

MATLAB Einführung

Christopher Basting

Numerische Mathematik für Physiker und Ingenieure, 04.05.2017

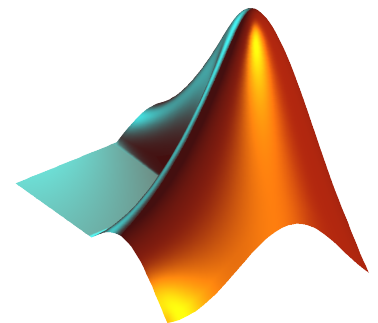
Numerisches Softwarewerkzeug MATLAB®

Eigenschaften

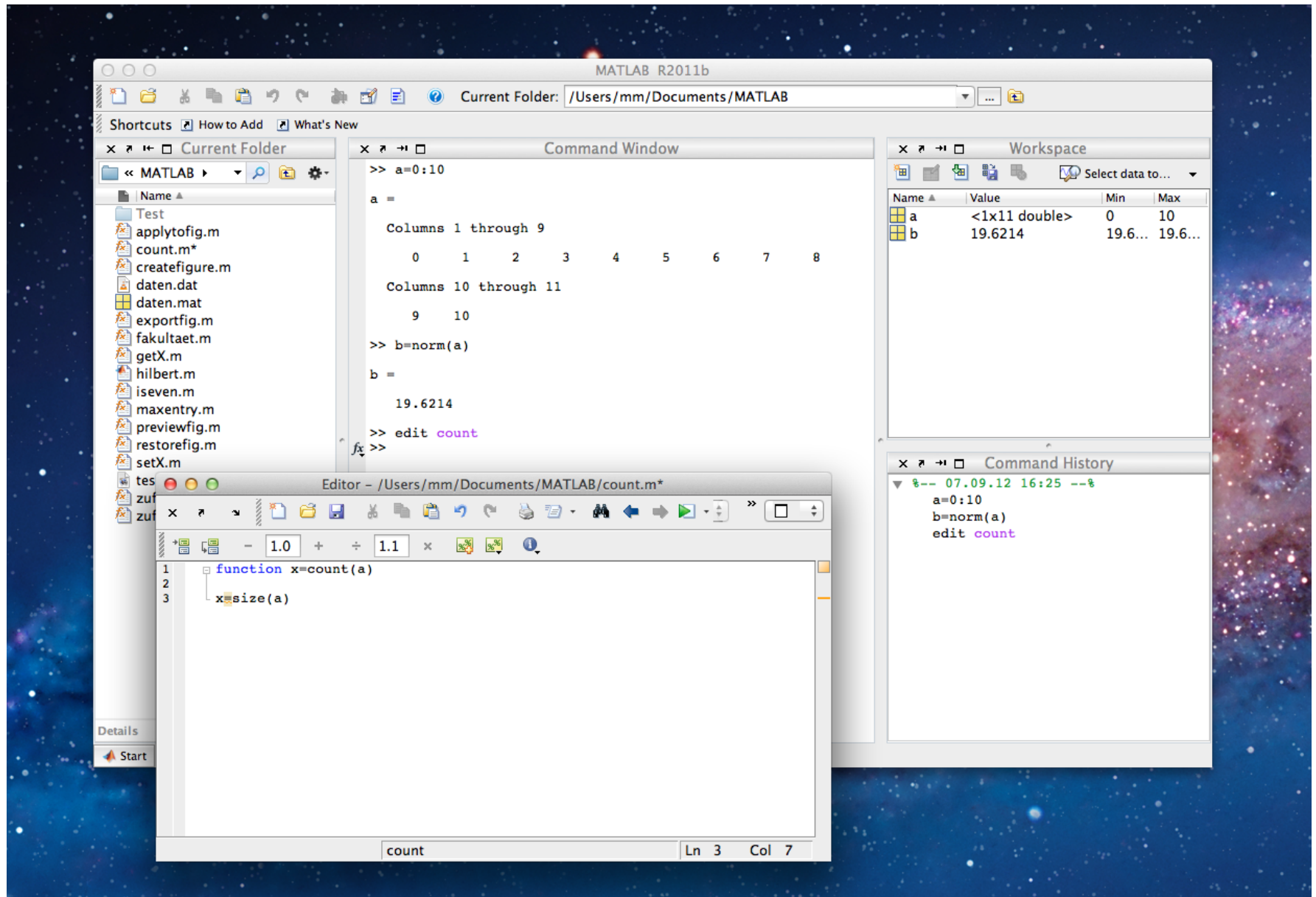
- Etablierte kommerzielle Software für numerische Berechnungen
- Unterstützung aller gängigen Betriebssysteme (M-files sind portabel)
- Interaktive, benutzerfreundliche Programmierumgebung (in Java)
- Sehr umfangreiche Bibliothek von mathematischen Funktionen
- Erweiterbar durch (kommerzielle) Toolboxes und kostenlose M-files
- Schnittstellen zu Hochsprachen wie C und Fortran sind vorhanden

Bezugsquelle

- Kommerzieller Vertrieb durch die Firma MathWorks®
<http://www.mathworks.de/>
- Studentenversion kostet ca. 100 Euro



MATLAB®Umgebung



Open-Source Alternative GNU Octave

Eigenschaften

- Weitgehend sprachkompatibel zu MATLAB® (M-files lauffähig)
- Funktionsumfang entspricht der Basisversion von MATLAB®
- Unterstützung aller gängigen Betriebssysteme
- Erweiterbar durch Packages <http://www.gnu.org/software/octave/>
- Benutzerfreundliche Programmierumgebung in Entwicklung
- Kostenlose Nutzung auf privaten Laptops möglich

Bezugsquelle

- Quellcode und z.T. Binärpakete frei verfügbar
<http://www.gnu.org/software/octave/>



MATLAB[®] vs. GNU Octave

- Grundfunktionalität beider Softwarepakete vergleichbar
- GNU Octave reicht zum Bearbeiten der Übungsaufgaben in diesem Kurs und in den Vorlesungen Numerik 1-2 aus
- MATLAB[®] bietet mehr Komfort beim Programmieren und Debuggen speziell von größeren Programmen (z.B. mit GUI)
- Funktionsumfang von MATLAB[®] lässt sich durch kommerzielle Toolboxes deutlich erweitern (Zielgruppe: Ingenieursanwendungen)
- MATLAB[®] ist insb. in den Ingenieurwissenschaften etabliert
- Eine Übersicht der Unterschiede liefert die Octave FAQ Abschnitt 10

Literatur zu MATLAB®/GNU Octave

- D.J. Higham, N.J. Higham, MATLAB Guide, SIAM, 2005.
- W. Schweizer, MATLAB kompakt, Oldenbourg-Verlag, 2009.
- G. Gramlich, W. Werner, Numerische Mathematik mit MATLAB: Eine Einführung für Naturwissenschaftler und Ingenieure. Dpunkt-Verlag, 2000.
- C. Überhuber, S. Katzenbeisser, D. Praetorius, MATLAB 7: Eine Einführung. Springer-Verlag, 2005.
- A. Quarteroni, F. Saleri, Scientific Computing with MATLAB and Octave, Springer, 2006.

Internet

- alternative Vorlesungsskripte und Kurzanleitungen findet man leicht mit gängigen Suchmaschinen
- Vorlesungsskript zum COP-Kurs: https://www.mathematik.tu-dortmund.de/sites/cop-kurs-ws14-15/download/vorlesung_komplett.pdf

Überblick: Crashkurs

- Integriertes Hilfesystem
- Variablen, Vektoren und Matrizen
- Mathematische Operationen
- Ein- und Ausgabe von Daten
- M-Dateien: Skripte und Funktionen
- Vergleichsoperatoren und -funktionen
- Logische Operatoren
- Verzweigungen und Schleifen

LIVE Demo:

M-Datei verfügbar → 

M-file: `intro.m`

Zusammenfassung: *Hilfesystem*

- `help <Thema>` gibt Hilfetexte auf dem Bildschirm aus
- `lookfor <Thema>` durchsucht Hilfetexte nach Stichwort
- `doc <Thema>` öffnet grafisches Hilfesystem/Browser
- `demo <Thema>` öffnet interaktive MATLAB Demos

Nützliche Hilfethemen

- `general` Generelle Befehle (`who`, `clear`, ...)
- `ops` Operationen (`+`, `-`, `*`, `/`, `^`, `[]`, ...)
- `elfunc` Mathematische Funktionen (`min`, `max`, `sqrt`, ...)
- `elmat` Matrix Funktionen
- `lang` Programmierung

Variablen

- Es wird zwischen Groß- und Kleinschreibung unterschieden
- Variablen werden dynamisch erzeugt und sind veränderbar
- Wenn keine Variable angegeben wird, dann wird das zuletzt berechnete Ergebnis in `ans` (=answer) gespeichert
- Eine Übersicht über alle Variablen liefern `who` bzw. `whos`
- Variablen lassen sich mittels `clear <Variablenname>` löschen;
alle Variablen werden mit `clear` oder `clear all` gelöscht
- Die Bildschirmausgabe lässt sich mittels Semikolon unterdrücken

Vektoren

- Zeilenvektoren können direkt erzeugt werden

```
>> a=[1 3 5 7 9 11 13 15 17 19]
```

```
a =
```

```
1 3 5 7 9 11 13 15 17 19
```

Vektoren

- Zeilenvektoren können direkt erzeugt werden

```
>> a=[1 3 5 7 9 11 13 15 17 19]
```

```
a =
```

```
1 3 5 7 9 11 13 15 17 19
```

oder mittels Doppelpunkt-Operator

```
>> a=1:2:20
```

```
a =
```

```
1 3 5 7 9 11 13 15 17 19
```

Vektoren

- Zeilenvektoren können direkt erzeugt werden

```
>> a=[1 3 5 7 9 11 13 15 17 19]
```

```
a =
```

```
1 3 5 7 9 11 13 15 17 19
```

oder mittels Doppelpunkt-Operator

```
>> a=1:2:20
```

```
a =
```

```
1 3 5 7 9 11 13 15 17 19
```

- Ohne Angabe der Schrittweite wird automatisch 1 angenommen

```
>> a=1:20
```

```
a =
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Vektoren

- Negative Schrittweiten sind zulässig, wenn gilt: Startwert $>$ Endwert

```
>> a=20:-2:1
```

```
a =
```

```
20 18 16 14 12 10 8 6 4 2
```

Vektoren

- Negative Schrittweiten sind zulässig, wenn gilt: Startwert > Endwert

```
>> a=20:-2:1
```

```
a =
```

```
20 18 16 14 12 10 8 6 4 2
```

- Anfangs-, Endwert und Schrittweite können nichtganzzahlig sein

```
>> a=0.3:0.1:0.7
```

```
a =
```

```
0.3000    0.4000    0.5000    0.6000    0.7000
```

```
>> b=0:pi/2:2*pi
```

pi ist als π vordefiniert

```
b =
```

```
0 1.5708    3.1416    4.7124    6.2832
```

Vektoren

- Spaltenvektoren können direkt mittels ';' erzeugt werden oder durch Transponieren des entsprechenden Zeilenvektors

```
>> a=[1; 2; 3; 4; 5]
```

a =

1
2
3
4
5

```
>> b=transpose(1:2:10)
```

b =

1
3
5
7
9

Vektoren

- Spaltenvektoren können direkt mittels `' ; '` erzeugt werden oder durch Transponieren des entsprechenden Zeilenvektors

```
>> a=[1; 2; 3; 4; 5]
```

a =

1
2
3
4
5

```
>> b=transpose(1:2:10)
```

b =

1
3
5
7
9

- Kurzform ist `a.'` und `b'` entspricht komplex konjugiert transponiert

```
>> a=(1:3).'
```

a =

1
2
3

```
>> b=[1+i 2+i 3+i]'
```

b =

1.0000 - 1.0000i
2.0000 - 1.0000i
3.0000 - 1.0000i

Vektoradressierung

- Einen einzelnen Vektoreintrag adressiert/verändert man mit

```
>> a=1:5; a(3)=pi
```

```
a =
```

```
1.0000    2.0000    3.1416    4.0000    5.0000
```

Vektoradressierung

- Einen einzelnen Vektoreintrag adressiert/verändert man mit

```
>> a=1:5; a(3)=pi
```

```
a =
```

```
1.0000    2.0000    3.1416    4.0000    5.0000
```

- Auf den letzten Vektoreintrag greift man mit end zu

```
>> b=1:5; b(end)=1
```

```
b =
```

```
1 2 3 4 1
```

Vektoradressierung

- Einen einzelnen Vektoreintrag adressiert/verändert man mit

```
>> a=1:5; a(3)=pi
```

```
a =
```

```
1.0000    2.0000    3.1416    4.0000    5.0000
```

- Auf den letzten Vektoreintrag greift man mit end zu

```
>> b=1:5; b(end)=1
```

```
b =
```

```
1 2 3 4 1
```

- Teilvektoren können direkt oder mittels ':' adressiert werden

```
>> a(2:4)=42
```

```
a =
```

```
1 42 42 42 5
```

```
>> b([1 3 end])=42
```

```
b =
```

```
42 2 42 4 42
```

Vektoradressierung

- Vektoren können zu größeren Vektoren zusammengesetzt werden

```
>> a=1:5; b=6:10; c=[a, b]
```

```
c =
```

```
1 2 3 4 5 6 7 8 9 10
```

Vektoradressierung

- Vektoren können zu größeren Vektoren zusammengesetzt werden

```
>> a=1:5; b=6:10; c=[a, b]
```

```
c =
```

```
1 2 3 4 5 6 7 8 9 10
```

Dabei kann das Komma bei Zeilenvektoren entfallen.

Vektoradressierung

- Vektoren können zu größeren Vektoren zusammengesetzt werden

```
>> a=1:5; b=6:10; c=[a, b]
```

```
c =
```

```
1 2 3 4 5 6 7 8 9 10
```

Dabei kann das Komma bei Zeilenvektoren entfallen.

- Vektoreinträge/Teilvektoren können mittels ' [] ' entfernt werden

```
>> c(3:8)=[]
```

```
c =
```

```
1 2 9 10 ← end entspricht jetzt dem Eintrag 4
```

Vektoroperationen

- min bzw. max berechnet kleinsten bzw. größten Werte eines Vektors

```
>> a=[2 5 4]; m=min(a)
```

```
m =
```

```
2
```

```
>> m=max(a)
```

```
m =
```

```
5
```


Vektoroperationen

- min bzw. max berechnet kleinsten bzw. größten Werte eines Vektors

```
>> a=[2 5 4]; m=min(a)
```

```
m =
```

```
2
```

```
>> m=max(a)
```

```
m =
```

```
5
```

- Die Position des kleinsten bzw. größten Eintrags liefert

```
>> [m i]=min(a)
```

```
m =      i =
```

```
2
```

```
1
```

```
>> [m i]=max(a)
```

```
m =      i =
```

```
5
```

```
2
```

Vektoroperationen

- min bzw. max berechnet kleinsten bzw. größten Werte eines Vektors

```
>> a=[2 5 4]; m=min(a)
```

```
m =
```

```
2
```

```
>> m=max(a)
```

```
m =
```

```
5
```

- Die Position des kleinsten bzw. größten Eintrags liefert

```
>> [m i]=min(a)
```

```
m =      i =
```

```
2
```

```
1
```

```
>> [m i]=max(a)
```

```
m =      i =
```

```
5
```

```
2
```

- sum berechnet die Summe, prod das Produkt der Vektoreinträge

```
>> sum(a)
```

```
ans =
```

```
11
```

$$\hat{=} \sum_{k=1}^n a_k$$

```
>> prod(a)
```

```
ans =
```

```
40
```

$$\hat{=} \prod_{k=1}^n a_k$$

Vektoroperationen

- dot berechnet das Skalarprodukt zweier Vektoren $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$

```
>> a=1:5; b=6:10; d=dot(a,b)
```

```
d =
```

```
130
```

$$\hat{=} (\mathbf{a}, \mathbf{b}) = \sum_{k=1}^n a_k b_k$$

Vektoroperationen

- `dot` berechnet das Skalarprodukt zweier Vektoren $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$

```
>> a=1:5; b=6:10; d=dot(a,b)
```

```
d =
```

```
130
```

$$\hat{=} (\mathbf{a}, \mathbf{b}) = \sum_{k=1}^n a_k b_k$$

- `norm` berechnet die Euklidische Norm ($p = 2$) eines Vektors $\mathbf{a} \in \mathbb{R}^n$

```
>> a=1:10; n=norm(a)
```

```
n =
```

```
19.6214
```

$$\hat{=} \|\mathbf{a}\| = \sqrt{\sum_{k=1}^n a_k^2}$$

Vektoroperationen

- dot berechnet das Skalarprodukt zweier Vektoren $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$

```
>> a=1:5; b=6:10; d=dot(a,b)
```

```
d =
```

```
130
```

$$\hat{=} (\mathbf{a}, \mathbf{b}) = \sum_{k=1}^n a_k b_k$$

- norm berechnet die Euklidische Norm ($p = 2$) eines Vektors $\mathbf{a} \in \mathbb{R}^n$

```
>> a=1:10; n=norm(a)
```

```
n =
```

```
19.6214
```

$$\hat{=} \|\mathbf{a}\| = \sqrt{\sum_{k=1}^n a_k^2}$$

- weitere Normen können mittels norm(a,p) berechnet werden

$p = 1$	Betragssummennorm	$\ \mathbf{a}\ _1 = \sum_{k=1}^n a_k $
$p = \infty$	Maximumsnorm	$\ \mathbf{a}\ _\infty = \max_{k=1, \dots, n} a_k $
$p \in \mathbb{R}$	p -Norm, $p \geq 1$	$\ \mathbf{a}\ _p = (\sum_{k=1}^n a_k ^p)^{1/p}$

MATLAB = MATrix LABoratory

- Eine $m \times n$ Matrix ist in MATLAB ein zweidimensionales Array mit m Zeilen (engl. *rows*) und n Spalten (engl. *columns*).

MATLAB = MATrix LABoratory

- Eine $m \times n$ Matrix ist in MATLAB ein zweidimensionales Array mit m Zeilen (engl. *rows*) und n Spalten (engl. *columns*).

- 2×3 Matrix

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

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

```
A =
```

```
1 2 3
```

```
4 5 6
```

Zeilen werden durch Semikolon getrennt (vgl. Spaltenvektoren)

MATLAB = MATrix LABoratory

- Eine $m \times n$ Matrix ist in MATLAB ein zweidimensionales Array mit m Zeilen (engl. *rows*) und n Spalten (engl. *columns*).

- 2×3 Matrix

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

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

```
A =
```

```
1 2 3
```

```
4 5 6
```

Zeilen werden durch Semikolon getrennt (vgl. Spaltenvektoren)

- `>> whos A`

Name	Size	Bytes	Class	Attributes
A	2x3	48	double	

Matrixdimensionen

■ Dimension der Matrix A

```
>> s = size(A)
```

```
s =  
    2    3
```

```
>> m = size(A,1)
```

```
m =  
    2
```

```
>> [m n] = size(A)
```

```
m =        n =  
    2        3
```

```
>> n = size(A,2)
```

```
n =  
    3
```

Matrixdimensionen

■ Dimension der Matrix A

```
>> s = size(A)
```

```
s =  
    2    3
```

```
>> m = size(A,1)
```

```
m =  
    2
```

■ Größte Dimension der Matrix A

```
>> l = max(size(A))
```

```
l =  
    3
```

```
>> [m n] = size(A)
```

```
m =        n =  
    2        3
```

```
>> n = size(A,2)
```

```
n =  
    3
```

```
>> l = length(A)
```

```
l =  
    3
```

Spezielle Matrizen

■ Nullmatrix

```
>> N = zeros(2)
```

```
N =
```

```
0 0  
0 0
```

```
>> N = zeros(2,3)
```

```
N =
```

```
0 0 0  
0 0 0
```

Spezielle Matrizen

■ Nullmatrix

```
>> N = zeros(2)
```

```
N =
```

```
0  0  
0  0
```

```
>> N = zeros(2,3)
```

```
N =
```

```
0  0  0  
0  0  0
```

■ „Einsmatrix“

```
>> E = ones(2)
```

```
E =
```

```
1  1  
1  1
```

```
>> E = ones(2,3)
```

```
E =
```

```
1  1  1  
1  1  1
```

Spezielle Matrizen

■ Nullmatrix

```
>> N = zeros(2)
```

```
N =
```

```
0 0
0 0
```

■ „Einsmatrix“

```
>> E = ones(2)
```

```
E =
```

```
1 1
1 1
```

■ Einheitsmatrix

```
>> I = eye(2)
```

```
I =
```

```
1 0
0 1
```

```
>> N = zeros(2,3)
```

```
N =
```

```
0 0 0
0 0 0
```

```
>> E = ones(2,3)
```

```
E =
```

```
1 1 1
1 1 1
```

```
>> I = eye(2,3)
```

```
I =
```

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

Spezielle Matrizen

■ Nullmatrix

```
>> N = zeros(2)
```

```
N =
```

```
0 0
0 0
```

```
>> N = zeros(2,3)
```

```
N =
```

```
0 0 0
0 0 0
```

■ „Einsmatrix“

```
>> E = ones(2)
```

```
E =
```

```
1 1
1 1
```

```
>> E = ones(2,3)
```

```
E =
```

```
1 1 1
1 1 1
```

■ Einheitsmatrix

```
>> I = eye(2)
```

```
I =
```

```
1 0
0 1
```

```
>> I = eye(2,3)
```

```
I =
```

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

■ Weitere spezielle Matrizen hilb, rand, magic, ... → doc elmat

Matrixoperationen

- Sämtliche Operationen wie $+$, $-$, $*$ und $^$ werden unterstützt

Matrixoperationen

- Sämtliche Operationen wie +, -, * und ^ werden unterstützt
- Vektoroperationen lassen sich (automatisch) spaltenweise anwenden

```
>> A=magic(3); [m i]=min(A)
```

```
m =
```

```
    3    1    2
```

```
i =
```

```
    2    1    3
```

$$A = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

Matrixoperationen

- Sämtliche Operationen wie +, -, * und ^ werden unterstützt
- Vektoroperationen lassen sich (automatisch) spaltenweise anwenden

```
>> A=magic(3); [m i]=min(A)
```

```
m =
```

```
    3    1    2
```

```
i =
```

```
    2    1    3
```

$$A = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

- oder gezielt spalten- bzw. zeilenweise durch Angabe der Dimension

```
>> p=prod(A,1)
```

```
p =
```

```
    96    45    84
```

```
>> p=prod(A,2)
```

```
p =
```

```
    48
```

```
   105
```

```
    72
```

Matrixoperationen

- `norm(A,p)` berechnet die p -Norm einer Matrix A
zulässige Werte sind $p=1, 2, \infty$ und 'fro' (Frobeniusnorm)

Matrixoperationen

- `norm(A,p)` berechnet die p -Norm einer Matrix A
zulässige Werte sind $p=1, 2, \text{inf}$ und `'fro'` (Frobeniusnorm)

- `inv` berechnet die Inverse A^{-1} einer regulären Matrix A

```
>> A=hilb(2), B=inv(A), C=A*B
```

A =	B =	C =
1.0000 0.5000	4.0000 -6.0000	1 0
0.5000 0.3333	-6.0000 12.0000	0 1

Matrixoperationen

- `norm(A,p)` berechnet die p -Norm einer Matrix A
zulässige Werte sind $p=1, 2, \text{inf}$ und `'fro'` (Frobeniusnorm)

- `inv` berechnet die Inverse A^{-1} einer regulären Matrix A

```
>> A=hilb(2), B=inv(A), C=A*B
```

A =	B =	C =
1.0000 0.5000	4.0000 -6.0000	1 0
0.5000 0.3333	-6.0000 12.0000	0 1

- `det` und `rank` bestimmen Determinante und Rang einer Matrix A

```
>> d=det(A)
```

```
d =  
0.0833
```

```
>> r=rank(A)
```

```
r =  
2
```

Matrixoperationen

- eig berechnet die Eigenwerte einer Matrix $A \in \mathbb{R}^{n \times n}$

```
>> A=hilb(2); l=eig(A)
```

```
l =
```

```
    0.0657
```

```
    1.2676
```

$$A\mathbf{x} = \lambda\mathbf{x}, \quad \lambda \in \mathbb{C}, \mathbf{x} \in \mathbb{C}^n \setminus \{0\}$$

Matrixoperationen

- eig berechnet die Eigenwerte einer Matrix $A \in \mathbb{R}^{n \times n}$

```
>> A=hilb(2); l=eig(A)
```

```
l =
```

```
    0.0657
```

```
    1.2676
```

$$A\mathbf{x} = \lambda\mathbf{x}, \quad \lambda \in \mathbb{C}, \mathbf{x} \in \mathbb{C}^n \setminus \{0\}$$

- Es können auch Eigenwerte und Eigenvektoren berechnet werden

```
>> [R D]=eig(A)
```

```
R =
```

```
    0.4719    -0.8817
```

```
   -0.8817    -0.4719
```

```
D =
```

```
    0.0657         0
```

```
         0    1.2676
```

Matrixoperationen

- eig berechnet die Eigenwerte einer Matrix $A \in \mathbb{R}^{n \times n}$

```
>> A=hilb(2); l=eig(A)
```

```
l =
```

```
0.0657
```

```
1.2676
```

$$A\mathbf{x} = \lambda\mathbf{x}, \quad \lambda \in \mathbb{C}, \mathbf{x} \in \mathbb{C}^n \setminus \{0\}$$

- Es können auch Eigenwerte und Eigenvektoren berechnet werden

```
>> [R D]=eig(A)
```

```
R =
```

```
0.4719    -0.8817
```

```
-0.8817    -0.4719
```

```
D =
```

```
0.0657         0
```

```
0    1.2676
```

Beachte: Bei der Eigenwertberechnung können Rundungsfehler auftreten.

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n}, \quad \mathbf{b} \in \mathbb{R}^n$$

können mittels Linksddivision (`mldivide`)

$$\mathbf{x} = A \backslash \mathbf{b} \quad \text{oder direkt} \quad \mathbf{x} = \text{inv}(A) * \mathbf{b}$$

gelöst werden (falls Matrix A invertierbar ist)

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n}, \quad \mathbf{b} \in \mathbb{R}^n$$

können mittels Linksddivision (`mldivide`)

$$\mathbf{x} = A \backslash \mathbf{b} \quad \text{oder direkt} \quad \mathbf{x} = \text{inv}(A) * \mathbf{b}$$

gelöst werden (falls Matrix A invertierbar ist)

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- `>> A=[1 2; 2 1]; b=[1; 2];`

`>> x=A\b`

`x =`

1

0

`>> x=inv(A)*b`

`x =`

1

0

Lineare Gleichungssysteme

- Lineare Gleichungssysteme der Form

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n \times n}, \quad \mathbf{b} \in \mathbb{R}^n$$

können mittels Linksddivision (`mldivide`)

$$\mathbf{x} = A \backslash \mathbf{b} \quad \text{oder direkt} \quad \mathbf{x} = \text{inv}(A) * \mathbf{b}$$

gelöst werden (falls Matrix A invertierbar ist)

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- `>> A=[1 2; 2 1]; b=[1; 2];`

`>> x=A\b`

`x =`

1

0

`>> x=inv(A)*b`

`x =`

1

0

- Linksddivision ist i.d.R. numerisch stabiler

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.“ vorangestellt wird

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.“ vorangestellt wird

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

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.“ vorangestellt wird

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

Produkt der Matrixeinträge

```
>> A.*B
```

```
ans =
```

```
1    4  
3    8
```

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.“ vorangestellt wird

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

Produkt der Matrixeinträge

```
>> A.*B
```

```
ans =
```

```
1    4
3    8
```

Division vom Typ „ b_{ij}/a_{ij} “

```
>> B./A
```

```
ans =
```

```
1.0000    1.0000
0.3333    0.5000
```

Elementweise Operationsausführung

- Operatoren $*$, $/$, \backslash , $^$ werden elementweise auf jeden Eintrag des Arrays angewendet, wenn ein „.“ vorangestellt wird

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

Produkt der Matriceinträge

```
>> A.*B
```

```
ans =
```

```
1    4
3    8
```

Division vom Typ „ b_{ij}/a_{ij} “

```
>> B./A
```

```
ans =
```

```
1.0000    1.0000
0.3333    0.5000
```

Beachte: A^k meint die k -fache Matrizenmultiplikation $A * \dots * A$ während $A.^k$ jeden Eintrag der Matrix exponenziert.

Teilmatrizen

- Matrizen können aus einzelnen Teilmatrizen aufgebaut werden

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

A =

```
1 2
3 4
```

```
>> B = [A zeros(2); ...
```

```
ones(2) eye(2)]
```

B =

```
1 2 0 0
3 4 0 0
1 1 1 0
1 1 0 1
```


Teilmatrizen

- Matrizen können aus einzelnen Teilmatrizen aufgebaut werden

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

A =

```
1 2
3 4
```

```
>> B = [A zeros(2); ...
        ones(2) eye(2)]
```

B =

```
1 2 0 0
3 4 0 0
1 1 1 0
1 1 0 1
```

- Blockmatrizen können mittels `repmat(A,m,n)` erzeugt werden

```
>> C = repmat(eye(2),2,3)
```

C =

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

Teilmatrizen

- Matrizen können aus einzelnen Teilmatrizen aufgebaut werden

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

A =

```
1 2
3 4
```

```
>> B = [A zeros(2); ...
        ones(2) eye(2)]
```

B =

```
1 2 0 0
3 4 0 0
1 1 1 0
1 1 0 1
```

- Blockmatrizen können mittels `repmat(A,m,n)` erzeugt werden

```
>> C = repmat(eye(2),2,3)
```

C =

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

```
>> D = repmat([1; 2],2)
```

D =

```
1 1
2 2
1 1
2 2
```

Diagonalmatrizen

- Diagonaleil einer Matrix kann mittels $\text{diag}(A,k)$ bestimmt werden

Diagonalmatrizen

- Diagonalelement einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

```
A =
```

```
1  4  7  
2  5  8  
3  6  9
```

Diagonalmatrizen

- Diagonalelement einer Matrix kann mittels $\text{diag}(A,k)$ bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

1	4	7
2	5	8
3	6	9

```
>> B=diag(A)
```

B =

1
5
9

Diagonalmatrizen

- Diagonalelement einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

1	4	7
2	5	8
3	6	9

```
>> B=diag(A)
```

B =

1
5
9

```
>> C=diag(A,1)
```

C =

4
8

Diagonalmatrizen

- Diagonaleil einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

1	4	7
2	5	8
3	6	9

```
>> B=diag(A)
```

B =

1
5
9

```
>> C=diag(A,1)
```

C =

4
8

- Diagonalmatrizen können mittels `diag(v,k)` erzeugt werden

Diagonalmatrizen

- Diagonalelement einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

```
1 4 7
2 5 8
3 6 9
```

```
>> B=diag(A)
```

B =

```
1
5
9
```

```
>> C=diag(A,1)
```

C =

```
4
8
```

- Diagonalmatrizen können mittels `diag(v,k)` erzeugt werden

```
>> D=diag(1:3)
```

D =

```
1 0 0
0 2 0
0 0 3
```


Diagonalmatrizen

- Diagonaleil einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

```
1 4 7
2 5 8
3 6 9
```

```
>> B=diag(A)
```

B =

```
1
5
9
```

```
>> C=diag(A,1)
```

C =

```
4
8
```

- Diagonalmatrizen können mittels `diag(v,k)` erzeugt werden

```
>> D=diag(1:3)
```

D =

```
1 0 0
0 2 0
0 0 3
```

```
>> E=diag(1:2,-1)
```

E =

```
0 0 0
1 0 0
0 2 0
```

Diagonalmatrizen

- Diagonalelement einer Matrix kann mittels `diag(A,k)` bestimmt werden

```
>> A=[1 4 7; 2 5 8; 3 6 9]
```

A =

```
1 4 7
2 5 8
3 6 9
```

```
>> B=diag(A)
```

B =

```
1
5
9
```

```
>> C=diag(A,1)
```

C =

```
4
8
```

- Diagonalmatrizen können mittels `diag(v,k)` erzeugt werden

```
>> D=diag(1:3)
```

D =

```
1 0 0
0 2 0
0 0 3
```

```
>> E=diag(1:2,-1)
```

E =

```
0 0 0
1 0 0
0 2 0
```

```
>> F=diag(diag(A))
```

F =

```
1 0 0
0 5 0
0 0 9
```

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

```
>> L=tril(A)
```

```
L =
```

```

1    0    0
2    5    0
3    6    9
```

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

```
>> L=tril(A)
```

L =

```
1    0    0
2    5    0
3    6    9
```

```
>> U=triu(A)
```

U =

```
1    4    7
0    5    8
0    0    9
```

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

```
>> L=tril(A)
```

L =

```
1  0  0
2  5  0
3  6  9
```

```
>> U=triu(A)
```

U =

```
1  4  7
0  5  8
0  0  9
```

```
>> T=tril(A,-1)
```

T =

```
0  0  0
2  0  0
3  6  0
```

Dreiecksmatrizen

- Die linke untere bzw. rechte obere Dreiecksmatrix von

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

kann mittels `tril(A,k)` bzw. `triu(A,k)` bestimmt werden

```
>> L=tril(A)
```

L =

```
1 0 0
2 5 0
3 6 9
```

```
>> U=triu(A)
```

U =

```
1 4 7
0 5 8
0 0 9
```

```
>> T=tril(A,-1)
```

T =

```
0 0 0
2 0 0
3 6 0
```

- Es gilt $A \equiv \text{tril}(A) + \text{triu}(A) - \text{diag}(\text{diag}(A))$
und $A \equiv \text{tril}(A,-1) + \text{triu}(A,1) + \text{diag}(\text{diag}(A))$

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```


Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

```
Standardendung ist .mat
```

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

Standardendung ist `.mat`

- `save Dateiname x y z` gespeichert nur die Variablen `x`, `y` und `z`

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

Standardendung ist `.mat`

- `save Dateiname x y z` gespeichert nur die Variablen `x`, `y` und `z`
- `save Dateiname Var*` speichert alle Variablen mit Namen `Var...`

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

Standardendung ist `.mat`

- `save Dateiname x y z` gespeichert nur die Variablen `x`, `y` und `z`
- `save Dateiname Var*` speichert alle Variablen mit Namen `Var...`
- Option `-ascii` speichert Daten als plattformunabhängige Textdatei

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

Standardendung ist `.mat`

- `save Dateiname x y z` gespeichert nur die Variablen `x`, `y` und `z`
- `save Dateiname Var*` speichert alle Variablen mit Namen `Var...`
- Option `-ascii` speichert Daten als plattformunabhängige Textdatei
- Option `-mat` speichert Daten als Binärdatei (Standardformat)

Laden und Speichern von Variablen

- Inhalt des Workspaces kann mittels `save` gespeichert werden

```
>> save
```

```
Saving to: matlab.mat
```

```
>> save Dateiname[.Ext]
```

Standardendung ist `.mat`

- `save Dateiname x y z` gespeichert nur die Variablen `x`, `y` und `z`
- `save Dateiname Var*` speichert alle Variablen mit Namen `Var...`
- Option `-ascii` speichert Daten als plattformunabhängige Textdatei
- Option `-mat` speichert Daten als Binärdatei (Standardformat)
- `load` liest gespeicherte Variablen aus einer Datei in den Workspace

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- Dateiendung `.m` kennzeichnet eine ausführbare Datei

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- <http://www.mathworks.com/matlabcentral/fileexchange/>

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- <http://www.mathworks.com/matlabcentral/fileexchange/>
- **Skripte** besitzen keine Ein- und Ausgabeparameter und arbeiten mit den global im Workspace vorhandenen Variablen

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- <http://www.mathworks.com/matlabcentral/fileexchange/>
- **Skripte** besitzen keine Ein- und Ausgabeparameter und arbeiten mit den global im Workspace vorhandenen Variablen
- **Funktionen** besitzen Ein- und Ausgabeparameter und arbeiten intern mit lokalen Variablen, die automatisch gelöscht werden

M-Dateien

- Bisher: Befehle wurden direkt in der MATLAB Konsole eingegeben
- (Oft) besser: Algorithmen können in einer M-Datei editiert/gespeichert und mehrfach (mit unterschiedlichen Daten) aufgerufen werden
- Dateiendung `.m` kennzeichnet eine ausführbare Datei
- <http://www.mathworks.com/matlabcentral/fileexchange/>
- **Skripte** besitzen keine Ein- und Ausgabeparameter und arbeiten mit den global im Workspace vorhandenen Variablen
- **Funktionen** besitzen Ein- und Ausgabeparameter und arbeiten intern mit lokalen Variablen, die automatisch gelöscht werden
- M-Dateien **müssen** eine gute Dokumentation enthalten ;-)

Skripte



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

Skripte



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry.m

```
%MAXENTRY    Betragsmäßig größter Matriceintrag  
% MAXENTRY berechnet betragsmäßig größten Wert von A  
B=abs(A); m=max(B); max(m)
```

Skripte



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry.m

```
%MAXENTRY    Betragsmäßig größter Matriceintrag  
% MAXENTRY berechnet betragsmäßig größten Wert von A  
B=abs(A); m=max(B); max(m)
```

- Kurzbeschreibung in H1-Zeile wird von help, lookfor verwendet

Skripte



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry.m

```
%MAXENTRY    Betragsmäßig größter Matriceintrag  
% MAXENTRY berechnet betragsmäßig größten Wert von A  
B=abs(A); m=max(B); max(m)
```

- Kurzbeschreibung in H1-Zeile wird von help, lookfor verwendet
- Hilfetext endet vor der ersten Zeile ohne Kommentarzeichen „%“

Skripte



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry.m

```
%MAXENTRY    Betragsmäßig größter Matriceintrag  
% MAXENTRY berechnet betragsmäßig größten Wert von A  
B=abs(A); m=max(B); max(m)
```

- Kurzbeschreibung in H1-Zeile wird von help, lookfor verwendet
- Hilfetext endet vor der ersten Zeile ohne Kommentarzeichen „%“
- Globale Variablen B und m bleiben im Workspace erhalten

Funktionen



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

Funktionen



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry1.m

```
function y = maxentry1(A)
%MAXENTRY1    Betragsmäßig größter Matrixeintrag
% MAXENTRY1 berechnet betragsmäßig größten Wert von A
B=abs(A); m=max(B); y=max(m);
```

- Funktions- und Dateiname müssen übereinstimmen

Funktionen



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry1.m

```
function y = maxentry1(A)
%MAXENTRY1    Betragsmäßig größter Matriceintrag
% MAXENTRY1 berechnet betragsmäßig größten Wert von A
B=abs(A); m=max(B); y=max(m);
```

- Funktions- und Dateiname müssen übereinstimmen
- Deklaration `function Ausgabe = name(Eingabe)`

Funktionen



Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A .

M-Datei: maxentry1.m

```
function y = maxentry1(A)
%MAXENTRY1    Betragsmäßig größter Matrixeintrag
% MAXENTRY1 berechnet betragsmäßig größten Wert von A
B=abs(A); m=max(B); y=max(m);
```

- Funktions- und Dateiname müssen übereinstimmen
- Deklaration `function Ausgabe = name(Eingabe)`
- Variablen `B` und `m` werden nach Verlassen der Funktion gelöscht

Funktionen mit mehreren Ausgabeparametern

Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A und bestimme dessen Zeilen- und Spaltennummer.

Funktionen mit mehreren Ausgabeparametern

Aufgabe: Berechne den betragsmäßig größten Eintrag der Matrix A und bestimme dessen Zeilen- und Spaltennummer.

M-Datei: maxentry2.m

```
function [y i j] = maxentry2(A)
%MAXENTRY2    Betragsmäßig größter Matrixeintrag
% MAXENTRY2 berechnet betragsmäßig größten Wert von A
[x k] = max(abs(A));
[y j] = max(x);
i      = k(j);
```

- Beim Aufruf können Ausgabeparameter von rechts nach links weggelassen werden, z.B. entfällt j bei `[y i] = maxentry2(A)`

Sichtbarkeit von Variablen

- Variablen in Skripten sind **global** und verbleiben im Workspace
- In Funktionen definierte Variablen sind **lokal**, d.h. sie sind außerhalb der Funktion nicht sichtbar, und werden automatisch gelöscht

Workspace

- definiere Variablen **a, b, c** im Workspace

Skript

- sieht Variablen **a, b, c**
- definiere Variable **x**

Funktion

- sieht Variablen **a, b, c, x**
- definiere Variable **y**

- Variablen **a, b, c, x** im Workspace vorhanden

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1,    b=2,    c=a==b
```

a =	b =	c =
1	2	0

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1,    b=2,    c=a==b
```

```
a =      b =      c =  
    1        2        0
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double
b	1x1	8	double
c	1x1	1	logical

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1,    b=2,    c=a==b
```

```
a =      b =      c =  
    1      2      0
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double
b	1x1	8	double
c	1x1	1	logical

=	==	gleich
≠	~=	ungleich
>	>	größer
≥	>=	größer gleich
<	<	kleiner
≤	<=	kleiner gleich

Vergleichsoperatoren

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1,    b=2,    c=a==b
```

```
a =      b =      c =  
    1        2        0
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double
b	1x1	8	double
c	1x1	1	logical

=	==	gleich
≠	~=	ungleich
>	>	größer
≥	>=	größer gleich
<	<	kleiner
≤	<=	kleiner gleich

Vergleichsoperatoren

- Neben double wird der Datentyp logical unterstützt

Vergleichsoperatoren

- Logische Variablen haben den Wert 1 (=wahr) oder 0 (=falsch)

```
>> a=1,    b=2,    c=a==b
```

```
a =      b =      c =  
    1        2        0
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double
b	1x1	8	double
c	1x1	1	logical

=	==	gleich
≠	~=	ungleich
>	>	größer
≥	>=	größer gleich
<	<	kleiner
≤	<=	kleiner gleich

Vergleichsoperatoren

- Neben double wird der Datentyp logical unterstützt
- Zuweisungsoperator (c=42) ist **kein** Vergleichsoperator (a==b)

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0     1  
1     0
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1  
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1  
1    1
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1  
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1  
1    1
```

- Vergleich zwischen Matrizen gleicher Größe

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1  
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1  
1    1
```

- Vergleich zwischen Matrizen gleicher Größe

```
>> diag([1; 1])==zeros(2)
```

```
ans =
```

```
0    1  
1    0
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1  
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1  
1    1
```

- Vergleich zwischen Matrizen gleicher Größe

```
>> diag([1; 1])==zeros(2)
```

```
ans =
```

```
0    1  
1    0
```

```
>> ones(3)==ones(2)
```

```
??? Error using ==> eq  
Matrix dimensions must agree.
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1
1    1
```

- Vergleich zwischen Matrizen **gleicher** Größe

```
>> diag([1; 1])==zeros(2)
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(3)==ones(2)
```

```
??? Error using ==> eq
```

```
Matrix dimensions must agree.
```

- Vergleichsfunktionen **is*** für **beliebige** Variablen

```
>> isequal(diag([1; 1]),...
           zeros(2))
```

```
ans =
```

```
0
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1
1    1
```

- Vergleich zwischen Matrizen **gleicher** Größe

```
>> diag([1; 1])==zeros(2)
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(3)==ones(2)
```

```
??? Error using ==> eq
```

```
Matrix dimensions must agree.
```

- Vergleichsfunktionen **is*** für **beliebige** Variablen

```
>> isequal(diag([1; 1]),...
           zeros(2))
```

```
ans =
```

```
0
```

```
>> isequal(ones(3),...
           ones(2))
```

```
ans =
```

```
0
```

Vergleichsfunktionen

- Vergleich zwischen Matrix und Skalar (Elementweiser Vergleich)

```
>> diag([1; 1])==0
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(2)==1
```

```
ans =
```

```
1    1
1    1
```

- Vergleich zwischen Matrizen **gleicher** Größe

```
>> diag([1; 1])==zeros(2)
```

```
ans =
```

```
0    1
1    0
```

```
>> ones(3)==ones(2)
```

```
??? Error using ==> eq
```

```
Matrix dimensions must agree.
```

- Vergleichsfunktionen **is*** für **beliebige** Variablen

```
>> isequal(diag([1; 1]),...
           zeros(2))
```

```
ans =
```

```
0
```

```
>> isequal(ones(3),...
           ones(3))
```

```
ans =
```

```
1
```


Übersicht über Vergleichsfunktionen

■ Vergleichsfunktionen mit skalarem Rückgabewert (doc is)

<code>isequal(A,B,...)</code>	Test, ob A , B , etc. identisch sind
<code>isempty(A)</code>	Test, ob A die leere Matrix <code>[]</code> ist
<code>islogical(A)</code>	Test, ob A vom Typ <code>logical</code> ist
<code>isnumeric(A)</code>	Test, ob A ein Zahlwert ist
<code>isscalar(A)</code>	Test, ob A ein Skalar (1×1) ist
<code>isvector(A)</code>	Test, ob A ein Vektor ($1 \times n$, $n \times 1$) ist

Übersicht über Vergleichsfunktionen

■ Vergleichsfunktionen mit skalarem Rückgabewert (doc is)

<code>isequal(A,B,...)</code>	Test, ob A , B , etc. identisch sind
<code>isempty(A)</code>	Test, ob A die leere Matrix <code>[]</code> ist
<code>islogical(A)</code>	Test, ob A vom Typ <code>logical</code> ist
<code>isnumeric(A)</code>	Test, ob A ein Zahlwert ist
<code>isscalar(A)</code>	Test, ob A ein Skalar (1×1) ist
<code>isvector(A)</code>	Test, ob A ein Vektor ($1 \times n$, $n \times 1$) ist

■ Vergleichsfunktionen mit mehrdimensionalem Rückgabewert

<code>isfinite(A)</code>	Test auf endliche Matrixeinträge
<code>isinf(A)</code>	Test auf unendliche Matrixeinträge vom Typ <code>Inf</code>
<code>isnan(A)</code>	Test auf unzulässige Matrixeinträge vom Typ <code>NaN</code>

Übersicht über Vergleichsfunktionen

■ Vergleichsfunktionen mit skalarem Rückgabewert (doc is)

<code>isequal(A,B,...)</code>	Test, ob A , B , etc. identisch sind
<code>isempty(A)</code>	Test, ob A die leere Matrix <code>[]</code> ist
<code>islogical(A)</code>	Test, ob A vom Typ <code>logical</code> ist
<code>isnumeric(A)</code>	Test, ob A ein Zahlwert ist
<code>isscalar(A)</code>	Test, ob A ein Skalar (1×1) ist
<code>isvector(A)</code>	Test, ob A ein Vektor ($1 \times n$, $n \times 1$) ist

■ Vergleichsfunktionen mit mehrdimensionalem Rückgabewert

<code>isfinite(A)</code>	Test auf endliche Matrixeinträge
<code>isinf(A)</code>	Test auf unendliche Matrixeinträge vom Typ <code>Inf</code>
<code>isnan(A)</code>	Test auf unzulässige Matrixeinträge vom Typ <code>NaN</code>

■ Vergleich `x==NaN` ist falsch, selbst wenn `x` „not a number“ ist

```
>> 0.0/0.0==NaN
ans = 0
```

```
>> isnan(0.0/0.0)
ans = 1
```

Logische Operatoren

■ Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

$$\begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 \& & 0\ 0\ 1\ 1 \\
 \hline
 = & 0\ 0\ 0\ 1 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 | & 0\ 0\ 1\ 1 & \\
 \hline
 = & 0\ 1\ 1\ 1 &
 \end{array}$$

$$\begin{array}{rcl}
 \sim & 0\ 1 & \\
 \hline
 = & 1\ 0 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 \text{xor} & 0\ 0\ 1\ 1 & \\
 \hline
 = & 0\ 1\ 1\ 0 &
 \end{array}$$

Logische Operatoren

■ Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

$$\begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 \& & 0\ 0\ 1\ 1 \\
 \hline
 = & 0\ 0\ 0\ 1 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 | & 0\ 0\ 1\ 1 & \\
 \hline
 = & 0\ 1\ 1\ 1 &
 \end{array}$$

$$\begin{array}{rcl}
 & & 0\ 1\ 0\ 1 \\
 \sim & 0\ 1 & \text{ xor } 0\ 0\ 1\ 1 \\
 \hline
 = & 1\ 0 & = 0\ 1\ 1\ 0
 \end{array}$$

Logische Operatoren

■ Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

$$\begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 \& & 0\ 0\ 1\ 1 \\
 \hline
 = & 0\ 0\ 0\ 1 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 | & 0\ 0\ 1\ 1 & \\
 \hline
 = & 0\ 1\ 1\ 1 &
 \end{array}$$

$$\begin{array}{rcl}
 \sim & 0\ 1 & \\
 \hline
 = & 1\ 0 &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0\ 1\ 0\ 1 & \\
 \text{xor} & 0\ 0\ 1\ 1 & \\
 \hline
 = & 0\ 1\ 1\ 0 &
 \end{array}$$

■ „short circuit“ Varianten && bzw. || für skalare Ausdrücke

Logische Operatoren

■ Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

$$\begin{array}{rcl}
 & 0 & 1 & 0 & 1 \\
 \& & 0 & 0 & 1 & 1 \\
 \hline
 = & 0 & 0 & 0 & 1
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0 & 1 & 0 & 1 \\
 | & 0 & 0 & 1 & 1 \\
 \hline
 = & 0 & 1 & 1 & 1
 \end{array}$$

$$\begin{array}{rcl}
 & & & 0 & 1 & 0 & 1 \\
 \sim & 0 & 1 & & & & \\
 \hline
 = & 1 & 0 & & & &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & & & 0 & 1 & 0 & 1 \\
 \text{xor} & 0 & 0 & 1 & 1 & & \\
 \hline
 = & 0 & 1 & 1 & 0 & &
 \end{array}$$

■ „short circuit“ Varianten && bzw. || für skalare Ausdrücke

■ Effizienz: *Ausdruck1* || *Ausdruck2*

wenn *Ausdruck1* wahr ist, dann wird *Ausdruck2* nicht ausgewertet

Logische Operatoren

■ Logische Operatoren (doc relop)

&	logisches Und
	logisches Oder
~	logisches Nicht
xor	logisches exklusives Oder

$$\begin{array}{rcl}
 & 0 & 1 & 0 & 1 \\
 \& & 0 & 0 & 1 & 1 \\
 \hline
 = & 0 & 0 & 0 & 1
 \end{array}
 \qquad
 \begin{array}{rcl}
 & 0 & 1 & 0 & 1 \\
 | & 0 & 0 & 1 & 1 \\
 \hline
 = & 0 & 1 & 1 & 1
 \end{array}$$

$$\begin{array}{rcl}
 & & & 0 & 1 & 0 & 1 \\
 \sim & 0 & 1 & & & & \\
 \hline
 = & 1 & 0 & & & &
 \end{array}
 \qquad
 \begin{array}{rcl}
 & & & 0 & 1 & 0 & 1 \\
 \text{xor} & 0 & 0 & 1 & 1 & & \\
 \hline
 = & 0 & 1 & 1 & 0 & &
 \end{array}$$

■ „short circuit“ Varianten && bzw. || für skalare Ausdrücke

■ Effizienz: $\text{Ausdruck1} \ || \ \text{Ausdruck2}$

wenn *Ausdruck1* wahr ist, dann wird *Ausdruck2* nicht ausgewertet

■ Fehlererkennung: $x > 0 \ \&\& \ \log(1/x) \leq 1$

verhindert Division durch 0 sowie Logarithmusberechnung für $x \leq 0$

find Funktion

- $k = \text{find}(\text{Ausdruck})$ findet alle Indizes k des Vektors \mathbf{v} , für die die im *Ausdruck* definierte Bedingung wahr ist

find Funktion

- $k = \text{find}(\text{Ausdruck})$ findet alle Indizes k des Vektors \mathbf{v} , für die die im *Ausdruck* definierte Bedingung wahr ist

```
>> v=1:10; k=find(v<=5)
```

```
k =
```

```
     1     2     3     4     5
```

find Funktion

- $k = \text{find}(\text{Ausdruck})$ findet alle Indizes k des Vektors \mathbf{v} , für die die im *Ausdruck* definierte Bedingung wahr ist

```
>> v=1:10; k=find(v<=5)
```

```
k =
```

```
     1     2     3     4     5
```

```
>> k=find(mod(v,2)==0)
```

```
k =
```

```
     2     4     6     8    10
```

mod elementweise zu betrachten

find Funktion

- $k = \text{find}(\text{Ausdruck})$ findet alle Indizes k des Vektors \mathbf{v} , für die die im *Ausdruck* definierte Bedingung wahr ist

```
>> v=1:10; k=find(v<=5)
```

```
k =
```

```
    1     2     3     4     5
```

```
>> k=find(mod(v,2)==0)
```

```
k =
```

```
    2     4     6     8    10
```

mod elementweise zu betrachten

```
>> v=1:10; k=find(v<=5 & mod(v,2)==0)
```

```
k =
```

```
    2     4
```

find Funktion

- $[i \ j \ s] = \text{find}(\text{Ausdruck})$ findet alle Indizes (i, j) der Matrix A , für die die im *Ausdruck* definierte Bedingung wahr ist und liefert die entsprechenden Werte in s

$$\left(\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} == \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

find Funktion

- $[i \ j \ s] = \text{find}(\text{Ausdruck})$ findet alle Indizes (i, j) der Matrix A , für die die im *Ausdruck* definierte Bedingung wahr ist und liefert die entsprechenden Werte in s

```
>> [i j]=find(ones(2)==diag([1 1]))
```

```
i =      j =  
    1    2    1    2
```

$$\left(\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} == \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Anwendung der `find` Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6];    k=find(v>0);    w=log(v(k))
```

```
w =
```

```
    0.6931    1.7918
```


Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6];    k=find(v>0);    w=log(v(k))
```

```
w =
```

```
    0.6931    1.7918
```

- Teilvektoren bzw. Teilmatrizen auswählen

Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6];    k=find(v>0);    w=log(v(k))
```

```
w =
```

```
    0.6931    1.7918
```

- Teilvektoren bzw. Teilmatrizen auswählen

```
>> A=rand(10);    k=find(A<=0.8);    l=length(k)
```

```
l =
```

```
    83
```

Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6];    k=find(v>0);    w=log(v(k))
```

```
w =
```

```
    0.6931    1.7918
```

- Teilvektoren bzw. Teilmatrizen auswählen

```
>> A=rand(10);    k=find(A<=0.8);    l=length(k)
```

```
l =
```

```
    83
```

- Alle Nicht-Nulleinträge auswählen

Anwendung der find Funktion

- Berechnung auf Teilvektoren bzw. Teilmatrizen einschränken

```
>> v=[0 -4 2 6];    k=find(v>0);    w=log(v(k))
```

```
w =
```

```
    0.6931    1.7918
```

- Teilvektoren bzw. Teilmatrizen auswählen

```
>> A=rand(10);    k=find(A<=0.8);    l=length(k)
```

```
l =
```

```
    83
```

- Alle Nicht-Nulleinträge auswählen

```
>> A=diag(10:20);    [i j s]=find(A)
```

```
i =
```

```
    1    2    ...
```

```
j =
```

```
    1    2    ...
```

```
s =
```

```
   10   11   ...
```

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen
- Einfachste Schleife ist der Doppelpunktoperator
 $A=1:5$ ist Kurzform für $A(1)=1$, $A(2)=2$, $A(3)=3$, ...

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen
- Einfachste Schleife ist der Doppelpunktoperator
 $A=1:5$ ist Kurzform für $A(1)=1$, $A(2)=2$, $A(3)=3$, ...
- Es gibt zwei unterschiedliche Schleifenarten
 - `for` Schleifen mit **fester** Anzahl an Wiederholungen
 - `while` Schleifen mit **variabler** Anzahl an Wiederholungen

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen
- Einfachste Schleife ist der Doppelpunktoperator
 $A=1:5$ ist Kurzform für $A(1)=1$, $A(2)=2$, $A(3)=3$, ...
- Es gibt zwei unterschiedliche Schleifenarten
 - `for` Schleifen mit **fester** Anzahl an Wiederholungen
 - `while` Schleifen mit **variabler** Anzahl an Wiederholungen
- Schleifen (auch unterschiedliche) können geschachtelt werden

Schleifen

- Schleifen dienen zur wiederholten Ausführung von Befehlen
- Einfachste Schleife ist der Doppelpunktoperator
 $A=1:5$ ist Kurzform für $A(1)=1$, $A(2)=2$, $A(3)=3$, ...
- Es gibt zwei unterschiedliche Schleifenarten
 - `for` Schleifen mit **fester** Anzahl an Wiederholungen
 - `while` Schleifen mit **variabler** Anzahl an Wiederholungen
- Schleifen (auch unterschiedliche) können geschachtelt werden
- Programmierfehler können zu Endlosschleifen führen, die mit der Tastenkombination `Strg+C` abgebrochen werden können

Schleifen mit fester Anzahl an Wdh.



Aufgabe: Berechne für 100 Zufallszahlen zwischen 0 und 1
die Anzahl der Zufallszahlen kleiner als 0.8.

Schleifen mit fester Anzahl an Wdh.



Aufgabe: Berechne für 100 Zufallszahlen zwischen 0 und 1 die Anzahl der Zufallszahlen kleiner als 0.8.

```
z=0; for i=1:100
    if rand(1)<0.8, z=z+1; end
end
```

Schleifen mit fester Anzahl an Wdh.



Aufgabe: Berechne für 100 Zufallszahlen zwischen 0 und 1
die Anzahl der Zufallszahlen kleiner als 0.8.

```
z=0; for i=1:100
    if rand(1)<0.8, z=z+1; end
end
```

```
for Variable=Ausdruck
    Befehlsfolge
end
```

Schleifen mit fester Anzahl an Wdh.



Aufgabe: Berechne für 100 Zufallszahlen zwischen 0 und 1 die Anzahl der Zufallszahlen kleiner als 0.8.

```
z=0; for i=1:100
    if rand(1)<0.8, z=z+1; end
end
```

```
for Variable=Ausdruck
    Befehlsfolge
end
```

■ Schleifen über Wertelisten sind möglich

```
for x=[pi pi/2 pi/4] →  $x^{(1)} = \pi$ ,  $x^{(2)} = \frac{1}{2}\pi$ ,  $x^{(3)} = \frac{1}{4}\pi$ 
```

Schleifen mit fester Anzahl an Wdh.



Aufgabe: Berechne für 100 Zufallszahlen zwischen 0 und 1 die Anzahl der Zufallszahlen kleiner als 0.8.

```
z=0; for i=1:100
    if rand(1)<0.8, z=z+1; end
end
```

```
for Variable=Ausdruck
    Befehlsfolge
end
```

- Schleifen über Wertelisten sind möglich

for x=[pi pi/2 pi/4] $\rightarrow x^{(1)} = \pi, \quad x^{(2)} = \frac{1}{2}\pi, \quad x^{(3)} = \frac{1}{4}\pi$

- Schleifen über Vektoren sind möglich

for x=eye(3) $\rightarrow x^{(1)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

Schleifen mit variabler Anzahl an Wdhlg.



Aufgabe: Berechne solange Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

Schleifen mit variabler Anzahl an Wdhlg.



Aufgabe: Berechne **solange** Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```


Schleifen mit variabler Anzahl an Wdhlg



Aufgabe: Berechne **solange** Zufallszahlen zwischen 0 und 1
bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```

```
while Ausdruck  
    Befehlsfolge  
end
```

Schleifen mit variabler Anzahl an Wdhlg



Aufgabe: Berechne **solange** Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```

```
while Ausdruck  
    Befehlsfolge  
end
```

- while Schleifen werden wiederholt solange *Ausdruck* wahr ist

Schleifen mit variabler Anzahl an Wdhlg



Aufgabe: Berechne **solange** Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```

```
while Ausdruck  
    Befehlsfolge  
end
```

- while Schleifen werden wiederholt solange *Ausdruck* wahr ist
- while 1, ..., end erzeugt eine Endlosschleife

Schleifen mit variabler Anzahl an Wdhlg



Aufgabe: Berechne *solange* Zufallszahlen zwischen 0 und 1 bis eine Zahl größer oder gleich 0.8 auftritt.

```
z=0; while rand(1)<0.8  
    z=z+1;  
end
```

```
while Ausdruck  
    Befehlsfolge  
end
```

- while Schleifen werden wiederholt solange *Ausdruck* wahr ist
- while 1, ..., end erzeugt eine Endlosschleife
- Endlosschleifen *müssen* mittels break verlassen werden

Tipps zum Umgang mit Schleifen

1. **Tip:** `while` Schleifen können zwei unterschiedliche Formen haben.

Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```

Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```

```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```

Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```



```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```


Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```

Für $n = 0$ gilt:
 $i = 0 \neq j = 1$

```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```

Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```

Für $n = 0$ gilt:
 $i = 0 \neq j = 1$

```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```

- Abbruchbedingung kann **zu Beginn** (vorprüfend) oder **am Ende** (nachprüfend) des Schleifendurchlaufs überprüft werden

Tipps zum Umgang mit Schleifen

1. **Tip:** while Schleifen können zwei unterschiedliche Formen haben.

```
i=0;  
while i<n  
    i=i+1;  
end
```

Für $n = 0$ gilt:
 $i = 0 \neq j = 1$

```
j=0;  
while 1  
    j=j+1;  
    if j>=n, break, end  
end
```

- Abbruchbedingung kann zu Beginn (vorprüfend) oder am Ende (nachprüfend) des Schleifendurchlaufs überprüft werden
- Hinweis für Programmierer: es gibt keine REPEAT-UNTIL Schleifen; verwende als Alternative `while 1, ..., end`

Tipps zum Umgang mit Schleifen

2. **Tip:** Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

■ Beispiel: Erzeuge Vektor $x = (x_i)$,
$$x_i = \begin{cases} i & \text{falls } i \text{ gerade} \\ \pi & \text{sonst} \end{cases}$$

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

- Beispiel: Erzeuge Vektor $x = (x_i)$,
$$x_i = \begin{cases} i & \text{falls } i \text{ gerade} \\ \pi & \text{sonst} \end{cases}$$

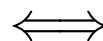
```
for i=1:n
    if mod(i,2) == 0
        x(i)=i;
    else
        x(i)=pi;
    end
end
```

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

- Beispiel: Erzeuge Vektor $x = (x_i)$,
$$x_i = \begin{cases} i & \text{falls } i \text{ gerade} \\ \pi & \text{sonst} \end{cases}$$

```
for i=1:n
    if mod(i,2) == 0
        x(i)=i;
    else
        x(i)=pi;
    end
end
```



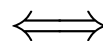
```
x(1:2:n) = pi;
x(2:2:n) = 2:2:n;
```

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

- Beispiel: Erzeuge Vektor $x = (x_i)$,
$$x_i = \begin{cases} i & \text{falls } i \text{ gerade} \\ \pi & \text{sonst} \end{cases}$$

```
for i=1:n
    if mod(i,2) == 0
        x(i)=i;
    else
        x(i)=pi;
    end
end
```



```
x(1:2:n) = pi;
x(2:2:n) = 2:2:n;
```

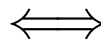
- Arrayoperationen sind übersichtlicher und bereits ausgetestet

Tipps zum Umgang mit Schleifen

2. Tip: Schleifen können oft durch Arrayoperationen ersetzt werden (Stichwort: Vektorisieren).

- Beispiel: Erzeuge Vektor $x = (x_i)$,
$$x_i = \begin{cases} i & \text{falls } i \text{ gerade} \\ \pi & \text{sonst} \end{cases}$$

```
for i=1:n
    if mod(i,2) == 0
        x(i)=i;
    else
        x(i)=pi;
    end
end
```



```
x(1:2:n) = pi;
x(2:2:n) = 2:2:n;
```

- Arrayoperationen sind übersichtlicher und bereits ausgetestet

Tipps zum Umgang mit Schleifen

■ Gauss Elimination (T. Driscoll, Learning MATLAB, SIAM, '09)

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

Tipps zum Umgang mit Schleifen

- Gauss Elimination (T. Driscoll, Learning MATLAB, SIAM, '09)

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

- Eine geschickte Implementierung kommt auch mit nur **einer** for-Schleife aus.

Tipps zum Umgang mit Schleifen

■ Gauss Elimination (T. Driscoll, Learning MATLAB, SIAM, '09)

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



```
n=length(A);  
for k=1:n-1  
    row=k+1:n;  
    col=k:n;  
    s=A(row,k)/A(k,k);  
    A(row,col)=...  
        A(row,col)-s*A(k,col);  
end
```

■ Eine geschickte Implementierung kommt auch mit nur **einer** for-Schleife aus.

Tipps zum Umgang mit Schleifen

■ Gauss Elimination (T. Driscoll, Learning MATLAB, SIAM, '09)

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



```
n=length(A);
for k=1:n-1
    row=k+1:n;
    col=k:n;
    s=A(row,k)/A(k,k);
    A(row,col)=...
        A(row,col)-s*A(k,col);
end
```

■ Eine geschickte Implementierung kommt auch mit nur **einer** for-Schleife aus.

CPU Zeit	n=200	400	600	800	1000
3 Schleifen	0.18	1.57	5.64	14.05	27.36
1 Schleife	0.02	0.17	0.64	0.88	3.15

Nicht behandelte Themen

Die folgenden Themen (und viele mehr...) wurden in diesem Crashkurs nicht angesprochen:

- Komplexe Zahlen
- Zeichenketten, Bildschirmausgabe
- Grafikausgabe in 2D, 3D (`plot`, `plot3d`, ...)
- Symbolisches Rechnen
- Fortgeschrittene Datenstrukturen (dünnbesetzte Matrizen, Zellvariablen, ...)
- Debugger und Profiler

Weiterführende Informationen zu diesen Themen finden Sie in der eingangs erwähnten Literatur.

Quelle

- Modifizierter MATLAB-Crashkurs von Steffen Basting (2014)
<https://www.mathematik.tu-dortmund.de/sites/cop-kurs-ws14-15/>