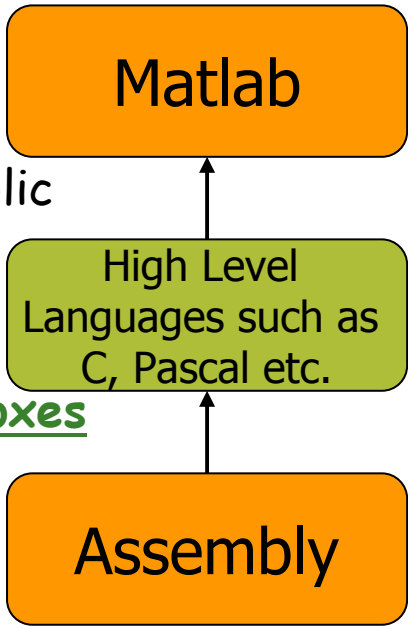
Introduction to Matlab

Outline

- What is Matlab?
- Matlab desktop & interface
- Scalar variables
- Vectors and matrices
  - **Exercise 1**
- Booleans
- Control structures
- File organization ⇨User defined functions
  - **Exercise 2**
- Structures
- Conclusion

What is Matlab?

- "MATLAB": acronym for MATrix LABoratory
- Scientific programming and visualization language based on matrix computations.
- Developed at the University of New Mexico in the early 1980's.
- Commercialized by The MathWorks, Inc. www.mathworks.com.
- Object-oriented programming language with built-in vectorization capability.
- Its core consists of a library of Fortran, C, and symbolic solvers.
- It offers very powerful graphics capabilities.
- **High level language** which has many **specialized toolboxes** for making things easier for us
- MATLAB is used in a wide range of scientific fields for R&D and industrial applications.

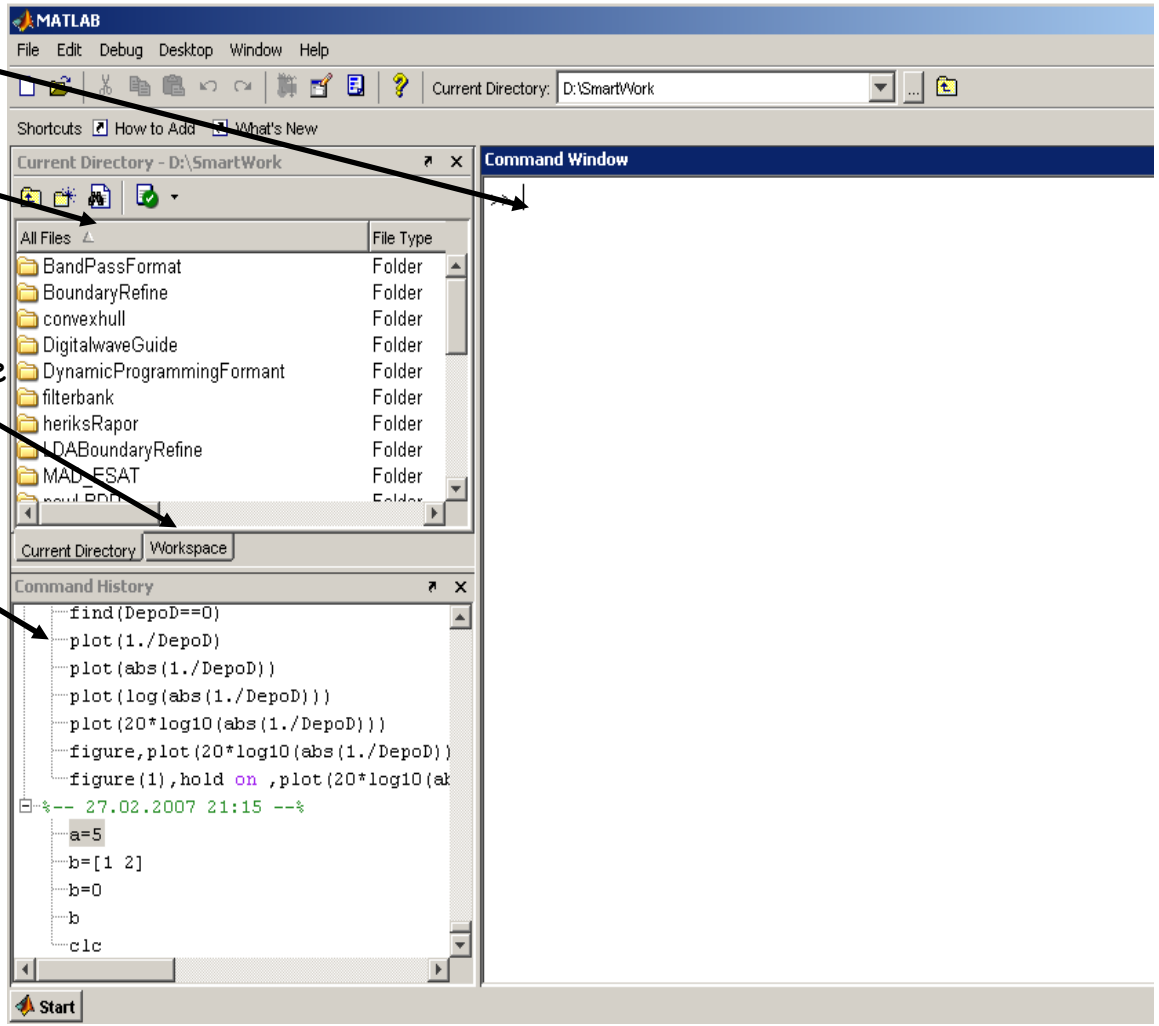


Non exhaustive list of basic capabilities

- Input/output data, create cell and structure arrays
- Matrix manipulations (A+B, A*B, eigen-solvers, etc..)
- Linear algebra (factorization, SVD, etc..)
- Basic and advanced mathematical functions
- Signal processing (FFT, IFFT, wavelets, etc..)
- Finite differences, numerical integration schemes
- Numerical optimization
- Resolution of Ordinary Differential Equations
- Statistics
- Graphical capabilities in 2D and 3D
- …

Matlab Desktop & Interface

- **Command Window**
  - Run commands
- **Current Directory**
  - View folders and m-files
- **Workspace**
  - View program variables
  - Double click on a variable to see it in the Array Editor
- **Command History**
  - View past commands
  - Save a whole session using diary
- **Additional windows**
  - Help Window
  - Editor Window
  - Array Window
  - Graphics Editor Window



Scalar variables: definition

- No need for type definition. i.e.,

```
int a;
double b;
float c;
```

- All variables are created with double precision unless specified and they are matrices.

Example:

```
>> a=5;
>> b=1.35;
>> c=pi;
>> d = 1.2 + 3.4i
```

- Scalar variable is stored internally as a (1 by 1) matrix.
- **whos** : To get a list of all the variables created in the workspace.
- **clear** : to clear all the variables in workspace or specified variable (ex : >> clear a)
- "**;**" : semi-column at the end of the command ⇨ to suppress the display of results

Scalar variables: operators & functions

Arithmetic Operators

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `^` power
- `'` complex conjugate transpose

Few Scalar functions

Description	MATLAB
$a+b$, $a-b$, ab, a/b	`a+b, a-b, a*b, a/b`
$\sqrt{a}$	`sqrt(a)`
a^b	`a^b`
$\|a\|$ (note: for complex arguments, this computes the modulus)	`abs(a)`
e^a	`exp(a)`
$\ln(a)$	`log(a)`
$\log_2(a)$, $\log_{10}(a)$	`log2(a), log10(a)`
$\sin(a)$, $\cos(a)$, $\tan(a)$	`sin(a), cos(a), tan(a)`
$\sin^{-1}(a)$, $\cos^{-1}(a)$, $\tan^{-1}(a)$	`asin(a), acos(a), atan(a)`
$\sinh(a)$, $\cosh(a)$, $\tanh(a)$	`sinh(a), cosh(a), tanh(a)`
n MOD k (modulo arithmetic)	`mod(n,k)`
Round to nearest integer	`round(x)`
Error function $\mathrm{erf}(x) = (2/\sqrt{\pi}) \int_0^x e^{-t^2} dt$	`erf(x)`

Vectors & Matrices: definition

- Vector

  >> b = [1 2 3 4]; ⇨ $b = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$

  >> c = [10 ; 20 ; 30 ; 40]; ⇨ $c = \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \end{pmatrix}$

- Vector index

  >> b(3) ⇨ ans = 3

  >> c(end) ⇨ ans = 40

- Matrix

  >> D = [1 2 3; 4 5 6] ⇨ $D = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

- Matrix index

  >> D(4) ⇨ ans = 5

  >> D(2,3) ⇨ ans = 6

  >> D(1,2:3) ⇨ ans = 2 3

  >> D(1, :) ⇨ ans = 1 2 3

Matrix indices begin from 1 (not 0 as in C) and must be positive integer

>> D(-2)
Error: ??? Subscript indices must either be real positive integers or logicals.

>> D(4,2)
Error: ??? Index exceeds matrix dimensions.

Matrices: concatenation & definition

- List

```
>> b = 1:3                    ⇨ ans = 1 2 3
>> c = 2:-0.5:-1              ⇨ ans = 2  1.5  0.5  0  -0.5  -1
>> x = linspace(0, 2*pi, 5)   ⇨ ans = 0  1.5708  3.1416  4.7124  6.2832
>> D = [1:3 ; 4:6]            ⇨ D =
```

$$D = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

- Concatenation

```
>> x = [1 2]; y = [4 5]; z=[3 6]
>> A = [ x y]                 ⇨ ans = 1  2  4  5
>> A = [ x; y]                ⇨ ans = 1  2
                                      4  5

>> A = [x y ; z]
Error:
??? Error using ==> vertcat CAT arguments dimensions are not consistent.
```

Matrices: generation from functions

- **zeros(M,N)** : MxN matrix of zeros

  >> x = zeros(1,3) ⇨ ans = 0 0 0

- **ones(M,N)** : MxN matrix of ones

  >> x = ones(1,3) ⇨ ans = 1 1 1

- **rand(M,N)** : MxN matrix of uniformly distributed random numbers on [0;1]

  >> x = rand(1,3) ⇨ ans = 0.6872 0.9871 0.2368

- **From loading external data files into Matlab memory**
  - Step 1: Import data file and create a file called **A.mat**
  - Step 2: Load **A.mat**

```
%...Definition of the matrix "A" in ASCI format
      1,    2,    3,
      4,    5,    6,      <- The coma is optional.
      7,    8,    9,
%...End of definition of matrix "A"
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrices: operations

Given A and B:

```
>> A = [1 2 3 ; 4 5 6 ; 7 8 9]

A =

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

```
>> B = [10 20 30 ; 40 50 60 ; 70 80 90]

B =

    10    20    30
    40    50    60
    70    80    90
```

Addition

```
>> A + B

ans =

    11    22    33
    44    55    66
    77    88    99
```

Subtraction

```
>> A - B

ans =

    -9   -18   -27
   -36   -45   -54
   -63   -72   -81
```

Product

```
>> A * B

ans =

   300   360   420
   660   810   960
  1020  1260  1500
```

Transpose

```
>> A'

ans =

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

Matrices: operations elements by elements

```
A = [1 2 3; 5 1 4; 3 2 1]
    A =
          1    2    3
          5    1    4
          3    2   -1
```

.* element-by-element multiplication

./ element-by-element division

.^ element-by-element power

```
x = A(1,:)

x=
     1   2   3
```

```
y = A(3 ,:)

y=
     3   4  -1
```

```
b = x .* y

b=
     3   8  -3
```

```
c = x . / y

c=
    0.33   0.5   -3
```

```
d = x .^2

d=
     1    4    9
```

```
>> x^2
Error:
 ??? Error using ==> mpower  Matrix must be square.
>> x*y
Error:
??? Error using ==> mtimes Inner matrix dimensions must agree.
```

Matrices: few functions

Description	MATLAB
Vector dot product $\vec{x} \cdot \vec{y} = \vec{x}^T \vec{y}$	`dot(x,y)`
Vector cross product $\vec{x} \times \vec{y}$	`cross(x,y)`
Matrix multiplication AB	`A * B`
Element-by-element multiplication of A and B	`A .* B`
Solve $A\vec{x} = \vec{b}$	`A\b` Warning: if there is no solution, MATLAB gives you a least-squares "best fit." If there are many solutions, MATLAB just gives you one of them.
Determinant of **A**	`det(A)`
Inverse of **A**	`inv(A)`
Trace of **A**	`trace(A)`
Compute AB^{-1}	`A/B`
Element-by-element division of A and B	`A ./ B`
Compute $A^{-1}B$	`A\B`
Square the matrix A	`A^2`
Raise matrix A to the k^{th} power	`A^k`
Raise each element of A to the k^{th} power	`A.^k`
Rank of matrix A	`rank(A)`
Set **w** to be a vector of eigenvalues of **A**, and **V** a matrix containing the corresponding eigenvectors	`[V,D]=eig(A)` and then `w=diag(D)` since MATLAB returns the eigenvalues on the diagonal of **D**

Note: 2 useful commands

⇨ for help to use a function
>> help *functionname*

⇨ to find functions
>> lookfor *keyword*

Exercise 1: basic graphic commands

- **Plot the function $e^{-t}sin^3(2\pi t)$ between $0 \leq t \leq 3$**
  - Define a time column-vector *t* of 100 samples starting at time 0.0 second, ending at time 3.0 second.
    - Function **linspace**
  - Calculate $y1 = sin^3(2\pi t)$
    - Function **sin** and power elements by elements **(.^)**
  - Calculate $y2 = e^{-t}$ and $y3 = e^{-t}sin^3(2\pi t)$
    - Function **exp** and multiplication elements by elements **(.*)**
  - Plot y1, y2 and y3 versus *t* on the same plot.
    Add a grid, legend, title and axis labels.
    - Functions **plot, grid, legend, title, xlabel, ylabel**.

Note: to save a copy of the figure on disk space, explore the figure menu (select "file" then "export") and export the figure to a graphics file

Exercise 1: Solution

- **Plot the function $e^{-t}sin^3(2\pi t)$ between $0 \leq t \leq 3$**
  - Define a time column-vector *t* of 100 samples starting at time 0.0 second, ending at time 3.0 second.

```
>> t=linspace(0,3,100);
```

  - Calculate $y1 = sin^3(2\pi t)$

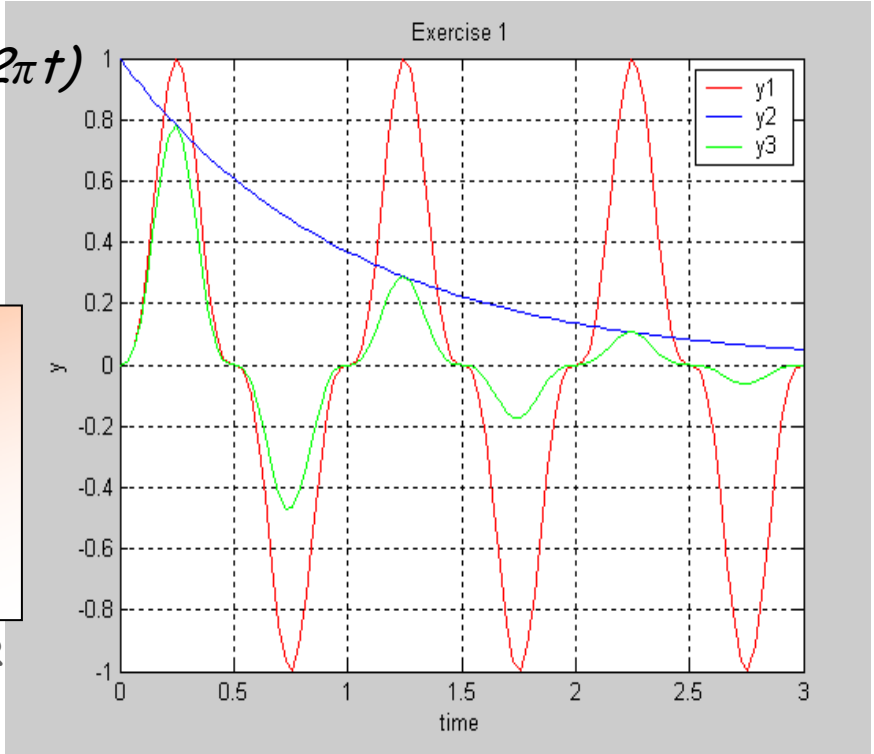
```
>> y1= sin(2*pi*t).^3;
```

  - Calculate $y2 = e^{-t}$ and $y3 = e^{-t}sin^3(2\pi t)$

```
>> y2= exp(-t);
>> y3 = y2.* y1;
```

  - Plot *y1*, *y2* and *y3* versus *t*

```
>> plot(t,y1,'-r',t,y2,'-b',t,y3,'-g');
>> grid;
>> legend('y1','y2','y3');
>> xlabel('time'); ylabel('y');
>> title('Exercise 1');
```



Booleans: definition

- Logical expressions with two values :
  - "0" for "false"
  - "1" for "true"

- Logical expression (X==-1) returns "1" if the numerical value of X is -1; returns "0" otherwise.

```
>> X = [0; 2; 2; 0; 0; 1];
>> find(X==1)
ans =    6
>> find(X==3)
ans =    Empty matrix: 0-by-1
```

```
>> X = [0; 2; 2; 0; 0; 1];
>> find(X==2)
ans =    2
         3
```

Booleans: Operators (relational, logical)

- == Equal to
- ~= Not equal to
- < Strictly smaller
- > Strictly greater
- <= Smaller than or equal to
- >= Greater than equal to
- & "And" operator
- | "Or" operator
- ~ "Opposite" operator (switch "0" and "1")

>> A = [0,0,1]; B = [1,0,1]; >> A&B ans = 0 0 1	>> A = [0,0,1]; B = [1,0,1]; >> A|B ans = 1 0 1

Control structures

- **IF**: evaluates a logical expression and executes a block of statements
  - **if-else-elseif-end**

```
if logical_expression
 statements
end
```

```
if logical_expression1
  statements
elseif logical_expression2
  statements
else
  statements
end
```

  - **Example:**

```
if ((N>=1) & (N<nMax))
    X = solver(A,B,C,D)
elseif (N<1)
    disp(['Erroneous index "N".']);
elseif (N>=nMax)
    disp(['Overflow error.']);
end
```

Control structures

- **SWITCH**: Switch works by comparing the input expression to each case value.
  - **switch-case-othewise-end**

```
switch expression
  case value1
    statements
  case value2
    statements
  ...
  otherwise
    statements
end
```

Useful to simplify code when the number of entries in a conditional control structure increases.

Control structures

- **FOR**: to execute a block of statements a predetermined number of times.
  - **for-end**

```
for index=start:increment:finish
  statements
end
```

  - **Example:**

```
for  k = 5:2:13
    fprintf('The square of %d is %d.\n', k, k^2)
end
```

```
The square of 5 is 25.
The square of 7 is 49.
The square of 9 is 81.
The square of 11 is 121.
The square of 13 is 169.
```

Control structures

- **WHILE**: executes its block of statements as long as the logical controlling **expresssion** evaluates as true
  - **while-end**

```
while expression
  statements
end
```

  - **Example:**

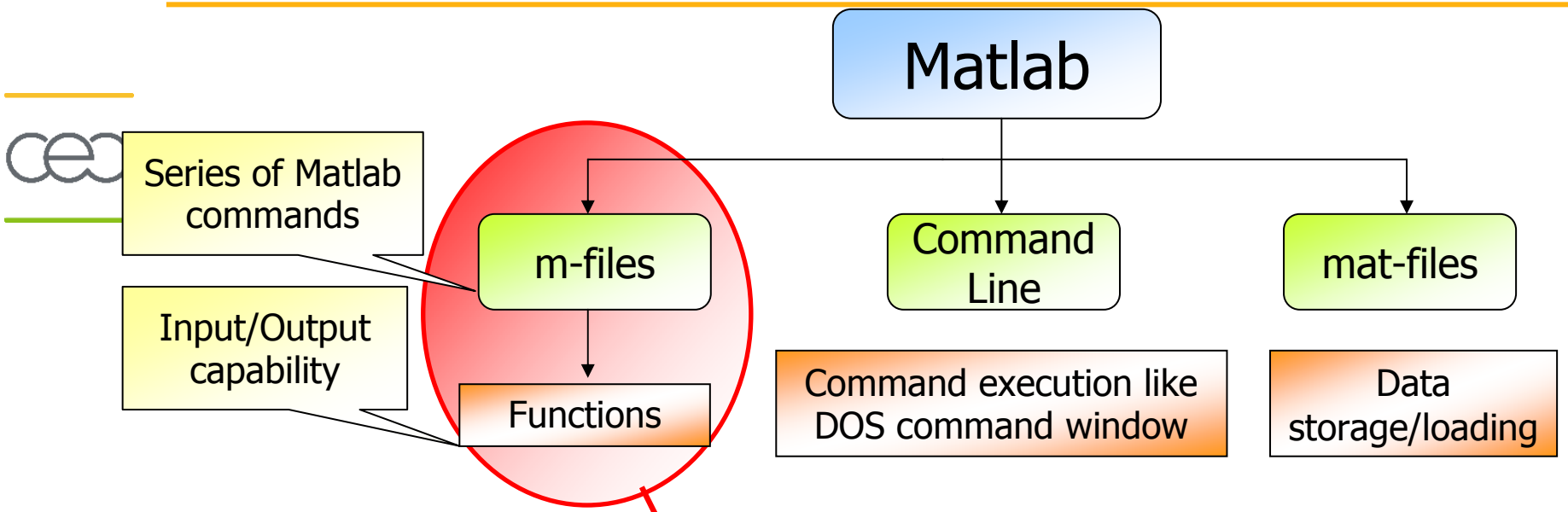
```
k=5;
while k<=13
    fprintf('The square of %d is %d.\n', k, k^2)
    k=k+2;
end
```

```
The square of 5 is 25.
The square of 7 is 49.
The square of 9 is 81.
The square of 11 is 121.
The square of 13 is 169.
```

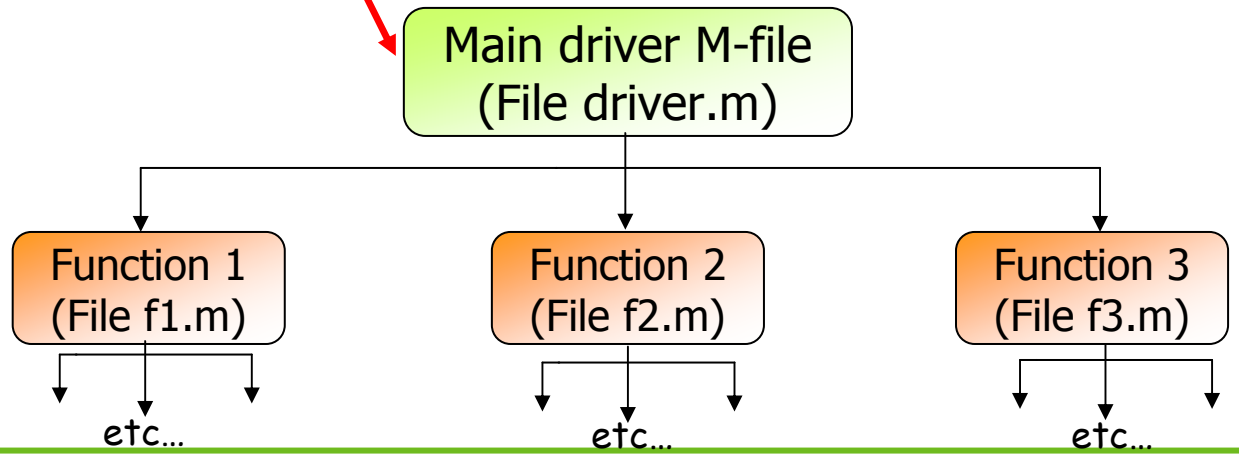
File organization

- Matlab can be operated in **3 modes**:
  - Trough an interactive session
    ⇨ commands are typed directly in the Matlab command window
  - By programming functions and executing **m-files**
    ⇨ the user writes functions in so-called “m-files”, then executes the functions
  - Using a GUI

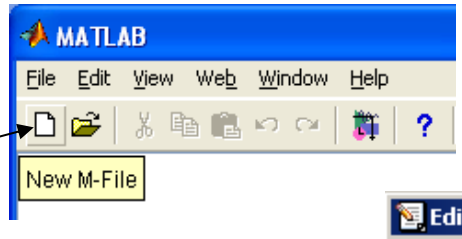
File organization



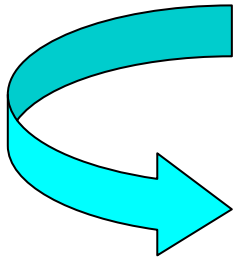
- A main program calls functions that execute various commands.

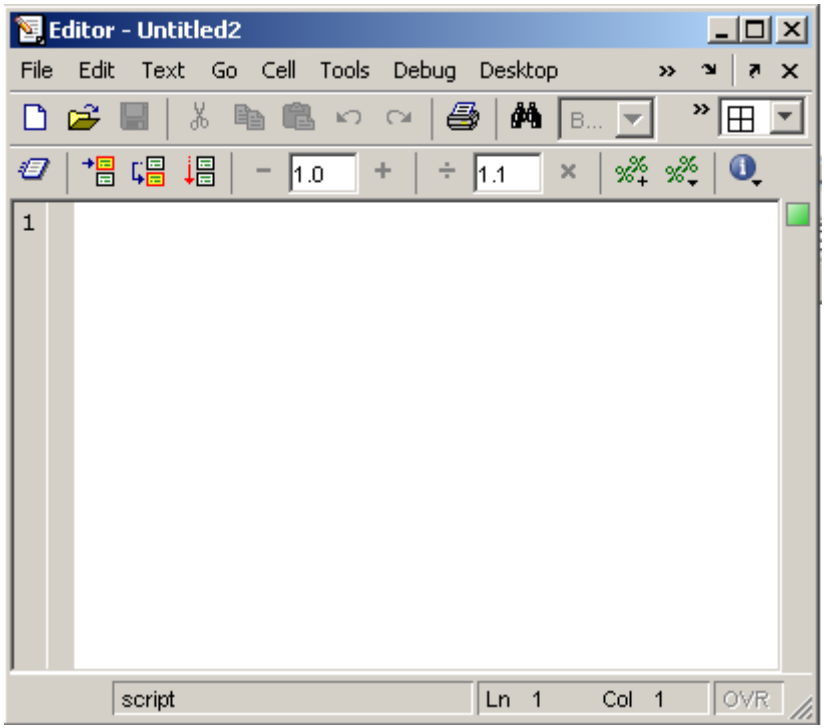


File organization : programming mode



Click to create
a new M-File

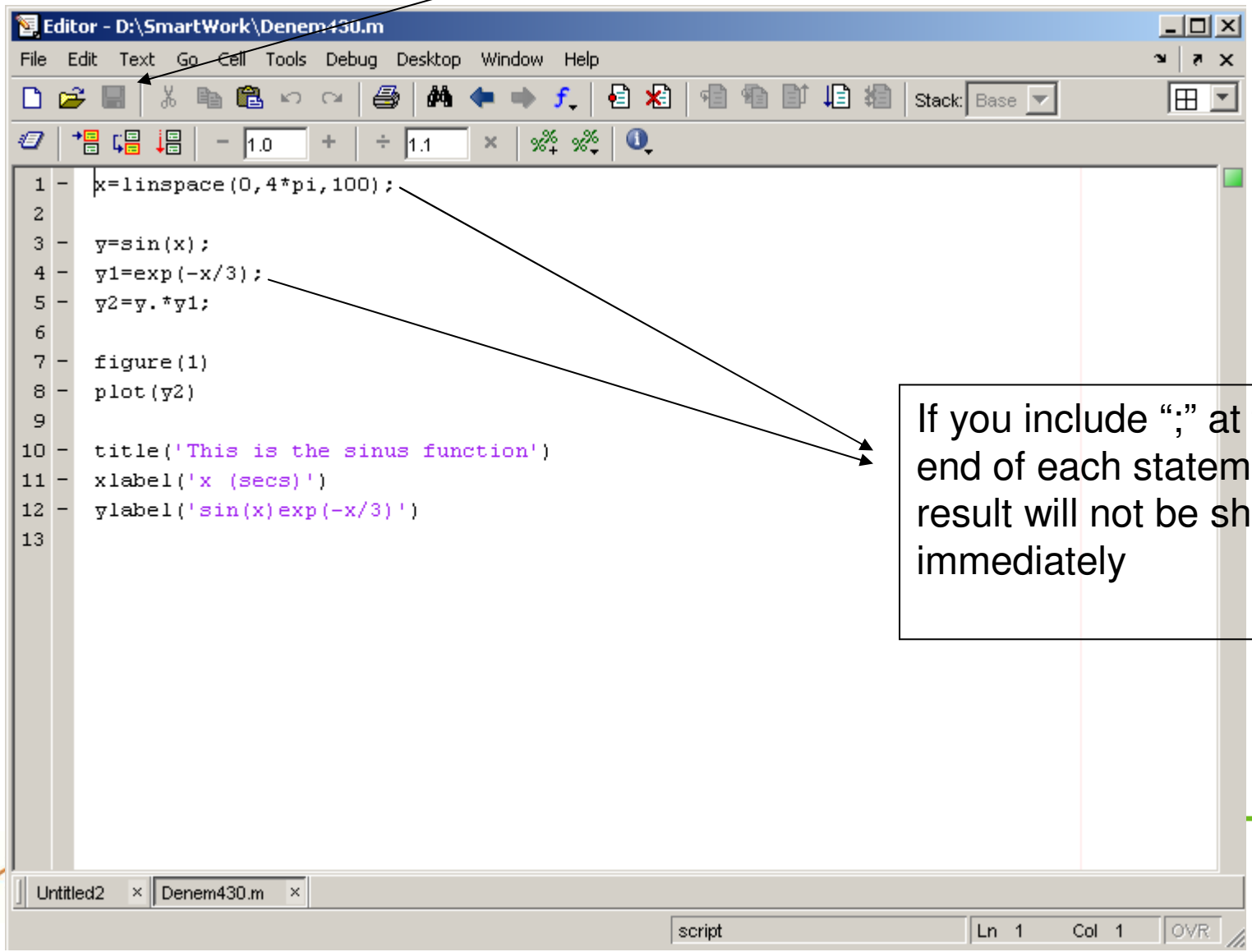




- Extension ".m"
- A text file containing script or function or program to run

File organization : programming mode

Save file as *exemple1.m*



If you include ";" at the end of each statement, result will not be shown immediately

File organization : user defined functions

- **Functions (or "m-files") :**
  - accept **a list of input arguments**
  - Return **a list of output arguments**

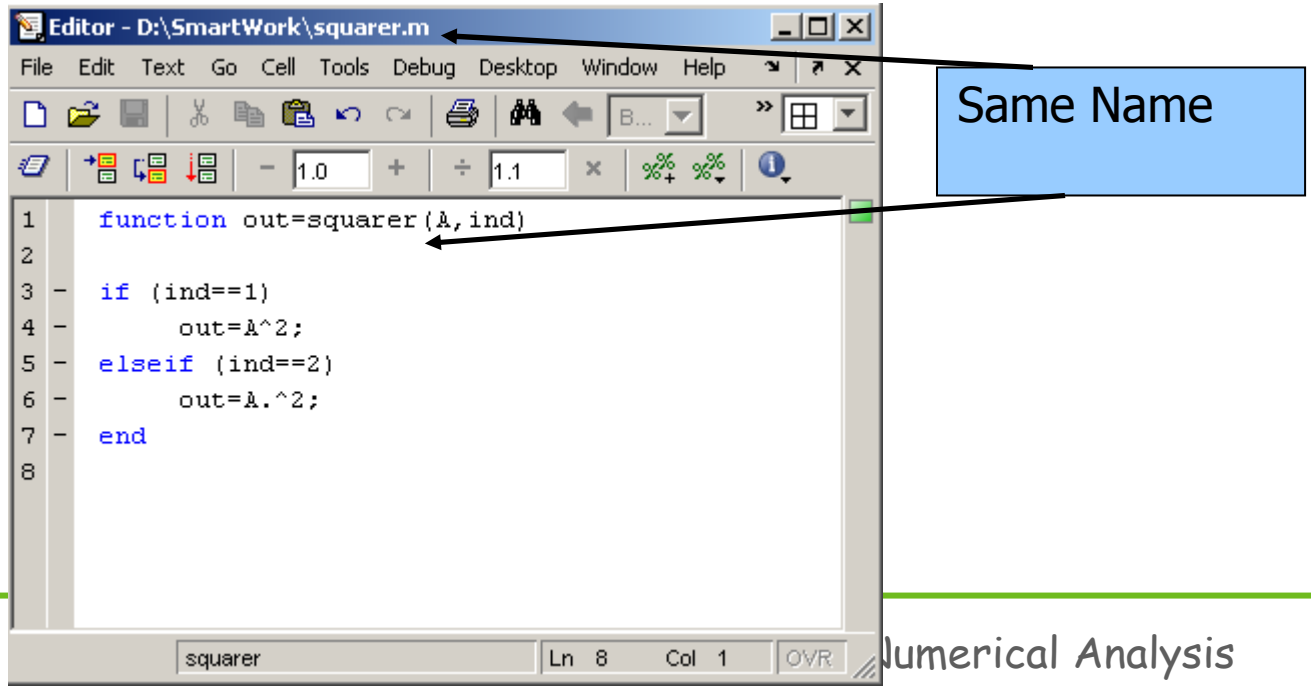
```
function output1 = functionname(input1)
function output1 = functionname(input1,input2,input3)
function [output1, output2] = functionname(input1,input2)
```

You should write this command at the beginning of the m-file

- For simplicity (not required), you should save the m-file with a file name same as the function name
- **All variables defined inside a function are local**
- Recurrent call to a function is allowed.

File organization : user defined functions

- **Example**
  - Write a function : out = squarer(A, ind)
    - Which takes the square of the input matrix if the input indicator is equal to 1
    - And takes the element by element square of the input matrix if the input indicator is equal to 2



File organization : user defined functions

Calling functions

- From Matlab window or another m-file

  >> [Solution, Residual, Norm] = solver(Matrix, Vector, Options)

- Some of input arguments can be omitted (if there are optional).

  >> [Solution, Residual, Norm] = solver(Matrix, Vector)

- Some of output arguments can be omitted.

  >> Solution = solver(Matrix, Vector, Options)

- Function calls can be integrated to other commands.

  >> sqrt(sum(solver(Matrix, Vector).^2))

Notes

- **%** is the neglect sign for Matlab (equivalent of "//" in C). Anything after it on the same line is neglected by Matlab compiler.
- **pause** : sometimes slowing down the execution is done deliberately for observation purposes.

```
pause      %wait until any key
pause(3)   %wait 3 seconds
```

- **Save** : save variable on disk space and create a file ***.mat***.
  Matlab recognizes that this extension identifies a data storage.

```
save FileName.mat A          % save variable A in binary format
save FileName.mat A –ascii   % save A in ascii format
save A                       % save A in A.mat
save FileName.mat A B C      % save variables A, B and C
save FileName.mat            % save all variables in workspace
```

- Matlab recognizes a limited number of Unix or MS-DOS commands such as **ls, dir, del, cd, pwd, del** and **delete**.

Exercise 2:

- **Step1: Write a function "simu_exp.m" that generates a random sample following an exponential distribution**

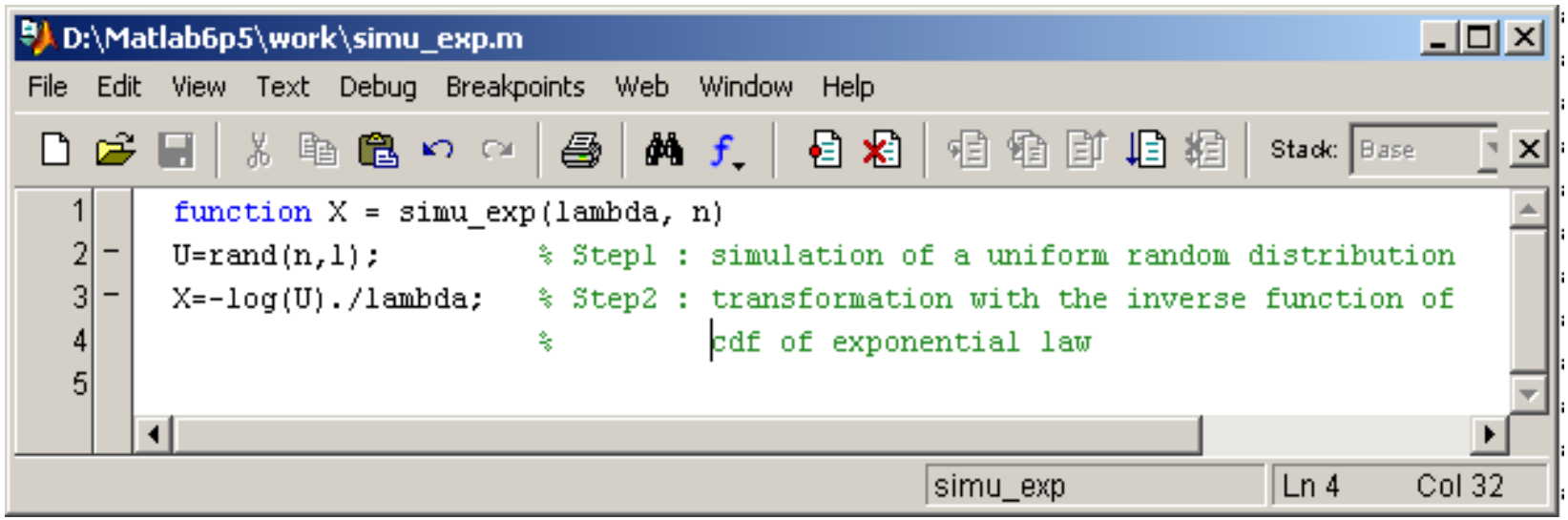
  - Input parameters :
    - Lambda : parameter of exponential law
    - N : size of sample
  - Output parameters :
    - X : vector of sample following an exponential distribution
  - **Use function rand**
  - Reminds :
    - Exponential distribution is defined on [0; +∞[, with parameter $\lambda > 0$
      - Probability density function : $\lambda e^{-\lambda x}$
      - Cumulative distribution function : $1 - \lambda e^{-\lambda x}$
    - Simulation with Inverse transform method
      - Given a probability law L of continuous cumulative distribution function F. If X ~U[0;1] and if the inverse function of F is defined, then $Y = F^{-1}(X)$ follows the probability law L.

Exercise 2:

- **Step2: Compare the empirical and the theoretical Cumulative Distribution Functions (CDF)**
  - Call ***simu_exp.m*** function to simulate a sample X of size N=100 following an exponential law of parameter λ=0.6
  - Plot the empirical CDF of X
    - Function **sort** and **stairs**
  - Compute the theoretical CDF and compare it with the empirical CDF on the same plot
    - Function **fplot**, **inline**, **hold**

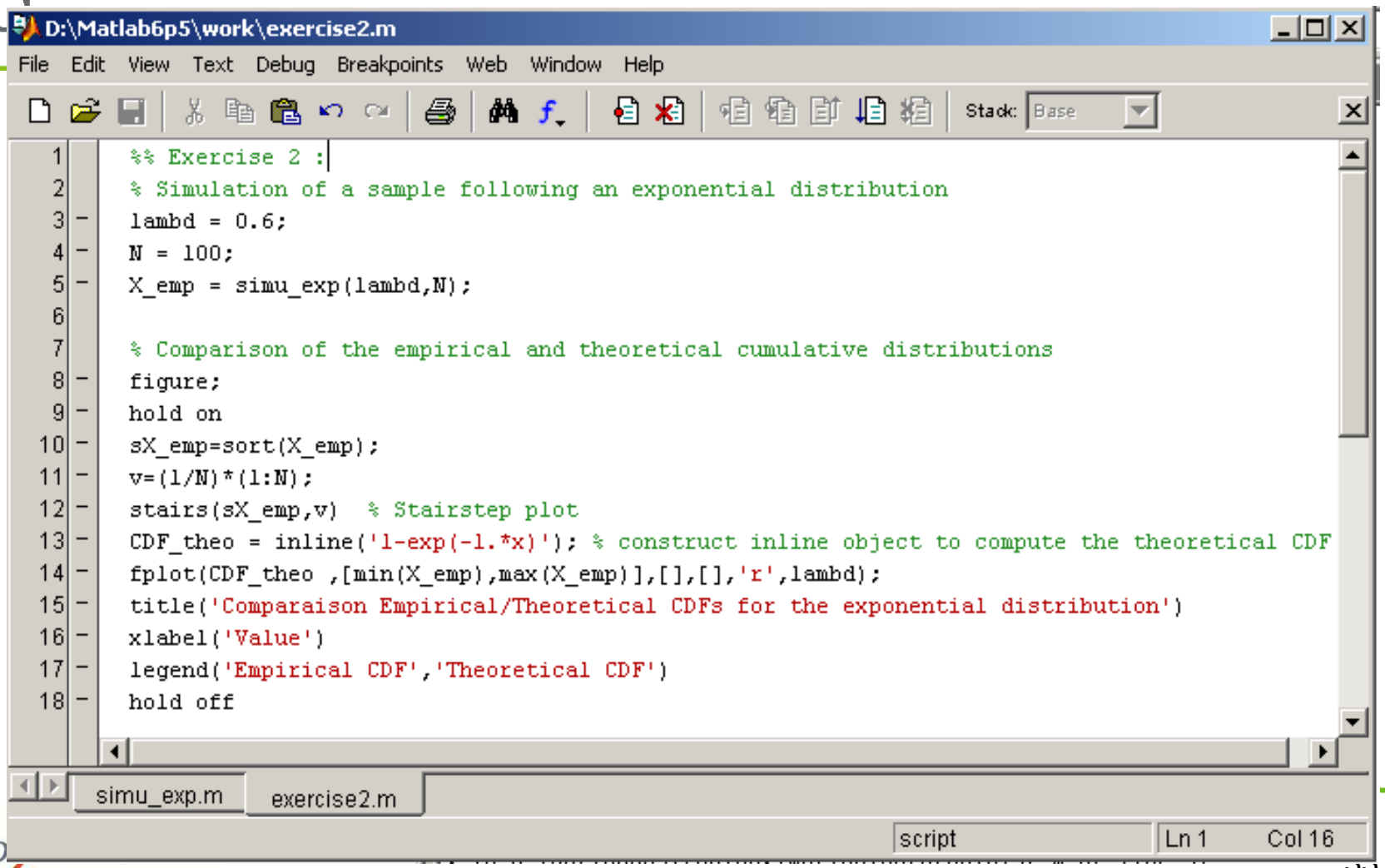
Exercise 2: Solution

- **Step1: Write a function "simu_exp.m" that generates a random sample following an exponential distribution**



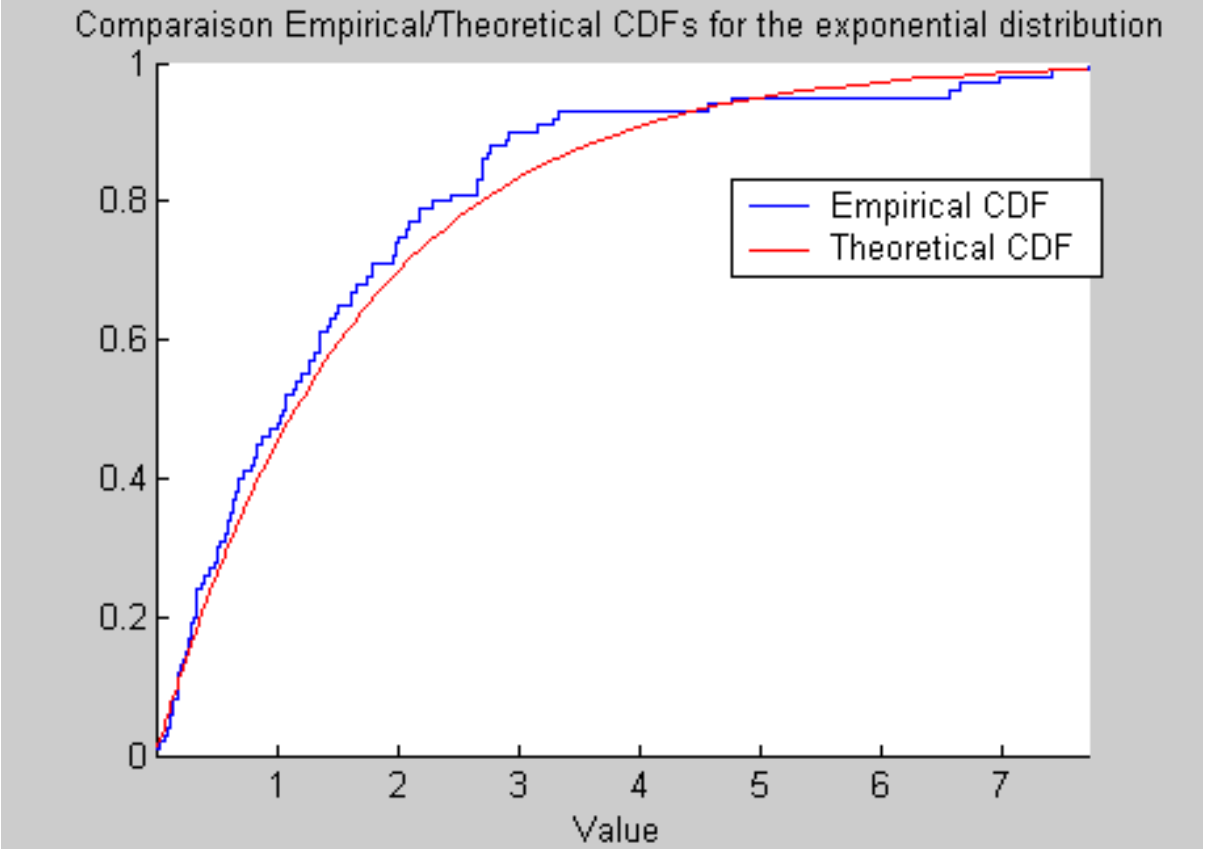
Exercise 2: Solution

- **Step2: Compare the empirical and the theoretical Cumulative Distribution Functions (CDF)**



Exercise 2: Solution

- **Step2: Compare the empirical and the theoretical Cumulative Distribution Functions (CDF)**



Structures

- **Structures are "meta-objects" capable of collecting any type of objects in a single variable.**
  - ⇨ useful way of grouping MATLAB variables that belong together
  - ⇨ useful when several (>10) objects must be passed on to functions
  - ⇨ useful to create a powerful storage capability.
- Example : structure ball

```
>> ball.mass = 10;
>> ball.position = [0, 0, 100];
>> ball.velocity = [0, 0, 0];
```

- The individual components of the structure are called **fields**.
- Examining a structure : to see the contents of a structure, just type it's name at the command prompt

```
>> ball
ball =
        mass: 10
    position: [0 0 100]
    velocity: [0 0 0]
```

Structures

- Access to a field:

```
>> ball.position
ans =
        0  0  100
```

- Add a new field to the data structure.

```
>> ball.radius = 2.0
ball =
        mass: 10
    position: [0 10 100]
    velocity: [0 0 0]
      radius: 2
```

- Arrays of structures to represent multiple objects.

```
>> Ball_array(2:3,1) = ball;
Ball_array =

3x1 struct array with fields:
    mass
    position
    velocity
    radius
```

Each of the three elements of the struct array is a "ball" structure.

```
>> Ball_array(2).mass = 5.0;
>> Ball_array(2)
ans =
        mass: 5
    position: [0 10 100]
    velocity: [0 0 0]
      radius: 2
```

Conclusion

- MATLAB is an **object-oriented programming language with built-in "vectorization" capability.**
- It offers very **powerful graphics capabilities** and has **many specialized toolboxes.**
- **Programming recommendations:**
  - Be modular, write and use functions.
  - Use the built-in Matlab functions as opposed to your own functions, except if you want to use a specific algorithm or guarantee numerical efficiency.
  - Use compact writing as much as possible.
  - One line of comment (symbol %) for one line of code.
  - Avoid if/then statement inside loops.
  - Always operate on the columns of the matrices inside loops to enable calculation "vectorization".
  - Pre-allocate memory before starting calculations.