

# Einführung in die Programmierung (MA8003)

Theorie 4.1: Datenorganisation, Profiler, nützliche Funktionen

Dr. Laura Scarabosio

Technische Universität München  
Fakultät Mathematik, Lehrstuhl für Numerische Mathematik M2

11.10.2019

<b>Theorie 1.1+1.2</b>	Mo (07.10.2019)	08:30 - 10:00 Uhr	HS BC2 0.01.17
Praxis 1.1	Mo (07.10.2019)	10:30 - 12:00 Uhr	HS BC2 0.01.17/16
Praxis 1.2	Mo (07.10.2019)	12:30 - 14:00 Uhr	HS BC2 0.01.17/16
<b>Theorie 2.1+2.2</b>	Mi (09.10.2019)	13:00 - 14:30 Uhr	HS BC2 0.01.17
Praxis 2.1	Mi (09.10.2019)	15:00 - 16:30 Uhr	HS BC2 0.01.17/16
Praxis 2.2	Mi (09.10.2019)	16:30 - 18:00 Uhr	HS BC2 0.01.17/16
<b>Theorie 3.1+3.2</b>	Do (10.10.2019)	08:30 - 10:00 Uhr	HS BC2 0.01.17
Praxis 3.1	Do (10.10.2019)	10:30 - 12:00 Uhr	HS BC2 0.01.17/16
Praxis 3.2	Do (10.10.2019)	12:30 - 14:00 Uhr	HS BC2 0.01.17/16
<b>Theorie 4.1+4.2</b>	Fr (11.10.2019)	08:30 - 10:00 Uhr	HS BC2 0.01.17
Praxis 4.1	Fr (11.10.2019)	10:30 - 12:00 Uhr	HS BC2 0.01.17/16
Praxis 4.2	Fr (11.10.2019)	12:30 - 14:00 Uhr	HS BC2 0.01.17/16
<b>Klausur</b>	Fr (25.10.2019)	xxx Uhr	xxx
Nachholklausur	Fr (22.11.2019)	xxx Uhr	xxx

Kurswebseite mit Infos, Folien und Übungsblättern:

<https://www-m2.ma.tum.de/bin/view/Allgemeines/...>  
...MA8003WS19

**Bitte melden Sie sich über TUM-Online für die Klausur an!**

- 1** Organisation von Daten
  - Strukturen (structures)
  - Cell Arrays
  
- 2** Nützliche Werkzeuge
  - Profiler
  - Code Analyzer
  - Dependency Report
  
- 3** Weitere nützliche Matlab-Befehle

- 1** Organisation von Daten
  - Strukturen (structures)
  - Cell Arrays
  
- 2** Nützliche Werkzeuge
  - Profiler
  - Code Analyzer
  - Dependency Report
  
- 3** Weitere nützliche Matlab-Befehle

## Einfaches Beispiel:

```
buch
|
|__ .autor  _____ 'Barry W. Boehm'
|
|__ .titel  _____ 'Software Engineering Economics'
|
|__ .preis  _____ 89.00
|
|__ .jahr   _____ 1981
```

Daten **beliebiger** Art können innerhalb einer Struktur gespeichert werden.

## Vorteile:

- sinnvoll geordnete Daten
- Zugriff mit konkreter Bezeichnung anstelle von Indizes
- multidimensionale Struktur möglich

- direkt mit Zuweisungen:

```
>> buch.autor = 'Barry W. Boehm';  
>> buch.titel = 'Software Engineering Economics';  
>> buch.preis = 89.00;  
>> buch.jahr = 1981;
```

- mit der struct-Funktion:

```
>> buch = struct('autor','Barry W. Boehm', ...  
               'titel','Software Engineering Economics', ...  
               'preis', 89.00, 'jahr', 1981);
```

## ■ Zugriff:

```
>> buch
buch =
    autor: 'Barry W. Boehm'
    titel: 'Software Engineering Economics'
    preis: 89
    jahr: 1981
>> buch.jahr
ans =
    1981
```

## ■ Anfügen weiterer Elemente: struct array (Index hinzufügen)

```
>> buch(2).autor = 'Steve McConnel'
>> buch(2).title = 'Code Complete'
>> ...
```

# Weitere Befehle zu Strukturen

Funktion	Beispiel	Beschreibung
fieldnames	<pre>&gt;&gt; fieldnames(buch); ans = 'autor'       'titel'       'preis'       'jahr'</pre>	alle Feldnamen holen
getfield	<pre>&gt;&gt; getfield(buch(2),             'titel') ans = Code Complete</pre>	Inhalt eines Feldes holen
setfield	<pre>&gt;&gt; setfield(buch(2),             'preis',59.90)</pre>	Feldinhalt setzen
isstruct	<pre>&gt;&gt; isstruct(buch)</pre>	wahr (= 1) für Struktur
isfield	<pre>&gt;&gt; isfield(buch,'titel')</pre>	wahr (= 1) für Feld



# Weitere Befehle zu Strukturen

Funktion	Beispiel	Beschreibung
rmfield	<pre data-bbox="364 225 917 443">&gt;&gt; rmfield(buch,'titel') ans = 1x2 struct array with fields:     autor     preis     jahr</pre>	Feld löschen
deal	<pre data-bbox="364 513 917 692">&gt;&gt; [a,b] = deal(buch.titel) a = Software Engineering Economics b = Code Complete</pre>	Feldinhalte in mehrere Variablen auslesen
struct2cell	<pre data-bbox="364 837 917 873">&gt;&gt; b_c = struct2cell(buch);</pre>	Struktur in ein Cell-Array umwandeln

- Vereinfachte Syntax, um über alle Felder einer Struktur zu operieren:

Feld in eckige Klammern einschließen, z.B.:

```
>> total = sum([buch.preis]);
```

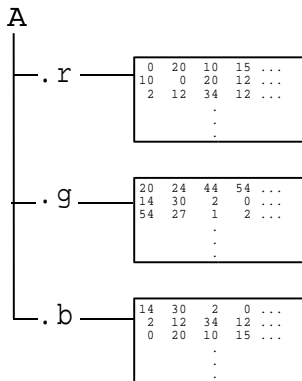
Dies entspricht einer durch Kommata getrennten Liste:

```
>> total = sum([buch(1).preis, buch(2).preis, ...]);
```

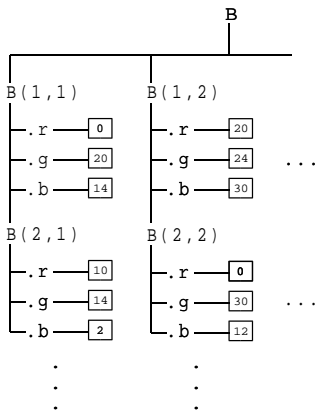
- Zwei Formen der Datenorganisation möglich, entweder ebenen- oder elementweise (siehe folgende Folien).

# Datenorganisation: Ebene vs. Einzel-Element

In einem Bild werden pro Pixel die **rot/grün/blau**-Werte gespeichert.



$$A.r(1,2) = 20$$



$$B.r(1,2) = 20$$

## 1) Teilbild holen:

### Ebenen-Organisation

```
>> r_sub = A.r(1:10,1:10);  
>> g_sub = A.g(1:10,1:10);  
>> b_sub = A.b(1:10,1:10);
```

### Einzel-Element-Organisation

```
>> sub = B(1:10,1:10);
```

## 2) Gesamte Farbebene holen:

### Ebenen-Organisation

```
>> r_ebene = A.r;
```

### Einzel-Element-Organisation

```
>> r_ebene = zeros(100,100);  
>> for k = 1:(100*100)  
    r_ebene(k) = B(k).r;  
end;
```

**Also:** Daten sollte man immer so organisieren, dass es für die am häufigsten benötigten Operationen günstig ist.

- Strukturen können beliebig geschachtelt werden.

```
>> A = struct('name','Hans',  
            'zeiten', [14,15,16; 15.3,14.8,13.6],  
            'adresse', struct('strasse','Hauptstr. 38','stadt','Berlin'))  
>> A(2).name = 'Max';  
>> A(2).zeiten = [14,15,16; 15.2,14.6,13.8];  
>> A(2).adresse.strasse = 'Albweg 3';  
>> A(2).adresse.stadt = 'Muenchen';
```

# Geschachtelte Strukturen

```
A
|
|-- A(1) ----- .name ----- Hans
|
|   |
|   |-- .zeiten ----- [ 14   15   16 ]
|   |                   [ 15.3 14.8 13.6]
|   |
|   |
|   |-- .adresse ----- .strasse -- Hauptstr. 38
|   |                   |
|   |                   |-- .stadt ----- Berlin
|   |
|
|-- A(2) ----- .name ----- Max
|
|   |
|   |-- .zeiten ----- [ 14   15   16 ]
|   |                   [ 15.2 14.6 13.8]
|   |
|   |
|   |-- .adresse ----- .strasse -- Albweg 3
|   |                   |
|   |                   |-- .stadt ----- Muenchen
```

- ein Cell Array kann mehrere Matlab-Felder enthalten, z.B:

<p>cell (1,1)</p> <table border="1" data-bbox="366 368 550 540"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9	<p>cell (1,2)</p> <table border="1" data-bbox="853 399 1125 532"><tr><td>'Hello world!'</td></tr></table>	'Hello world!'
1	2	3									
4	5	6									
7	8	9									
'Hello world!'											
<p>cell (2,1)</p> <table border="1" data-bbox="267 736 648 829"><tr><td>[1.25 3.75 0.34]</td></tr></table>	[1.25 3.75 0.34]	<p>cell (2,2)</p> <table border="1" data-bbox="946 762 1030 809"><tr><td>25</td></tr></table>	25								
[1.25 3.75 0.34]											
25											

## Möglichkeit 1 – Cell Indexing:

```
>> A(1,1) = {[1 2 3; 4 5 6; 7 8 9]};  
>> A(1,2) = {'Hello world!'};  
>> A(2,1) = {[1.25 3.75 0.34]};  
>> A(2,2) = {25};
```

## Möglichkeit 2 – Content Indexing:

```
>> A{1,1} = [1 2 3; 4 5 6; 7 8 9];  
>> A{1,2} = 'Hello world!';  
>> A{2,1} = [1.25 3.75 0.34];  
>> A{2,2} = 25;
```



## Beispiele für Zugriff:

```
>> A(2,2)
ans = [25]

>> A(2,2)+5
??? Error using ==> +
Function '+' not
defined for variables
of class 'cell'.

>> A{2,2}
ans = 25

>> A{2,2}+5
ans = 30
```

```
>> A{1,2}
ans =
Hello world!

>> A{1,1}(1,1)
ans = 1

>> A{:,1}
ans =
     1     2     3
     4     5     6
     7     8     9

ans =
1.250    3.750    0.340
```

- Zellen löschen mit eckigen Klammern, z.B.:

```
>> A(2,2) = []  
>> A(:,2) = []
```

- Dimensionen des Cell Arrays umformen:

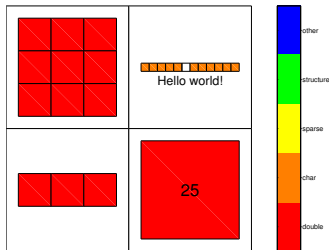
```
>> B = reshape(A,4,1);
```

- Speicher für neues Cell Array reservieren:

```
>> C = cell(5,3);
```

- Struktur des Cell Arrays graphisch anzeigen:

```
>> cellplot(A, 'legend')
```



<b>Funktion</b>	<b>Beschreibung</b>
cell2struct	Cell Array in Struktur-Feld umwandeln
celldisp	gesamtes Cell-Array anzeigen
cellfun	eine Funktion auf jede Zelle eines Cell Arrays anwenden
deal	Cell Array in mehrere Variablen auslesen
iscell	wahr (= 1) für ein Cell Array

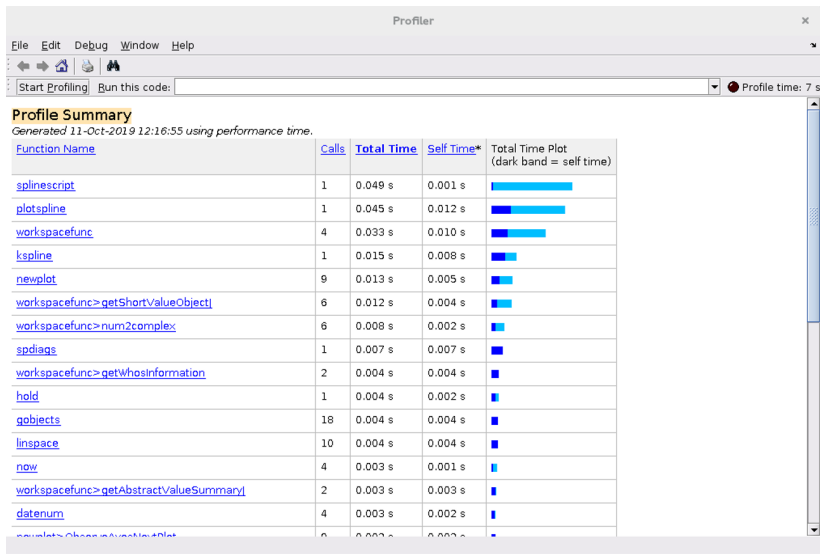
## Sonstiges

- gesamte Structures können in einer Zelle abgespeichert werden
- Cell Arrays können beliebig tief geschachtelt werden  
(Zugriff dann z.B. mit  $A\{2,3\}\{1,2\}\{1,5\}$ )

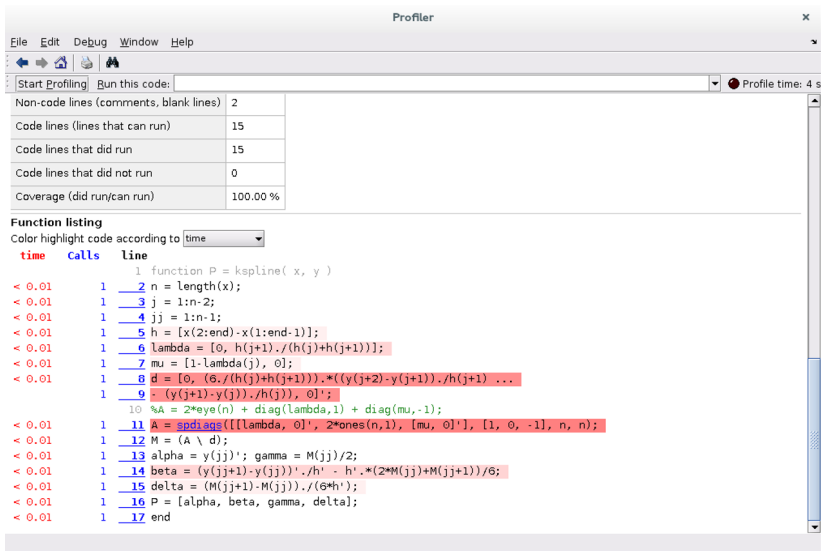
- 1 Organisation von Daten
  - Strukturen (structures)
  - Cell Arrays
- 2 Nützliche Werkzeuge
  - Profiler
  - Code Analyzer
  - Dependency Report
- 3 Weitere nützliche Matlab-Befehle

- Detaillierte Rechenzeitanalyse: Wieviel Zeit wurde für welche Zeile oder Funktion verbraucht?
- Auflistung geordnet nach Relevanz
- Gut geeignet zum Auffinden von Effizienzproblemen
- Falls optimiert werden soll, wo wäre Optimierung sinnvoll?
- Vorgehen:
  - 1 Profile Aufzeichnung starten mit `profile on`
  - 2 m-File aufrufen
  - 3 Profile report ansehen mit `profile viewer`.
  - 4 Optionen: Abspeichern von Berichten, Detail-Level setzen (z.B. interne Funktionen in Laufzeitmessung einschließen), ...

# Beispiel: Profiler, Zusammenfassung



# Beispiel: Profiler, Details



The screenshot shows the MATLAB Profiler interface. At the top, there is a menu bar (File, Edit, Debug, Window, Help) and a toolbar with navigation icons. Below the toolbar, there are buttons for 'Start Profiling' and 'Run this code:'. A dropdown menu shows 'Profile time: 4 s'. A table displays execution statistics:

Category	Value
Non-code lines (comments, blank lines)	2
Code lines (lines that can run)	15
Code lines that did run	15
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

Below the table is the 'Function listing' section. It has a dropdown menu set to 'time'. The listing shows the following code with columns for 'time', 'calls', and 'line':

```
time  calls  line
< 0.01  1    2  1 function P = kspline( x, y )
< 0.01  1    3  2 n = length(x);
< 0.01  1    4  3 j = 1:n-2;
< 0.01  1    5  4 jj = 1:n-1;
< 0.01  1    6  5 h = [x(2:end)-x(1:end-1)];
< 0.01  1    7  6 lambda = [0, h(j+1)./(h(j)+h(j+1))];
< 0.01  1    8  7 mu = [1-lambda(j), 0];
< 0.01  1    9  8 d = [0, (6./(h(j)+h(j+1))).*((y(j+2)-y(j+1))./h(j+1) ...
1    10  9  -(y(j+1)-y(j))./h(j)), 0]';
1    11 10 %A = 2*eye(n) + diag(lambda,1) + diag(mu,-1);
< 0.01  1   11 11 A = spdiags([lambda, 0]', 2*ones(n,1), [mu, 0]', [1, 0, -1], n, n);
< 0.01  1   12 12 M = (A \ d);
< 0.01  1   13 13 alpha = y(jj)'; gamma = M(jj)/2;
< 0.01  1   14 14 beta = (y(jj+1)-y(jj))./h' - h'.*(2*M(jj)+M(jj+1))/6;
< 0.01  1   15 15 delta = (M(jj+1)-M(jj))./(6*h');
< 0.01  1   16 16 P = [alpha, beta, gamma, delta];
< 0.01  1   17 17 end
```

Web Browser - Code Analyzer Report

Code Analyzer Report

## Code Analyzer Report

This report displays potential errors and problems, as well as opportunities to improve your MATLAB programs ([Learn More](#)).

Report for folder /scratch/users/scalaura/work/TUM/teaching/EinfProgr19/tests

<a href="#">kspline.m</a> 1 message	<b>E</b> Use of brackets [] is unnecessary. Use parentheses to group, if needed.
<a href="#">plotspline.m</a> 1 message	<b>E</b> Terminate statement with semicolon to suppress output (in functions).
<a href="#">splinescript.m</a> No messages	



# Dependency Report

Web Browser - Dependency Report

Dependency Report

The Dependency Report shows dependencies among MATLAB files in a folder ([Learn More](#)).

Show child functions  Show parent functions (current folder only)  
 Show subfunctions

Built-in functions and files in toolbox/matlab are not shown

Report for Folder /scratch/users/scalaura/work/TUM/teaching/EinfProgr19/tests

MATLAB File List	Children (called functions)	Parents (calling functions, current dir. only)
<a href="#">kspline</a>		plotspline
<a href="#">plotspline</a>	current dir : <a href="#">kspline</a>	splinescript
<a href="#">splinescript</a>	current dir : <a href="#">plotspline</a>	

- 1 Organisation von Daten
  - Strukturen (structures)
  - Cell Arrays
  
- 2 Nützliche Werkzeuge
  - Profiler
  - Code Analyzer
  - Dependency Report
  
- 3 Weitere nützliche Matlab-Befehle

# Berechnung von Eigenwerte

Der Befehl `eig` berechnet die Eigenwerte einer Matrix.

- `lambda = eig(A)`: Eigenwerte von A im Spaltenvektor `lambda`.
- `[V,Lambda] = eig(A)`: Eigenwerte und Eigenvektoren, mit Lambda Diagonalmatrix ( $A*V = V*D$ ).
- `[V,Lambda] = eig(A,B)`: verallgemeinerte Eigenwerte und Eigenvektoren ( $A*V = B*V*D$ ).

```
>> A = [2 0 0 ; 0 -3 0; 0 0 5];  
>> eig(A)  
  
ans =  
  
    -3  
     2  
     5
```

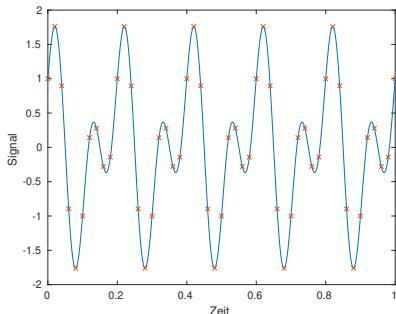
```
>> [V,D] = eig(A)  
  
V =  
  
     0     1     0  
     1     0     0  
     0     0     1  
  
D =  
  
    -3     0     0  
     0     2     0  
     0     0     5
```

Hinweis: Siehe auch `eigs`.

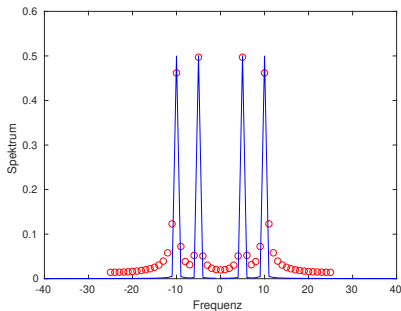
# Fourier-Transformation

fft berechnet die schnelle Fourier-Transformation (*Fast Fourier Transform*) einer Funktion.

Inverse Fourier-Transformation mit ifft.



```
g = @(t)sin(20*pi*t)+cos(10*pi*t);  
N=51;  
t = linspace(0,1,51);  
gvals = g(t);
```



```
G = fft(gvals);  
G = G/N;  
G = abs(G);
```

Ende Theorie 4.1

**Fragen?**