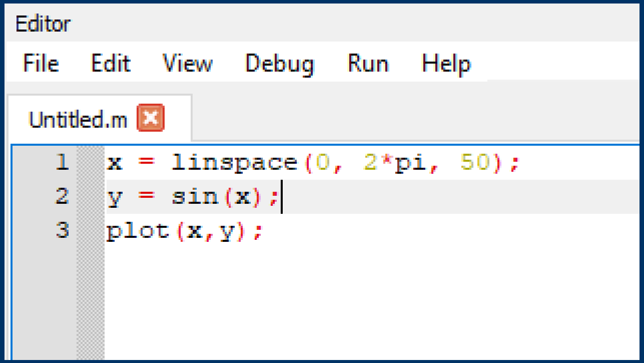
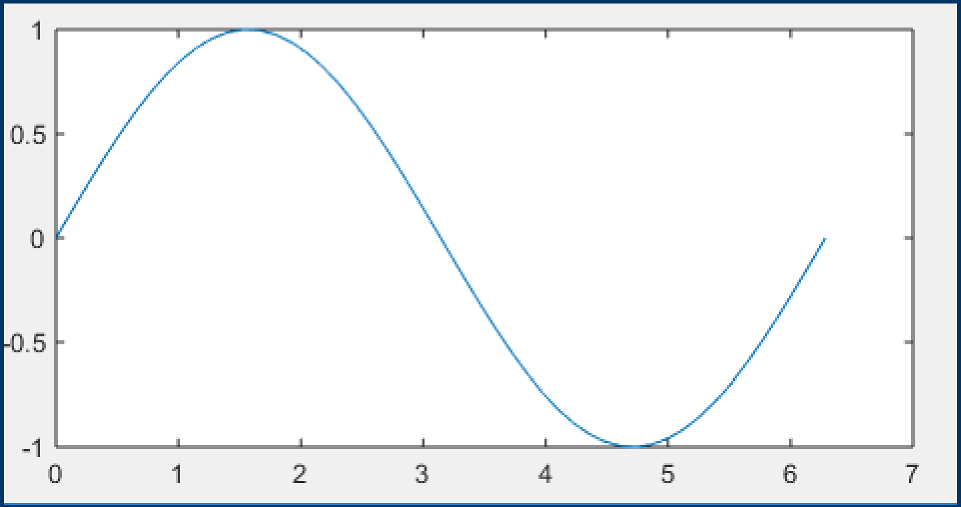
Bauhaus-Universität Weimar

~~MATLAB~~ GNU Octave for Engineers

Tutorial 1: Variables, vectors, operations and plotting



```
x = linspace(0, 2*pi, 50);
y = sin(x);
plot(x,y);
```

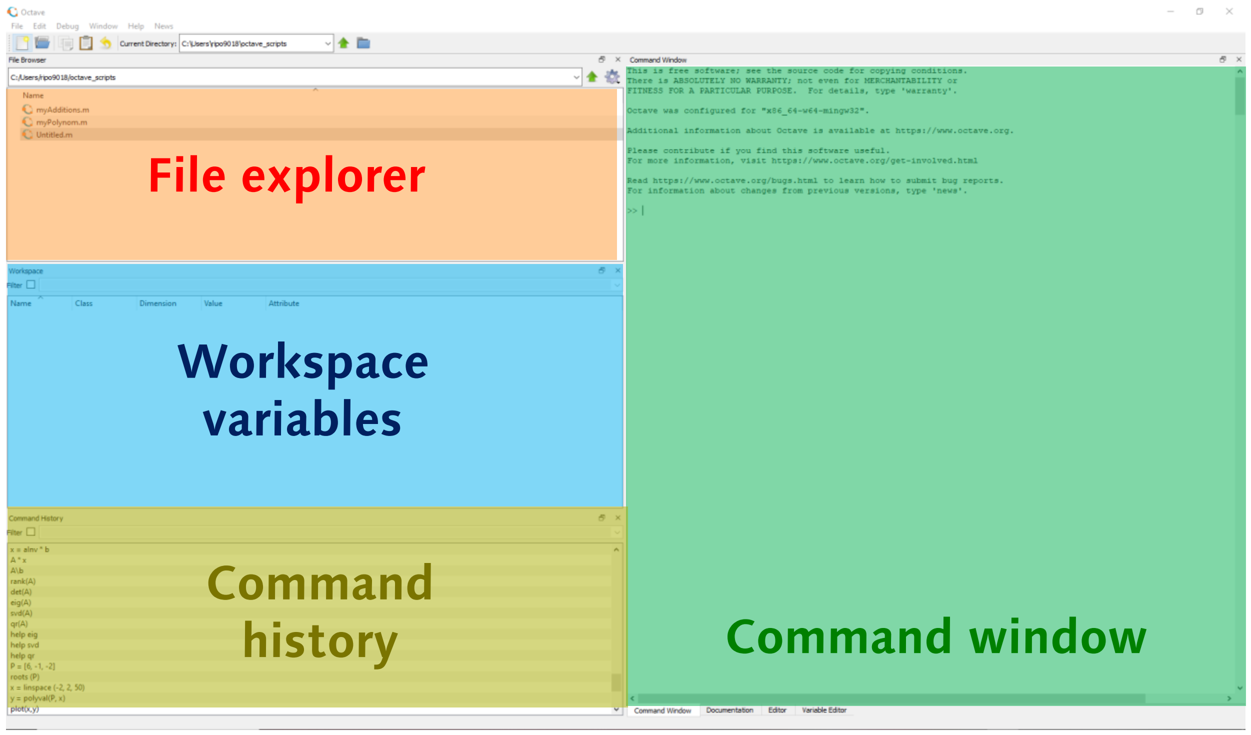


Mathias Artus

- Getting started
- Auto completion and Help
- Variables
- MATLAB as a calculator (command line)
- Vectors and matrices
- Vector and matrix operations
- Plotting
- Scripting

Getting started

- Octave is an equivalent Open Source tool to MATLAB but not identical
- Download: https://www.gnu.org/software/octave/download.html
- Online Editor: https://octave-online.net/
  please register if you want to use scripts and we want to use them
  If you don't want to register take the download version
- Default layout see below. Feel free to rearrange

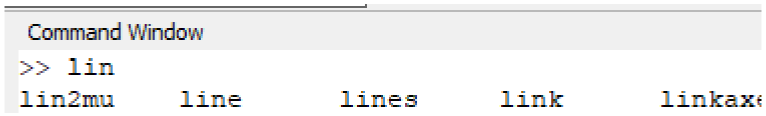

Octave
File Edit Debug Window Help News
Current Directory: C:\Users\ripo9018\octave_scripts
File Browser
C:/Users/ripo9018/octave_scripts
Name
myAdditions.m
myPolynom.m
Untitled.m
File explorer
Workspace
Filter
Name Class Dimension Value Attribute
Workspace variables
Command History
Command history
Command Window
Command window
Command Window Documentation Editor Variable Editor

```
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> |
```

```
x = aInv * b
A * x
A\b
rank(A)
det(A)
eig(A)
svd(A)
qr(A)
help eig
help svd
help qr
P = [6, -1, -2]
roots (P)
x = linspace (-2, 2, 50)
y = polyval(P, x)
plot(x,y)
```

Auto completion

- Type the start of a function name
- Press 2x ***tab*** to show possible completions

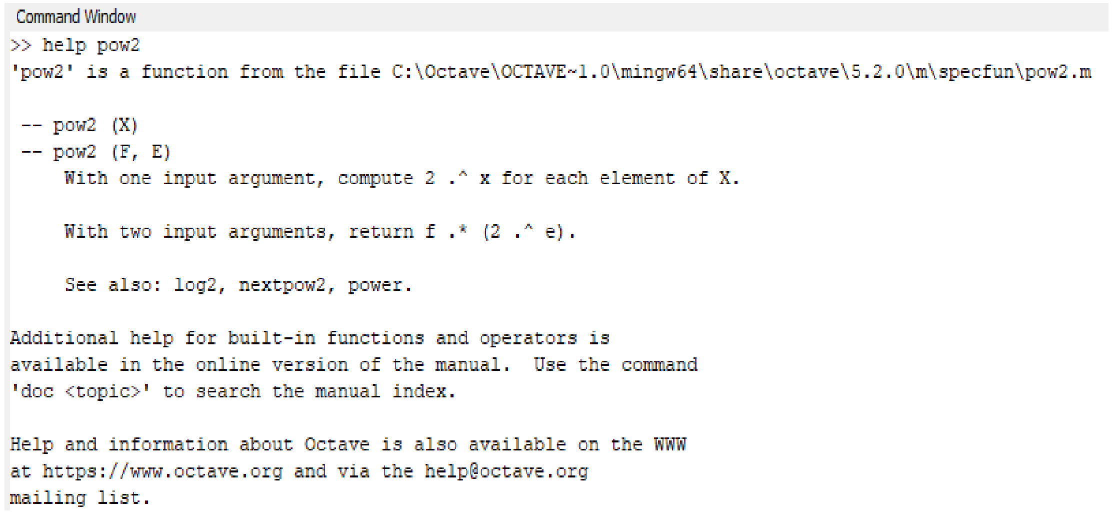


Command Window

```
>> lin
lin2mu      line        lines       link        linkax
```

Help

- Type „*help*“ and the function name



Command Window

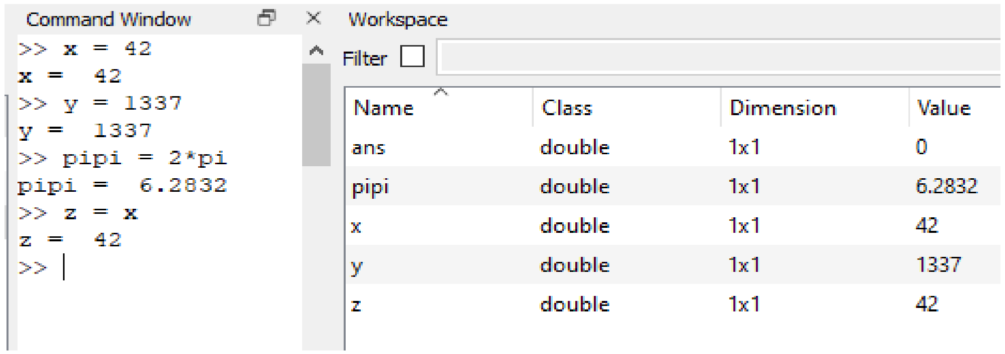
```
>> help pow2
'pow2' is a function from the file C:\Octave\OCTAVE~1.0\mingw64\share\octave\5.2.0\m\specfun\pow2.m

 -- pow2 (X)
 -- pow2 (F, E)
     With one input argument, compute 2 .^ x for each element of X.

     With two input arguments, return f .* (2 .^ e).

     See also: log2, nextpow2, power.

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

TASKS

- ☐ **Find trigonometric functions**
- ☐ **Get help for *linspace***

Variables

- Int, double, char, string, object
- No type binding
- Stay in workspace, reuseable
- „***clear***“ deletes actual workspace
- Case sensitive **(x ≠ X)**
- Starting with character



```
>> x = 42
x =  42
>> y = 1337
y =  1337
>> pipi = 2*pi
pipi =  6.2832
>> z = x
z =  42
>>
```

Name	Class	Dimension	Value
ans	double	1x1	0
pipi	double	1x1	6.2832
x	double	1x1	42
y	double	1x1	1337
z	double	1x1	42

Predefined variables

- **Pi** (3.14159…)
- **Inf**, **-inf** (infinity)
- **nan** (not an number)
- **i** and **j** (complex notation)

TASKS

- Define a variable ***myString*** with the value ‚Hello World' (Attention to the quotation marks)
- Define a variable ***Euler*** with the value **2.71828**
- ***myResult*** shall be the natural logarithm of ***Euler***

MATLAB as a calculator

- **Command line** as instant test method for statements
- When „**variable =**" are omitted, then „**ans**" gets the result
  - Therefore: „2*5" results in the line „ans = 10"
  - **ATTENTION: „ ans"** can be reused.
- Check syntax
- Check results

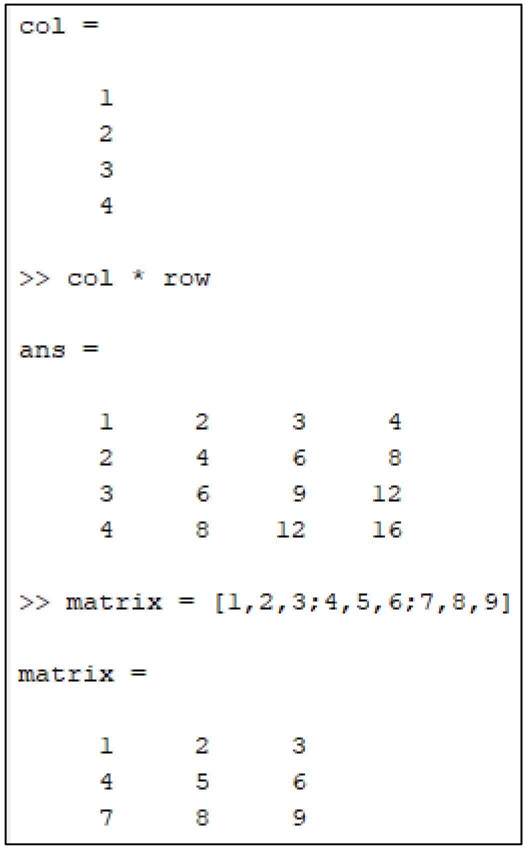
TASKS

- Calculate the **square root of 2**
- Calculate the **sine of ¾ * π**
- Test the function **factorial** with different values (e.g. 2 or 5). What does it calculate?

Vectors and matrices

- MATLAB is made for Vectors: **MAT**rix **LAB**oratory
- Row vector: **row = [1, 2, 3 ,4]**
- Column vector: **col = [1; 2; 3; 4]**
- Matrix: **matrix = [1, 2, 3; 4, 5, 6; 7, 8, 9]**
- Shown in Workspace
  - Try **double-klick** on the variable‘s name
  - Try typing „**row**“, „**col**“ and „**matrix**“ in the command window
- Matrices **Zeros**, **ones**, **eye**.
- Indexing by round brackets ( ) → e.g. **matrix(6)**, **matrix(2,1)**
  - Try **row(2:3)**, **matrix(2:5)**

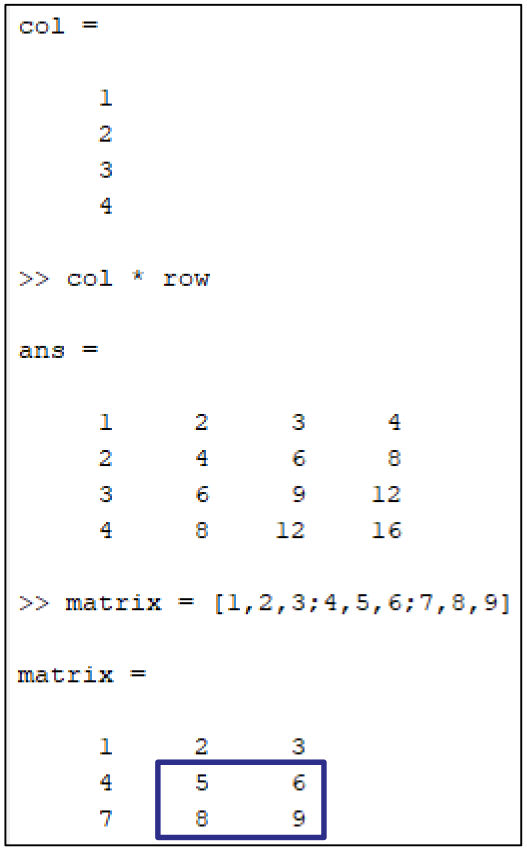
ATTENTION: MATLAB indexing starts with 1



```
col =

     1
     2
     3
     4

>> col * row

ans =

     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16

>> matrix = [1,2,3;4,5,6;7,8,9]

matrix =

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

TASKS

- Try typing „**row**“, „**col**“ and „**matrix**“ in the command window
- Define a **row vector** with ones
- Define a **column vector** with zeros
- Define a **eye matrix** of 3rd order
- Let MATLAB print the **lower right 2x2 Matrix** from **matrix**



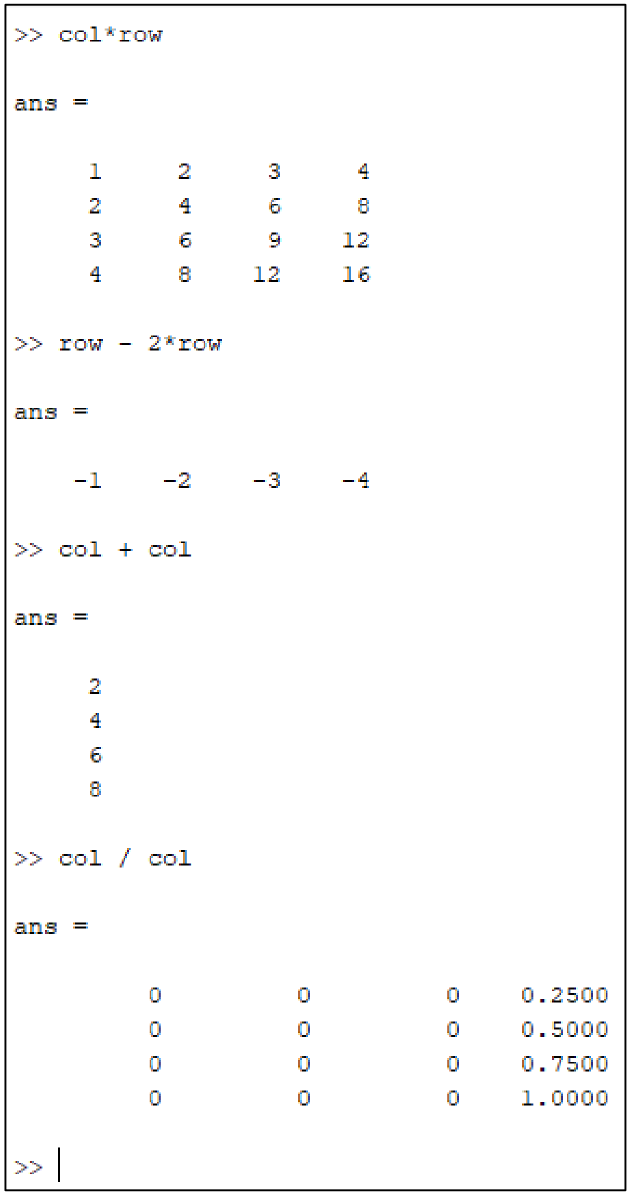
```
col =

     1
     2
     3
     4

>> col * row

ans =

     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16

>> matrix = [1,2,3;4,5,6;7,8,9]

matrix =

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

Vector and matrix operations

- For matrices, the same scalar operations are used
  - Addition: **+**
  - Substraction: **-**
  - Multiplication: *****
  - Division: **/**

PLEASE NOTE:

- You must **respect dimensions**!!!
- Operation over each element,
  use the **dot**: **.*   ./   .^**
- Transpose of a matrix: **transpose**(x)
- Inverse of a matrix: **inv**(x)



```
>> col*row

ans =

     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16

>> row - 2*row

ans =

    -1    -2    -3    -4

>> col + col

ans =

     2
     4
     6
     8

>> col / col

ans =

          0          0          0     0.2500
          0          0          0     0.5000
          0          0          0     0.7500
          0          0          0     1.0000

>> |
```

TASKS

- Calculate a Matrix with the order 5 with 3's on the main diagonal and all other values 0

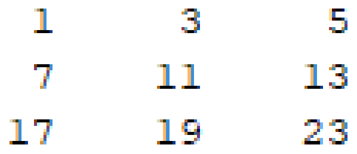
- Calculate the dot product of two column vectors {4,2,1,3,3,7}

```
row = [1, 2, 3 ,4]
col = [1; 2; 3; 4]
```

- Take **row** and **col** and calculate the multiplication and division

- Calculate the sine for 100 x-values

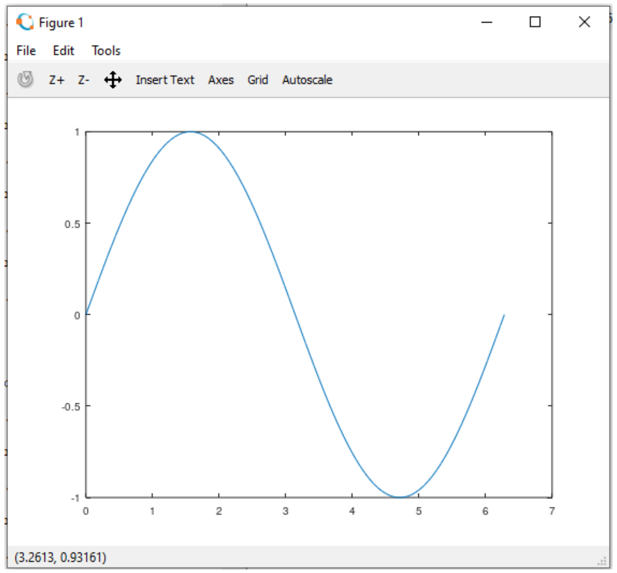
- Calculate the inverse of



```
 1    3    5
 7   11   13
17   19   23
```

Plotting

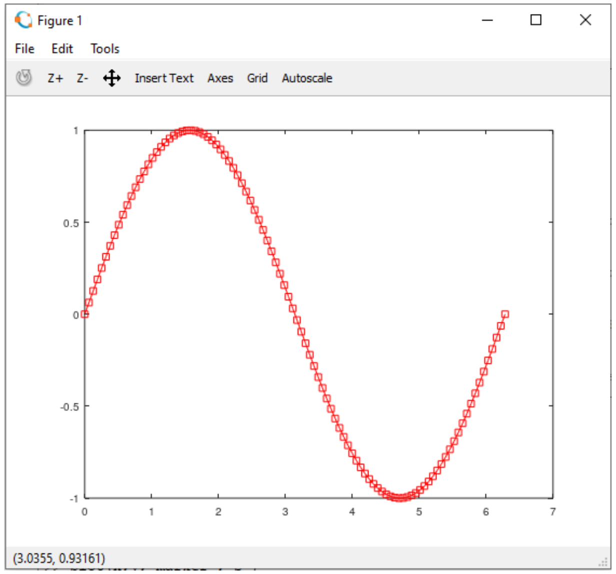
- *plot (x,y)*
- *plot (x,y,linespec)*
  - plot (x,y,'--sr')
- For more options see **help**
- *plot (matrix): each column as line*
- Export with **File/Save As** in the figure window



TASKS

- Plot **sine** and **cosine** functions
- Plot a circle
- Plot 3 random sequences of numbers in one chart

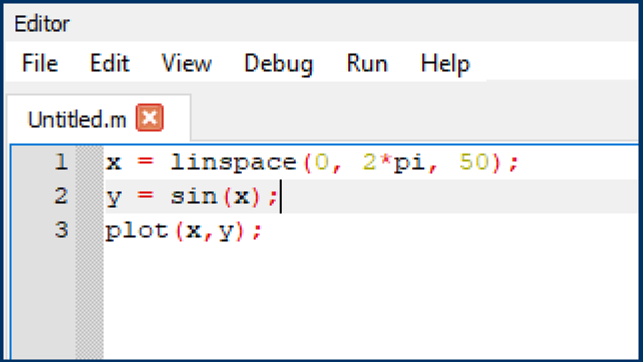
HINT: Use **rand()**



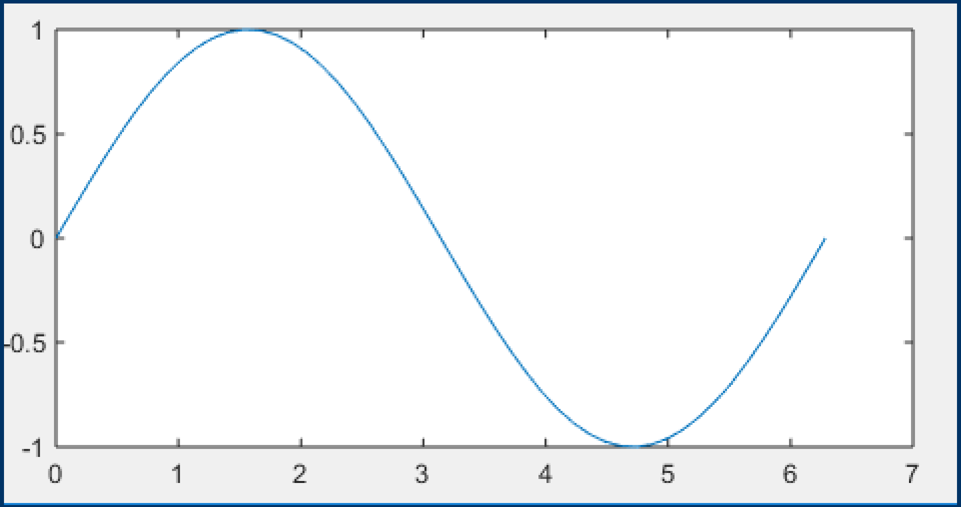
Bauhaus-Universität Weimar

~~MATLAB~~ GNU Octave for Engineers

Tutorial 2: Scripts and functions



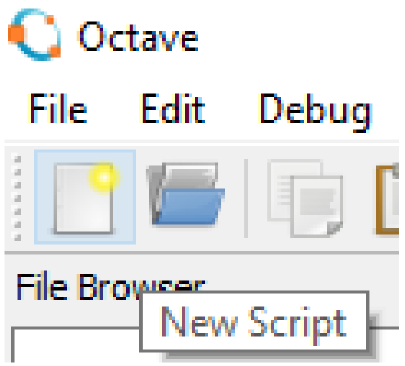
```
x = linspace(0, 2*pi, 50);
y = sin(x);
plot(x,y);
```

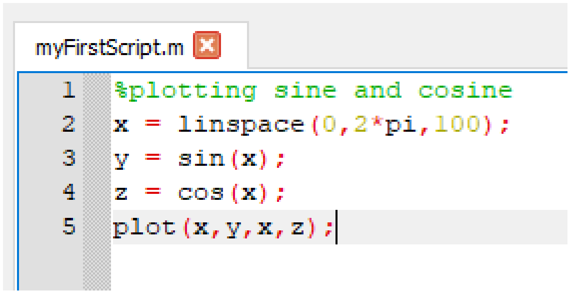


Mathias Artus

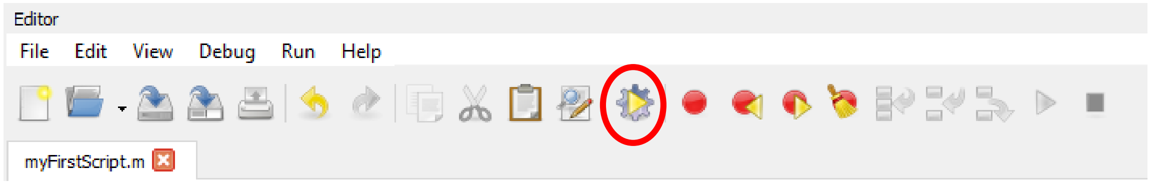
Scripting and Comments

- All things shown can be saved in a file as script for reusing
- If in the working directory exists .m-files, they'res accessable
- Start loving the semicolon (end of instruction)
- Save the shown script as „myFirstScript.m“
- Go into the command window and type *myFirstScript*
- Alternative: Press the play Button
- Comment your code with „%“
- Actual variables in your workspace accessed by Scripts
- Variable stay in workspace after execution





```
%plotting sine and cosine
x = linspace(0,2*pi,100);
y = sin(x);
z = cos(x);
plot(x,y,x,z);
```



TASK Write a script which adds 1 to the existing vector **row** and plots it

Flow control

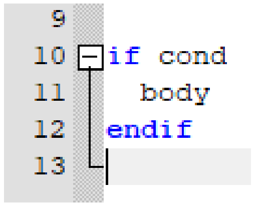
- Condition: **if, elseif, else**
- Loops: **for, while** (no do-while)

- Relational operators:

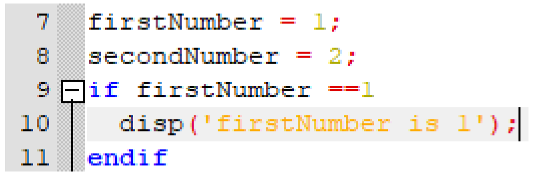
Equal	==
Not equal	~=
Greater than	>
Less than	<
Greater or equal	>=
Less or equal	<=

- Logical operators:

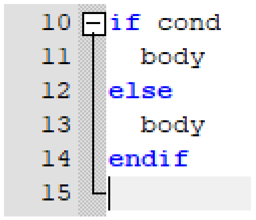
	El.-wise	scalar
And	&	&&
Or	|	||
Not	~	
Xor	xor	
All true	all	
Any true	any	



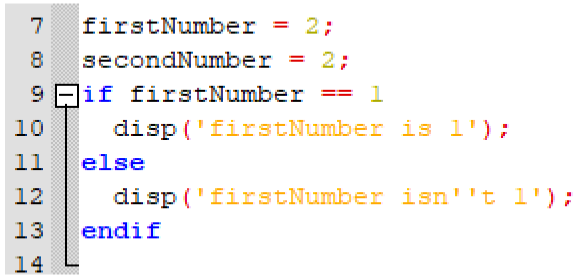
```
if cond
   body
endif
```



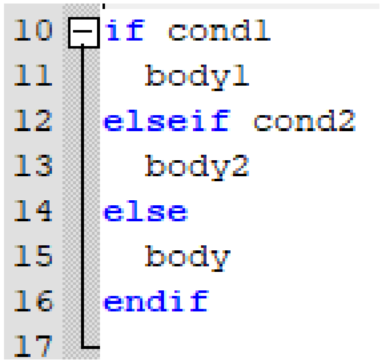
```
firstNumber = 1;
secondNumber = 2;
if firstNumber ==1
  disp('firstNumber is 1');
endif
```



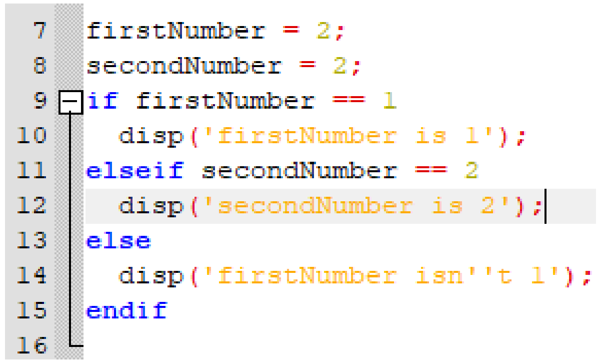
```
if cond
   body
else
   body
endif
```



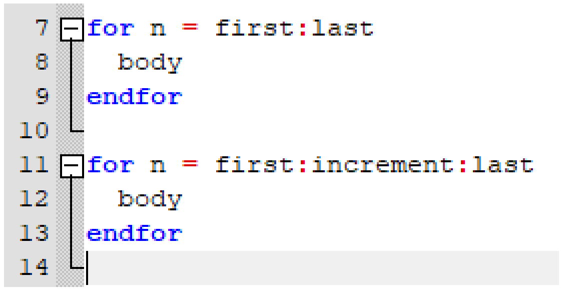
```
firstNumber = 2;
secondNumber = 2;
if firstNumber == 1
  disp('firstNumber is 1');
else
  disp('firstNumber isn''t 1');
endif
```



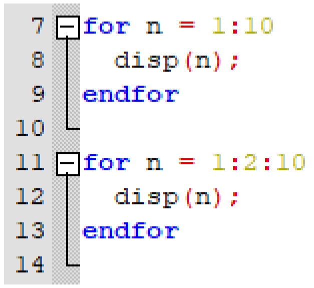
```
if cond1
   body1
elseif cond2
   body2
else
   body
endif
```



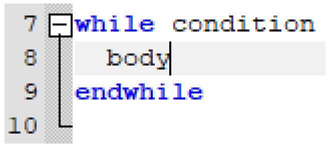
```
firstNumber = 2;
secondNumber = 2;
if firstNumber == 1
  disp('firstNumber is 1');
elseif secondNumber == 2
  disp('secondNumber is 2');
else
  disp('firstNumber isn''t 1');
endif
```



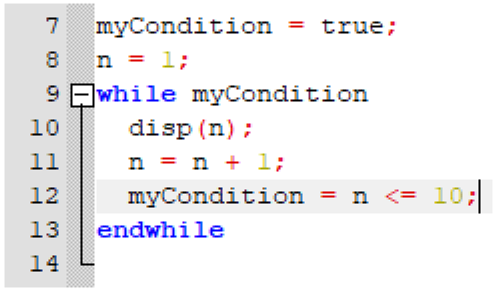
```
for n = first:last
   body
endfor

for n = first:increment:last
   body
endfor
```



```
for n = 1:10
   disp(n);
endfor

for n = 1:2:10
   disp(n);
endfor
```



```
while condition
   body
endwhile
```



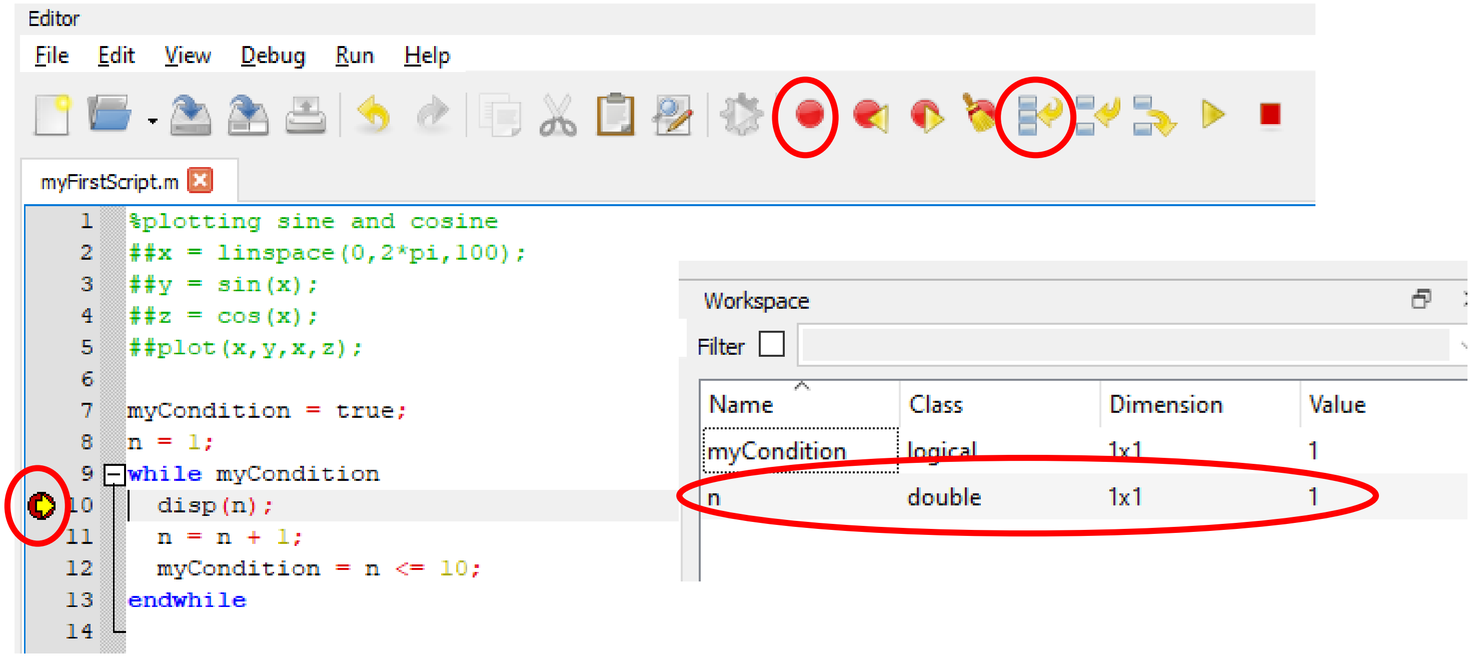
```
myCondition = true;
n = 1;
while myCondition
   disp(n);
   n = n + 1;
   myCondition = n <= 10;
endwhile
```

TASKS

- Write a script which defines two variables
- Compare them
- Display which is greater (use *disp*)
- Extension: if the greater one is 42 display „the answer to life, the universe and everything"
- Write a script which displays the even numbers between 1 and 10

Debugging

- Finding bugs in code
- Breakpoints (conditioned)
- Variable analysis (hover the mouse over the variable
- Step wise debugging
- **Think about where the error can come from!!!**
- **The computer doesn‘t know it**

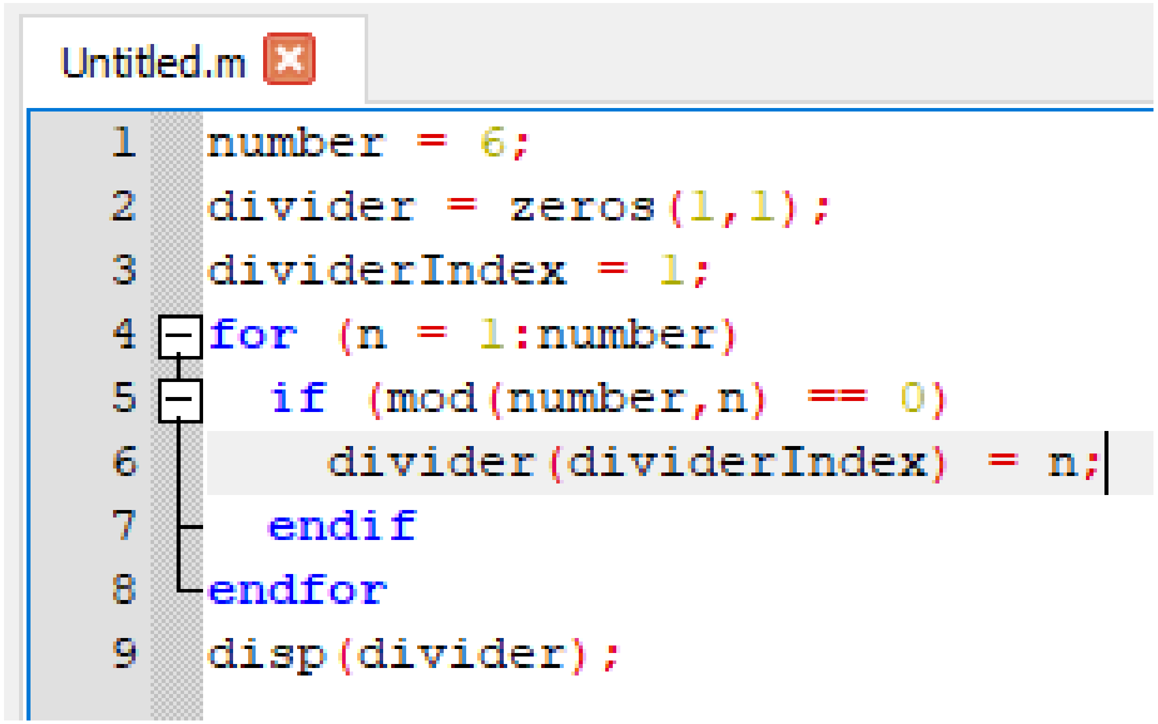


```
%plotting sine and cosine
##x = linspace(0,2*pi,100);
##y = sin(x);
##z = cos(x);
##plot(x,y,x,z);

myCondition = true;
n = 1;
while myCondition
  disp(n);
  n = n + 1;
  myCondition = n <= 10;
endwhile
```

Name	Class	Dimension	Value
myCondition	logical	1x1	1
n	double	1x1	1

TASKS

- The script shall display all deviders of 6
- *Use* ***mod***: calculate the reminder of a division
- Find the bug and fix it

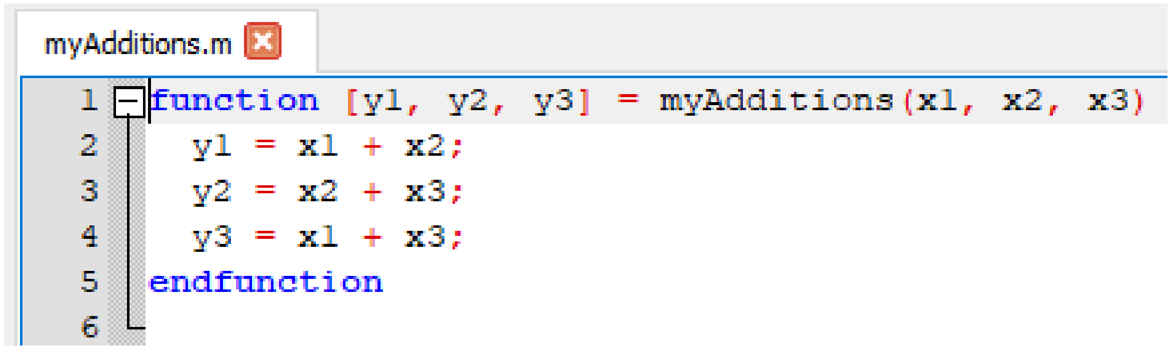


```
number = 6;
divider = zeros(1,1);
dividerIndex = 1;
for (n = 1:number)
  if (mod(number,n) == 0)
    divider(dividerIndex) = n;
  endif
endfor
disp(divider);
```

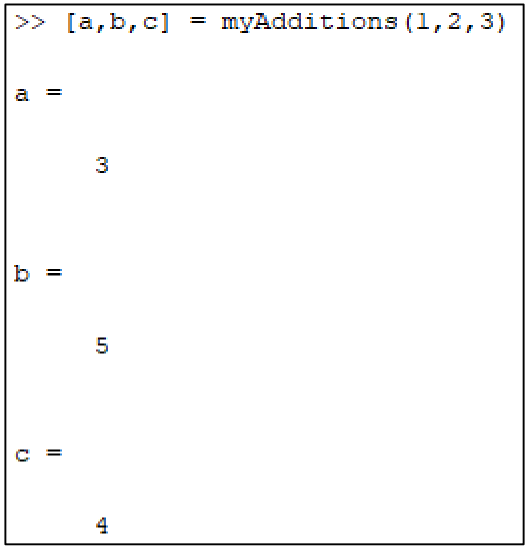
Functions

- Functions take parameters and calculate a result
- They can be called from command window or in other scripts
- They are saved in a *.m*-file
- **ATTENTION: One function per file**

  File name = function name
- At call result must have same structure like defined


myAdditions.m

```
function [y1, y2, y3] = myAdditions(x1, x2, x3)
  y1 = x1 + x2;
  y2 = x2 + x3;
  y3 = x1 + x3;
endfunction

```

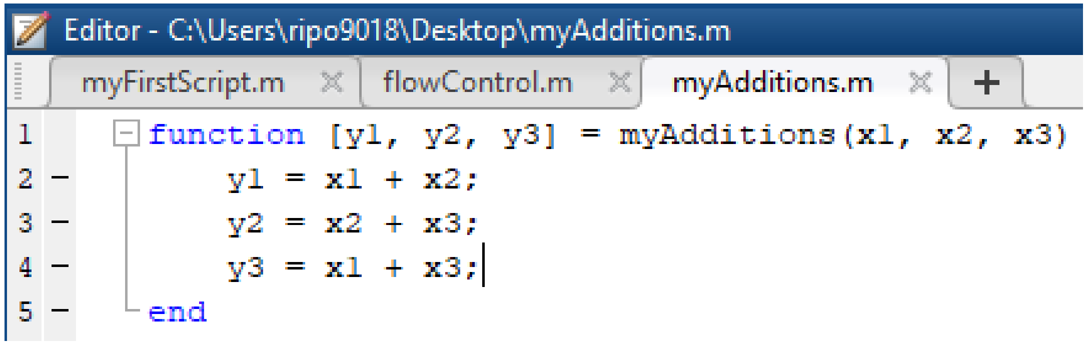


```
>> [a,b,c] = myAdditions(1,2,3)

a =

     3


b =

     5


c =

     4
```

TASKS

- Rewrite the function that it takes and returns vectors instead of 3 individual elements.

  Hint: length(x) returns number of elements in a vector



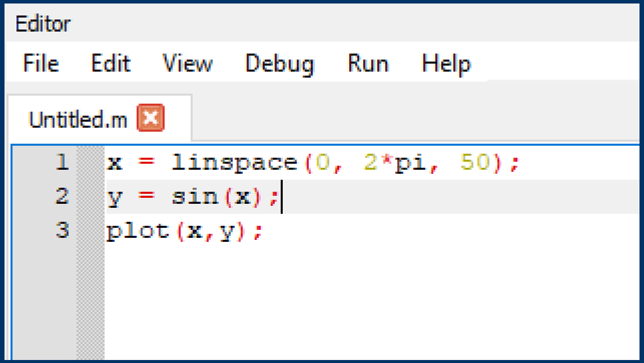
```
function [y1, y2, y3] = myAdditions(x1, x2, x3)
    y1 = x1 + x2;
    y2 = x2 + x3;
    y3 = x1 + x3;
end
```

- Write a function that calculates the sum of a vector

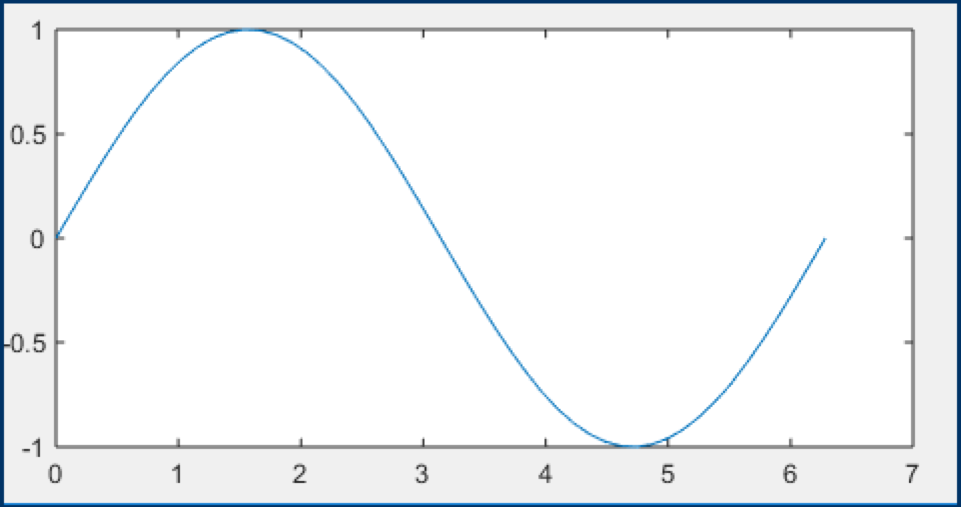
Bauhaus-Universität Weimar

MATLAB for Engineers

Tutorial 3: Linear algebra and equation solving



```
x = linspace(0, 2*pi, 50);
y = sin(x);
plot(x,y);
```



Mathias Artus

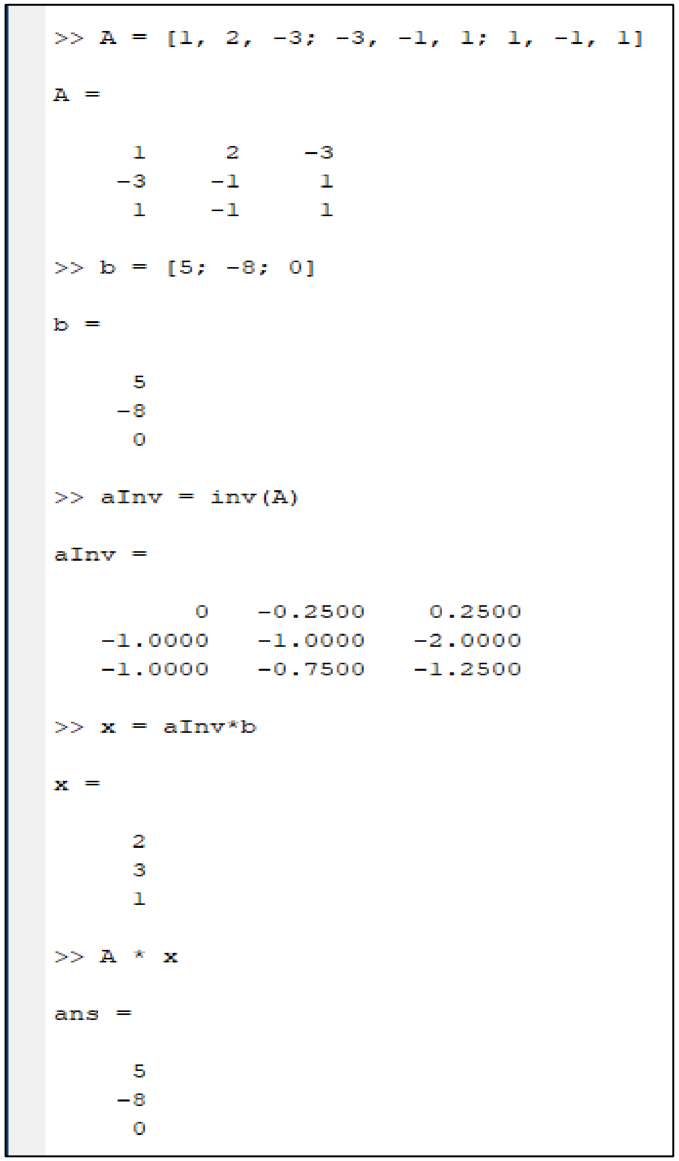
Linear Algebra

- **Given equations:**
  - $x_1 + 2x_2 - 3x_3 = 5$
  - $-3x_1 - x_2 + x_3 = -8$
  - $x_1 - x_2 + x_3 = 0$
- **Matrix equation: A * x = b**
  - **The unknown variables are: x_1, x_2, x_3**

$$\begin{bmatrix} 1 & 2 & -3 \\ -3 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -8 \\ 0 \end{bmatrix}$$

- $x = A^{-1} * b$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -3 \\ -3 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix}^{-1} * \begin{bmatrix} 5 \\ -8 \\ 0 \end{bmatrix}$$



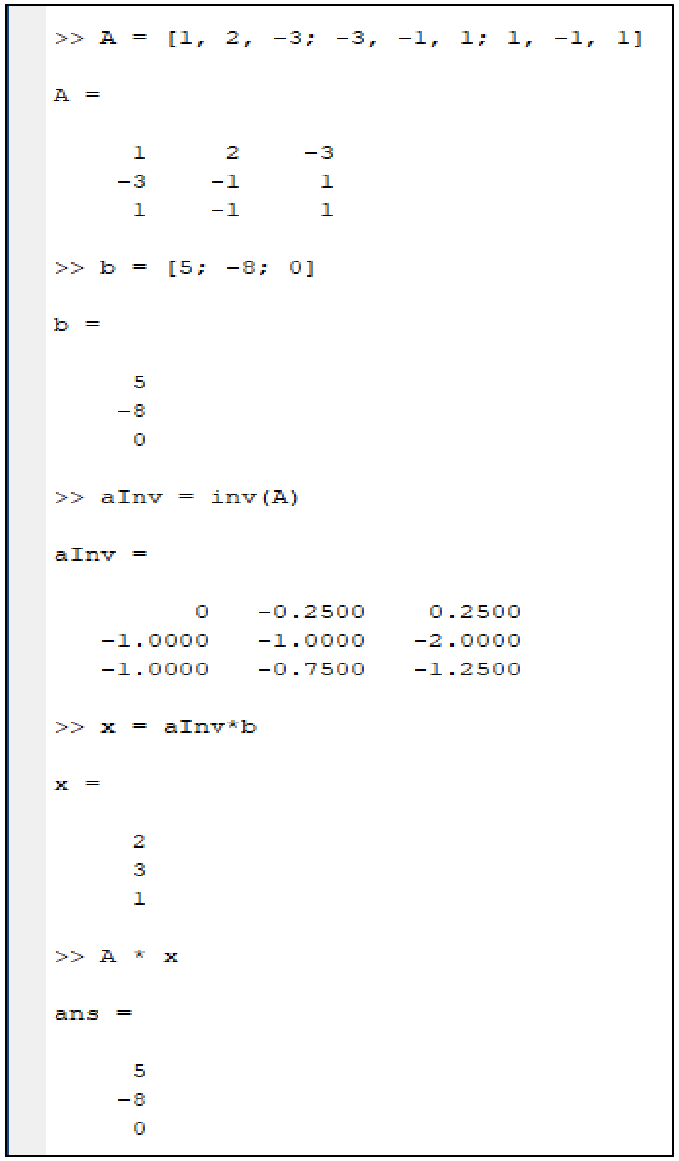
```
>> A = [1, 2, -3; -3, -1, 1; 1, -1, 1]

A =

    1    2   -3
   -3   -1    1
    1   -1    1

>> b = [5; -8; 0]

b =

    5
   -8
    0

>> aInv = inv(A)

aInv =

         0   -0.2500    0.2500
   -1.0000   -1.0000   -2.0000
   -1.0000   -0.7500   -1.2500

>> x = aInv*b

x =

    2
    3
    1

>> A * x

ans =

    5
   -8
    0
```

TASKS

- Given the equations:

$$2x_1 + 5x_2 - x_3 = 9$$
$$-3x_1 - x_2 + 4x_3 = 7$$
$$- x_2 + x_3 = 1$$

- Define the matrix and vectors for the calculation
- Calculate the result in Matlab



```
>> A = [1, 2, -3; -3, -1, 1; 1, -1, 1]

A =

     1     2    -3
    -3    -1     1
     1    -1     1

>> b = [5; -8; 0]

b =

     5
    -8
     0

>> aInv = inv(A)

aInv =

         0   -0.2500    0.2500
   -1.0000   -1.0000   -2.0000
   -1.0000   -0.7500   -1.2500

>> x = aInv*b

x =

     2
     3
     1

>> A * x

ans =

     5
    -8
     0
```

Useful functions

- Rank of a matrix: **rank(matrix)**
  - Linear independent rows/columns
- Determinat: **det(matrix)**
  - Only for square matrices
- Decompositions
  - Eigen: **eig(matrix)**
  - Singular value decomposition: **svd(matrix)**
  - QR: **qr(matrix)**
- „\“ calculates least squares solution and works with rectangular matrices too
  - x = A\b

Polynomials

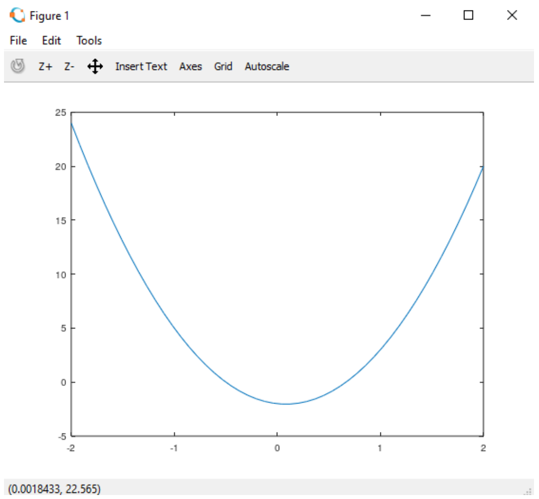
For a polynom: $a_1 + a_2x + a_3x^2 + \dots a_nx^n$

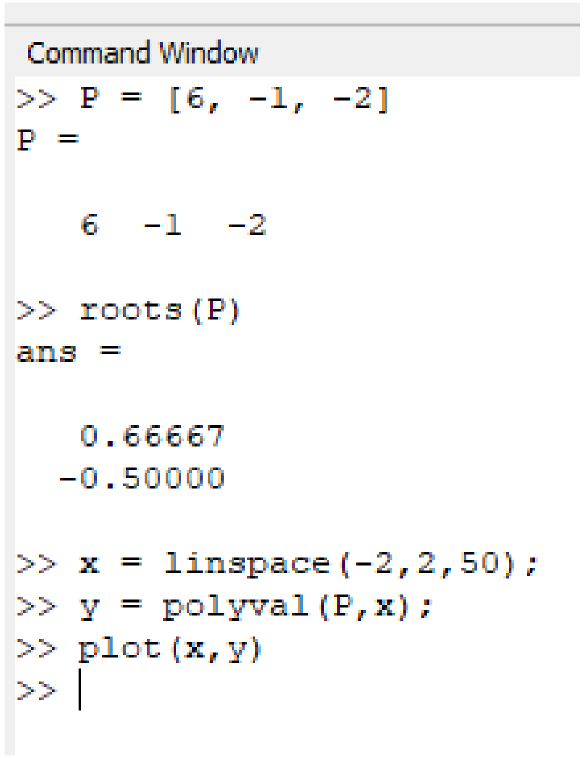
- $P = [a_n, a_{n-1}, a_{n-2}, \dots, a_1]$

Given the example: $6x^2 - x - 2$

- P = [6, -1, -2]

  P of Polynom n-th order has the length N+1
- **roots(P)** computes the roots
- **Polyval(P,x)** computes the results





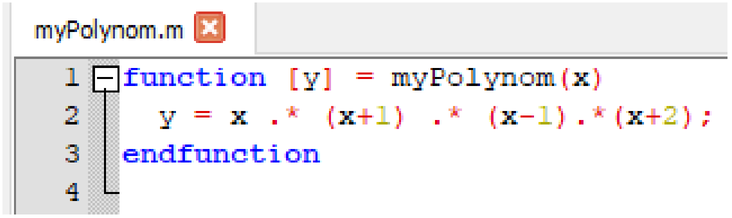
```
>> P = [6, -1, -2]
P =

   6  -1  -2

>> roots(P)
ans =

   0.66667
  -0.50000

>> x = linspace(-2,2,50);
>> y = polyval(P,x);
>> plot(x,y)
>> |
```

TASKS

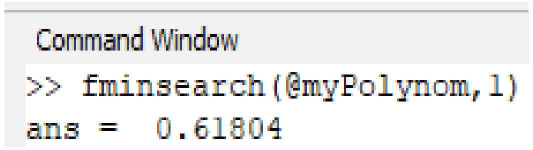
- Given: $x^2 - 2x = f(x)$
- Compute the roots
- Plot the function

Optimization

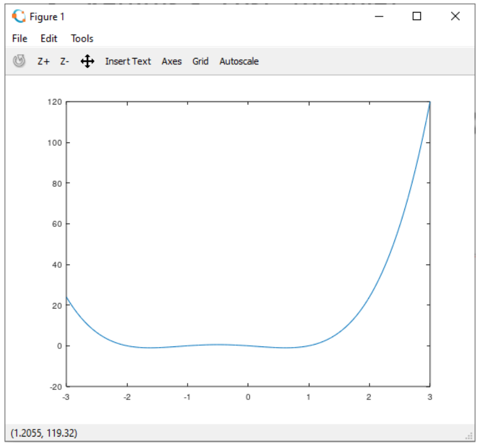
- Minimizing the residual of a function
- Define f(x) as a function in MATLAB (already seen)
  - Returns a „cost“ (double)
  - Takes one Parameter double
- Unconstraint search: *fminsearch*(‚myPolynom‘, 0.1)
  - Starts the search at 0.1
- Constraint minimum search: *fminbnd*(‚myPolynom‘, 0, 3)
  - Search between 0 and 3


myPolynom.m

```
function [y] = myPolynom(x)
  y = x .* (x+1) .* (x-1).*(x+2);
endfunction
```

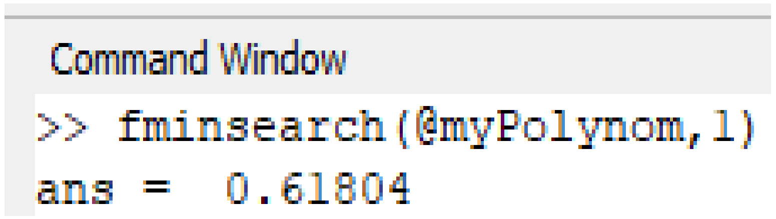

Command Window

```
>> fminsearch(@myPolynom,1)
ans =  0.61804
```



TASKS

- Given: (x – 0.5) * (x + 0.75) * (x - 4) * (x + 3) = f(x)
- Compute the Minimum
- Plot the function for evaluation



```
>> fminsearch(@myPolynom,1)
ans =   0.61804
```

Additional solver

- *glpk*: linear programming
- *qp*: quadratic programming
- …

Further tools

- Optimization package (optim)
- Neural network package (nnet)
- Statistics package
- …

Material based on „Introduction to MATLAB“ from MIT.
Remember: GNU Octave is not MATLAB!

https://ocw.mit.edu/resources/res-18-002-introduction-to-matlab-spring-2008/

Exercises for improving your Octave skills:

https://octave.org/doc/v5.2.0/

Examples for matrix equations:

http://www.math-exercises.com/matrices/matrix-equations

Examples for equation systems:

http://www.math-exercises.com/equations-and-inequalities/systems-of-linear-equations-and-inequalities

Bauhaus-Universität Weimar

Enjoy Coding, simulating and optimizing with ~~MATLAB~~ GNU Octave