac{\pi}{2}$  bei einer Fehleingabe.
- ▶ Definieren Sie einen Vektor `x` von 0 bis zum Eingabewert in  $\frac{\pi}{12}$  Schritten. Berechnen Sie den Kosinus des Vektors `x` und weisen Sie das Ergebnis der Variablen `y` zu.
- ▶ Zeichnen Sie die Kurve und beschriften Sie das Diagramm.
- ▶ Schreiben Sie in die ersten Zeilen einige „hilfreiche“ Kommentare.

148

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Funktionen

Mit anonymen Funktionen haben wir bisher gelernt, einfache Funktionen schnell zu implementieren. Mit Skripten haben wir gelernt, wie wir umfangreiche eigene Programme zur Erweiterung des Funktionsumfangs von MATLAB schreiben können.

Kombinieren wie diese beiden Aspekte, so erhalten wir Funktionen.

Grundsätzlich ist die `m`-Datei einer Funktion genauso aufgebaut wie die eines Skripts. Der wesentliche Unterschied liegt in der ersten Zeile der Datei: hier muss die Funktion deklariert werden.

Dies geschieht über das Schlüsselwort `function` gemäß der Syntax

```
function [out1, out2, ...] = funcname(in1, in2, ...).
```

150

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Achtung!



Die MATLAB-Sprache sieht vor, dass der Name `funcname` einer Funktion auch gleichzeitig der Dateiname der Funktion ist.

### Funktionen

Ähnlich wie bisherige Statements wird eine Funktion durch das Schlüsselwort `end` beendet.

Der Funktion werden die Argumente `in1, in2, ...` übergeben. Optional kann die Funktion auch einen Vektor mit Ausgabewerten `out1` etc. zurückgeben.

### Achtung!



Zu beachten ist, dass alle Variablen innerhalb einer Funktion auch nur dort sichtbar sind.

151

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

malzwei.m

```
function y = malzwei(x)
%MALZWEI Verdopplung
%
% y = malzwei(x) verdoppelt das Argument x

y = 2*x;
end
```

Command Window

```
>> y = malzwei(5)

y =
fx 10
```

152

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Funktionen II

Das vorgehende Beispiel ist natürlich für das, was die Funktion tut, sehr umständlich. Schauen wir uns also ein komplexeres Beispiel aus der Analysis an.

#### Algorithm 1: Bisektionsverfahren

**Input** : Ein Intervall  $[a, b]$ , eine auf  $[a, b]$  stetige Funktion  $f$ , ein Toleranzwert  $\epsilon > 0$ .

**Output**: Ein Näherungswert  $x$  für eine Nullstelle von  $f$ .

```
1 if f(a)f(b) > 0 then
2   Break
3 while (b - a)/2 > epsilon do
4   x = (a + b)/2
5   if f(x) = 0 then
6     Break
7   if f(a)f(x) < 0 then
8     b = x
9   else
10    a = x
11 return x
```

153

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

```
function x = mybisect(f, a, b, TOL)

if f(a)*f(b) > 0
disp('Bedingung_für_den_ZWS_nicht_erfüllt!')
return
end

while (b-a)/2 > TOL
x = (a+b)/2;

if f(x) == 0
disp('Nullstelle_gefunden!')
break
end

if f(a)*f(x) < 0
b = x;
else
a = x;
end

end
end
```

154

Notizen

---

---

---

---

---

---

---

---

**AUFGABE 11 | Eigene Funktionen I**

1. Schreiben Sie eine MATLAB-Funktion die ein  $n \in \mathbb{N}$  Als Eingabe erwartet. Ferner sei  $m \gg 1$  eine große Zahl. Erzeugen Sie  $n$  zufällige  $m \times m$ -Matrizen und zählen sie, wie viele dieser  $m$  Matrizen invertierbar sind.
2. Schreiben Sie eine Funktion  
 $y = \text{myableitung}(f, x0, h)$   
die die Ableitung einer (anonymen) Funktion  $f$  an einer Stelle  $x0$  mit Genauigkeit  $h$  approximiert.

Notizen

---

---

---

---

---

---

---

---

**AUFGABE 12 | Eigene Funktionen II**

Schreiben Sie ein MATLAB-Skript, welches das Newtonverfahren aus Algorithmus 2 für die Funktion  $f: \mathbb{R} \rightarrow \mathbb{R}, f(x) := -(x-2) \cdot \exp(x)$  implementiert.

**Algorithm 2:** (Einfaches) Newtonverfahren

**Input** : Ein Startpunkt  $x^0$ , eine stetige Funktion  $f$  und deren Ableitung  $f'$ , eine hinreichend große Zahl  $n$

**Output:** Ein Näherungswert  $x$  für eine Nullstelle von  $f$ .

```
1 for k = 1, ..., n do
2    $x^k = x^{k-1} - \frac{f(x^{k-1})}{f'(x^{k-1})}$ 
3 return x
```

Notizen

---

---

---

---

---

---

---

---

**Oft benötigte Befehle innerhalb von Funktionen**

Wir wollen uns zum Abschluss des Kurses noch kurz mit dem Abfangen von Fehlern in Skripten und Funktionen beschäftigen. Mit der Funktion `isempty` überprüft man, ob eine Variable leer ist<sup>10</sup>.

Mit einer bedingten Zuweisung können so Fehler während der Laufzeit abgefangen werden.

<sup>10</sup>Die Variable wurde also angelegt, hat aber keinen Inhalt; z. B. `x = []`.

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Beispiel zu isempty

```
x = []  
  
if isempty(x)  
    x = 1  
end
```

### Command Window

```
x =  
[]  
  
x =  
1
```

158

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### Oft benötigte Befehle innerhalb von Funktionen

Um während der Laufzeit einer Funktion zu überprüfen, ob alle Ein- und Ausgabeparameter übergeben wurden, benutzt man die Befehle

`nargin` bzw. `nargout`

benutzen.

Tritt hier ein Fehler auf, so benötigen wir einen Mechanismus, die Funktion mit einer erklärenden Fehlermeldung vorzeitig abbrechen zu können.

Die benötigte Funktion hierzu heißt `error`. Mit ihr kann man Fehlermeldungen in Skripten und Funktionen gemäß

`error('Fehlermeldung')`

einbinden.

159

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

### malzwei.m

```
function y = malzwei(x)  
%MALZWEI Verdopplung  
%  
% y = malzwei(x) verdoppelt das Argument x  
  
if nargin == 0  
    error('Nicht_genug_Eingabeargumente!')  
end  
  
y = 2*x;  
end
```

### Command Window

```
>> malzwei()  
Error using malzwei (line 7)  
Nicht genug Eingabeargumente!
```

160

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

Der folgende Aufruf hingegen löst keine Fehlermeldung aus, da das Eingabeargument `x` zwar übergeben wird, aber leer ist.

```
Command Window
>> malzwei([])

ans =
fx  []
```

Die Existenz einer Variablen – nicht zu verwechseln mit einer existierenden, aber leeren Variablen – wird mit der Funktion `exist` überprüft<sup>11</sup>.

<sup>11</sup>`exist` prüft mehr als die bloße Existenz einer Variablen. Weitere Informationen liefert die MATLAB Dokumentation

Notizen

---

---

---

---

---

---

---

---

## Skripte & Funktionen

```
Command Window
>> x = 5; exist x

ans =
1

>> x = []; exist x

ans =
1

>> exist malzwei

ans =
fx  2
```

Notizen

---

---

---

---

---

---

---

---

Notizen

---

---

---

---

---

---

---

---