About the course

Organisation

- Theoretical lectures on Tuesday 8:00 - 9:45 in BIN 2.A.01
- Exercise classes on Wednesday 8:00 - 9:45 in BIN 0.B.06

Homework

- Every 2 weeks
- Published on Tuesday morning (8:00) of week n
- Deadline on Tuesday morning (8:00) of week $n + 1$

During exercise class

- Correct homework
- New exercises
- Hands on moments (easy ones)
- We solve together (difficult ones)

MATLAB crash course

– First 2 lessons

Material

– Slides of Programming in MATLAB course 2018

– Notes Felix Fontain notes 2013

– Help toolbox of MATLAB

– Google

Outline

Introduction

Flow control (if-statements, loops, ...

Arrays

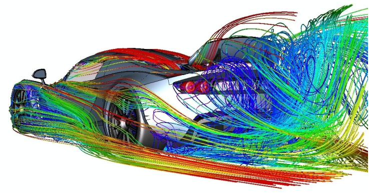
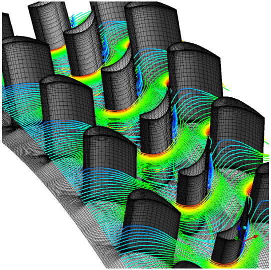
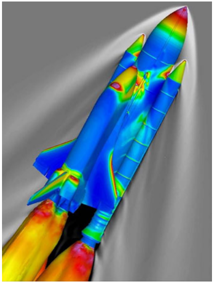
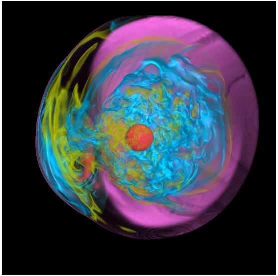
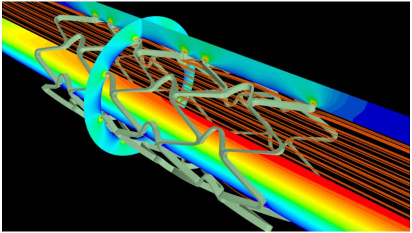
Visualisation

Exercises lesson 2

About MatLab

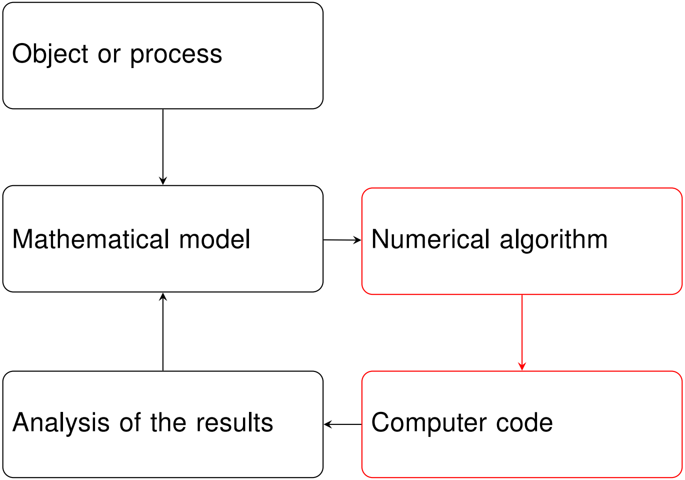
- MATrix LABoratory
- originally a FORTRAN program
- Data elaboration with floating points
- ready-to-use Toolboxes (e.g.: Image processing, financial tools, etc.)
- symbolic computations and exact arithmetic (Symbolic Math Toolbox) (but Mathematica is preferable for this..)
- equivalent software: Octave, Scilab

Numerical simulations

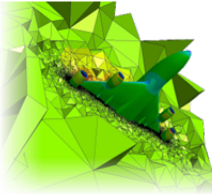
Mathematical modeling

The solution of a real industrial problem can be roughly represented by the following scheme.



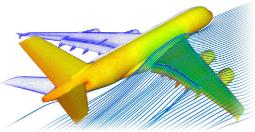
Object

CAD model
3D mesh
Navier-Stokes equations
Numerical methods



Software
HPC simulation

Results
Data visualization
Analysis



Brief Design of Computers

Computer components:

- **CPU:** elaborates instructions in a machine code
- **computer memory:** stores data & instructions
- **bus:** communication system for data between components(CPU, memory, etc.)
- **peripheral:** everything connected to a computer (keyboard, usb, etc.)
- **software, produced via <u>Programming languages</u>:**
  - low-level
  - high-level

Python to Matlab

NumPy	Matlab	Notes
a and b	a&&b	-
a or b	a || b	-
array([[1.,2.,3.], [4.,5.,6.]])	[1 2 3; 4 5 6]	2x3 matrix literal
a[-1]	a(end)	last element a
a[1] or a[1,:]	a(2,:)	second row of a

Python to Matlab

NumPy	Matlab	Notes
a.transpose() or a.T	a.'	transpose of a
a.conj().transpose() or a.conj().T	a'	conjugate transpose of a
a.dot(b)	a * b	matrix multiply
a * b	a .* b	element-wise multiply
a/b	a./b	element-wise divide
a**3	a.ˆ 3	element-wise exponentiation

Python to Matlab

Main differences:

- Indexing starts from 1
- Array/vector access is through ()
- Everything are arrays!
- Arrays have pass-by-value semantics as opposed to reference
- Possible to do Object Oriented, but widely not used

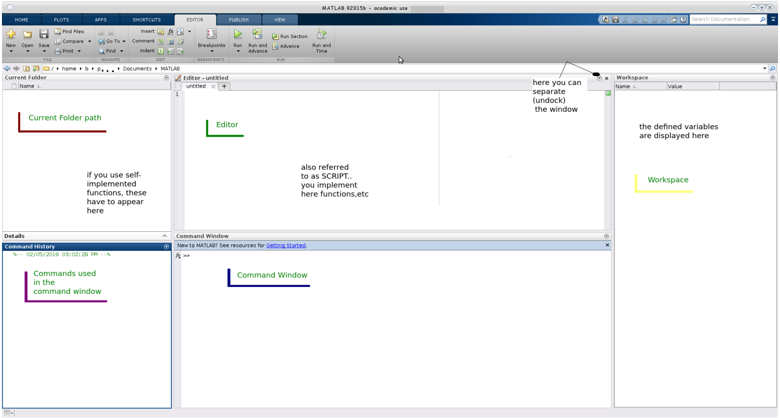
More info: https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

MATLAB

Install MATLAB on your machine with UZH license

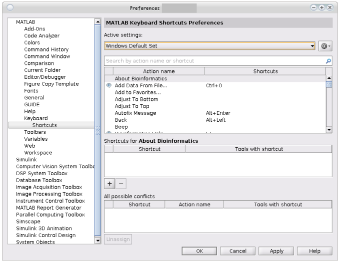
1. Instructions on [UZH website]
2. Type Thinlinc password

The Matlab interface



Watch out: shortcuts!

Setting up the Keyboard shortcuts: Preferences > Matlab > Keyboard > Shortcuts > Windows Default Set



About precision…

Matlab uses *floating-point numbers* $\mathbb{F}$ - different from $\mathbb{R}$!

Example

```
>> 1/7
ans =
0.1429
```

Output formats:

```
>> format Type
```

Type	
`rat`	1/7
`short`	0.1429
`short g`	0.14286
`short e`	1.4286e-01
`long`	0.142857142857143
`long g`	0.142857142857143
`long e`	1.428571428571428e-01

Storing of numbers

Representation of **floating-point numbers**[1]:

$$x = (-1)^s \cdot (0.a_1 a_2 \dots a_t) \cdot \beta^e = (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j} = (-1)^s \cdot m \cdot \beta^{e-t}$$

with

- $s \in \{0, 1\}$
- $a_1 \neq 0$
- $\beta \geq 2$ is the basis adopted by the computer (usually 2)
- m, the *mantissa*, is an integer whose length is the number of digits t
- $a_j \in \{0, \dots, \beta - 1\}$ are the stored digits
- $e \in \{L, \dots, U\}$ is an integer number called *exponent*

$\rightarrow$ floating-point numbers since decimal point not fixed

$\rightarrow$ digits $a_1 a_2 .. a_p$ are also called p first significant digits of x.

[1] Reference: Quarteroni, Saleri, Gervasio - Scientific Computing with Matlab and Octave, IVth ed.

Exercises diy

Representation of **floating-point numbers**:

$$x = (-1)^s \cdot (0.a_1 a_2 ... a_t) \cdot \beta^e = (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j} = (-1)^s \cdot m \cdot \beta^{e-t}$$

$$a_1 \neq 0$$

Use:

$\beta = 2 \quad t = 3 \quad e \in \{-4, \ldots, 3\}$

– What is the smallest number that you can represent?

– What is the biggest number?

– What is the smallest absolute value of a number that you can represent?

– How many bits do you need to store one number?

Exercises diy

Representation of **floating-point numbers**:

$$x = (-1)^s \cdot (0.a_1 a_2 ... a_t) \cdot \beta^e = (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j} = (-1)^s \cdot m \cdot \beta^{e-t}$$

$$a_1 \neq 0$$

Use:

$\beta = 2 \quad t = 3 \quad e \in \{-4, \ldots, 3\}$

- What is the smallest number that you can represent? -7
- What is the biggest number? 7
- What is the smallest absolute value of a number that you can represent? $\frac{1}{32}$
- How many bits do you need to store one number?
  $s : 1, a_i : t-1 = 2, e : 3 \Rightarrow$ 6 bits

Storing in MatLab[1]

General: $\mathbb{F}(\beta, t, L, U)$ characterizes the set $\mathbb{F}$
(namely the basis, # significant digits and range of e via (L, U).
MatLab: $\mathbb{F} = \mathbb{F}(2, 53, -1021, 1024)$ are stored in 8 bytes (called *double precision*)

Smallest and largest positive values of $\mathbb{F}$:

$$x_{min} = \beta^{L-1}, \quad x_{max} = \beta^{u}(1 - \beta^{-t}).$$

MatLab:

```
>> realmin
ans =
2.2250738585072e-308

>> realmax
ans =
1.79769313486232e+308
```

[1] Reference: Quarteroni, Saleri, Gervasio - Scientific Computing with Matlab and Octave, IVth ed.

Exercises diy

Represent the following floating points in base 2, or explain why this is not possible to be done exactly.

a) 15

b) 0.5

c) 0.1

d) 1/3

e) 1/256

Round-off error[1]

Definition

It is the error generated when a real number is replaced with a floating-point number.

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_M$$

with $\epsilon_M = \beta^{1-t}$ called the machine epsilon.

Matlab: $\epsilon_M = 2^{-52} \sim 2.22 \cdot 10^{-16}$:

```
>> eps
ans =
2.22044604925031e-16
```

[1] Reference: Quarteroni, Saleri, Gervasio - Scientific Computing with Matlab and Octave, IVth ed.

Good to know: Sources of computational errors

Computational errors can arise due to

1. Incompleteness of mathematical model.
2. Errors in the initial (input) data (e.g. from experimental measurements).
3. Approximate methods to solve the equations of the mathematical model.
4. Fixed-size data storage format on computers, round-off errors.
5. ...

Exercises… Get started: Scalar assignment

1. Type in the command line:

```
>> a= 2.45
>> A= 3.1
```

Note: >> is called prompt.

Exercises... Get started: Scalar assignment

Typing in the command line:

```
>> a = 2.45
>> A = 3.1
>> A = 7.2
>> a= 1.2,
>> a= 1.7, b=2.45
>> a= 1.2;
>> a= 1.7; b=2.45
>> a= 1.7, b=2.45
>> a= 1.7; b=2.45
```

Get started: Cleaning of memory

– delete variable *A* from workspace:

```
>> clear A
```

– delete **all variables** from workspace:

```
>> clear
>> clear all
```

– **clean up** the visualization window:

```
>> clc
>> home
```

Good habits: Clean your workspace and visualization always before starting a new exercise.

Get started: Discover new instructions

– to see how to use an instruction, ex. called `instruction`:

```
>> help instruction
```

Example:

```
>> help format
```

Mathworks website

Expressions

Arithmetic operators (in decreasing precedence order)

`( )`	brackets	operations precedence
`^ * /`	power/multiplic./division	arithmetical operations
`+ -`	addition/subtraction	arithmetical operations
`== ~=`	equivalent to/unequal to...	equivalence operators
`< <= > >=`	less/greater than...	equivalence operators
`& | ~`	and/or/not	Logic symbols

To know more:

```
>> doc 'operator precedence'
```

Expressions

Numerical expressions:

```
>> 1

>> 0.23, % or
>> .23

>> 23e-2 % or
>> 23*10^(-2)

>> 5+4i % or
>> 5+4j
```

Functions:

call with `functionName(parameter1, parameter2, ...)`

sin	sqrt	log	floor
cos	abs	log2	ceil
tan	exp	log10	round

Predefined Variables

`ans`	results of last computation
`pi`	$\pi = 3.14..$
`eps`	machine precision
`i,j`	$\sqrt{-1}$
`nan`	not a number (ex. as result of 0/0)
`inf`	infinity (ex. as result of 1/0)
`realmin, realmax`	smallest/biggest fluctuation-point number

Attention:

don't overwrite these variables!!

To check the complete word list:

» `iskeyword`

Scripts

many commands ⇒ **Scripts** convenient

- click "new script"
- write the following:

```
>> z=23;
>> hallo=12*z
```

- save the script with the name "*FirstScript.m*".
- run the script by clicking on the run button
- run the script from command line via `FirstScript`
- by typing the inital letters of `FirstScript` and TAB: autofill
- workspace shows z and hallo variables

Comments

→ Good habit: comment whatever you write

Comment types

```
% single line comment

%{
multi line
comment
%}

%% describes a codeblock, splits into different sections
```

Why comments?

- ease in reading codes
- easy to maintain
- documentation: describe in the beginning of the script a comment with
  - what the script does
  - what variables it takes as input or output
  - further useful info
- these comments are displayed when typing

  ```
  >> help FirstTest
  ```

  or

  ```
  >> FirstTest + F1
  ```

Functions

- none, one or many inputs (arguments) and outputs
- go to New> Function
- change it to:

```
function y = power8( x )
% Compute y=x^8 (This is a comment, displayed in the help)
y=x^8;
end
```

- save it to `power8.m`
- type in command line:

```
>> power8(2)
```

Note: The function name must be the same of the .m file!

Functions: multiple arguments

– in case of multiple inputs or outputs

```
function [y1, y2] = Multi_inout_function(x1,x2,x3)
  % example function with more than one input
  y1=x1;
  y2=x2+x3;
```

– call from command line:

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

– if not interested in some outputs:

```
>> a = L103_function(1,2,3)
>> [~,b]=L103_function(1,2,3)
```

– Variable inputs/outputs possible: see `varargin`, `varargout`, `nargin`, `nargout`.

Multiple functions in a file

– multiple functions in a single file are possible:

```
function y=power8(x)
y=mypower8(x);
end

function y=mypower8(x)
y=x^8;
end
```

Attention: only the function with the same name as the file can be called by the command line or other scripts!!

Local variables in functions

interface between function and the workspace:
only via input output arguments

```
  function y=power8_local(x)
  % none ;  -> this appears in the function description
  y=x^8
  % the next line does not generate any
  % variable in the workspace
  myVar = 5;
end
```

This means: everything which does not appear in the first line of a function stays *local*

→ useful if variables are called the same in different functions

Anonymous Functions

– Functions (e.g. equations) written without .m:

Example

```
>> f = @(x,y) x^y
>> f(2,3)
```

– the right hand side is an anonymous function.

– `f` is a `function_handle`.

```
>> class(f)
```

Condition statement

Definition

→ commands in the if-statement are **only considered** if statement fulfilled.

```
if (condition statement)
(matLab commands)
end
```

Example

```
if x == 5   % is carried through, if x=5
  y=1;
elseif x==6
  z=2;
else
  y=3;
end
```

for-loops

Definition

$\rightarrow$ carries out same commands for several times (e.g. for the terms in a vector)

```
for index=start:end
  ( commands )
end
```

Example

```
for k=1:10
  disp(k^2)
end
```

- `continue`: jumps to the loop starting
- `break`: interrupts the loop

while-loops

→ Execute loops, until a condition is specified.

```
a=0;
while a<5
  a=a+1;
end
```

Example (bad example - Ctrl+C interrupts script)

```
a=6;
while a>5
  a=a+1;
end
```

Exercises... Condition statements & Loops

1.home

a) Write a script which contains:
   - if
   - for
   - while
   - continue
   - break
   - a call to a self implemented function

b) Check the execution through the debugger.

2.live

- Compute the sum of the square of the first 1000 integer numbers, i.e., $\sum_{k=1}^{1000} k^2$ using a for loop
- Compute the product of the first 100 integers $\prod_{k=1}^{100} k = 100!$ using a while loop
- Compute the sums of the even integers < 100 and odd integers < 100 in one loop.

Arrays

Very important part of Matlab. Two types of arrays:

1. scalar (1×1), vector ($n \times 1$ or $1 \times n$) and matrix ($n \times m$)
2. cell array of objects

Exercise... Arrays (live)

a) Try the following in the command line:

```
>> a=[1 2 3 4]
>> b=[1, 2, 3, 4]
>> c=[1; 2; 3; 4]
```

b) Check the workspace and look at the information about a, b, c.

c) What do: space or , or ; do?

d) Compute

```
>> a+b
>> a+c
```

Matrices

Example

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

Matrices in Matlab

- definition via `[ ]`
- space or `,` divides columns
- `;` divides lines
- to access values:

  ```
  >> A(2,3)  % 2nd line, 3rd column (=6)
  ```

- change of single values:

  ```
  >> A(2,3)=7
  ```

Size of matrices

Check the dimension of the array

- check the workspace
- call the variable through the command line without ;
- use Matlab-inbuilt function `size`

```
>> size(a)
>> size(c)
>> size(A)
>> size(A,1)
>> size(A,2)
```

- to check the length of vectors:

```
>> length(a)
>> length(c)
>> length(A)
```

- go to last element of vector *a*:

```
>> a(end)
```

Built-in structures of matrices

A few examples of the matrix built-in layouts:

- $n \times n$ identity matrix

```
>> eye(5)
```

- 3×8 matrix of ones

```
>> ones(3,8)
```

- 4×6 matrix of zeros

```
>> zeros(4,6)
```

- matrix with random values

```
>> rand(5)
>> randn(3,4)
```

- equispaced points from a to b

```
>> linspace(a,b,N)
>> linspace(-1.2,2.5, 12)
```

- magic square

```
>> magic(5)
```

- empty matrix

```
>> A=[]
```

- & many more..

Operations with matrices

<table>
<tr><td>+</td><td>-</td><td></td><td>element-by-element sum/division</td></tr>
<tr><td>'</td><td>.'</td><td></td><td>conjugate transposed (ex. A'), transposed (ex. A.')</td></tr>
<tr><td>*</td><td></td><td></td><td>multiplication (ex. B*A, 3*B)</td></tr>
<tr><td>^</td><td></td><td></td><td>Power for square matrix</td></tr>
<tr><td>.*</td><td>./</td><td>.\</td><td>element-by-element multiplication/division</td></tr>
<tr><td>.^</td><td></td><td></td><td>element-by-element power</td></tr>
<tr><td>|</td><td>&</td><td>~</td><td>element-by-element logic operations</td></tr>
<tr><td>==</td><td>~=</td><td>< > <= >=</td><td>element-by-element comparing operators</td></tr>
<tr><td>inv</td><td></td><td></td><td>inverse of matrix</td></tr>
<tr><td>\</td><td></td><td></td><td>$A\backslash b$ solves the system of equations $A * x = b$</td></tr>
<tr><td>diag</td><td></td><td></td><td>diagonal matrix (ex. B=diag([1,2])
or extract diagonal elements ex. diag(magic(5))
Note: diag(A,k) gives the k-th diagonal
diag([3,2],k) builds a matrix with k-th diagonal</td></tr>
</table>

Broadcasting operations

Attention: new versions of MATLAB (from 2016b) are broadcasting the operations! You can add a scalar to a matrix

```
>> A = [1, 2; 3, 4];
>> A + 1
   ans =
     2     3
     4     5
```

or vector to a matrix

```
>> b = [3, 2];
>> A+b
   ans =
     4     4
     6     6
```

Broadcasting operations

Attention: new versions of MATLAB (from 2016b) is broadcasting the operations! You can also add a vector to a vector

```
>> b = [3, 2];
>> a = [1; 4];
>> a+b
   ans =
      4     3
      7     6
```

Useful commands when dealing with matrices

– assembling matrices

```
>> C=[b A]
>> D=[C; A B]
```

– change shape (columns are filled first)

```
>> A2=reshape(A,3,2)
```

– most functions react on matrices element-wise

```
>> sin(A)
```

– submatrices:

```
>> A3(1:3, 2:4)
>> A3(1:3, [1 4 3])
>> A3(1:3, 1:2:6)
>> A3(1:5)
>> A3(1, 1:end)
>> A3(1, 1:end-1)
>> A3(:,1:3)
>> A3(:)
```

– the expression

```
a:b:c
```

Generates (row-)vector with values: a to c with step b. If b is omitted it defaults to 1.

Example

```
>> 1:3:10
>> .1:.2:5
>> plot(sin(0:.01:2*pi))
```

Some tips when dealing with matrices

- clear columns or lines:

```
>> A=rand(5)
>> A(:,[2,4])=[]
```

- array dimension is changed automatically when new elements added:

```
>> A=[1 2 3 4]'
>> size(A)
>> A(25)=3
>> size(A)
```

Useful functions

- `max`, `min`: find the maximum/ minimum of a vector.
  If applied to matrices: column-wise

```
>> max(A(:))
>> max(A)
```

- similarly: `sum`, `mean`, `median`, `std`

- to get the **dimension along which the function operates**:

```
>> mean(A,2)
```

Exercises... Matrices

1.home Construct in Matlab the following matrices in an efficient way (use `help diag`):

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 & 1 & 3 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 4 & 0 \\ 2 & 2 & 2 & 2 & 0 & 0 & 5 \\ 2 & 2 & 2 & 2 & 6 & 6 & 7 \\ 2 & 2 & 2 & 2 & 6 & 6 & 8 \end{pmatrix}$$

$$C = \begin{pmatrix} 5 & 1 & 0 & 0 & 0 & 5 & 5 & 6 & 7 & 8 \\ 6 & 0 & 2 & 0 & 0 & 6 & 1 & 0 & 3 & 0 \\ 7 & 0 & 0 & 3 & 0 & 7 & 0 & 2 & 0 & 4 \\ 8 & 0 & 0 & 0 & 4 & 8 & 5 & 6 & 7 & 8 \end{pmatrix}$$

Exercises… Matrices (home except last)

2.home Generate a 10×10 random matrix.
Find the values and indexes of the biggest element of each row.

Possible hints: `sort`, `max`, `find`

3.home Do the inverse of a random vector (entry–wise).

Exercises... Matrices

4.home Write a function, which shifts cyclically a vector or matrix by a certain number of rows down and columns to the right (do **not use** `circshift`).

5.home Checkboard:

a) Generate a random (`rand`) matrix A with size $n \times n$ and a second matrix with the same size as A and which takes some elements of A in the black fields, while zeros everywhere else. The structure of the second matrix should be similar to a chessboard.

b) Make sure that your code works for the cases when n is even and odd .

To check if you succeeded try `spy`.

Exercises... Matrices

6.home What in-built functions of MatLab can you use to compute, given a matrix of type $m \times n$:

- average,
- median,
- mode.

7.home Check what the following functions do:

- `rank`,
- `null`,
- `rref`,
- `orth`.

Exercises... Matrices

8.home Consider A to be a square matrix, as

```
>> A=[1+i, 1+2i; 2, 4i];
```

Check the following expressions:

```
>> A'
>> A.'
>> A.^A
```

9.home Consider `A=rand(10)` and check

```
>> diag(diag(A)) + diag(diag(A,1),1)+ diag(diag(A,2),2)
```

10.diy Generate a table with N values of the function $f(x)$ on the interval $[a, b]$, s.t. you get an $N \times 2$ matrix having on each row x and $f(x)$. Consider the following functions:

a) $f(x) = x^5 - 4x + 1$

b) $f(x) = exp(i\frac{x}{10})$

Displaying results

Example (Plot of a vector)

```
>> x=linspace(0,2*pi,200);
>> y1=sin(x);
>> y2=cos(x);
>> plot([y1',y2'])
>> plot(y1)
>> plot(y2)
```

→ We are interested also in x:

```
>> plot(x,[y1', y2'])
>> plot(x,y1,x,y2)
```

Try:

```
>> plot(y1, y2)
>> plot([y1; y2])
```

what changes?

Pictures complete of information

For **complete** and **useful** pictures, some features have to be added:

– Legend - when more items displayed, makes it clear what is what

```
>> legend('sin', 'cos')
```

– Title

```
>> title(' My plot')
```

– Colors and line tipology (if not specified: automatic colors)

```
>> plot(x, y1, 'r*-', x, y2, 'k:')
```

– axis labels

```
>> xlabel('x')
>> ylabel('y')
```

– change x-axis ranges to $[0, 2\pi]$ and y-axis to $[1.5, 1.5]$

```
>> axis([0 2*pi -1.5 1.5])
```

How to save pictures?

To save your plots you can

– in your command window

```
>> fig = figure()
>> % call your plot()
>> savefig('my filename0')
>> savefig(fig,'my filename1')
>> saveas(fig, 'my filename2')
```

– on the figure window go to File > Save As and choose the format

Hint: to change the output format `saveas(fig, 'my filename', FORMAT)`, where you can choose eps, png, etc.

Multiple information on same plot

– when for example in loops, an alternative to

```
>> plot(x, y1, x, y2)
```

is to apply `hold on`:

```
>> plot(x, y1)
>> hold on
>> plot(x, y2)
>> hold off
```

Multiple plots

– if visualization desired in different figures: either

```
>> figure()
>> plot(x,y1)
```

or specifying

```
>> figure(1) % Choose figure 1
>> plot(x,y1)
>> figure(2) % Choose figure 2
>> plot(x,y2)
```

Theory..Sub-figures

For **multiple sub-figures** within a figure:

– `subplot(a,b,n)` with `n` the place where the plot is collocated, while `a,b` the dimension of the fictive matrix where it is collocated.

Example

```
>> subplot(1,2,1)
>> plot(x,y1)
>> subplot(1,2,2)
>> plot(x,y2)
```

Good to know:

`clf` to **clear active figures**

`close` or `close all` to **close the last or all figures**.

Different scale plots

Plot the functions $f(x) = e^{3x}$, $g(x) = x^x$, $h(x) = \log(x)$ and $j(x) = 3x$ on $[0.01, 100]$.

- Use `plot`
- Use `semilogx`
- Use `semilogy`
- Use `loglog`.

Which plot is better for each function?

This is usefull when we are interested in logarithmic and exponential functions. (Very common in error plots)

Exercises... Plots

1.live Plot in one figure the functions $f(x) = e^{x/10} \sin(2\pi x)$ and $g(x) = log(3 + x) \cos(4\pi x)$ on the interval $[0, 1]$. The plot be such that:

- f is plotted in red colour and dashed lines,
- g is in blue and it is alternating dots and dashes,
- it should contain a title "Cute functions"
- the x axis is delimited by 0 and 1 and it has a label "Time"
- the y axis is delimited by -2 and 2 and it has a label "Money"
- there is a legend: for f is "Marc" and for g is "John".

Then, save the plot as "my_cute_functions.fig" and close the figure (using the functions we learned).

Exercises... Plots

2.home Check `help plot`. Create with the obtained information the plots for:

a) a function $f = x^2 - 0.5$ on $[-1, 1]$ with red and dashed lines;
b) the functions $f = sin(s * x)$ with $s = 1, 2, 3, 4$ with different colors and a legend;
c) $f = sin(2 * pi * x)$. Here the function values have to be displayed as little black circles at the points $x = n/10$.

Exercises...

1.live a) Write a function that approximates the derivative of an anonymous function f in a point x_0 using the Taylor expansion

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}. \tag{1}$$

b) Apply the function on $f(x) = \sin(x)$ in $x_0 = 1.2$.

c) Test the algorithm with different hs and plot the error for different hs.

Exercises...

2.home

a) Write a function `fibo(n)` that computes the members of the Fibonacci sequence $1, 1, 2, 3, 5, 8, ..$

- once without loops as a recursive function
  $f(n) = f(n-1) + f(n-2) \quad \forall n > 2, \quad f(1) = f(2) = 1.$
- once with loops

b) Write a documentation text for the implemented function

Exercises...

3.live Check numerically :

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

4.diy Bisection method

a) Write a function, that takes a continuous function f and two values (boundary) a, b, such that $f(a) \cdot f(b) < 0$. The function will return the x for which $f(x) = 0$. This is done through the bisection method on the interval $[a, b]$.

b) Check the function on `sin` to compute `pi`.

Exercises...

5.home Newton method

a) Write a function, that takes as input a function f, its derivative f' and an initial value x_0. This function will give as output a zero of f found with the Newton method, that reads

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Be careful, $f'(x) \neq 0$, so, choose properly x_0.

b) Check your implemented function using the function `sin` to compute `pi`.