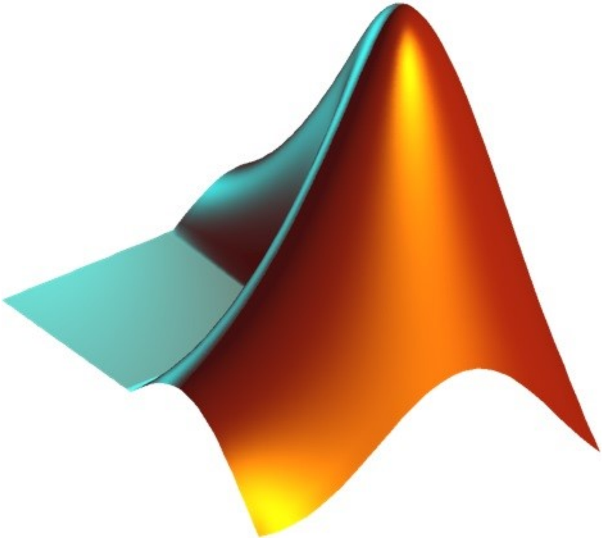
Practical Course MATLAB/SIMULINK

MATLAB Fundamentals



Objectives & Preparation " MATLAB Fundamentals "

- Which MathWorks products are covered?
  - ⇨ MATLAB

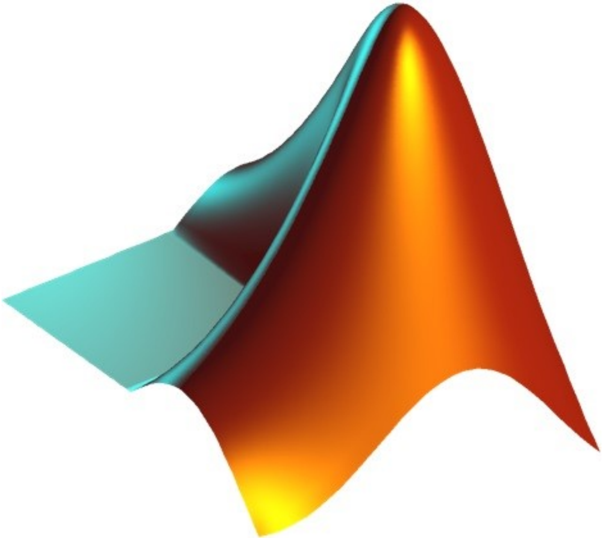
- What skills are learnt?
  - ⇨ MATLAB interfaces – how to get around
  - ⇨ Help & documentation
  - ⇨ Basic coding skills (variables, expressions, code structures…)
  - ⇨ Debugging

- How to prepare for the session?
  - ⇨ MathWorks Tutorials:
    - – https://matlabacademy.mathworks.com/R2017b/portal.html?course=mlbe
    - – https://matlabacademy.mathworks.com/R2017b/portal.html?course=gettingstarted

Outline

1. Introduction
2. Graphical User Interface
3. MATLAB help
4. Variables and Expressions
    1. Commands and Assignmen
    2. Arrays, Vectors and Matric
    3. Data Types
5. Scripts and Functions
6. Debugging
7. List of Useful Commands

1. Introduction



Introduction

- **MAT**rix **LAB**oratory is a numerical computing environment and fourth-generation programming language

- Developed by **Cleve Moler**, chairman of the computer science department at the University of New Mexico, in the late 1970s

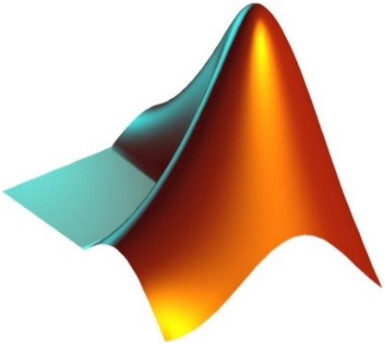
www.mathworks.de

- Initially designed to give students **easy access** to the software libraries **LINPACK** (numerical linear algebra) and **EISPACK** (numerical computation of eigenvalues and eigenvectors)

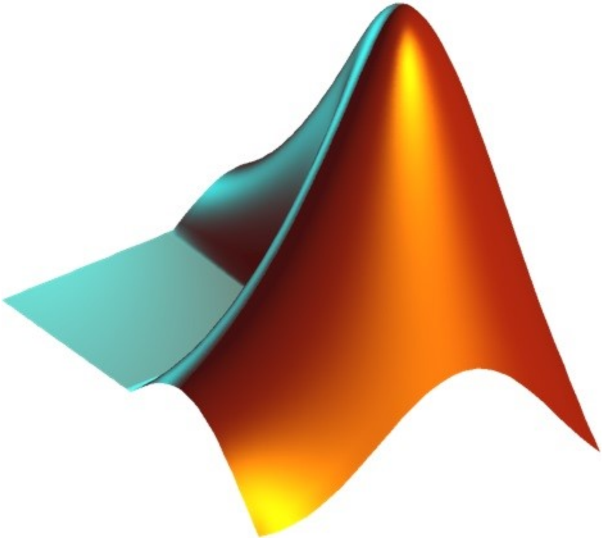
- Recognizing the commercial potential, the engineer **Jack Little** joined Moler along with **Steve Bangert** and founded **The MathWorks**

- Today, MathWorks has over **3500 employees** and a yearly **revenue of approximately $1.05 billion**

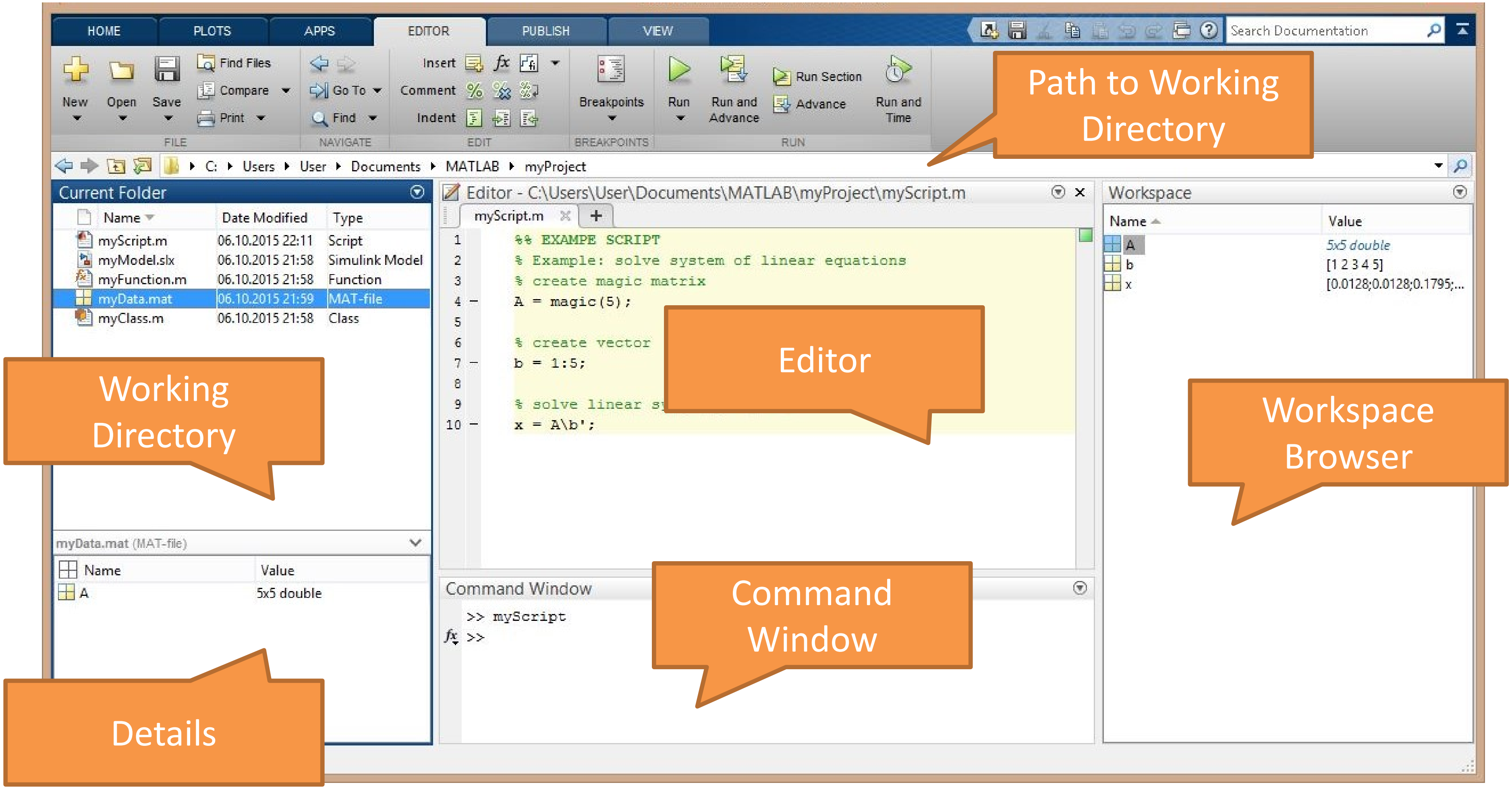
- MATLAB logo displays **L-shaped membrane** from Moler‘s PhD thesis



2. Graphical User Interface



MATLAB Default Layout



Name	Date Modified	Type
myScript.m	06.10.2015 22:11	Script
myModel.slx	06.10.2015 21:58	Simulink Model
myFunction.m	06.10.2015 21:58	Function
myData.mat	06.10.2015 21:59	MAT-file
myClass.m	06.10.2015 21:58	Class

```
%% EXAMPE SCRIPT
% Example: solve system of linear equations
% create magic matrix
A = magic(5);

% create vector
b = 1:5;

% solve linear sy
x = A\b';
```

Name	Value
A	5x5 double
b	[1 2 3 4 5]
x	[0.0128;0.0128;0.1795;...

Name	Value
A	5x5 double

```
>> myScript
>>
```

Path to Working Directory

Editor

Working Directory

Workspace Browser

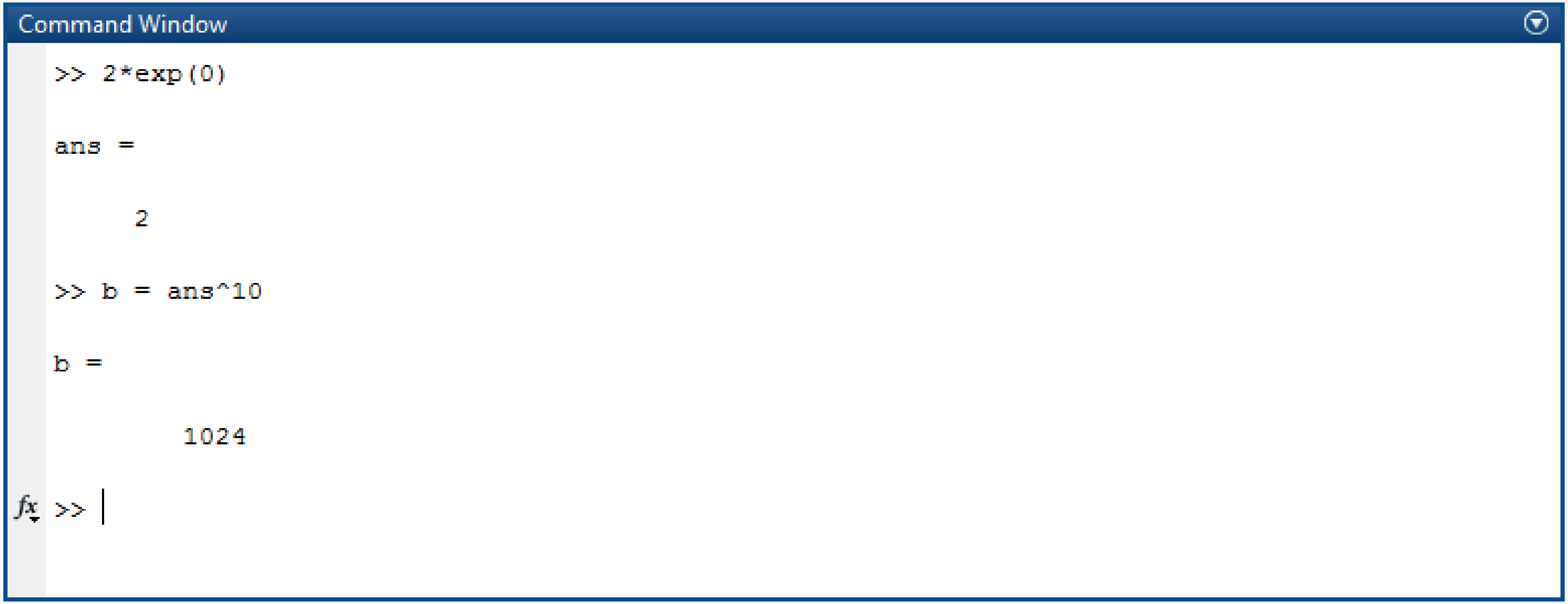
Command Window

Details

Graphical User Interface

Command window

- Used to directly type and execute commands
- Displays function return



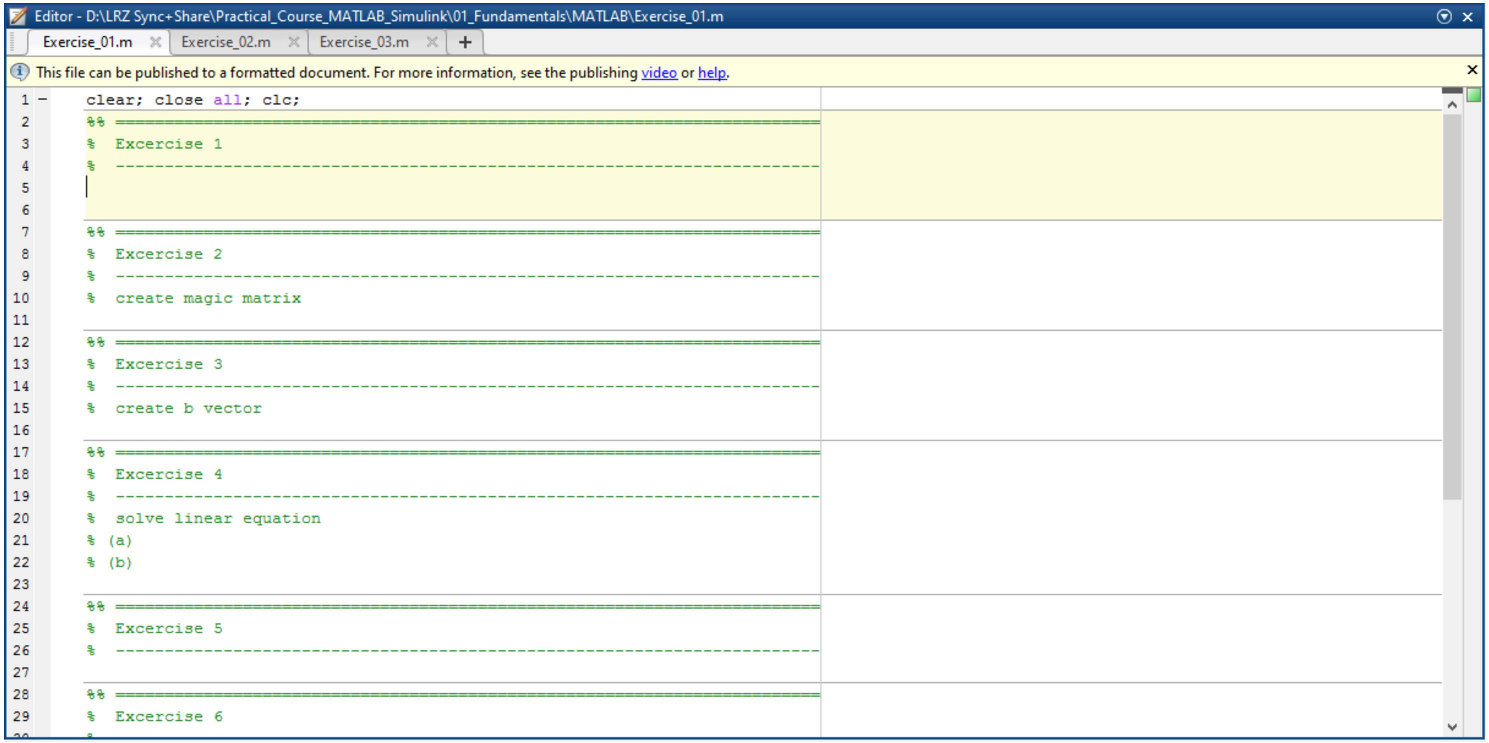
```
>> 2*exp(0)

ans =

     2

>> b = ans^10

b =

        1024

>> 
```

Graphical User Interface

Editor

Used to open, edit and save programs (e.g. scripts and functions)

```
>> edit Excercise_01.m

>>
```



Editor - D:\LRZ Sync+Share\Practical_Course_MATLAB_Simulink\01_Fundamentals\MATLAB\Exercise_01.m
Exercise_01.m
Exercise_02.m
Exercise_03.m
+

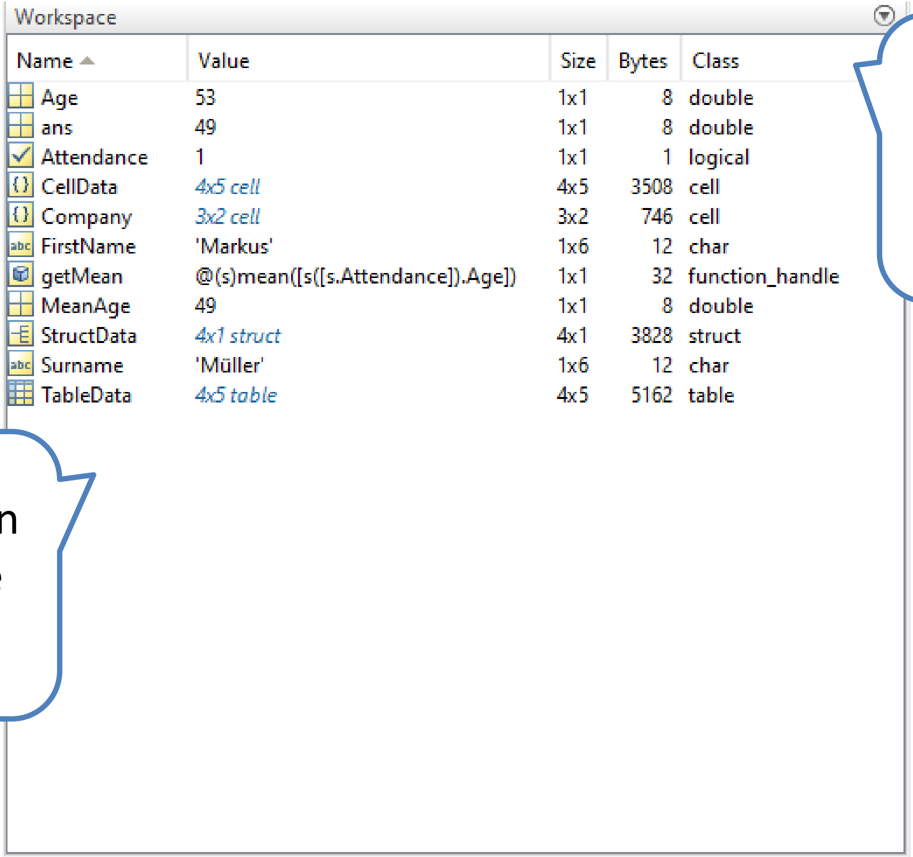
This file can be published to a formatted document. For more information, see the publishing video or help.

```
clear; close all; clc;
%% ========================================================================
%  Excercise 1
%  ------------------------------------------------------------------------


%% ========================================================================
%  Excercise 2
%  ------------------------------------------------------------------------
%  create magic matrix

%% ========================================================================
%  Excercise 3
%  ------------------------------------------------------------------------
%  create b vector

%% ========================================================================
%  Excercise 4
%  ------------------------------------------------------------------------
%  solve linear equation
% (a)
% (b)

%% ========================================================================
%  Excercise 5
%  ------------------------------------------------------------------------

%% ========================================================================
%  Excercise 6
```

Graphical User Interface

Workspace Browser

Used to view and edit variables in the current workspace



Workspace

Name ▲	Value	Size	Bytes	Class
Age	53	1x1	8	double
ans	49	1x1	8	double
Attendance	1	1x1	1	logical
CellData	*4x5 cell*	4x5	3508	cell
Company	*3x2 cell*	3x2	746	cell
FirstName	'Markus'	1x6	12	char
getMean	@(s)mean([s([s.Attendance]).Age])	1x1	32	function_handle
MeanAge	49	1x1	8	double
StructData	*4x1 struct*	4x1	3828	struct
Surname	'Müller'	1x6	12	char
TableData	*4x5 table*	4x5	5162	table

Different properties of each variable can be displayed by right clicking the header row

The variable editor can be opened by double clicking a variable

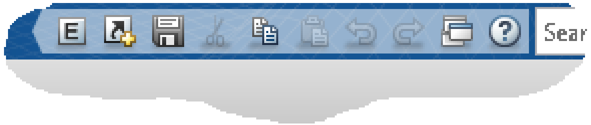
Graphical User Interface

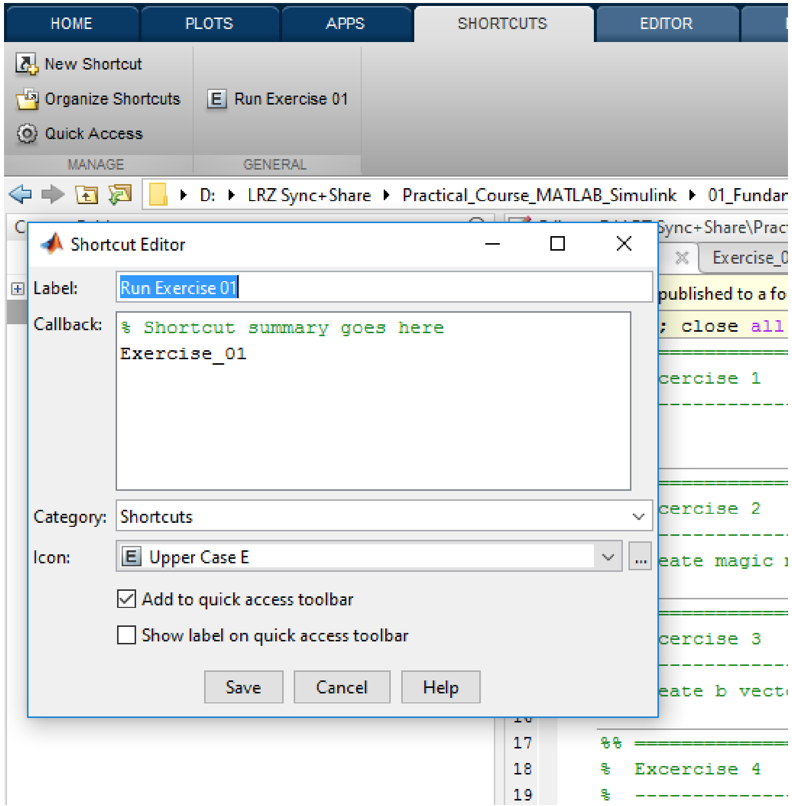
Shortcuts

You can create shortcuts to rerun commands that are used often. Some examples may be:

- `format compact`
- `clear`
- `workspace`
- `filebrowser`
- `clc`

Create shortcuts by selecting “New Shortcut” from the SHORTCUTS ribbon or the quick access toolbar.





Label: Run Exercise 01

Callback:

```
% Shortcut summary goes here
Exercise_01
```

Category: Shortcuts

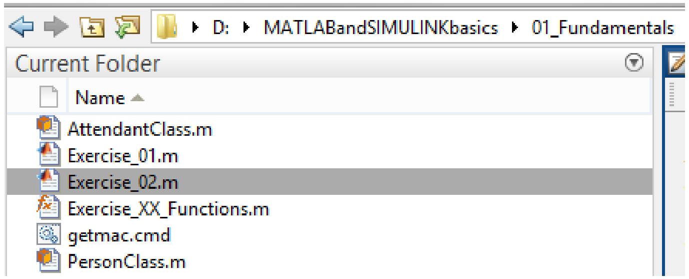
Icon: Upper Case E

☑ Add to quick access toolbar

□ Show label on quick access toolbar

Graphical User Interface

Current Working Directory


D: ▸ MATLABandSIMULINKbasics ▸ 01_Fundamentals

Current Folder

Name

AttendantClass.m
Exercise_01.m
Exercise_02.m
Exercise_XX_Functions.m
getmac.cmd
PersonClass.m

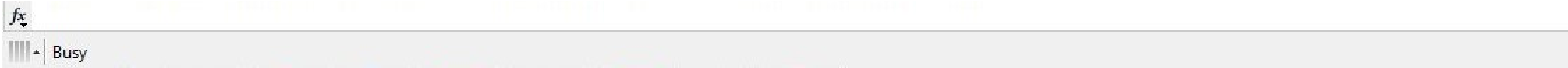
- Current Working Directory contain active files, that can be called from a program
- The Current Folder Window gives an overview of the current working directory
- Change current working directory:
  - Interactively
  - By using the cd command
- Additional folders can be added to the MATLAB search path
  - Interactively by right clicking the folder
  - By using the addpath command

```
>> cd ..
>> addpath Matlab
```

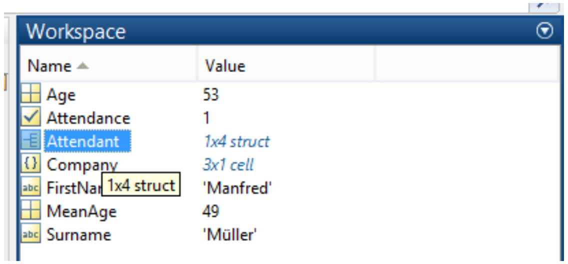
Graphical User Interface

MATLAB Status

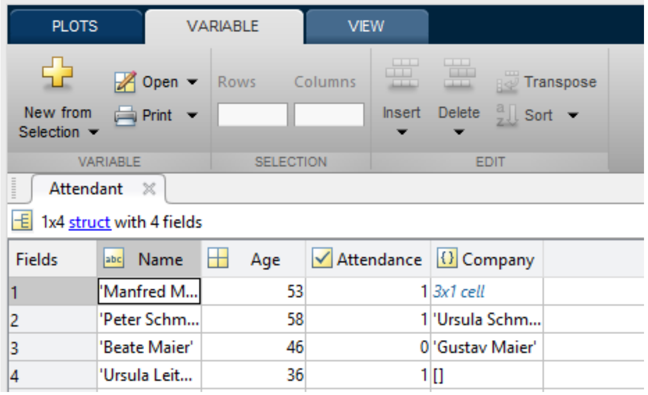
Current status is displayed in the MATLAB status bar



Variable editor


Workspace

Name	Value
Age	53
Attendance	1
Attendant	1x4 struct
Company	3x1 cell
FirstNa 1x4 struct	'Manfred'
MeanAge	49
Surname	'Müller'



1x4 struct with 4 fields

Fields	Name	Age	Attendance	Company
1	'Manfred M...	53	1	3x1 cell
2	'Peter Schm...	58	1	'Ursula Schm...
3	'Beate Maier'	46	0	'Gustav Maier'
4	'Ursula Leit...	36	1	[]

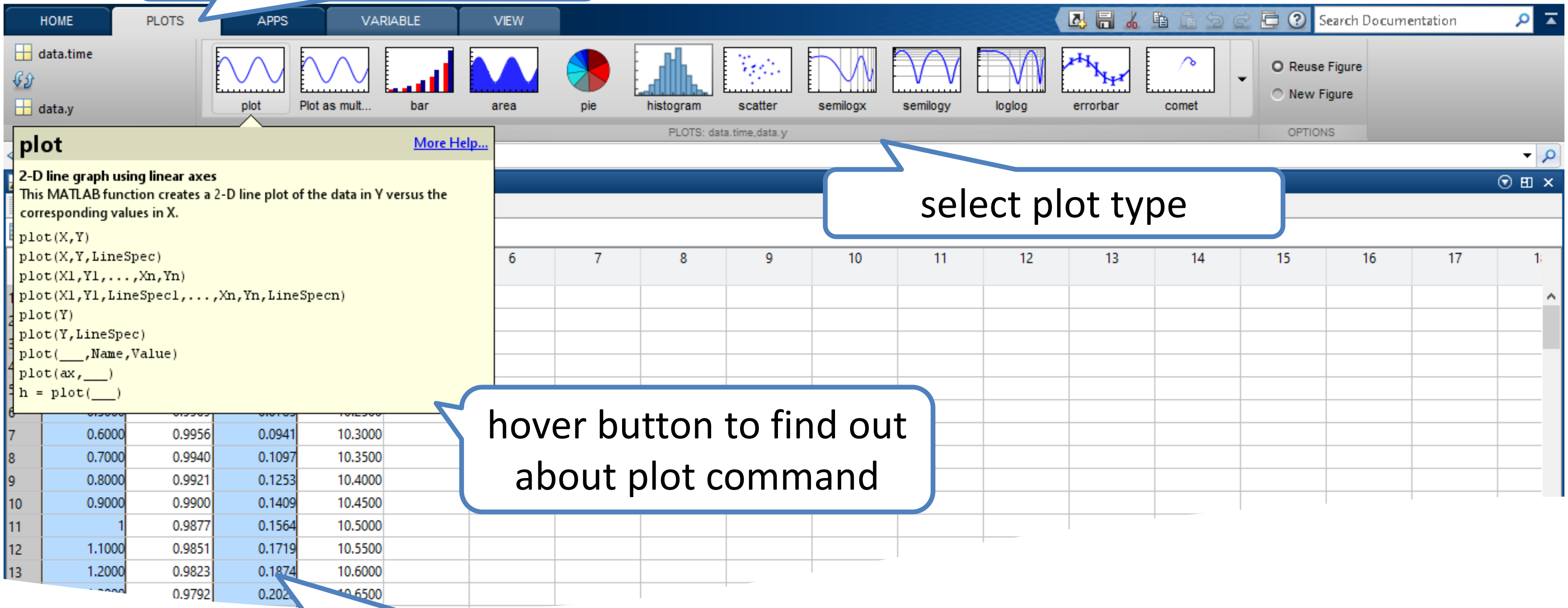
- Inspect and edit variable in the MATLAB workspace
- Open
  - Interactively by double clicking the variable
  - By using the openvar command

```
>> openvar Attendant
```

Graphical User Interface

Variable editor

Data can be plotted interactively from the Variable editor



choose PLOTS ribbon

select plot type

plot More Help...

2-D line graph using linear axes

This MATLAB function creates a 2-D line plot of the data in Y versus the corresponding values in X.

```
plot(X,Y)
plot(X,Y,LineSpec)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn)
plot(Y)
plot(Y,LineSpec)
plot(___,Name,Value)
plot(ax,___)
h = plot(___)
```

7	0.6000	0.9956	0.0941	10.3000
8	0.7000	0.9940	0.1097	10.3500
9	0.8000	0.9921	0.1253	10.4000
10	0.9000	0.9900	0.1409	10.4500
11	1	0.9877	0.1564	10.5000
12	1.1000	0.9851	0.1719	10.5500
13	1.2000	0.9823	0.1874	10.6000
	[illegible]	0.9792	0.202	10.6500

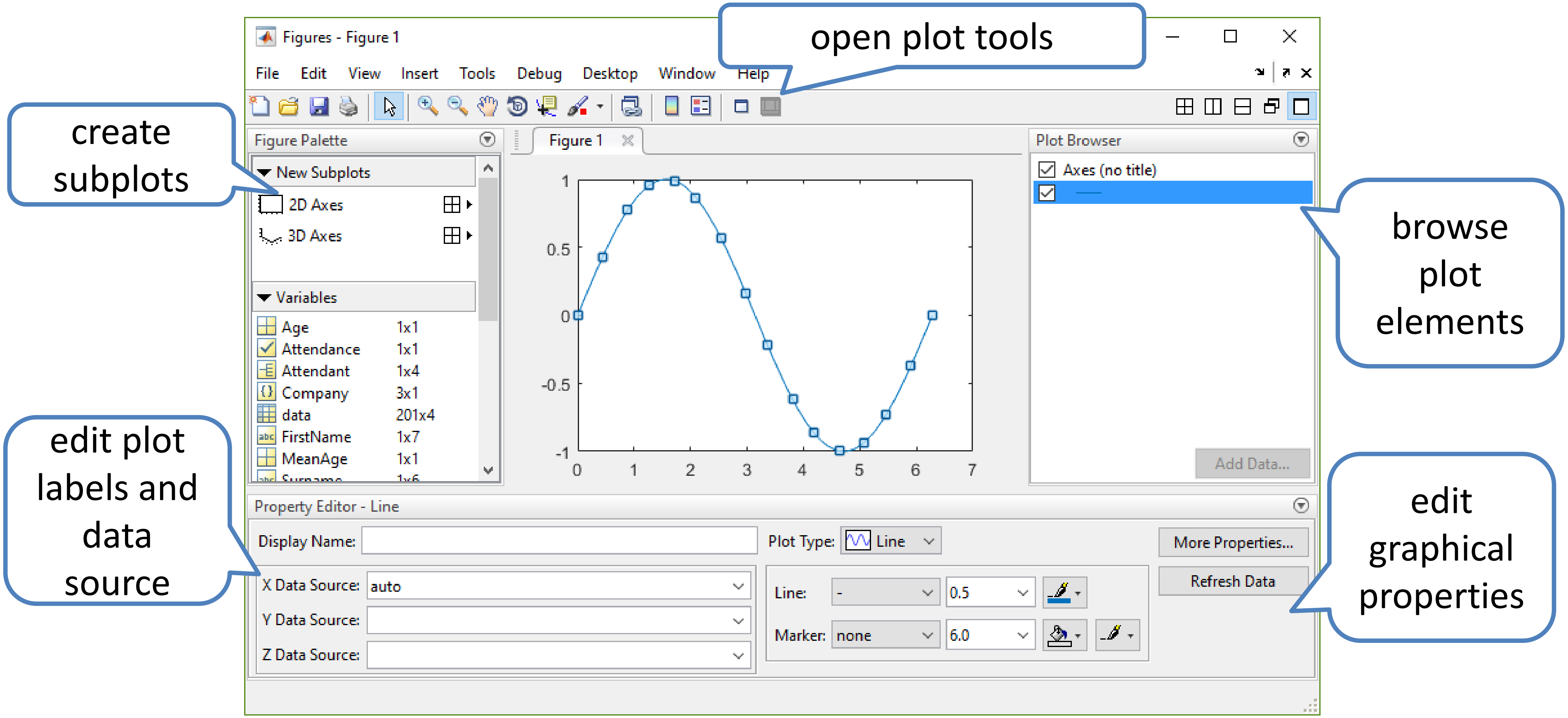
hover button to find out about plot command

choose columns to plot

Graphical User Interface

Plot Tools

Plots can be edited interactively using the plot tools



Age	1x1
Attendance	1x1
Attendant	1x4
Company	3x1
data	201x4
FirstName	1x7
MeanAge	1x1
Surname	1x6

Display Name:

Plot Type: Line

X Data Source: auto

Y Data Source:

Z Data Source:

Line: - 0.5

Marker: none 6.0

open plot tools

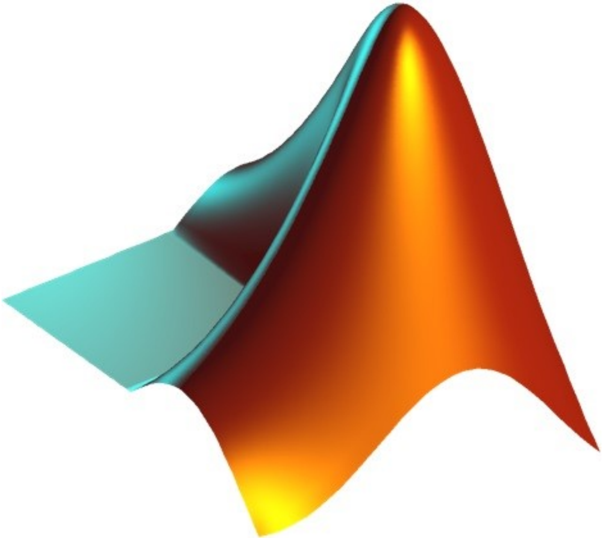
create subplots

browse plot elements

edit plot labels and data source

edit graphical properties

3. MATLAB Help



MATLAB Help

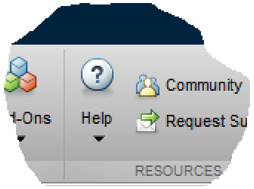
Help is one of the most important features in MATLAB. There are several ways to access help

To open the help browser

- Use the doc command

```
>> doc mean
```

- Or click the Help Button from the HOME ribbon

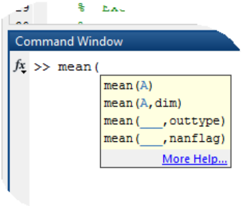


To view quick help

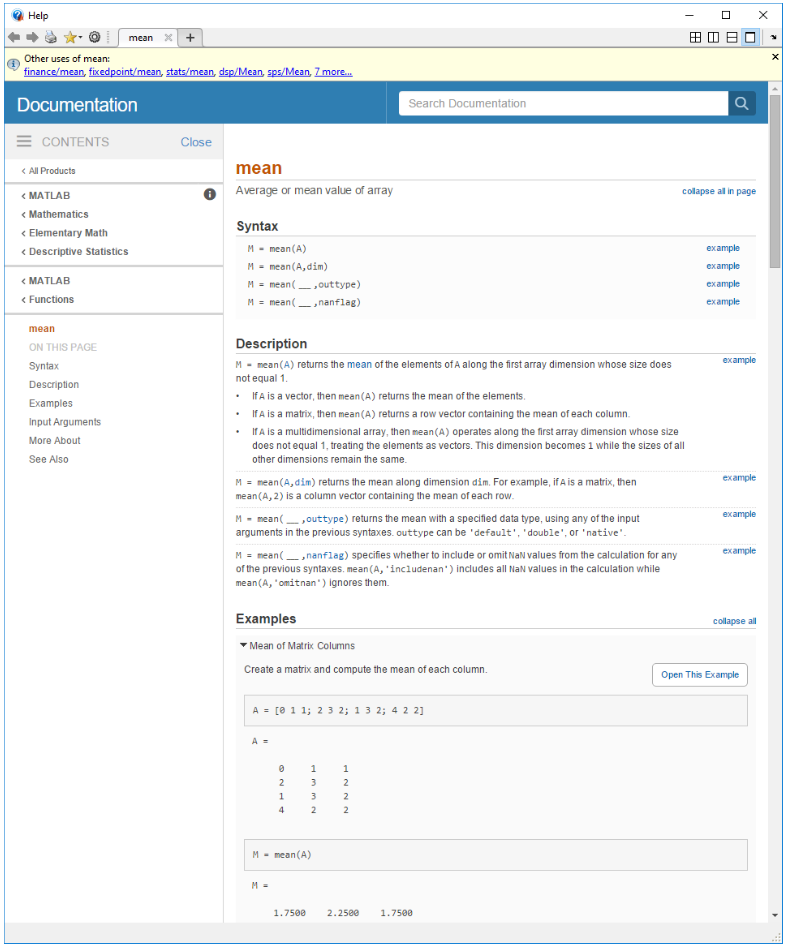
- use the help command

```
>> help mean
```

- or just start typing the command



```
>> mean(
mean(A)
mean(A,dim)
mean(___,outtype)
mean(___,nanflag)
```



mean

Average or mean value of array

Syntax

```
M = mean(A)
M = mean(A,dim)
M = mean(__,outtype)
M = mean(__,nanflag)
```

Description

`M = mean(A)` returns the mean of the elements of A along the first array dimension whose size does not equal 1.

- If A is a vector, then `mean(A)` returns the mean of the elements.
- If A is a matrix, then `mean(A)` returns a row vector containing the mean of each column.
- If A is a multidimensional array, then `mean(A)` operates along the first array dimension whose size does not equal 1, treating the elements as vectors. This dimension becomes 1 while the sizes of all other dimensions remain the same.

`M = mean(A,dim)` returns the mean along dimension `dim`. For example, if A is a matrix, then `mean(A,2)` is a column vector containing the mean of each row.

`M = mean(__,outtype)` returns the mean with a specified data type, using any of the input arguments in the previous syntaxes. `outtype` can be `'default'`, `'double'`, or `'native'`.

`M = mean(__,nanflag)` specifies whether to include or omit NaN values from the calculation for any of the previous syntaxes. `mean(A,'includenan')` includes all NaN values in the calculation while `mean(A,'omitnan')` ignores them.

Examples

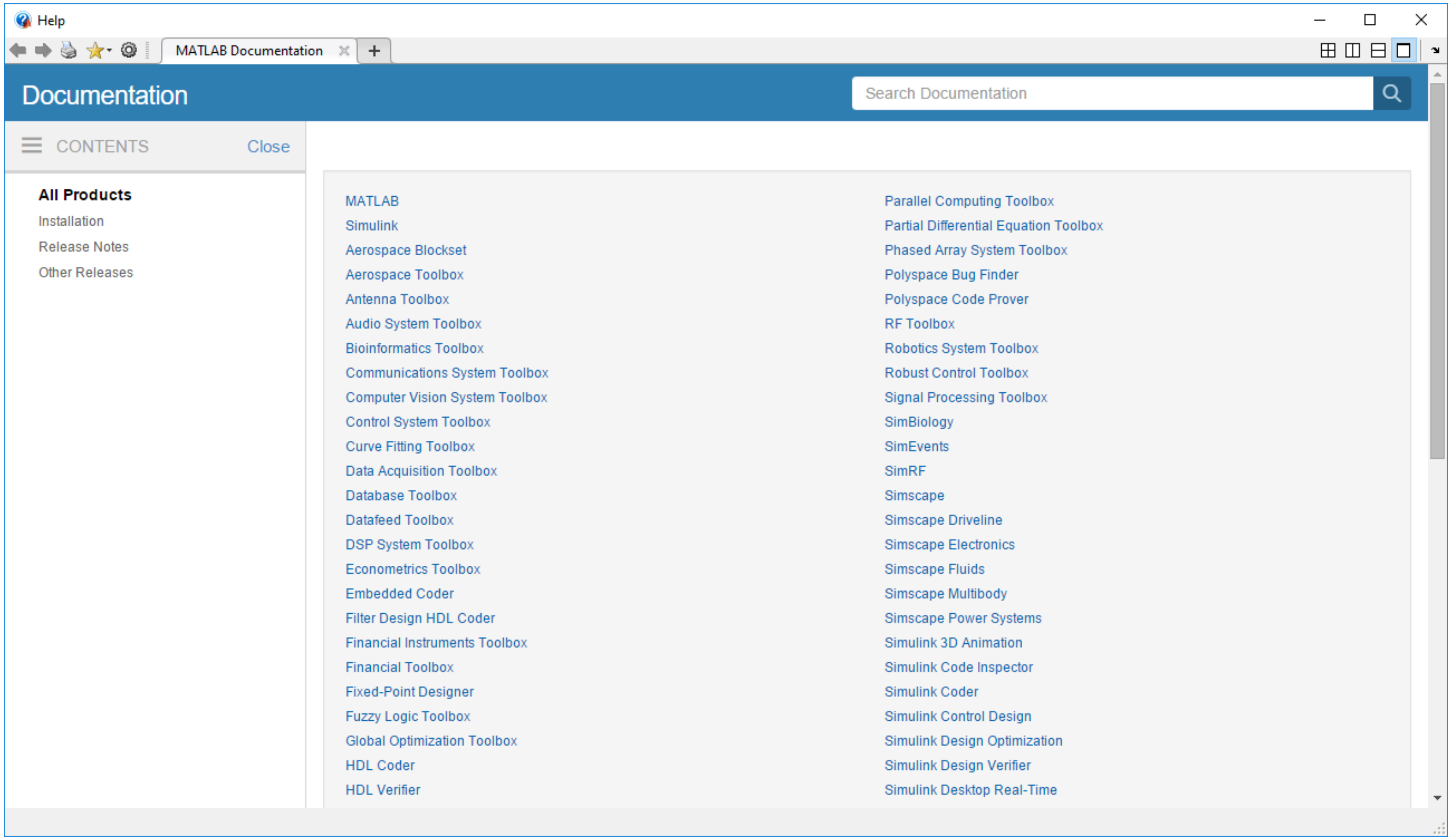
Mean of Matrix Columns

Create a matrix and compute the mean of each column.

```
A = [0 1 1; 2 3 2; 1 3 2; 4 2 2]

A =

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

M = mean(A)

M =

    1.7500    2.2500    1.7500
```

MATLAB Help

Use the documentation browser to view help for all toolboxes in MATLAB. Many of them contain quick start guides and easy examples to demonstrate the functionality.



All Products

Installation

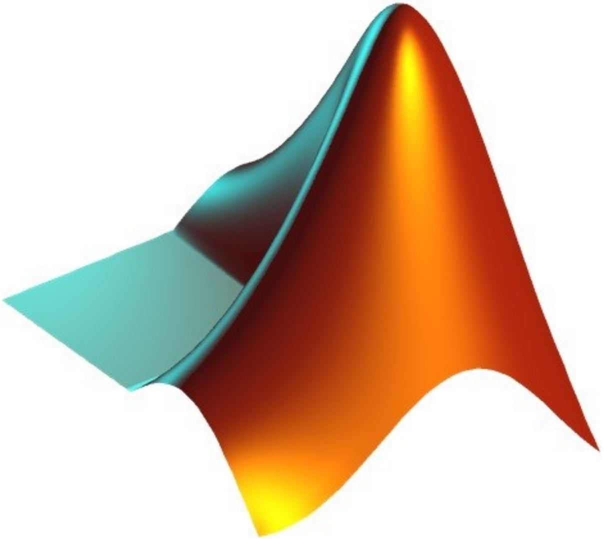
Release Notes

Other Releases

MATLAB
Simulink
Aerospace Blockset
Aerospace Toolbox
Antenna Toolbox
Audio System Toolbox
Bioinformatics Toolbox
Communications System Toolbox
Computer Vision System Toolbox
Control System Toolbox
Curve Fitting Toolbox
Data Acquisition Toolbox
Database Toolbox
Datafeed Toolbox
DSP System Toolbox
Econometrics Toolbox
Embedded Coder
Filter Design HDL Coder
Financial Instruments Toolbox
Financial Toolbox
Fixed-Point Designer
Fuzzy Logic Toolbox
Global Optimization Toolbox
HDL Coder
HDL Verifier

Parallel Computing Toolbox
Partial Differential Equation Toolbox
Phased Array System Toolbox
Polyspace Bug Finder
Polyspace Code Prover
RF Toolbox
Robotics System Toolbox
Robust Control Toolbox
Signal Processing Toolbox
SimBiology
SimEvents
SimRF
Simscape
Simscape Driveline
Simscape Electronics
Simscape Fluids
Simscape Multibody
Simscape Power Systems
Simulink 3D Animation
Simulink Code Inspector
Simulink Coder
Simulink Control Design
Simulink Design Optimization
Simulink Design Verifier
Simulink Desktop Real-Time

4. Variables and Expressions



Commands and Assignments

ans

Variable, that automatically stores most recent answer when no output argument is specified.

```
>> 1 + 1
```

```
ans =

     2
```

variable assignment

Command return is assigned to variable, when specified. The colon (;) suppresses output in the command window.

```
>> b = ans * 2
```

```
b =

     4
```

```
>> c = b^2;

>> c
```

```
c =

    16
```

Commands and Assignments

variable names

- starts with a letter, followed by letters, numbers and underscores
- case sensitive
- maximal length of variable name is the return value of the `namelengthmax` command
- MATLAB keywords are not allowed as variable names
- examples for invalid variable names:

```
>> while = 1;
>> 6x = 1;
>> n! = 1;
>> my home = 1;
```

clc

Clears the command window.

```
>> clc
```

```
>>
```

Commands and Assignments

Basic math functions

In MATLAB, a large variety of built-in math functions is available. You can find an overview by typing doc mathematics into the command window

```
>> sin(pi/2)
ans =
     1
```

$\sin\left(\frac{\pi}{2}\right)$

```
>> exp(i*pi)+1
ans =
   0.0000e+00 + 1.2246e-16i
```

$e^{i\pi} + 1$

```
>> eps
ans =
   2.2204e-16
```

machine epsilon

```
>> 2^10
ans =
        1024
```

2^{10}

```
>> doc mathematics
```

Notice, that by default calculations are performed numerically!

Arrays, Vector and Matrices

Creation

- Use the [ ] operator to create arrays
  - Columns are separated by a comma (,)
  - Rows are separated by a semicolon (;)
- The colon operator (:) can be used to create number series’
- Use the size command to determine the matrix‘ dimension

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

     1     2     3

     4     5     6
```

```
>> Matrix2 = [[1;5],[2:4;6:2:10]]
Matrix2 =

     1     2     3     4

     5     6     8    10
```

```
>> [size(Matrix), size(Matrix, 2)]
ans =

     2     3     3
```

Arrays, Vector and Matrices

Special matrices

Special matrices can be created by using various commands including:

diag(), eye(), true(), false(), linspace(), logspace(), meshgrid(), ngrid(), ones(), zeros(), rand(), randn(), nan()…

```
>> diag([1,2,3])
```

```
ans =

     1     0     0

     0     2     0

     0     0     3
```

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

```
ans =

     1     0     0

     0     1     0
```

```
>> linspace(0,10,6)
```

```
ans =

     0     2     4     6     8    10
```

Arrays, Vector and Matrices

Concatenation

Several matrixes can be concatenated to a combined matrix using the cat, vertcat, horzcat or [ ] commands

```
>> horzcat(Matrix1,Matrix2)
ans =
       1       2       5       6
       3       4       7       8
```

```
>> cat(3,Matrix1,Matrix2)
ans(:,:,1) =
       1       2
       3       4
ans(:,:,2) =
       5       6
       7       8
```

```
>> size([Matrix1; Matrix2])
ans =
       4       2
```

Arrays, Vector and Matrices

Operations

Common matrix operations can be performed in MATLAB

- plus/minus

```
>> Vector1 = [1, 2, 3]; Vector2 = [1, 1, 1]; Vector1 + Vector2;
```

```
ans =

     2     3     4
```

- transpose and multiply

```
>> Vector1*Vector2'
```

```
ans =

     6
```

- inverse, determinate and eigenvalues

```
>> Matrix = magic(2); disp(inv(Matrix)); det(Matrix)
```

```
   -0.2000    0.3000
    0.4000   -0.1000

ans =

   -10
```

Arrays, Vector and Matrices

Array vs. Matrix operations: element-wise operator

Using the element-wise operator (.), scalar operations can be performed on each element of two arrays with equal dimensions

```
>> Vector1 = [1, 2, 3]; Vector2 = 3:-1:1; Vector1.*Vector2
```

```
ans =

     3     4     3
```

```
>> Matrix.^2
```

```
ans =

     1     9

    16     4
```

```
>> Matrix^2
```

```
ans =

    13     9

    12    16
```

Arrays, Vector and Matrices

Sorting and reshaping

There are various possibilities to sort and reshape arrays

```
>> reshape(Matrix,[1,4])
ans =
     1     4     3     2
>> ans(:)
ans =
     1
     4
     3
     2
```

Reshaping to a column vector equal to

```
>> reshape(ans,[],1)
```

```
>> repmat(sort(ans),[1,4])
ans =
     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4
```

Arrays, Vector and Matrices

Indexing

In MATLAB, there are three ways to select a subset of an array or matrix

- Subscript indexing: use ( ) operator to access subscript range of the matrix

```
>> A = magic(3); [A(2,[1,3]) A(1,:)]
ans =

     3     7     8     1     6
```

Using a single colon operator (:) is equivalent to typing `1:end` and can be used to select entire rows/columns

- Linear indexing: in MATLAB, elements can be accessed using a linear index which acts if the matrix has been reshaped to a column vector

```
>> A(:)=1:numel(A); A(2:4)
ans =

     2     3     4
```

```
>> A
A =

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

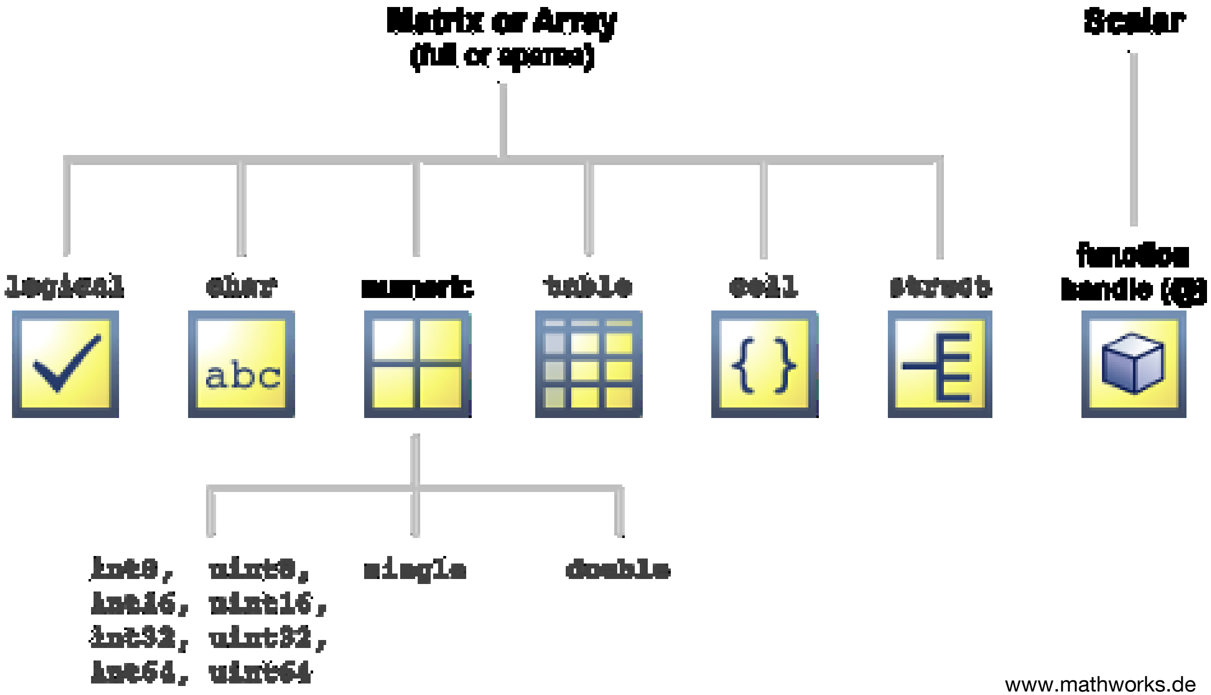
- Logical indexing: access all non-zero entries of a logical matrix of the size of A

```
>> A(mod(A,2)==0)'
ans =

     2     4     6     8
```

```
>> mod(A,2)==0
ans =

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

Data Types

- Several data types, or classes, can be used in MATLAB to work with different data
- the data type is automatically set by MATLAB when assigning a variable
- common data types include those below



www.mathworks.de

Data Types

Logical Data Type

Boolean data can be stored using MATLAB‘s logical data type (zeros are treated as false, all other numeric values as true)

```
>> A = true; B = 0; C = logical(B); whos A B C
```

```
  Name         Size                Bytes  Class      Attributes
  A            1x1                     1  logical
  B            1x1                     8  double
  C            1x1                     1  logical
```

Using these variables Boolean operations can be performed

```
>> [A&C, A|C, xor(A,B), ~A]
```

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

Short Circuit Logical operations (&& and ||) can be more efficient:

- if the first operand determines the solution, the second is not evaluated
- i.e., since A is true, `A||B` always returns true and B does not have to be evaluated
- similarly, since B is false, `A&&B` will always return false and A does not have to be evaluated

Data Types

Character Arrays

- Character arrays, i.e. arrays of numerical values that represent Unicode characters, can be used to represent text in MATLAB

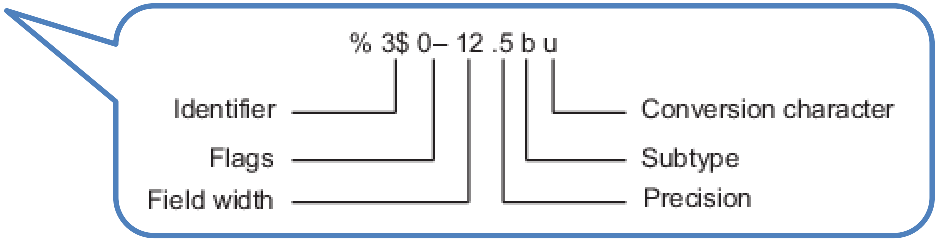
```
>> s = [72    101    108    108    111     32     87    111    114    108    100     33];
>> s = char(s)
```

```
s =

Hello World!
```

- Besides regular array operations, special operations can be performed such as
  - parsing: `strfind`, `sscanf`, `strsplit`…
  - comparing: `strcmp`, `strcmpi`, `strncmp`…
  - modification: `upper`, `lower`, `deblank`, `strjust`…
- data can be formatted into a string using the `sprintf` command

```
>> sprintf('The number pi is %2$8.5g and e is %1$4.5g',exp(1),pi)
```

```
ans =

The number pi is   3.1416 and e is 2.7183
```



Data Types

Strings

- Strings are created by enclosing a piece of text in double quotes. In contrary to character arrays, it is possible to concatenate pieces of text into an array:

```
>> str = ["Flight", "System", "Dynamics"]
str = 1x3 string array
     "Flight"     "System"      "Dynamics"
```

- There are many built-in functions to manipulate strings known from other programming languages, e.g. the plus operator:

```
>> str(1) + " " + str(2) + " " + str(3) + "!"
str =

"Flight System Dynamics!"
```

- Besides easier handling and manipulation, string arrays are more efficient than corresponding cell arrays

```
>> str = ["Flight", "System", "Dynamics"]; cll = {'Flight','System','Dynamics'};
>> whos str cll

  Name      Size            Bytes  Class     Attributes
   cll       1x3              376   cell
   str       1x3              298   string
```

Data Types

Numeric Data Type

- by default, numeric data is stored as double-precision floating point

```
>> a = 25; whos a
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
```

- the data type can be converted to a different class using the corresponding command (e.g.

```
>> b = single(a); whos b
  Name        Size              Bytes  Class     Attributes

  b           1x1                   4  single
```

- similarly, other classes (such as strings) can be converted to numeric values

```
>> s = 'Hello World';
>> int8(s)
ans =

   72   101   108   108   111    32    87   111   114   108   100
```

Data Types

Full and Sparse Data

Numeric values can be stored as sparse data to reduce

- memory demand

```
>> A = zeros(1000); whos A
  Name          Size                   Bytes  Class     Attributes

  A          1000x1000               8000000  double
```

```
>> B = sparse(A); whos B
  Name          Size                 Bytes  Class     Attributes

  B          1000x1000                8024  double    sparse
```

- and the number of arithmetic operations (and thus computation time)

```
>> tic; A*rand(size(A));toc
Elapsed time is 0.114932 seconds.
>> tic; B*rand(size(A));toc
Elapsed time is 0.025196 seconds.
```

Data Types

Cell Array

A cell array is a data type with indexed data containers called cells, where each cell can contain any type of data.

- Use the ( ) operator to refer to the cell

```
>> PatientData = {'Smith',38,71;'Johnson',43,69;'Williams',38,64;'Jones',40,67}
>> PatientData(:,2)'
```

```
ans =
    [38]    [43]    [38]    [40]    [49]
```

- and the { } operator to refer to its content.

```
>> [PatientData{:,2}]
```

```
ans =
    38    43    38    40    49
```

- use `cellfun` to apply a function to every cell of a cell array

```
>> PatientData(:,2) = cellfun(@(a){a+1},PatientData(:,2)); >> PatientData(:,2)'
```

```
ans =
    [39]    [44]    [39]    [41]    [50]
```

Data Types

Structures and Structure Arrays

Structure arrays contain data in fields that can be accessed by name.

- Create a structure by assigning a value to a field

```
>> PatientStruct(2).Name = 'Johnson';
```

- Or by using the `struct` command

```
>> PatientStruct =
struct('Name',PatientData(:,1),'Age',PatientData(:,2),'Height',PatientData(:,3))
PatientStruct =
5x1 struct array with fields:
    Name
    Age
    Height
```

- Data can be assigned or accessed using the '.' operator

```
>> {PatientStruct(2).Name, PatientStruct(2).Height}
ans =
    'Johnson'    [69]
```

Data Types

Tables

A table is a data type for collecting heterogeneous data and metadata properties, such as variable names, row names, descriptions, and variable units, in a single container.

```
>> PatientTable = struct2table(PatientStruct(1:3))
```

```
PatientTable =

       Name        Age    Height

    __________     ___    ______

    'Smith'        39     71

    'Johnson'      44     69

    'Williams'     39     64
```

Data can be accessed similarly to structures

```
>> PatientTable.Name(1)
```

```
ans =

    'Smith'
```

The Properties field of the table contains information about the table

```
>> TableProperties = PatientTable.Properties
```

Data Types

Function handles

A function handle stores an association to a function. Indirectly calling a function enables you to invoke the function regardless of where you call it from.

- A function handle an be created using the @ command

```
>> f = @ones
```

```
f =

     @ones
```

```
>> f(1,2)
```

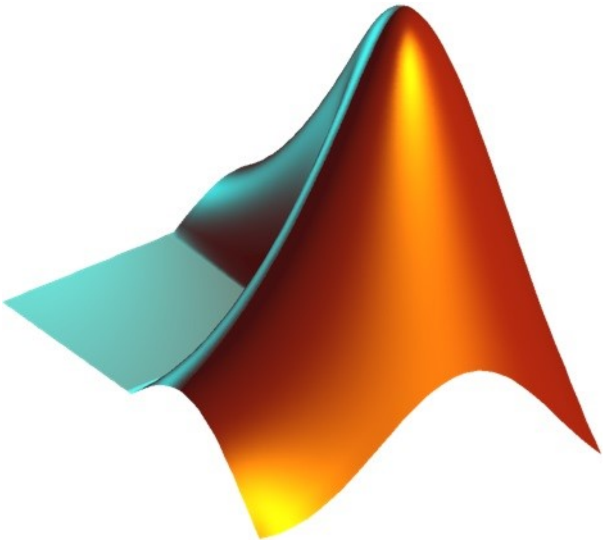
```
ans =

      1      1
```

- A function handle can store anonymous functions, which is a one-line expression without program file (see Chapter 5: Scripts and Functions)

```
>> f = @(x)x^2;f(2)
```

```
ans =

      4
```

5. Scripts and Functions



Scripts and Functions

Scripts and functions contain programs that consist of a series of MATLAB statements, which can be edited using the MATLAB editor and stored in a .m-file.

- Scripts are the simplest types of programs used to automate commands that have to be performed repeatedly from the command line
- Function offer additional programming flexibility
  - Input and outputs
  - Individual workspace (separate from the base workspace)

```
function [out1, out2] = FuncName(in1, in2)
% calculate area
out1 = in1 * in2;

% calculate circumference
out2 = 2*(in1 + in2);
end
```

```
>> [area, circumference] = FuncName(3,4)
area =
    12
circumference =
    14
```

Scripts and Functions

Syntax

Functions have to be saved in a m-file named according to the function name.

Element	Description
Function keyword (required)	function
Output arguments (optional)	names of output variables that are set within the function
Function name (required)	Valid function names follow the same rules as variable names
Input arguments (optional)	names of input variables that are used within the function

```
% rectangleProps.m
function [area, cf] = rectangleProps(a, b)
% calculate area
area = a * b;

% calculate circumference
cf = 2*(a + b);
end

function dispPi
% print pi
fprintf('%.4g\n',pi)
end
```

Scripts and Functions

Function Types

One program file (.m) can contain several functions – the main function and a combination of local and nested functions.

Type	Description	Location
Local functions	subroutines that are available to any other function within the same file	Same file
Nested functions	Completely contained in another function, can use variables defined in parent function	Same file
Private functions	Like local functions, but can be used by any function within a folder immediately above the private folder	Subfolder called “private”
Anonymous functions	Function that consists of one single expression with no file but completely stored within a function handle	No file

```
% myFunction.m
function b = myFunction(a)
b = squareMe(a) + doubleMe~~(a);~~

   function y = doubleMe
      y = 2.*a;
   end
end

function y = squareMe(x)
y = x.^2;
end
```

Scripts and Functions

Variable-length input/output

MATLAB supports functions with a variable number of input and output arguments.

keyword	Description
nargin	Holds the number of input arguments passed to the function
varargin	Holds in the input arguments in a row cell array
nargout	Holds the number of input arguments the function needs to return
varargout	Row cell output in which the variable number of output arguments have to be stored

```
function [ varargout ] = VarArgsFun( varargin )
%print the number of inputs and outputs
fprintf('Number of Input Arguments: %i\n', nargin);
fprintf('Number of Output Arguments: %i\n', nargout);

% if there are any inputs --> display in the command
window
if nargin > 0
   fprintf('The Inputs are:\n');
   for i = 1:nargin
     display(varargin{i});
   end
end

% if there are any outputs --> create a number
sequence
if nargout > 0
   fprintf('Creating Outputs:\n')
   varargout = cell(1,nargout);
   for i = 1:nargout
    varargout{i} = i;
   end
end
end
```

Scripts and Functions

Conditional Statements

Conditional statements enable selecting which code block to execute at run time.

- if statement
  Apply conditions using the keywords `if`, `elseif` and `else`

```
function Compare(a, b)
if a < b
   disp('smaller')
elseif a > b
   disp('larger')
else
   disp('equal')
end
end
```

```
>> Compare(1,10)
smaller
>> Compare(11,10)
larger
>> Compare(10,10)
equal
```

Scripts and Functions

Conditional Statements

Conditional statements enable selecting which code block to execute at run time.

- switch statement
  Test for equality against a set of known values

```
function WeekDay(dayString)
switch dayString
   case 'Monday'
      disp('Start of the work week')
   case 'Tuesday'
      disp('Day 2')
   case 'Wednesday'
      disp('Day 3')
   case 'Thursday'
      disp('Day 4')
   case 'Friday'
      disp('Last day of the work week')
   otherwise
      disp('Weekend!')
end
end
```

```
>> WeekDay(‘Tuesday’)
```

```
Day 2
```

```
>> WeekDay(‘Saturday’)
```

```
Weekend!
```

Scripts and Functions

Loop Control Statements

Loop control statements allow for repeated execution of code blocks.

- for statement
  loops through a code block for a specific number of times using prespecified values for a loop iterator (similar to the foreach loop in C++)

```
% myScript.m
a = zeros(1,10);
for iter = [3, 5:2:10]
   a(iter) = iter/2*(iter-1);
end
a(5:end)
```

```
>> myScript
ans =
     10      0     21      0     36      0
```

Scripts and Functions

Loop Control Statements

Loop control statements allow for repeated execution of code blocks.

- for statement
  loops through a code block for a specific number of times using prespecified values for a loop iterator (similar to the foreach loop in C++)

```
% myScript.m
a = zeros(1,10);
for iter = [3, 5:2:10]
   a(iter) = iter/2*(iter-1);
end
a(5:end)
```

```
>> myScript
```

```
ans =

     10      0     21      0     36      0
```

Scripts and Functions

Loop Control Statements

Loop control statements allow for repeated execution of code blocks.

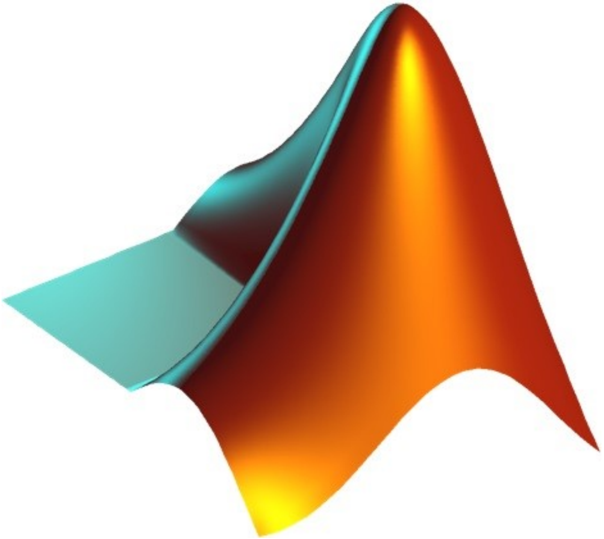
- while statement
  loops through a code block as long as a condition remains true (similar to the while loop in C++)

```
% myScript.m
a = zeros(1,10);
iter = 1;
while iter <= 10
   a(iter) = iter/2*(iter-1);
   iter = iter + 1;
end
a(5:end)
```

```
>> myScript
ans =

    10    15    21    28    36    45
```

Use the `break` statement to exit the loop, or skip to the next iteration using the `continue` statement

6. Debugging



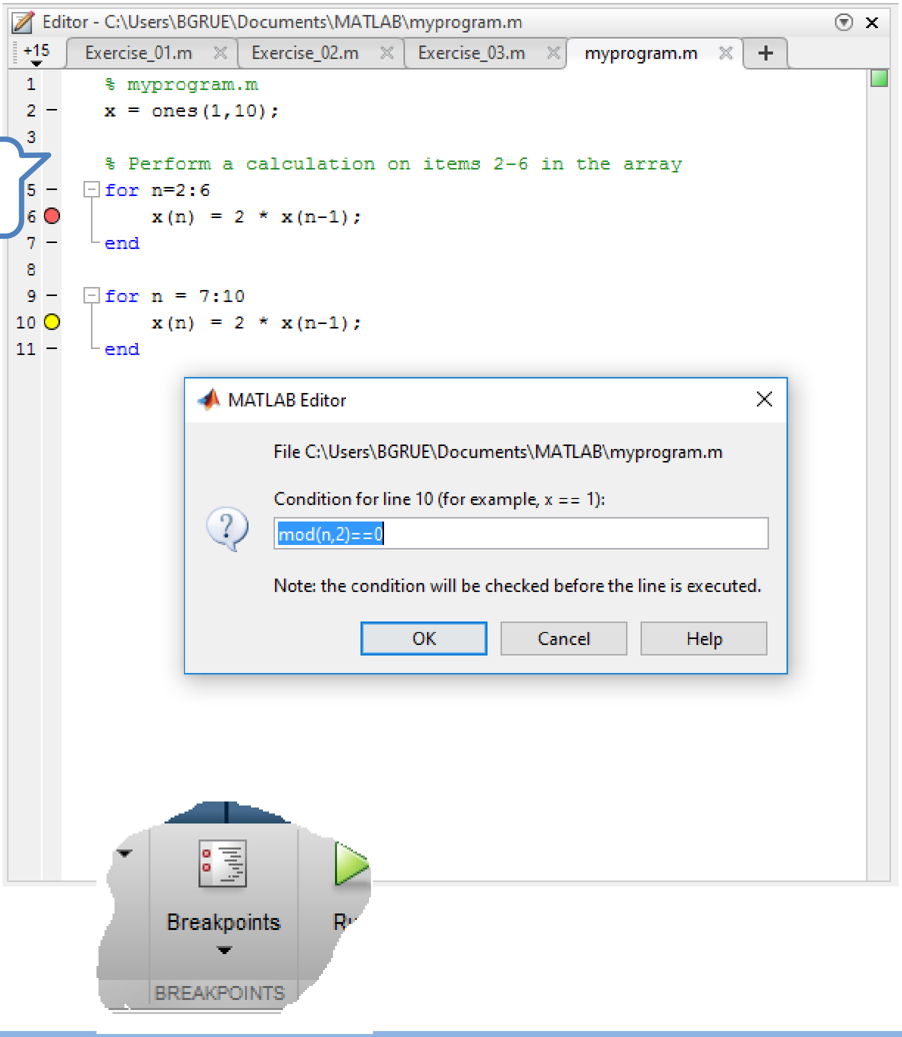
Debugging

Diagnosing Problems in code is a key task in programming. MATLAB provides several features to facilitate this.

Breakpoint alley

Breakpoints

- Standard
  A standard breakpoint can be added by clicking the breakpoint alley next to an executable line (marked with a dash (-)). The program will be stopped once it reaches this line.
- Conditional
  A conditional breakpoint can be set by right clicking the breakpoint alley and selecting “Set Conditional Breakpoint…”. A condition can be stated to stop the code when it is fulfilled.
- Error
  To stop the execution of the code on errors select the Breakpoints button from the Editor ribbon and specify one of the following
  - Break on errors
  - Break on warnings
  - Or chose more error and warning handling options



```
1       % myprogram.m
2  -    x = ones(1,10);
3
4       % Perform a calculation on items 2-6 in the array
5  -    for n=2:6
6  ●        x(n) = 2 * x(n-1);
7  -    end
8
9  -    for n = 7:10
10 ○        x(n) = 2 * x(n-1);
11 -    end
```

File C:\Users\BGRUE\Documents\MATLAB\myprogram.m

Condition for line 10 (for example, x == 1):

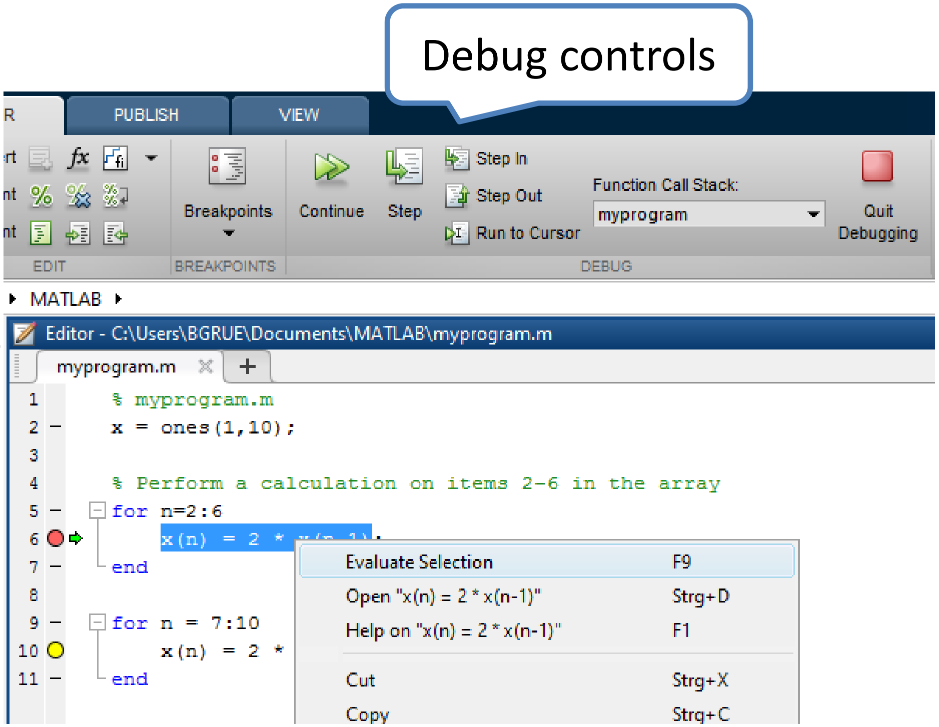
mod(n,2)==0

Note: the condition will be checked before the line is executed.

Debugging

To diagnose a program the following steps can be taken:

- Click “Run” to run the script of function to investigate
- The code will be stopped at the first active breakpoint
    - Evaluate parts of the code by right clicking a selection and selecting “Evaluate Selection” or pressing the F9 key
    - Step through the program using the controls in the DEBUG panel of the Editor ribbon
- Finish debugging by
    - clicking the “Quit Debugging” Button
    - Using the “Continue Button” to run the code until the end of the script or function



Debug controls

```
% myprogram.m
x = ones(1,10);

% Perform a calculation on items 2-6 in the array
for n=2:6
    x(n) = 2 * x(n-1);
end

for n = 7:10
    x(n) = 2 *
end
```

Debugging

Try, Catch statement

Errors can be mitigated within the code by using a try catch statement. Using a try, catch statement, error information can be retrieved from an MException object created by MATLAB.

```
% myprogram.m
x = ones(1,10);

% Perform a calculation on items in the array
for n=2:6
   x(n) = 2 * x(n-1);
end

try
   for n = 7:11
      x(n) = 2 * x(n);
   end
catch ME
   error(ME.message);
end
```

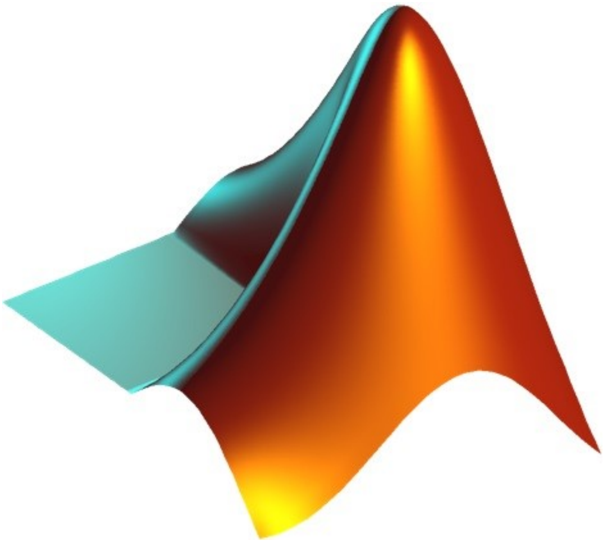
Within the for-loop n becomes bigger than the number of elements in x

The error command throws another Exception and outputs a message to the command window

```
>> myprogram
Error using myprogram (line 14)
Index exceeds matrix dimensions.
```

Within the for-loop n becomes bigger than the number of elements in x

7. List of Useful Commands



List of Commands

Command	Explanation	Slide #
`edit`	Edit or create file	8
`format`	Set Command Window output display format	11
`clear`	Remove items from workspace, freeing up system memory	11
`workspace`	Open Workspace browser to manage workspace	11
`filebrowser`	Open Current Folder browser, or select it if already open	11
`clc`	Clear Command Window	11
`cd`	Change current folder	12
`addpath`	Add folders to search path	12
`openvar`	Open workspace variable in Variables editor or other graphical editing tool	13
`doc`	Reference page in Help browser	17
`help`	Help for functions in Command Window	17
`ans`	Most recent answer	20
`namelengthmax`	Maximum identifier length	20

Command	Explanation	Slide #
`clc`	Clear Command Window	21
`sin`	Sine of argument in radians	22
`exp`	Exponential	22
`eps`	Floating-point relative accuracy	22
`diag`	Create diagonal matrix or get diagonal elements of matrix	23
`eye`	Identity matrix	23
`linspace`	Generate linearly spaced vector	23
`cat, vertcat, horzcat`	Concatenate arrays along specified dimension	24
`size`	Array Dimensions	24
`magic`	Magic square	25
`disp`	Display value of variable	25
`inv`	Matrix inverse	25
`det`	Matrix determinant	25
`reshape`	Reshape array	27

List of Commands

Command	Explanation	Slide #
`repmat`	Repeat copies of array	27
`sort`	Sort array elements	27
`numel`	Number of array elements	28
`mod`	Remainder after division (modulo operation)	28
`logical`	Convert numeric values to logicals	30
`whos`	List variables in workspace, with sizes and types	30
`char`	Convert to character array (string)	31
`sprintf`	Format data into string	31
`single`	Convert to single precision	32
`int8, int16, int32`	Convert to 8/16/32-bit signed integer	32
`zeros`	Create array of all zeros	33
`sparse`	Create sparse matrix	33
`rand`	Uniformly distributed random numbers	33

Command	Explanation	Slide #
`tic`	Start stopwatch timer	33
`toc`	Read elapsed time from stopwatch	33
`cellfun`	Apply function to each cell in cell array	34
`struct`	Create structure array	35
`struct2table`	Convert structure array to table	36
`try, catch`	Execute statements and catch resulting errors	51