

# Einführung in Matlab und Python - Einheit 1

## Einleitung, Vektoren und Matrizen

Jochen Schulz

Georg-August Universität Göttingen 

- Anmeldung → StudIP  
<https://studip.uni-goettingen.de/>  
[Einführung in Matlab und Python \(Mathematische Anwendersysteme\)](#)
- Alle Unterlagen (Aufgabenblätter, Vorlesungsfolien, Beispiele, Musterlösungen) → StudIP

- Anmeldung → StudIP  
<https://studip.uni-goettingen.de/>  
[Einführung in Matlab und Python \(Mathematische Anwendersysteme\)](#)
- Alle Unterlagen (Aufgabenblätter, Vorlesungsfolien, Beispiele, Musterlösungen) → StudIP

## Dozent

Jochen Schulz

NAM, Zimmer 04 (Erdgeschoß)

**Telefon:** 39-4525

**Email:** [schulz@math.uni-goettingen.de](mailto:schulz@math.uni-goettingen.de)

**XMPP:** [schulz@jabber.num.math.uni-goettingen.de](xmpp:schulz@jabber.num.math.uni-goettingen.de)

# Ablauf der Veranstaltung

- Blockveranstaltung vom 9.9 - 20.9.2013
- **Vorlesung:** 9.15 - ca. 11.30 (MN55)
- **Übungsbetrieb/Praktikum:** ca. 11:30 - ca. 17:00 (NAM WAP-Raum)
  - 1 Übungszettel/Tag.
  - Besprechung Aufgaben vom Vortag (individuell)
  - Betreuung: Hilfestellung beim Bearbeiten der Aufgaben
- **Klausur:** 27.9.2013; 10:00 - 11:30; Anmeldung über FlexNow.

# Inhalt der Vorlesung

1. **Tag** Organisatorisches, Basis von Matlab und Python
2. **Tag** Programmieren, Datenstrukturen
3. **Tag** Rekursionen, Grafik
4. **Tag** ? Polynome, Interpolation, Debugging
5. **Tag** ? Mehrdimensionale Arrays, Funktionen, Numerische Lineare Algebra, Dünnbesetzte Matrizen
6. **Tag** ? Numerische Mathematik, Profiler
7. **Tag** ? Visualisierung und Validierung
8. **Tag** ? Schnittstelle zu C (Optional)
9. **Tag** ? Wunschvorlesung
10. **Tag** Fragestunde

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen



# Programmieren für den Wissenschaftler

- Daten erzeugen oder erheben (Simulation, Experiment)
- Weiterverarbeitung von Daten
- Visualisierung und Validierung
- Ergebnisse veröffentlichen bzw. kommunizieren

Wir wollen: eine *High-Level* Sprache:

- Programmieren ist leicht
- Vorhandene Elemente nutzen
- geeignet für Prototyping und Debugging (Interaktion)
- Möglichst nur ein Werkzeug für alle Probleme

# MATLAB

- MATLAB steht für **Mat**rix **lab**oratory; ursprünglich speziell Matrizenrechnung.
- Interaktives System für numerische Berechnungen und Visualisierungen (Skriptsprache).

## Vorteile

- Vielfältige Visualisierungsmöglichkeiten.
- Viele zusätzliche Toolboxes (Symb. Math T., PDE T., Wavelet T.)
- Ausgereifte und integrierte Oberfläche.

## Nachteile

- Kostenintensiv.
- Ein/Ausgabe von Dateien kann umständlich sein.
- Spezialisierter Funktionsumfang macht manche Programmierung schwer.

# Python: NumPy, SciPy, SymPy

- Modulare Skriptsprache.

## Vorteile

- Viele Module mit wissenschaftlichen Fokus.
- Klare Code-Struktur.
- Ebenso viele Module für den nicht-wissenschaftlichen Gebrauch (nützlich z.B. für Ein-/Ausgabe).
- Frei und open-source.

## Nachteile

- Entwicklungsumgebung etwas komplizierter (Spyder,ipython).
- Nicht alle spezialisierten Möglichkeiten anderer Software.

## MATLAB



Matlab online-help :-).



Introduction to Scientific Computing, C.F. van Loan, Prentice Hall, New Jersey, 1997,



Scientific Computing with MATLAB, A. Quarteroni, F. Saleri, Springer, 2003,

## Python



NumPy, SciPy SciPy developers (<http://scipy.org/>),



SciPy-lectures, F. Perez, E. Gouillart, G. Varoquaux, V. Haenel (<http://scipy-lectures.github.io>),



Matplotlib (<http://matplotlib.org>)



scitools (<https://code.google.com/p/scitools/>)



mayavi (<http://docs.enthought.com/mayavi/mayavi/mlab.html>)

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen



# Struktur von Skript-Files

- Skript-Files bestehen aus einer Sequenz von Befehlen, die nacheinander abgearbeitet werden.
- operiert auf Variablen im *Workspace*.
- Gestartet wird das Programm `name.m` durch Eingabe von `name`.
- Beschreibung des Skript-Files (Auch Funktion):

```
help sin
```

```
sin      Sine of argument in radians.  
sin(X)  is the sine of the elements of X.
```

# Struktur von Function-Files

```
function [Out_1,...,Out_k] = myfunction(In_1,...,In_l)
% Beschreibung der Funktion
..
Out_1=..
..
Out_k=..
```

Soll keine Variable zurückgegeben werden, so besteht die erste Zeile aus

```
function myfunction(In_1,...,In_k)
```

- Call-by-value
- Variablen lokal, d.h.
  - Variablen des Workspace sind nicht verfügbar.
  - definierte Variablen werden nicht im Workspace gespeichert.

**Wichtig:** Funktionsname = Dateiname.



# Priorität beim Programmaufruf

## Beispiel-Programmaufruf

name

Testet ob, ..

- 1 Variable
- 2 Unterfunktion. Eine Unterfunktion ist ein Programm/Funktion, die in derselben Datei wie der Aufruf steht.
- 3 Programm im aktuellen Verzeichnis.
- 4 *private function*.
- 5 Programm im Suchpfad.

Bei gefundenem Namen wird die Suche beendet.

Der Suchpfad (Variable `path`) enthält Verzeichnisse in einer geordneten Liste.

- Abarbeitung erfolgt der Ordnung gemäss.
- Suchpfade hinzufügen:

```
addpath <pfadname>
```

- Suchpfade entfernen:

```
rmpath <pfadname>
```

# Bedienung: Kurzreferenz

- `doc <name>`  
öffnet grafisches Hilfefenster zum jeweiligen Programm.
- **F5**  
führt offene Datei im command window aus.
- **F9**  
führt markierte Code-Zeilen aus.
- **clear**  
löscht Variablen im Workspace
- `lookfor <name>`  
Suche nach `name` in den Kommentaren zu den Funktionen (auch: grafisches Hilfefenster).
- `what`  
m-Files im aktuellen Verzeichnis.
- `type <name>`  
Inhalt von `name.m` (Command Window).
- `which <name>`  
absoluter Pfad der Datei, in dem die Funktion `name` gespeichert ist.

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

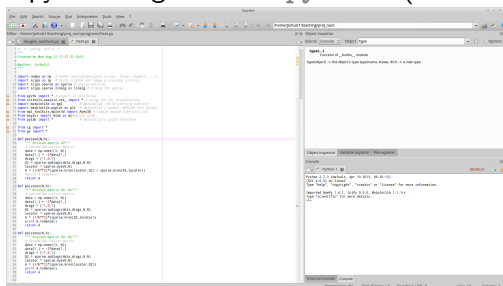
## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

# Spyder Fenster-Aufbau

Starten von Spyder: Eingabe von `spyder &` (in einem Terminal).



- **Editor:** Dateimanipulation
- **Console:** Befehlseingabe und Standardausgabe
- **Object Inspector:** Hilfe und Variablenansicht
- **Grafik:** in separaten Fenstern

# Struktur von Skript-Files

- Skript-Files bestehen aus einer Sequenz von Befehlen, die nacheinander abgearbeitet werden, sowie von definierten Funktionen
- enthält geladene Module.
- operiert auf globalen Variablen.
- Gestartet wird das Programm `name.py` durch F5 im Editor oder in einem terminal.

# Funktionen

Eine Funktion kann man wie folgt definieren:

```
def <name>(<Argumente>) :  
    """ Beschreibung """  
    <Code Block>  
    return <Rueckgabe>
```

- Variablen lokal.
- Call-by-reference.
- **Objekt-Methoden:**

Objekte besitzen Funktionen, sogenannte Objekt-Methoden. Ein Objekt kennt alle auf sich selbst anwendbaren Funktionen.

```
f = 3.14  
f.is_integer()
```

# Priorität beim Programmaufruf

Beispiel-Programmaufruf

name

Testet ob,..

- 1 Variable
- 2 Funktion

Bei gefundenem Namen wird die Suche beendet.



Programm: spyder

- `<name>?`  
öffnet Hilfe zur jeweiligen Funktion.
- **F5**  
führt offene Datei in dedizierter oder vorhandener Shell aus.
- **F9**  
führt markierte Code-Zeilen aus.
- `<name>.TAB`  
zeigt alle Objektmethoden.

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

# Graph eines Polynoms

## Aufgabe:

Zeichnen Sie den Graphen eines Polynoms

$$p(x) = \sum_{i=0}^N a_i x^i, \quad a_i \in \mathbb{R}$$

Zu Werten  $(x_i)_{i=1}^n$  muß man  $(p(x_i))_{i=1}^n$  berechnen.

# Skalare Version

matlab

```
function y=ausw_poly1(a,x)

n = length(a);
aux_vector = x.^(0:n-1);
y = aux_vector*transpose(a);
```

python

```
def ausw_poly1(a,x):
    n = len(a)
    aux_vector = x**np.array(range(0,n))
    return dot(aux_vector,a)
```

# Vektorielle Version

matlab

```
function y = ausw_poly2(a,x)
n = length(a);
k = length(x);
A = repmat(transpose(x),1,n);
B = repmat(0:(n-1),k,1);
y = (A.^B)*transpose(a);
```

python

```
def ausw_poly2(a,x):
    n = len(a)
    k = len(x)
    xm = np.array([x])
    A = sp.repeat(xm.T, n,1)
    B = sp.repeat(np.array([range(0,n)]), k,0)
    return dot(A**B,a)
```

# Plotten des Polynoms

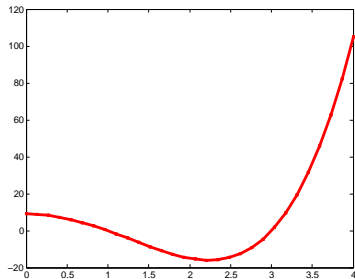
matlab

```
a = [9 0 -10 0 1]; % Koeffizienten  
  
x = linspace(0,4,30); % Betrachte [0,4]  
y = ausw_poly2(a,x);  
% Plotten  
plot(x,y,'r*-', 'LineWidth',3, 'MarkerSize',4)
```

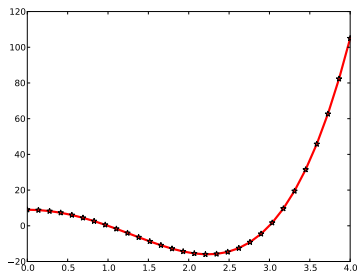
python

```
a = np.array([9,0,-10,0,1])  
  
x = np.linspace(0,4,30) # Betrachte [0,4]  
y = ausw_poly2(a,x)  
#Plotten  
plot(x,y,'r*-',linewidth=3,markersize=8)
```

# Plotten des Polynoms



Matlab



Python

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen



- **Bezeichner** sind Namen, wie z.B.  $x$  oder  $f$ . Sie können im mathematischen Kontext sowohl Variablen als auch Unbestimmte repräsentieren.
- Bezeichner sind aus Buchstaben, Ziffern und Unterstrich `_` zusammengesetzt.
- Man unterscheidet zwischen Groß- und Kleinschreibung.
- Bezeichner dürfen nicht mit einer Ziffer beginnen.

## Beispiele

- zulässige Bezeichner: `x`, `f`, `x23`, `_x_1`
- unzulässige Bezeichner: `12x`, `p~`, `x>y`, `Das System`

- Der **Wert** eines Bezeichners ist ein **Objekt** eines bestimmten **Datentyps**.
- Ein **Datentyp** ist durch seine Eigenschaften gegeben.  
**Beispiel:** ganze Zahlen (Integers), Zeichenketten (Strings), Gleitkommazahlen (float) ...
- Matlab: 2D-Arrays vom Typ Double vorherrschend
- Python: integer, float, numpy.ndarray, lists,..
- Ein **Objekt** ist eine Instanz (Einheit) eines Datentyps.

# Zuweisungsoperator =

```
<bezeichner> = <wert>
```

Zuweisung des Wertes `wert` zu dem Bezeichner `bez`.

- `func(arg)=expr(arg)`: Definition der Funktion `func` mit dem Argument `arg` und Zuweisung des Ausdrucks `expr` zu (abhängig von `arg`)
- **Warnung:** Unterscheiden Sie stets zwischen dem Zuweisungsoperator `=` und dem logischen Operator `==`.
- Löschen von Zuweisungen/Variablen.
  - Python: `%clear <bezeichner>`
  - Matlab: `clear <bezeichner>`

# Python: Listen und Tuple

- Eine **Liste** ist in Python mit [...] gekennzeichnet

```
liste = [21,22,24,23]  
liste.sort(); liste
```

[21, 22, 23, 24]

- Ein **Tuple** ist in Python mit (...) gekennzeichnet

```
tuple = (liste[0], liste[2])  
tuple, tuple[0]
```

(( 21, 24), 21)

- Liste von ganzen Zahlen von a bis b

```
range(a,b+1)
```

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

# Vektoren I

- Erzeugen 'per Hand'

```
b = [1 2 4]
```

```
b = np.array([1,2,4])
```

- Abfragen der Einträge von  $b$

```
b(2)
```

```
b[1]
```

Index  $\equiv$  Position im Vektor

**Achtung:** *Matlab*: Indizes beginnen immer mit 1!

*Python*: Indizes beginnen immer mit 0!

# Doppelpunkt - Notation

`x:s:z` erzeugt einen Vektor der Form

$$(x, x + s, x + 2s, x + 3s, \dots, z).$$

```
>> a = 2:11
```

```
a =
```

```
2 3 4 5 6 7 8 9 10 11
```

```
>> c = -2:0.75:1
```

```
c =
```

```
-2.0000 -1.2500 -0.5000 0.2500 1.0000
```



# Vektoren II

- `length(a)`(Matlab) `len(a)`(Python)  
Länge des Vektors  $a$  an.

- `linspace(x1,x2,N)`  
Vektor

$$x_1, x_1 + \frac{x_2 - x_1}{N-1}, x_1 + 2\frac{x_2 - x_1}{N-1}, \dots, x_2$$

der Länge  $N$ .

```
linspace(1,2,4)
```

```
ans =  
    1.0000    1.3333    1.6667    2.0000
```

- `logspace(x1,x2,N)`  
wie `linspace`, nur logarith. Skalierung

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

# Erzeugen von Matrizen

- Erzeugen 'per Hand'

```
B = [1 3 4; 5 6 7]
```

```
B = np.array([[1,3,4],[5,6,7]])  
#alternativ (Unterschiedlicher Datentyp!)  
B = matrix([[1,3,4],[5,6,7]])
```

- `eye(n,m)`  
( $n \times m$ )-EinheitsMatrix)

```
eye(2,3)
```

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

(1 auf der Hauptdiagonalen und 0 sonst).

# Erzeugen von Matrizen II

- `zeros(n,m)` (Matlab) `zeros((n,m))` (Python)  
 $(n \times m)$ - Matrix mit 0 als Einträge.
- `ones(n,m)` (Matlab) `ones((n,m))` (Python)  
 $(n \times m)$ - Matrix mit 1 als Einträge.
- Blockmatrizen

```
C = [B zeros(2,2); eye(2,3) eye(2,2)]
```

```
C = vstack([hstack([B, zeros((2,2))]) , hstack([eye(2,3), eye(2,2)])])
```

C =

1	3	4	0	0
5	6	7	0	0
1	0	0	1	0
0	1	0	0	1

**Achtung:** Matrizen in einer Zeile müssen dieselbe Zeilenanzahl haben und Matrizen in einer Spalte dieselbe Spaltenanzahl.

# Erzeugen von Matrizen III

- `repmat(A,n,m)` (Matlab) `tile(A,(n,m))` (Python)  
Blockmatrix mit  $(n \times m)$  aus  $A$  bestehenden Blöcken zusammenhängen

```
D = repmat(B,1,2)
```

```
D = tile(B,(1,2))
```

D =

1	3	4	1	3	4
5	6	7	5	6	7

- `diag(v,k)`  
Matrix der Größe  $(n + |k|) \times (n + |k|)$  mit den Einträgen des Vektors  $v$  (Länge  $n$ ) auf der  $k$ -ten Nebendiagonalen.

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- **Manipulation von Matrizen**
- Matrix- und Vektoroperationen

# Beispiel- und Spezial-Matrizen

- Beispiel

```
E = vander(linspace(1,3,3))
```

E =

1	1	1
4	2	1
9	3	1

- Hilbert-Matrix: `hilb`(Matlab) und `scipy.linalg.hilbert`(Python)
- Hadamard-Matrix: `hadamard` (Matlab) und `scipy.linalg.hadamard`(Python)
- use.

# Matlab: Zugriff auf Matrizen

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

A =

1	2	3
4	5	6
7	8	9

## Abfragen eines Eintrags

```
>> A(2,1)
ans =
    4
```

## Abfrage von Blöcken

```
>> A(2:3,1:2)
ans =
    4    5
    7    8
```

## Abfrage einer Zeile

```
>> A(2,:)
ans =
    4    5    6
```

## Abfrage mehrerer Zeilen

```
>> A([1 3], :)
ans =
    1    2    3
    7    8    9
```



# Python: Zugriff auf Matrizen

```
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

A =

1	2	3
4	5	6
7	8	9

Abfragen eines Eintrags

```
>>> A[1,0]  
ans =  
4
```

Abfrage von Blöcken

```
>>> A[1:3,0:2]  
ans =  
4 5  
7 8
```

Abfrage einer Zeile

```
>>> A[:,0]  
ans =  
4 5 6
```

Abfrage mehrerer Zeilen

```
>>> A[(0,2),:]  
ans =  
1 2 3  
7 8 9
```

## 1 MATLAB und Python Basis

- Einleitung
- Grundlegende Bedienung MATLAB
- Grundlegende Bedienung Python (Spyder)
- Erstes Beispiel

## 2 Programmieren: Basis

## 3 Vektoren und Matrizen

- Erzeugen von Vektoren
- Erzeugen von Matrizen
- Manipulation von Matrizen
- Matrix- und Vektoroperationen

# Matrizenoperationen

## Standard-Matrix Operationen +, -, \*

```
A = [1 2; 3 4]; B = 2*ones(2,2);
```

```
A = np.array([[1,2],[3,4]]); B = 2*ones((2,2))
```

### Multiplikation

```
>> A*B  
ans =
```

```
     6     6  
    14    14
```

### Addition

```
>> A+B  
ans =
```

```
     3     4  
     5     6
```

### Subtraktion

```
>> A-B  
ans =
```

```
    -1     0  
     1     2
```

```
>>> dot(A,B)  
[[ 6.,  6.],  
 [14., 14.]]
```

```
>>> A + B  
([[ 3.,  4.],  
 [ 5.,  6.]])
```

```
>>> A - B  
([[ -1.,  0.],  
 [ 1.,  2.]])
```

# Andere Operatoren

- $A \setminus B$  (Matlab) `solve(A,B)` (Python)  
Lösung  $X$  von  $A * X = B$ .
- $A / B$  (Matlab) `solve(A.T,B.T)` (Python)  
Lösung  $X$  von  $X * A = B$ .
- `inv(A)`  
Inverse von  $A$ .
- $A'$  (Matlab) `A.conj().T` (Python)  
komplex Transponierte von  $A$ .
- $A.'$  (Matlab) `A.T` (Python)  
Transponierte von  $A$ .
- $A^z$  (Matlab) `power(A,z)` (Python)  
(quadratische Matrizen)  $\underbrace{A * A * \dots * A}_{z\text{-mal}}$
- `size(A)` (Matlab) `A.shape` (Python)  
Größe einer Matrix  $A$ .

# Matlab: Punktnotation

```
A = [1 2; 3 4]; B = 2*ones(2,2);
```

- $C(i,j) = A(i,j) * B(i,j)$ .

```
>> C = A.*B
```

```
C =
```

```
     2     4
     6     8
```

- $C(i,j) = A(i,j)/B(i,j)$ .

```
>> C = A./B
```

```
C =
```

```
    0.5000    1.0000
    1.5000    2.0000
```

# Matlab: Punktnotation

- $C(i,j) = B(i,j)/A(i,j)$ .

```
>> C = A.\B  
C =  
    2.0000    1.0000  
    0.6667    0.5000
```

- $C(i,j) = A(i,j)^{B(i,j)}$

```
>> C = A.^B  
C =  
    1     4  
    9    16
```

**Achtung:** Dimension von  $A$  und  $B$  gleich.

Matrizen können durch Skalare ersetzt werden, z.B.  $A.^2$ .

# Skalarprodukt

- Vektoren  $a = (a_1, \dots, a_n)$ ,  $b = (b_1, \dots, b_n)$
- Skalarprodukt:  $ab^t$
- Summe der Einträge von  $a$ :  $(1, \dots, 1)a^t$

Beispiel:

```
>> a=1:100; b=linspace(0,1,100);  
>> a*transpose(b)  
3.3667e+03  
>> ones(1,100)*transpose(a)  
5050
```

```
>>> a=linspace(1,100,100); b=linspace(0,1,100)  
>>> dot(a,b)  
3366.666666666667  
>>> dot(ones((1,100)),a)  
array([ 5050.] )
```

**Wichtig:** komponentenweise Multiplikation:  $*$ (Python)  $.*$ (Matlab)!