MATLAB and Simulink

Prof. Dr. Ottmar Beucher

University of Applied Sciences Karlsruhe
Germany

National Chung Hsing University, Taichung
Summer School 17th - 30th July 2013
What is MATLAB?

- A tool for numerical calculation and simulation
- A higher programming language
- An interpreted programming language
Elementary MATLAB operations:

- arithmetic operations
- logical operations
- mathematical functions
- graphics functions
- I/O-operations (data exchange)
The MATLAB User Interface (since Release 2012b)

1. Command Window
2. Command History Window
3. Current Folder Browser
4. Workspace Browser
5. Current Folder Toolbar
6. Quick Access Toolbar
7. Toolstrip
8. Search Field

Prof. Dr. Ottmar Beucher
MATLAB and Simulink - Summer School NCHU, Taichung
July 2013
Defining Numerical Arrays (MATrices)

≫ rowvector = [1 5 −3]
rowvector =
  1 5 −3≫

columnvector = [2; 4; 3; −1; 1 −4*j]
columnvector =
  2.0000
  4.0000
  3.0000
  −1.0000
  1.0000 − 4.0000i≫

aMatrix = [3 1+2*i 2; 4 0 −5]
aMatrix =
  3.0000 1.0000 + 2.0000i 2.0000
  4.0000 0 −5.0000
Defining Numerical Arrays (MATrices)

≫ % appending a row vector (note: semicolon)
   aMatrix = [aMatrix; 1 2 3]≫

≫ % appending a row vector
   aMatrix = [aMatrix; rowvector]≫

≫ % appending a column vector (note: comma)
   v = [1;2;3];≫
   aMatrix = [aMatrix, v]≫

≫ % access to a component
   element23 = aMatrix(2,3)≫

≫ % setting to a component
   aMatrix(2,3) = 25≫

≫ % cancelling a row vector
   aMatrix(1,: )=[]≫

≫ % cancelling a column vector
   aMatrix(:,2)=[]≫

≫ % BUT
   aMatrix(2,3) = []
   Subscripted assignment dimension mismatch.
The *Variables* Window:

![Variables Window](image)

<table>
<thead>
<tr>
<th>#</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.0000 + 0.0000i</td>
<td>1.0000 + 2.0000i</td>
<td>2.0000 + 0.0000i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.0000 + 0.0000i</td>
<td>0.0000 + 0.0000i</td>
<td>-5.0000 + 0.0000i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 - 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MATLABs Memory Stack - The "Workspace"

Workspace

x
columnvector
aMatrix
rowvector
....

MATLAB

Simulink

all kind of data
1. Within MATLAB define the following matrices and vectors and save them in corresponding variables:

\[
M = \begin{pmatrix}
1 & 0 & 0 \\
0 & j & 1 \\
j & j+1 & -3
\end{pmatrix},
\]

\[
k = 2.75,
\vec{v} = \begin{pmatrix}
1 \\
3 \\
-7 \\
-0.5
\end{pmatrix},
\]

\[
\vec{w} = \begin{pmatrix}
1 & -5.5 & -1.7 & -1.5 & 3 & -10.7
\end{pmatrix},
\]

\[
\vec{y} = \begin{pmatrix}
1 & 1.5 & 2 & 2.5 & \cdots & 100.5
\end{pmatrix}.
\]
2. Consider the matrix $M$ of the previous exercise:
   - Enlarge matrix $M$ to a $6 \times 6$-matrix $V$, such that
     \[
     V = \begin{pmatrix}
     M & M \\
     M & M \\
     
     \end{pmatrix}.
     \]
   - Cancel from matrix $V$ the 2nd row and the 3rd column.
   - Save the 4th row of matrix $V$ in a new vector $r_4$.
   - Change component $(4, 2)$ of matrix $V$ to $j + 5$.

3. Cancel all variables of the workspace and reconstruct matrix $V$ afterwards using the automatically saved definition commands for $V$ and the ↑- and ↓-keys. Additionally try to reconstruct other variables using the Command History Window.

4. Overwrite the 5th row of matrix $V$ with zeros using the Variables Window.
Matrix algebra operations: $M \cdot N$, $M + N$, $\lambda \cdot M$

```matlab
≫ M = [1 2 3; 4 -1 2];  % defines 2x3–matrix M
≫ N = [1 2 -1; 4 -1 1; 2 0 1]  % defines 3x3–matrix N
≫ V = M*N  % product M\times N ALLOWED !
V =

15  0  4
4   9 -3

≫ W = N*M;  % product N\times M NOT ALLOWED !
??? Error using ==> mtimes Inner matrix dimensions must agree.

≫ K = [1 5 3; 2 -1 -2];  % defines another 2x3–matrix K
≫ W = N*M;  % sums N+M, M+N ALLOWED !
≫ lambda = 2;  % defines a scalar value (1x1–matrix)
≫ V = lambda*M;  % product lambda\times M, M\times lambda ALLOWED !
```
Field operations: $M \cdot N, M^k, M/N$

% defines 2x3–matrix M
M = [1 2 3; 4 -1 2];
N = [1 2 -1; 4 -1 1; 2 0 1]; % defines 3x3–matrix N

V = M.*N % field product M.*N NOT ALLOWED!
Error using .*
Matrix dimensions must agree.

% defines another 2x3–matrix K
K = [1 5 3; 2 -1 -2];
V = M.*K % field products M.*K, K.*M ALLOWED!

V =
1 10 9
8 1 -4

k = 3; % defines a scalar value (1x1–matrix)
M.^k;
% field powers ALLOWED!

W = M./K % field divisions ALLOWED!
W =
1.0000 0.4000 1.0000
2.0000 1.0000 -1.0000
Exercises

5. Solve the following problems:

- Calculate using matrix- or/and field-operations the standard scalar product of the vectors
  \[ \vec{x} = \left( \begin{array}{cccc} 1 & 2 & \frac{1}{2} & -3 \\ 0 & -1 & -3 & -1 \end{array} \right) \quad \text{and} \quad \vec{y} = \left( \begin{array}{cccc} 2 & 0 & -3 & \frac{1}{3} \\ 2 & 1 & 3 & 2 \end{array} \right). \]

- Calculate the product of the matrices
  \[ A = \left( \begin{array}{ccc} -1 & 3.5 & 2 \\ 0 & 1 & -1.3 \\ 1.1 & 2 & 1.9 \end{array} \right) \quad \text{and} \quad B = \left( \begin{array}{ccc} 1 & 0 & -1 \\ -1.5 & 1.5 & -3 \\ 1 & 1 & 1 \end{array} \right). \]

- Starting from
  \[ A = \left( \begin{array}{ccc} -1 & 3.5 & 2 \\ 0 & 1 & -1.3 \\ 1.1 & 2 & 1.9 \end{array} \right) \]
  calculate the matrix
  \[ C = \left( \begin{array}{ccc} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1.9 \end{array} \right) \]
  using appropriate field operations!
Logical Operations

Operations:

- and (\&)
- or (\|)
- negation (\sim)
- exclusive Or (\texttt{xor})

- always \textit{Field} operators
- result: logical values, logical arrays
Logical Operations (Examples)

\[
A = \begin{bmatrix} 1 & -3 \\ 0 & 0 \end{bmatrix}
\]

\[
B = \begin{bmatrix} 0 & 5 \\ 0 & 1 \end{bmatrix}
\]

\[
\text{Res} = A \& B \quad \% \text{ Logical AND}
\]

\[
\text{Res} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}
\]

\[
\text{Res} = A \mid B \quad \% \text{ Logical OR}
\]

\[
\text{Res} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}
\]

\[
\text{Res} = \text{xor}(A, B) \quad \% \text{ Exclusive OR}
\]

\[
\text{Res} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]

\[
\text{Res} = \neg B \quad \% \text{ Negation}
\]

\[
\text{Res} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}
\]
Relational Operations

Operations:

- greater than (>)
- less than (<)
- greater or equal (≥)
- less or equal (≤)
- equal (==)
- not equal (∼=)

always **Field operators**

result: logical values, logical arrays
Logical Operations (Examples)

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

$$B = \begin{bmatrix} 0 & 5 \\ 0 & 1 \end{bmatrix}$$

$$Res = A > B \quad \% \text{ greater}$$

$$Res = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$Res = A < B \quad \% \text{ less or equal}$$

$$Res = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$Res = A == B \quad \% \text{ equal}$$

$$Res = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

$$Res = A \neq B \quad \% \text{ not equal}$$

$$Res = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$
Logical and Numerical Indexing

Two types of indexing:

- **with integer values**

  \[
  A = \begin{bmatrix}
  1 & -3 \\
  0 & 0 
  \end{bmatrix}
  \]

  \[
  a = A(2,1) \quad \text{% access to element (2,1)}
  \]

  \[
  a = 0
  \]

  \[
  z = A(1:2,1) \quad \text{% access to elements (1,1) and (2,1)}
  \]

  \[
  z = \begin{bmatrix}
  1 \\
  0 
  \end{bmatrix}
  \]

- **with logical values**

  \[
  \text{vect} = [-2, 3, -4, 5, 1]
  \]

  \[
  \text{select} = \text{vect} > 2 \quad \text{% check elements > 2}
  \]

  \[
  \text{select} = \begin{bmatrix}
  0 & 1 & 0 & 1 & 0
  \end{bmatrix}
  \]

  \[
  \text{whos}
  \]

  \[
  \begin{array}{lllll}
  \text{Name} & \text{Size} & \text{Bytes} & \text{Class} \\
  \text{select} & 1x5 & 5 & \text{logical} \\
  \text{vect} & 1x5 & 40 & \text{double}
  \end{array}
  \]

  \[
  \text{pick} = \text{vect} (\text{select}) \quad \text{% select by logical indexing}
  \]

  \[
  \text{pick} = \begin{bmatrix}
  3 \\
  5
  \end{bmatrix}
  \]
Consider the following matrix:

\[
C = \begin{pmatrix}
1 & 2 & 3 & 4 & 10 \\
-22 & 1 & 11 & -12 & 4 \\
8 & 1 & 6 & -11 & 5 \\
18 & 1 & 11 & 6 & 4
\end{pmatrix}.
\]

Using appropriate relational operators set all entries \( > 10 \) and \( < -10 \) of the matrix \( C \) to 0.

**Hint:** first try to realize the comparisons with \( > 10 \) and \( < -10 \) and then use the results to set the entries to 0 with appropriate field operations.

Consider the matrix:

\[
D = \begin{pmatrix}
7 & 2 & 3 & 10 \\
-2 & -3 & 11 & 4 \\
8 & 1 & 6 & 5 \\
18 & 1 & 11 & 4
\end{pmatrix}.
\]

Using logical fields select the diagonal of \( D \) and save it to a vector called \texttt{diag}. 
**Act as field operators:**

\[
\gg \quad t = (0:1:4) \quad % \text{call with a vector}
\]

\[
\begin{align*}
t &= \\
0 & 1 2 3 4 \\
\end{align*}
\]

\[
\gg \quad s = \sin(t) \quad % \text{values of } \sin(t)
\]

\[
\begin{align*}
s &= \\
0 & 0.8415 0.9093 0.1411 -0.7568 \\
\end{align*}
\]

\[
\gg \quad \text{mess} = [25.5 \ 16.3 \ 18.0; \ \ldots] \quad % \text{call with a matrix}
\]

\[
\begin{align*}
\text{mess} &= \\
2.0 & 6.9 3.0; \ \ldots \\
0.05 & 4.9 1.1; \\
\end{align*}
\]

\[
\gg \quad \text{dBmess} = 20 \times \log10(\text{mess})
\]

\[
\begin{align*}
\text{dBmess} &= \\
28.1308 & 24.2438 25.1055 \\
6.0206 & 16.7770 9.5424 \\
-26.0206 & 13.8039 0.8279 \\
\end{align*}
\]

**call equivalent to loops like:**

\[
\begin{align*}
\text{double } s[6]; \\
\text{for}(i = 0; i < 6; i++) \\
\quad \{ \\
\quad \quad s[i] = \sin(i); \\
\quad \}
\end{align*}
\]

\[
\begin{align*}
\text{double } \text{dBmess}[3,3]; \\
\text{for}(i = 0; i < 4; i++) \\
\quad \{ \\
\quad \quad \text{for}(k = 0; k < 4; k++) \\
\quad \quad \quad \{ \\
\quad \quad \quad \quad \text{dBmess}[i,k] = 20 \times \log10(\text{mess}[i,k]); \\
\quad \quad \quad \}
\quad \}
\end{align*}
\]
8. For a time vector between 0 and 10 with equidistant spacing of 0.1 calculate the values of the signal (function):

\[ s(t) = \sin(2\pi 5t) \cos(2\pi 3t) + e^{-0.1t}. \]

9. For a time vector between 0 and 10 with equidistant spacing of 0.1 calculate the values of the signal (function):

\[ s(t) = 20 \sin(2\pi 5t). \]

Then round the values first towards \( \infty \) and then towards 0. Find the appropriate functions using MATLAB’s help mechanisms.

In both cases print the first 6 values of \( s(t) \) together with their rounded values in a two row matrix and then interpret the somehow strange result.

10. Using the appropriate elementary mathematical MATLAB function, calculate the corresponding vector of binary and decadic logarithms for

\[ \vec{b} = \left( \begin{array}{ccccc} 1024 & 1000 & 100 & 2 & 1 \end{array} \right). \]
MATLAB’s plot facilities:

- using plot functions in interactive mode and in MATLAB programs
  - $x - y$-plots with `plot`, `stem`, `stairs`, ...
  - line style options
  - annotation functions
  - three-dimensional plots with `surf`, `mesh`, ...
  - plotting multiple functions with `subplot`

- using the *Plot Tools Window* in interactive mode
  - customizing plots interactively
  - documentation of plots
MATLAB’s $x - y$-Plot Commands

- Using standard plot commands

```matlab
≫ t=(0:1:5);≫
s=sin(t);≫
plot(t,s); % The simplest form≫

% plotting multiple signals≫
t=(0:0.01:2);≫
sinfkt=sin(2*pi*5*t);≫
cosfkt=2*cos(2*pi*3*t);≫
expfkt=exp(-2*t);≫
plot(t,[sinfkt; cosfkt; expfkt]);≫

% much better≫
plot(t,sinfkt,'k--', t, cosfkt, ... 
   'b--', t, expfkt, 'm.');≫
```

- Customizing line styles and annotation

```matlab
≫ t=(0:1:5);≫
s=sin(t);≫
plot(t,s,'mo'); % line style circles % color magenta≫

xlabel('time / s') % annotating x-axis≫
ylabel('ampl. / V') % and y-axis % figure title≫
title('A sine function')≫

grid % show grid≫
axis([0, 0.5, 0, 2]) % show only a section≫

% adding a text≫
plot(0.75,0,'\leftarrow signal',... 
   'FontSize',18)≫

% and much more ...
```
The Plot Tools Window
Multiple plots with `subplot`

```matlab
≫ % Example: plotting absolute value and argument of a complex function
≫ x = (0:0.01:1);
f = (1+j*x)./(j-2*x); % the complex function
≫ subplot(211) % setting upper plot window
≫ plot(x, abs(f), 'b-') % plot abs−value
≫ grid
≫ xlabel('x−values')
≫ ylabel('modulus of f(x)')
≫ subplot(212) % setting lower plot
≫ plot(x, angle(f), 'r-') % plot argument
≫ grid
≫ xlabel('x−values')
≫ ylabel('phase angle of f(x)')
```

Advantage:
- plot of multiple functions with same argument but different unity for function values
Exercises

11 Insert the following MATLAB commands to the command window and try to interpret the somehow strange graphical result:

```
≫ t = (0:0.5:10);
≫ sinfkt = sin(2*pi*5*t);
≫ cosfkt = 2*cos(2*pi*3*t);
≫ expfkt = exp(-2*t);
≫ plot(t,[sinfkt; cosfkt; expfkt])
```

12 Try to experiment with the MATLAB functions `semilogx`, `semilogy` and `loglog`. For this purpose define a frequency vector

\[ \omega = (0.01, 0.02, 0.03, 0.04, \ldots, 5) \text{ rad/s} \]

and try to plot modulus and phase angle of the the so called transfer functions

\[ H(\omega) = \frac{1}{j\omega} \]

and

\[ H(\omega) = \frac{1}{1 + j\omega}. \]

13 Plot the second transfer function above in a superposed way using a logarithmic scale for the frequency and for the modulus but not for the phase angle (this is called a Bode diagram).
MATLAB’s I/O-Operations

MATLAB’s I/O facilities:

- using I/O functions in interactive mode and in MATLAB programs
  - save workspace or variables with `save`
  - load or variables with `load`
  - the ASCII interface
  - specialized functions for many formats
- using the Import Data Tool in interactive mode
  - open the Import Data Tool with `uiimport`
  - open the Import Data Tool with menu or double click
Using your MATLAB editor create a vector and/or a matrix of real numbers and save it to a text file.
Delete all matrices of the workspace with `clear`.
Then, using the `load` command, try to input the contents of the text file into the workspace and analyze the result.

With MATLAB create a column vector of complex numbers. Save this vector in MATLAB’s binary format using the `save` command.
The delete the Workspace with `clear` and then re-input the contents of the saved file to MATLAB’s workspace using `load` or the Import Tool!
Compare the result with that of the previous exercise.

Search for an appropriate MATLAB function which may be used to insert the *.wav-file `Tada.wav` of the companion software to the workspace.
Then display the audio signal graphically. Take care that the time axis is annotated correctly.
Then multiply the signal by factor 10 and save it in a *.wav-file using a different file name.
Listen to the audio signals.
Useful commands for initialization in MATLAB programs:

- Initializing with zeros:
  ```matlab
  >> M = zeros(2,2);
  ```

- Initializing with ones:
  ```matlab
  >> K = ones(2,2);
  ```

- Initializing with a unit matrix:
  ```matlab
  >> E5 = eye(5);
  ```

- Initializing with an empty vector:
  ```matlab
  >> N = [];
  ```

- Determine length and size of arrays:
  ```matlab
  >> x1 = [1,2,3,4,5,6];
  >> k = length(x1);
  >> [n,m] = size(M);
  ```

- Determine last component:
  ```matlab
  >> last = zVec(end);
  ```

Reorganizing vectors and matrices:

- Transpose a matrix:
  ```matlab
  >> M = [1 2; 3 -j; -1 j+1];
  >> N = M';
  ```

- Transpose a matrix (field-operation):
  ```matlab
  >> M = [1 2; 3 -j; -1 j+1];
  >> N = M.';
  ```

- Reorganize as column vector:
  ```matlab
  >> mVec = M(:);
  ```

- Reorganize as matrix:
  ```matlab
  >> L = repmat(M,2,2);
  ```
Using MATLAB’s help mechanisms, find a function that turns
\[ \vec{y} = (1, 1.1, 1.2, 1.3, 1.4, \cdots, 9.8, 9.9, 10) \]

into
\[ \vec{z} = (10, 9.9, 9.8, 9.7, \cdots, 1.2, 1.1, 1). \]

Consider the vector
\[ \vec{z} = (1, 1.5, 2, \cdots, 98.5, 99, 99.5, 100). \]

Using logical operations and using `repmat`, form this vector construct a new vector \( \vec{w} \) containing every third component of \( \vec{z} \)!
% Script—File funkbsp
%
% Aufruf:  funkbsp
%
% A MATLAB—Script—File example
%
% Autor:  Prof. Dr. Ottmar Beucher
% HS Karlsruhe — Technik und Wirtschaft
% Version: 1.01
% Date: 14.11.2012

t=(0:0.01:2);
sinfkkt=sin(2*pi*5*t);
cosfkt=2*cos(2*pi*3*t);
expfkt=exp(-2*t);
plot(t,[sinfkt; cosfkt; expfkt])
xlabel('Time / s')
ylabel('Amplitude')
title('Three beautiful signals')

% launch in command window≫
funkbsp
Example:

```matlab
function [t, sinfkt, cosfkt, expfkt] = ...
    funkbsp2(f1, f2, damp)
%
% Function funkbsp2
%
% Call: see below
%
% First example of a MATLAB–Function

 t=(0:0.01:2);
 sinfkt=sin(2*pi*f1*t);
 cosfkt=2*cos(2*pi*f2*t);
 expfkt=exp(-damp*t);
 plot(t,[sinfkt;cosfkt;expfkt])
xlabel('Time / s')
ylabel('Amplitude')
title('Three beautiful signals')
```

Features:

- collect MATLAB commands to form a program unit capable of being parameterized
- are executable in Command Window with file name as command and with parameters
- may also use MATLAB’s help mechanism
- clear input/output-parameter interface
- act on their own variable stack!
MATLAB’s Function Calling Mechanism (Call by Value)

Call of $\text{thesum} = \text{funkbsp3}(v1,v2)$

1. workspace
   - $v1$
   - $v2$
   - copies $v1, v2$ into $a, b$

2. stack of function funkbsp3
   - $a$
   - $b$
   - executes commands of funkbsp3

3. copies sum in thesum
   - sum
   - (contains $a+b$)

4. the stack of function funkbsp3 is cancelled
   - a
   - (contains $a^2$)
   - to nirvana
19 Change the code of function \texttt{funkbsp3} in such way, that you may externally have access to the contents of componentwise squared variable \texttt{a}.

20 Change the code of function \texttt{funkbsp2} in such way, that you may additionally pass \texttt{colors} and \texttt{line styles} for the graphs of the signals. The function should then plot the signals in that way.
MATLAB as a Programming Language

- MATLAB as a Procedural Programming Language:
  
  **Control flow**:
  
  - `if` — Conditionally execute statements.
  - `else` statements.
  - `elseif` end
  - `for` — Repeat statements a specific number of times.
  - `while` — Repeat statements an indefinite number of times.
  - `...`
  - `switch` — Switch among several cases based on expressions.
  - `...`

- Features:
  
  - procedural programming approach
  - commands and constructs similar to other programming languages (like C/C++)
  - special commands for MATLAB functions with variable parameter lists
  - not treated here.

MATLAB as an OOP Language:

```matlab
classdef exampleclass
    properties
        % define class properties
    end
    methods
        % define class methods
    end
end
```
21 Write a MATLAB function that determines for a given vector of numbers the maximum of its entries using an appropriate loop construct. The vector should be the only input parameter, the found maximum the only output parameter.

22 Write a MATLAB function that determines for a given vector of numbers the positive entries using an appropriate loop construct. The given vector should be the only input parameter of the MATLAB function. The result (the positive entries) should be organized in a vector and this vector should be the output parameter of the MATLAB function.

23 Write a MATLAB function that plots the signal \( \sin(x) \) between 0 and \( 2\pi \) in a given color. The color should be the input parameter of the MATLAB function having the following strings as allowed values: ‘red’, ‘blue’, ‘green’, ‘magenta’. Use a switch...case construct to determine internally, according to the input parameter value, which color to use in the plot command.
• Function handles
  - a special kind of pointer:

  ```
  >> my_cos = @cos;
  whos
  Name      Size Bytes Class
  my_cos    1x1    16     function_handle
  value = cos(1)
  value =
    0.5403
  value = my_cos(1)
  value =
    0.5403
  ```

• Features:

  - a "pointer" to a MATLAB function
  - main usage:
    pass MATLAB functions to other MATLAB functions via parameter list!
Revisit the previous exercise (23) and enlarge the functionality of the program in a way that enables the user to pass a signal and the plot interval to the MATLAB function. Suppose that the signal is programmed in a separate MATLAB function or that it is a built-in MATLAB function.

The MATLAB function \texttt{fzero} calculates (using an appropriate numerical procedure) the root of a real valued function. This real function, realized as a MATLAB function, has to be passed to \texttt{fzero} as a \texttt{function handle}. Familiarize with the calling convention of \texttt{fzero} and calculate the zeros of example functions calling \texttt{fzero} in the \texttt{Command Window}. 
MATLAB’s ODE Solvers

Calling MATLAB’s ODE solvers:

```plaintext
≫ help ode23
ode23 Solve non-stiff differential equations, low order method.

[TOUT,YOUT] = ode23(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the system of differential equations y' = f(t,y) from time T0 to TFINAL with initial conditions Y0.

ODEFUN is a function handle.

For a scalar T and a vector Y, ODEFUN(T,Y) must return a column vector corresponding to f(t,y).
Each row in the solution array YOUT corresponds to a time returned in the column vector TOUT.
To obtain solutions at specific times T0, T1, ..., TFINAL, use TSPAN = [T0 T1 ... TFINAL].
...

See also ode45, ode113, ode15s, ode23s, ode23t, ode23tb, ode15i, odeset, odeplot, odephas3, odeprint, ...

Reference page in Help browser doc ode23
```

Features:

- solve initial value problems of type \( y' = f(t, y) \)
- solve ODEs of first order or systems of first order ODEs
- need a user-defined ODE function
- need a time range and a vector of initial values
- deliver numerical solution at internally defined supporting points
- are optimized by using step size steering
- Runge-Kutta-based algorithms for non-stiff and stiff ODEs
Example:

Mathematical model:

initial value problem of second order:

\[ \ddot{\alpha}(t) = -\frac{g}{l} \cdot \sin(\alpha(t)), \quad \alpha(0), \dot{\alpha}(0). \]

or equivalently:

\[ \dot{\alpha}_1(t) = \alpha_2(t), \]
\[ \dot{\alpha}_2(t) = -\frac{g}{l} \cdot \sin(\alpha_1(t)) \]

\[ \vec{\alpha}(0) = \begin{pmatrix} \alpha(0) \\ \dot{\alpha}(0) \end{pmatrix} = \begin{pmatrix} \alpha_1(0) \\ \alpha_2(0) \end{pmatrix}. \]
The ODE-file Structure

**Structure of an ODE-File (example):**

```matlab
function [alphadot] = pendgl(t, alpha)
% % Function pendgl
% % Call: only called by ODE-solvers
% % Example of an ODE-file for use with
% % ode23, ode45 a.s.o.
%
l = 10; % length of pendulum
g = 9.81; % grav. acceleration

%% Initialization
alphadot = [0; 0];

%% Representation of the first order ODEs
    % first equation
alphadot(1) = alpha(2);
    % second equation
alphadot(2) = -(g/l) * sin(alpha(1));
```

**Features:**

- represents the system of 1st order ODEs
- has a fixed parameter interface
- does'nt make sense as a function of it’s own! Is called only by MATLAB’s solvers
- has to be passed to MATLAB’s solvers by a function handle
How Solvers Use ODE-file’s Information

Initial values
(initialized during call of ode23)

memory area
for the solutions

memory area
for the derivatives of the solutions

(initialized by pendgl)

$\alpha_1(0)$
$\alpha_2(0)$

$\alpha_1(t_1)$
$\alpha_2(t_1)$

$\alpha_1(t_2)$
$\alpha_2(t_2)$

$\dot{\alpha}_1(t_1)$
$\dot{\alpha}_2(t_1)$

$\dot{\alpha}_1(t_2)$
$\dot{\alpha}_2(t_2)$
Resistor-Capacitor Lowpass

Example:

Mathematical model:

\[
\frac{d}{dt} u(t) = -\frac{1}{RC} u(t) + \frac{1}{RC} u_1(t), \quad u(0) = u_0.
\]
Exercises

28 Solve the initial value problem for the resistor-capacitor lowpass using different functions as input signals of the lowpass (instead of $u_1(t)$ take for instance a sinusoidal input).

29 Solve the problem:

$$\dot{y}_1(t) = -2y_1(t) - y_2(t) \quad y_1(0) = 1,$$
$$\dot{y}_2(t) = 4y_1(t) - y_2(t) \quad y_2(0) = 1.$$  

Compare the numerical solution with an analytical one (which you may calculate "by hand").

30 Solve the problem:

$$\ddot{y} + 3\dot{y} + 3y + y = u_1(t), \quad y(0) = 0, \dot{y}(0) = 1, \ddot{y}(0) = 1.$$  

The function $u_1(t)$ denotes the unit step function.
The Symbolic Math Toolbox

- Delivers Computer Algebra facilities
- two different interfaces

  - the MuPAD-Notebook: user interface with its own command syntax
  - the MATLAB-adaption: user interface with MATLAB’s command syntax
Symbolic Calculus Example

≫

```matlab
syms x y v
whos
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>1x1</td>
<td>60</td>
<td>sym</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1x1</td>
<td>60</td>
<td>sym</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>1x1</td>
<td>60</td>
<td>sym</td>
<td></td>
</tr>
</tbody>
</table>

```

≫

f = sin(x*y^2)*cos(v*x*y) % define a function

f =

```

sin(x*y^2)*cos(v*x*y)
```

≫

```
dfy = diff(f, 'y') % derivative with respect to y
dfy =
```

```
2*x*y*cos(x*y^2)*cos(v*x*y) - v*x*sin(x*y^2)*sin(v*x*y)
```

≫

```
dfv = diff(f, 'v') % derivative with respect to v
dfv =
```

```
-x*y*sin(x*y^2)*sin(v*x*y)
```

≫

```
pretty(f) % display expression "pretty"
```

```
2
sin(x y) cos(v x y)
```

Prof. Dr. Ottmar Beucher MATLAB and Simulink - Summer School NCHU, Taichung July 2013 45
Export Symbolic to Numeric

syms x % define symbolic variables
f = x^2*cos(x); % define function
df = diff(f, 'x'); % calculate symbolic derivative

dt = 0.1; % define sampling step size
t = (0:dt:10); % define sampling points
fnum = t.^2.*cos(t); % calculate function numerically

dfnum = diff(fnum)/dt; % differentiate function numerically
plot((dt:dt:10), dfnum, 'b--') % plot numerical derivative

hold
dfnum2 = subs(df, x, t); % convert symbolic solution to
plot(t, dfnum2, 'r') % a numerical solution with the
% subs-command and plot it

grid
xlabel('x')
ylabel('derivative')
31. Integrate the function

\[ g(x) = \sin(5x - 2) \]

two times symbolically.

32. Calculate the 3rd-order Taylor-polynomial in \( x_0 = 1 \) of the function \( g(x) \).

33. Using the symbolic math function \texttt{dsolve} calculate the solution of the following ordinary differential equation:

\[ \dot{y} = xy^2. \]
Example:  

Modelling the dynamics:

\[
y(t) = \frac{1}{C} \int_{0}^{t} i(\tau) \, d\tau,
\]

\[
y(t) = x(t) - i(t) \cdot R \quad \left(\leftrightarrow i(t) = \frac{1}{R}(x(t) - y(t))\right).
\]
The idea behind Simulink:

- **draw a block diagram of your dynamical system**
- **let Simulink convert it to a system of differential equations**
- **let Simulink solve this system of differential equations by using one of MATLAB’s numerical ODE solvers**
- **let Simulink display and save the signals (input signals, output signals (normally ODE solutions) and internal states**

Simulink may be used to:

- **directly model a given real dynamical system by its block diagram and simulate the dynamical behavior of that system**
- **solve a system of ODEs by converting the ODEs into a block diagram and simulating the resulting dynamical system**
Constructing a Simulink Model - The Library Browser

1. Search for blocks
2. Open new model
3. Block Library Tree
4. Commonly Used Blocks
5. Blocks of the Continuous Library

Prof. Dr. Ottmar Beucher
MATLAB and Simulink - Summer School NCHU, Taichung
July 2013
Constructing a Simulink Model - The Model Window

1. Model Browser
2. Model Construction Window
3. Parameterize models
4. Start and simulate models
5. Open and save models
Simulating a Simulink Model - The Parameter Panel

1. Select tree

2. Simulation time
   - Start time: 0.0
   - Stop time: 10.0

3. Solver options
   - Type: Variable-step
   - Max step size: auto
   - Min step size: auto
   - Initial step size: auto
   - Relative tolerance: 1e-3
   - Absolute tolerance: auto
   - Shape preservation: Disable All

4. Stepsize selection (depending on solver)

Tasking and sample time options
- Tasking mode for periodic sample times: Auto
- Automatically handle rate transition for data transfer
- Higher priority value indicates higher task priority

Zero-crossing options
- Zero-crossing control: Use local settings
- Time tolerance: 10^-128*eps
- Signal threshold: auto
- Number of consecutive zero crossings: 1000
Check the test system `sl_test1.slx` for different step sizes using a solver with internal step size control (variable step solvers). Compare different calculation times when using a step size of 0.00001 directly with the fixed step solver `ode3` on one hand and when using the variable step solvers `ode23` on the other hand.

Design a Simulink test system `sl_LsgDiff.slx` for the derivative block `Derivative`. The quickest way to do so is to alter the system `sl_test1.slx`. Then experiment like in the previous exercise.

Explain why the result of the test system `sl_test1.slx` represents the solution of the initial value problem

\[ \dot{y}(t) = x(t), \quad y(0) = 0. \]

Which one of the signals is \(x(t)\), which one is \(y(t)\)?
Dynamical Systems - Three Examples

Three Examples:

- **an undamped harmonic oscillation**
  \[
  \dot{y}(t) = -y(t), \quad y(0) = 1, \quad \dot{y}(0) = 0.
  \]

- **the resistor-capacitor lowpass**
  \[
  T \cdot \dot{y}(t) + \cdot y(t) = x(t), \quad T := \frac{1}{RC}
  \]

- **rapid mechanical oscillations in air and fluids**
  \[
  m\ddot{x}(t) + b \cdot \text{sgn}(\dot{x}(t)) \cdot \dot{x}^2(t) + cx(t) = 0.
  \]
37 Design a Simulink system to solve the nonlinear differential equation

\[ \ddot{\alpha}(t) = -\frac{g}{l} \cdot \sin(\alpha(t)), \quad \alpha(0) = a_0, \quad \dot{\alpha}(0) = a_1 \]

of the mathematical pendulum (i.e. to simulate the dynamical behavior of the mathematical pendulum).

38 Creating an appropriate Simulink system solve the following initial value problem of third order:

\[ \dddot{y}(t) + 3\ddot{y}(t) + 4\dot{y}(t) + 2y(t) = 0, \quad y(0) = 1, \quad \dot{y}(0) = 1, \quad \ddot{y}(0) = 0. \]

39 With an appropriate Simulink system solve the ODE-system

\[ \dot{y}_1(t) = -3y_1(t) - 2y_2(t), \quad y_1(0) = 1, \]
\[ \dot{y}_2(t) = 4y_1(t) + 2y_2(t), \quad y_2(0) = 1. \]

Compare the numerical Simulink solution with an exact one which you may calculate by hand or using the Symbolic Math Toolbox.
Methods: The function block (Fcn)

![Function Block Parameters: Fcn](image)

- General expression block. Use "u" as the input variable name.
  - Example: \( \sin(u(1) \exp(2.3 \cdot (-u(2)))) \)

- Expression:
  - \( \sin(u(1) \exp(2.3 \cdot (-u(2)))) \)

- Sample time (-1 for inherited):
  - -1
Methods:

Creating Subsystems

System zur Simulation eines Mechanischen Schwingers bei Newton-Reibung
(System zur Lösung einer nichtlinearen Differentialgleichung 2. Ordnung)
40 Simplify the Simulink system for the solution of exercise (39) by defining appropriate \texttt{subsystem} blocks.

41 Create a Simulink system for the solution of exercise (39) using appropriate \texttt{Fcn} blocks.
# Interaction with MATLAB - an Overview

<table>
<thead>
<tr>
<th>Simulink $\Rightarrow$ MATLAB</th>
<th>MATLAB $\Rightarrow$ Simulink</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>Define variables in the workspace and use them as system- or block parameters</td>
</tr>
<tr>
<td><strong>Scope</strong> block</td>
<td>—</td>
</tr>
<tr>
<td><em>(History - Save data to workspace)</em></td>
<td></td>
</tr>
<tr>
<td><strong>To Workspace</strong> block</td>
<td><strong>From Workspace</strong> block</td>
</tr>
<tr>
<td>Model Configuration Parameters <em>(Data Import/Export panel)</em></td>
<td>Model Configuration Parameters <em>(Data Import/Export panel)</em></td>
</tr>
<tr>
<td>– Save to workspace</td>
<td>– Load from workspace</td>
</tr>
<tr>
<td>• time vector <em>(Time)</em></td>
<td>• time vector <em>(Input)</em></td>
</tr>
<tr>
<td>• Export via <strong>Output</strong> <em>(Output)</em></td>
<td>• Import via <strong>Input</strong> <em>(Input)</em></td>
</tr>
<tr>
<td>Return value of MATLAB function <strong>sim</strong></td>
<td>Parameter of MATLAB function <strong>sim</strong></td>
</tr>
<tr>
<td>MATLAB function <strong>get_param</strong></td>
<td>MATLAB function <strong>set_param</strong></td>
</tr>
</tbody>
</table>
Interaction with MATLAB - the Config Parameters Panel

1. **Input from workspace via import**

2. **Export time to workspace**

3. **Output to workspace via output**

4. **Export as signal object**

5. **Export as signal object**
Design a Simulink system to solve the linear differential equation

\[ m\ddot{y}(t) + 2\dot{y}(t) + 4y(t) = F(t), \quad y(0) = 1, \ \dot{y}(0) = 0. \]

This equation models the movement of an oscillating object with mass \( m \) disturbed by an external force \( F(t) \).

In the first experiment use a step function to model the force \( F(t) \). The change from 0 (no disturbing force) to \( c \) (constant disturbing force) should occur at time \( t_0 > 0 \).

Use workspace variables to parameterize the system.

Solve the problem of the previous exercise (42) once again by creating a new Simulink system that inputs the disturbing external force \( F(t) \) with a From Workspace block and that outputs the result with a To Workspace block.

Solve the problem of the previous exercise (42) once more by creating a new Simulink system that inputs the disturbing external force \( F(t) \) via an Inport and that outputs the result with an Outport.
Interaction with MATLAB - Data Exchange with Functions

- **Set parameters of a Simulink system with** `set_param`:

  ```matlab
  >> % Example: set parameter StopTime of system sl_Pendel to 30
  >> set_param( 'sl_Pendel', 'StopTime', '30')
  >>
  % Example: set parameter Gain of Block Gain in system sl_Pendel to -1
  >> set_param( 'sl_Pendel/Gain', 'Gain', '-1')
  >>
  % Example: set parameter Gain of Block Gain in system sl_Pendel to -g/l
  g = 9.81; % gravity acceleration
  l = 25; % length of pendulum
  >> set_param( 'sl_Pendel/Gain', 'Gain', num2str(-g/l));
  ```

- **Get parameters of a Simulink system with** `get_param`:

  ```matlab
  >> % Example: get parameter StopTime of system sl_Pendel
  >> get_param( 'sl_Pendel', 'StopTime')
  >>
  % Example: get parameter Gain of Block Gain in system sl_Pendel
  >> set_param( 'sl_Pendel/Gain', 'Gain');
  >>
  % Example: get initial condition parameter of the integrator
  % block in system sl_Pendel
  >> set_param( 'sl_Pendel/Integrator1', 'InitialCondition');
  ```

- **Note**: This works also from within MATLAB functions!
Revisit the Simulink system of the previous exercise (42) and alter the parameters via `set_param` before starting it.

Launch the Simulink system `sl_Pendel2.slx` of the companion software. This system simulates the mathematical pendulum. Alter the parameter `pendulum length` via `set_param` to 5 m and then to 8 m before starting the simulation.
The `sim` command

- simulates (an opened) Simulink system
- may parameterize the Simulink system via its parameter list
- returns simulation data via a return parameter (the `Simulink.SimulationOutput` object)
- logged signals (see `Data Import/Export` panel) may be included in the `SimulationOutput` object
The *sim* Command - the Config Parameters Panel

- **Select:**
  - Solver
  - Data Import/Export
  - Optimization
  - Diagnostics
  - Hardware Implementation
  - Model Referencing
  - Simulation Target

- **Load from workspace:**
  - Input: \([t, u]\)
  - Initial state: \(x_{\text{Initial}}\)

- **Save to workspace:**
  - Time, State, Output
    - Time: \(t\)
    - States: \(x_{\text{out}}\)
    - Output: \(y_{\text{out}}\)
    - Final states: \(x_{\text{Final}}\)
  - Export time to Simulink.Simulation.Output object via outport
    - Format: Array
    - Limit data points to last: 1000
    - Decimation: 1
    - Save complete SimState in final state

- **Signals:**
  - Signal logging: \(\text{logsout}\)
  - Signal logging format: Dataset

- **Data Store Memory:**
  - Data stores: \(\text{dsmout}\)

- **Save options:**
  - Save simulation output as single object
  - Record and inspect simulation output

**Export to Simulink.Simulation.Output object via outport**

**Output to Simulink.Simulation.Output object via outport**
Automize Simulink simulations with `sim` - an example:

```matlab
function [zeit, simuerg] = simPendel2(pendellaenge)

% Open Simulink system if it's not open
open('sl_Pendel2.slx');

% Set Gain block parameters

g = 9.81; % gravity acceleration

% update with set_param
set_param('sl_Pendel2/Gain', 'Gain', num2str(-g / pendellaenge));

% Simulate Simulink system with sim
% make sure that sim returns a Simulink.SimulationOutput object
% (if sim is launched only with the Simulink system name
% only the time vector will be returned)

simOut = sim('sl_Pendel2', 'SaveOutput', 'on');

% Access the results with the get method
% and save them into the output variables
% of function simPendel2

zeit = simOut.get('t');
simuerg = simOut.get('yout');
```

**Note:** the system `sl_Pendel2` may be launched several times this way, using a `for` loop and saving the results in matrices or cell arrays!!
47 Write a MATLAB function with which you may simulate the Simulink system sl_Pendel2.slx for a given vector of pendulum lengths.

48 Design a Simulink system that creates sinusoidal signals using the sine generator block Sine Wave. Then write a MATLAB function that is able to launch this Simulink system several times varying the frequency of the sine wave.

49 Design a Simulink system that solves the initial value problem

\[ \ddot{y}(t) + \dot{y}(t) + y(t) = f(t), \quad y(0) = 0, \dot{y}(0) = 0. \]

The function \( f(t) \) should be a step function (use block Step) with a step from 0 to \( h \). Then write a MATLAB function that is able to launch this Simulink system several times varying the parameter \( h \).
Look Up Tables

Parameter panel for 1-D Lookup Table block

Parameter panel for 1-D Lookup Table block

Lookup Table blocks
Tension-current characteristic of a solar cell:
Consider the following characteristic:

\[ k(x) = \begin{cases} 
0 & \text{für } x < 0, \\
\sqrt{x} & \text{für } x \in [0, 4], \\
2 & \text{für } x > 4.
\end{cases} \]

Design a Simulink system that realizes this characteristic with a 1-D Lookup Table. Input a sine wave signal to the Lookup Table and analyze its output signal. Experiment with different amplitudes of the sine wave.

Consider the following two-dimensional characteristic:

\[ k(x, y) = x^2 + y^2 \quad \text{für } x, y \in [-1, 1]. \]

Design a Simulink system that realizes this characteristic with a 2-D Lookup Table. Then activate the characteristic at points \((x, x)\) with \(x \in [-1, 1]\). For this purpose design an appropriate input signal which you should read from workspace with a From Workspace block.

Concerning the parameterization of the 2-D Lookup Table compare the steps needed for creating three dimensional plot (e.g. the function meshgrid)!