Westfälische
Wilhelms-Universität
Münster

# Advanced Matlab

living.knowledge
WWU Münster

Mario Ohlberger,
Martin Drohmann,
Stefan Girke

10/05/2012

# Cell Array

- Datentyp, der beliebige Werte (nicht nur Zahlen) in Zellen speichert, die wie einer Matrix oder einem Vektor angeordnet sein können.
- Werte können sein
  - Zahlen (ganzzahlige, Fließkomma- oder komplexe Zahlen),
  - Matrizen/Vektoren, Zeichenketten,
  - function handles,
  - Cell Arrays und Struct arrays
  - (und Objekte).
- Initialisierung mit Befehl `cell(rows, columns)` oder `{wert1, wert2; wert3, wert4}`.
- Zugriff auf Werte mit eckigen Klammern `c{1,2}`

# Cell Array: Fortsetzung

## Beispiel: Cell Array von Zeichenketten und Zahlen

```
1  zellen = {'Holland', 0; 'Deutschland', 2};
2
3  disp(zellen{1,2})
4  0
5
6  disp(zellen{2,1})
7  Deutschland
```

# Cell Array: Fortsetzung

## Achtung!

Mit runden Klammern kann man auch auf Zellen zugreifen, es werden wieder **Cell-Arrays** zurückgegeben.

```
1  zellen = {'Holland', 0; 'Deutschland', 2};
2
3  disp(zellen(1,2))
4  [0]
5
6  disp(zellen(2,:))
7  'Deutschland'    [2]
```

# Cell Array: Fortsetzung

## Unterschied zu Matrix

- Nur mit Matrizen kann man Matrix-Vektor-Rechnungen und die lineare Algebra Funktionen von Matlab verwenden.
- Der Zugriff auf Matrizen ist *viel* schneller!
- Matrizen können nur Zahlwerte speichern.

Hilfe unter `doc cell` oder `web([docroot '/techdoc/matlab_prog/br04bw6-98.html'])`

# Struct Array

- Datentyp der beliebige Werte in sogenannten Feldern mit eindeutigen Bezeichnern speichert.
- Matlab Hilfe unter `doc struct` oder `web([docroot '/techdoc/matlab_prog/br04bw6-38.html'])`

## Beispiel

```
1  struktur.feld = 12;
2  struktur.anderes_feld = 'string';
3  struktur.drittes_feld = {'cell', 'of', 'strings'};
4
5  feldnamen = fieldnames(struktur)
6  feldnamen =
7
8     'feld'
9     'anderes_feld'
10    'drittes_feld'
```

# Struct Array (Forts.)

- Der Zugriff auf Felder kann auch über Zeichenketten geschehen mit der Syntax `struktur.('feldname')`

## Beispiel: Allen Feldern den Wert [] zuweisen

```
1  feldnamen = fieldnames(struktur);
2  for i=1:length(feldnamen)
3    struktur.(feldnamen{i}) = [];
4  end
5
6  struktur
7 struktur =
8
9              feld: []
10     anderes_feld: []
11     drittes_feld: []
```

# Function Handle

- Assoziert einen Variablennamen mit einer vorhanden Funktion.
- Die assoziierte Funktion kann anschließen mit diesem Variablennamen erneut aufgerufen werden.

## Verwendungsmöglichkeiten

1. Übergabe von Funktionen an andere Funktionen
2. Verwendung von Funktionen außerhalb ihres Gültigkeitsbereichs

Hilfe unter `doc function_handle` oder `web([docroot '/techdoc/matlab_prog/f2-38133.html'])`.

# Function Handle (Fortsetzung)

## 1. Übergabe von Funktionen

Die Funktion `quad` zum numerischen Integrieren von Funktionen erwartet einen Function Handle als erstes Argument.

```
1  fhandle = @sin;
2  quad(fhandle, 0, pi)
3  2
4
5  fhandle = @cos;
6  quad(fhandle, 0, pi)
7  0
8
9  fhandle = @(x) x;
10 quad(fhandle, 0, pi)
11 4.9348
```

# Dünn besetzte Matrizen (sparce matrices)

- Speichern nur die Elemente einer Matrix, die *nicht null* sind.
- Vermindert Berechnungszeiten durch Ignorieren von Operationan auf Nullelementen.

## Vorteil durch dünn besetzte Struktur einer Matrix

```
M_full = magic(1100);           % Create 1100-by-1100 matrix.
M_full(M_full > 50) = 0;        % Set elements >50 to zero.
M_sparse = sparse(M_full);      % Create sparse matrix of same.

whos
  Name            Size                Bytes  Class     Attributes

  M_full       1100x1100            9680000  double
  M_sparse     1100x1100               5004  double    sparse
```

Hilfe unter `web([docroot '/techdoc/math/f6-32006.html'])`

# Package Ordner

Ein Ordner, dessen Name mit einem '+' beginnt, ist ein sogenanntes *'Package'*.
**Ziel:** In größeren Projekten wie RBmatlab ($\approx$ 600 M-Files) ist es schwierig Ordnung zu halten / Übersichtlichkeit zu wahren.

- Daher: viele lange Dateinamen und
- viele Ordner, (alle mit `addpath` dem Workspace hinzugefügt)
- Alternative: Packages

## Beispiel

Nehmen wir folgende Ordner-Struktur an:

```
+Exercise2/                % ein Package Ordner 'Exercise2' mit zwei M-Files
+Exercise2/model.m
+Exercise2/script.m
```

Dann erfolgt der Zugriff auf die M-File Funktionen mit:

```
Exercise2.model            % und
Exercise2.script
```

# Prozedurale Programmierung

- Ist ein *Programmierparadigma*, d.h. eine Art ein Programm aufzuschreiben.
- Sehr simpel: Programm besteht aus Funktionen (Prozeduren), die weitere Funktionen aufrufen können.

# Objektorientierte Programmierung

- Anderes Programmier-Paradigma (gleiche Funktionalität wie bisher!)
- Bei modernen Programmiersprachen oftmals wenigstens optional verfügbar.

**Idee**

Zusammengehörige *Daten* und *Funktionen* werden in einem *Objekt* zusammengefasst.

# Einige Ziele

- Besseres Verständnis durch Abstraktion
- Einfache Erweiterbarkeit durch Vererbung und Polymorphismus
- Ordnung durch Kapselung (Daten können nur verändert werden, wenn es sinnvoll ist)

# OOP in Matlab

- Ein Objekt hat einen *Objektbezeichner* und eine *Klasse*.
- Beispiel: Initialiseren wir das Objekt `automodell = Auto(200, rot)` so dat es den Bezeichner `automodell` und die Klasse `Auto`.
- Man sagt auch: `autmodell` ist eine *Instanz* der Klasse `Auto`.
- Hilfe unter `web([docroot '/techdoc/matlab_oop/ug_intropage.html'])`.

## Klasse

Eine Klasse beinhaltet Bezeichner für die gespeicherten Daten des Objekts (wie bei einer Struktur) und implementiert die Funktionen.

```
1  classdef Auto
2    properties
3      farbe;
4      hoechstgeschwindigkeit;
5    end
6    methods
7      function a = Auto(kmh, farbe)  % Dies ist ein Konstruktor!!!
8        a.farbe = farbe;
9        a.hoechstgeschwindigkeit = kmh;
10     end
11
12     function fahren(ziel)
13       % ...
14     end
15   end
16 end
```

# Klassen (Fortsetzung)

- Die Funktionen eines Objekts werden
  - mit `automodell.fahren('Berlin')` oder
  - alternativ mit `fahren(automodell, 'Berlin')`

  aufgerufen.
- Der *Konstruktor* ist eine spezielle Funktion mit dem Namen der Klasse und wird bei der Instanziierung aufgerufen.

## Vererbung

Eine Klasse kann von einer anderen Klasse erben:

```
1 classdef Cabrio < Auto
2   methods
3     function verdeck_oeffnen()
4       % ...
5     end
6   end
7 end
```

Die `Cabrio` erbt alle Eigenschaften und Methoden von `Auto` und implementiert nur den Unterschied, also die Funktion `verdeck_oeffnen`.

## Abstrakte Klassen

- Es können auch Klassen definiert werden, in denen nicht alle Eigenschaften und Funktionen implementiert sind.
- Diese Klassen nennen wir auch Interfaces.
- Vereinfacht das Sprechen über Objekte:
  - *Statt:* Das erste Argument ist ein Datentyp, auf den ich eine Methode fahren anwenden kann, und das zwei Datenfelder für Spritverbrauch und Geschwindigkeit hat
  - Der erste Argument ist ein Objekt, das das Interface `IAuto` erfüllt.
- Beispiel: `Exercise2.IAuto`.