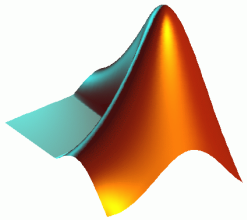
CS@CU

COMS W3101-2

Programming Languages: MATLAB



Lecture 3

Spring 2010

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw3101-2.html

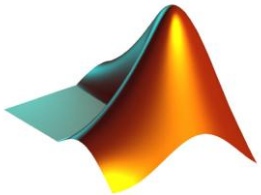
Run MATLAB as executable

- Type from command line:
  - `matlab -nodisplay -r command`

Tells MATLAB not to initialize the visual interface
NOTE: this works only for Linux

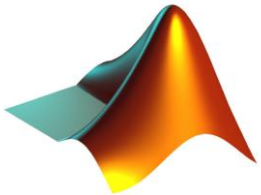
Tells MATLAB to execute the following command

- This will start MATLAB, open its environment without showing the GUI, and execute the specified command within the environment
- NOTE: MATLAB will stay open after this!!! We have to explicitly close the environment



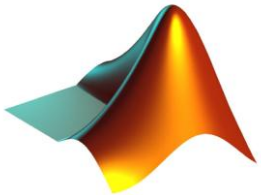
Run MATLAB as executable

- Example 1:
  - `matlab -nodisplay -r x=3`
- Example 2:
  - Prepare script *myscript.m*
    - `x = rand(30);`
    - `save('my_x','x');`
  - `matlab -nodisplay -r myscript`



Run MATLAB as executable

- Example 1:
  - `matlab -nodisplay -r x=3`
- Example 2:
  - Prepare script *myscript.m*
    - `x = rand(30);`
    - `save('my_x','x');`
    - `exit`
  - `matlab -nodisplay -r myscript`



I/O – Saving/Loading Data

MATLAB stores data in specific files, with extension *.mat*

- `save`
  - **Example**
  - `x = rand(7, 3);`
  - `y = 'cool';`
  - `save('myfile','x');`
  - `save('myfile2','x','y');`

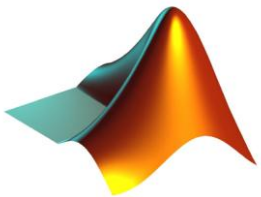
General Form

`save('namefile(.mat)','variable');`
`save('namefile(.mat)','var1','var2',…);`

- `load`
  - **Example**
  - `load myfile2;`
  - `newX = load('myfile2','x');`

General Form

`load 'namefile(.mat)' ;`
`var = load('namefile(.mat)');`



I/O – User Input

MATLAB allows scripts or functions to read data inserted by users in the command window, using the function *input()*

- `input()`
  - **Example** 1
  - `weight = input('Insert weight: ');`

  General Form

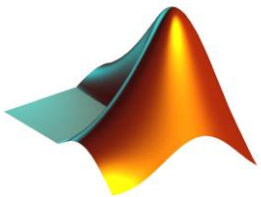
  `var = input('string');`

  - **Example** 2
  - `name = input('Insert name:\n ','s');`

  General Form

  `var = input('string','s');`

This specifies that *var* is going to be a string

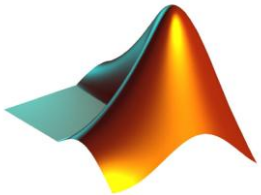


I/O – Text Files

- `fopen`
  - `fid = fopen('myfile.txt','r')`

File ID — Filename — Mode: 'r' – read, 'w' – write, 'a' – append

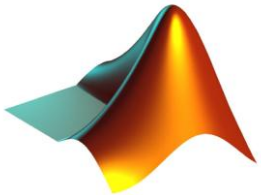
- Operations on file `fid`
- `fclose`
  - `fclose(fid)`



I/O – Text Files

Example

```
fid = fopen('myfile.txt','r');
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    disp(tline)
end
fclose(fid);
```



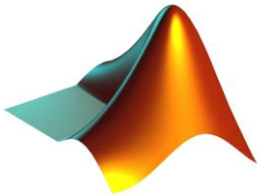
I/O – Text Files

Example

```
fid = fopen('myfile.txt','r');
while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
    disp(tline)
end
fclose(fid);
```

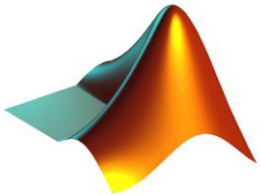
Reads single line in file

Checks format of argument



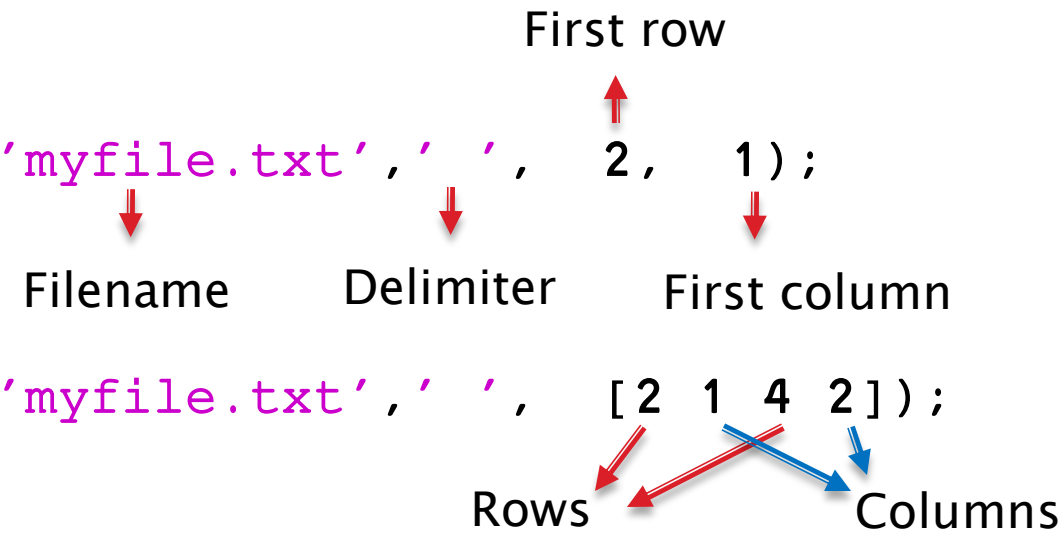
I/O – Text Files

- MATLAB has special functions to deal with files containing formatted data
- dlmread
  - `data = dlmread('myfile.txt',' ',  2,  1);`
  - `data = dlmread('myfile.txt',' ',  [2 1 4 2]);`
- dlmwrite

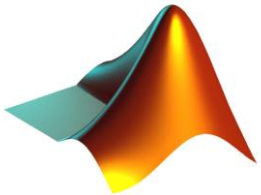


I/O – Text Files

- MATLAB has special functions to deal with files containing formatted data
- dlmread
  - `data = dlmread('myfile.txt',' ', 2, 1);`
  - `data = dlmread('myfile.txt',' ', [2 1 4 2]);`



- dlmwrite

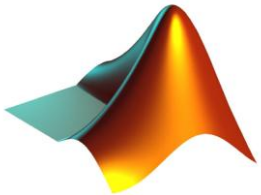


I/O – Text Files

- MATLAB has special functions to deal with files containing formatted data
- dlmread
  - `data = dlmread('myfile.txt',' ', 2, 1);`

Starts counting with 0 !

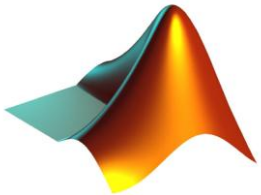
  - `data = dlmread('myfile.txt',' ', [2 1 4 2]);`
- dlmwrite



I/O – Text Files

- MATLAB has special functions to deal with files containing formatted data
- dlmread
  - `data = dlmread('myfile.txt',' ', 2, 1);`
  - `data = dlmread('myfile.txt',' ', [2 1 4 2]);`
- dlmwrite
  - `dlmwrite('myfile2.txt',data,'\t', '-append');`

Filename Variable Delimiter



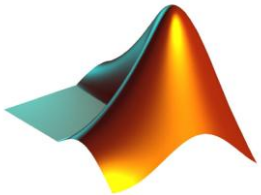
I/O – Text Files

- MATLAB has special functions to deal with files containing formatted data

- `csvread`
- `csvwrite`

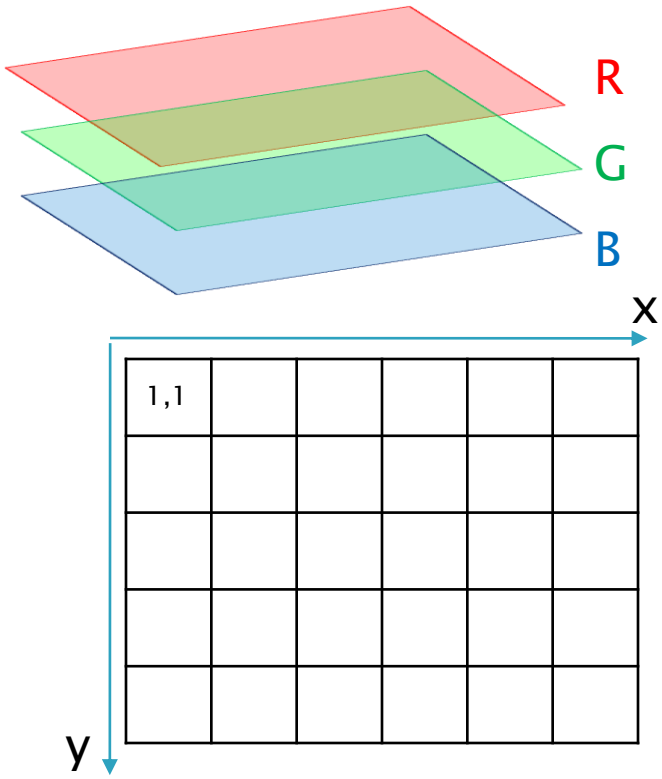
  } For comma separated files

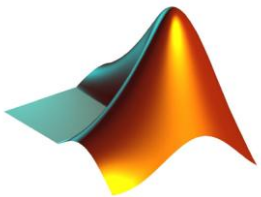
  - `dataCSV = csvread('myfile.csv')`
  - `dataCSV2 = dlmread('myfile.csv',',')`



I/O – Images

- Images are matrices
  - Color images are [nxmx3] matrices
  - Grayscale images are [nxm] matrices
- Reading Images
- imread
  - `Im = imread('mypic.jpg');`
- Saving Images
- imwrite
  - `imwrite(Im,'mypic2.png');`





Advanced Data Structures

- Structs are data structures that allow to keep different data types in the same variable

- Struct
  - `s = struct('field1', var1, 'field2', var2, ...);`

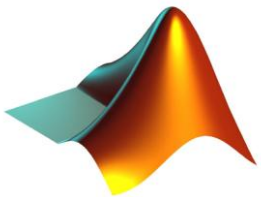
  Example
  - `s = struct('num', [1:10], 'str', 'cool');`
  - `s.str`
  - `s.newField = 3;`

Note: when we save multiple variables in a .mat file, and later try to load them assigning to a single variable, they get saved as fields of a struct

Advanced Data Structures

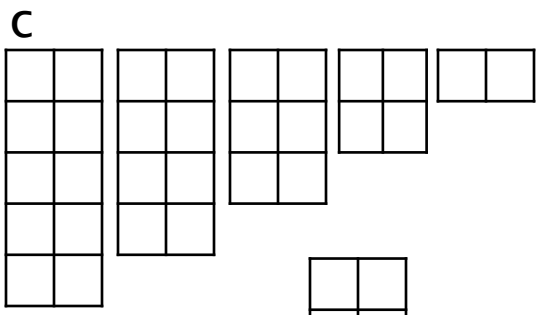
- Cells also allow to keep different data types in the same variable

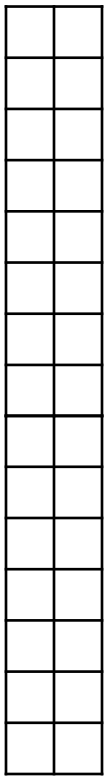
- Cell
  - `c = cell(n);`
  - `c = cell(m,n);`
  - `c = {'one','two','three'};`
  - `c = {'one','two',3};`
  - `c = {[1] [2 3 4]; [5; 9] [6 7 8; 10 11 12]};`
  -
```
for in=1:5
  c{in} = rand(in,2);
end
```



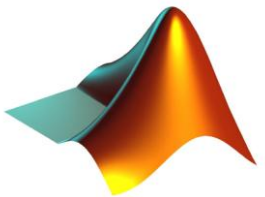
Advanced Data Structures

- Cell2mat
  - `for in=1:5`
  - `  c{in} = rand(in,2);`
  - `end`
  - `cMat = cell2mat(c);` Error!
  - `cMat = cell2mat(c');`



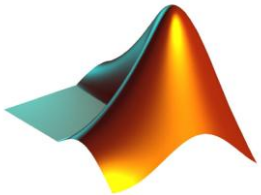


Matrices dimensions must agree!



Advanced Data Structures

- cell2struct
  - `fields = {'number','name','value'}`
  - `c = {'one','Luke',3; 'two','Don',7};`
  - `cStruct = cell2struct(c,fields,2);`

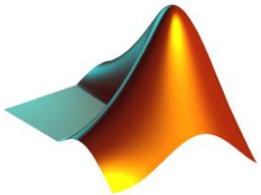


I/O – Text Files

- `textscan()`

- Read data from text file, convert, and write to cell array

  - `fid = fopen('myfile.txt');`
  - `C = textscan(fid,'format');`
  - `fclose(fid);`



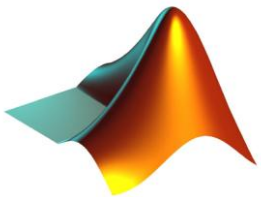
I/O – Text Files

inputTextscan.txt

```
Sally  Level1 12.34 45 1.23e10 inf NaN Yes
Joe    Level2 23.54 60 9e19 -inf 0.001 No
Bill   Level3 34.90 12 2e5 10 100 No
```

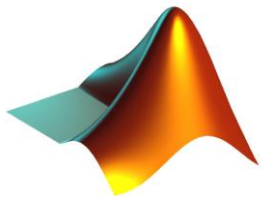
- `textscan()`
- Read data from text file, convert, and write to cell array
- Example
  - `fid = fopen('inputTextscan.txt');`
  - `C = textscan(fid,'%s %s %f32 %d8 %u %f %f %s');`
  - `fclose(fid);`
  - `C = [1x8] cell`

```
{3x1 cell}      %s
{3x1 cell}      %s
[3x1 single]    %f32
[3x1 int8]      %d8
[3x1 uint32]    %u
[3x1 double]    %f
[3x1 double]    %f
{3x1 cell}      %s
```



Operations on Strings

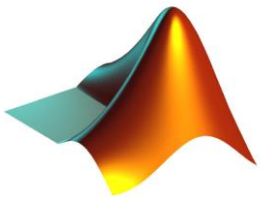
- lower
  - `S = 'ABCDE';`
  - `sL = lower(S);` `'abcde'`
- upper
  - `Sagain = upper(sL);`
- strtok
  - `S = 'try it out! It is fun';`
  - `[part res] = strtok(S,'!');`



Operations on Strings

- str(n)cmp(i)
  - res = strcmp('hi','Hi'); 0
  - res = strcmpi('hi','Hi'); 1

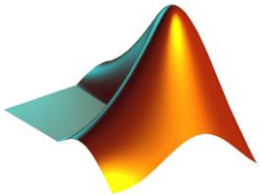
- strfind
  - S = 'this is my long long string';
  - index = strfind(S,'i');



Strings – Conversions

- str2num
  - S = '334345';
  - sNum = str2num(S);

- num2str
  - n = 33;
  - s = str2num(n);

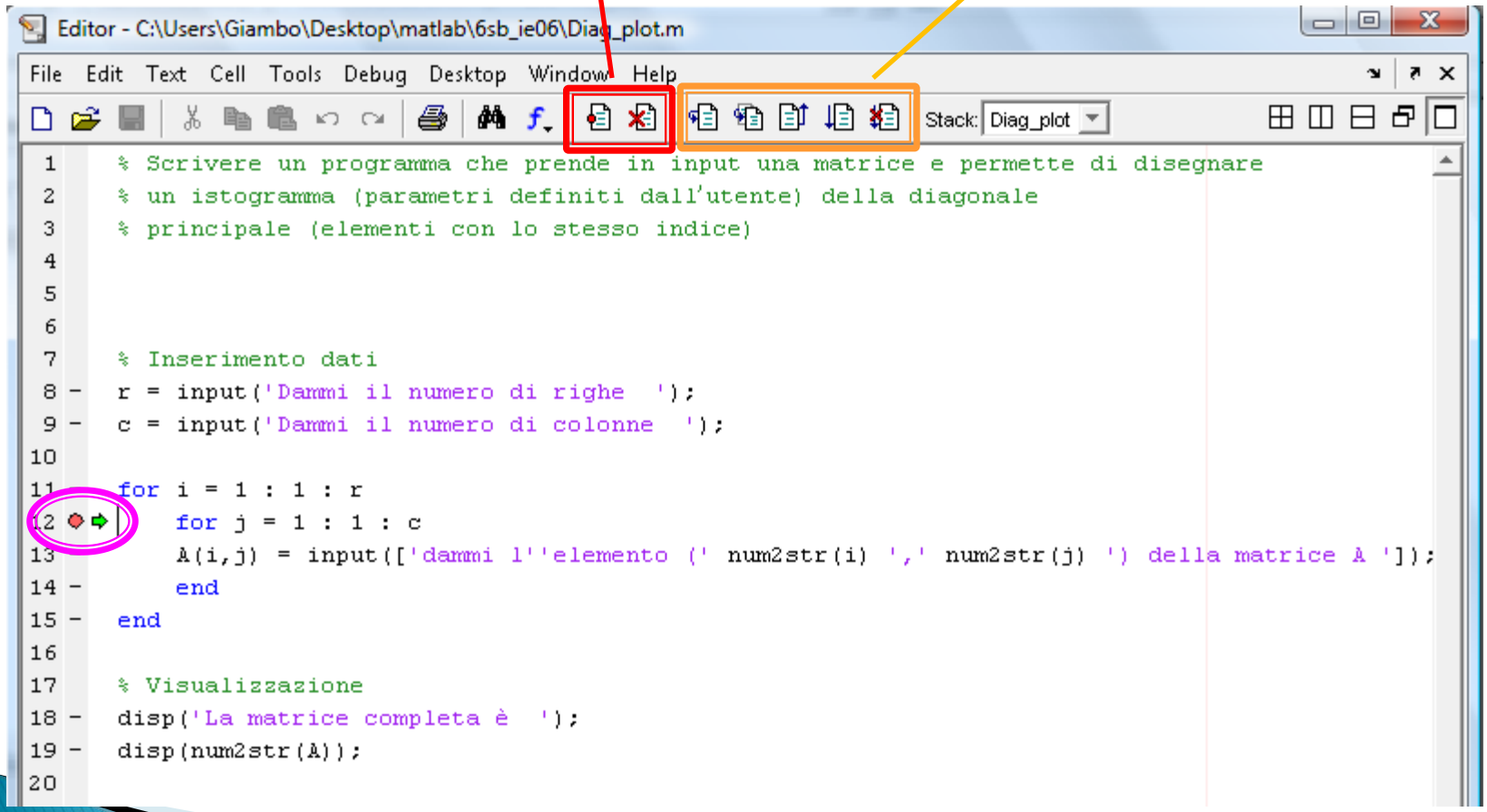


Debugging

- interface

Insert/remove breakpoint

Continue, step, jump, stop debugging



Functions

- Definition

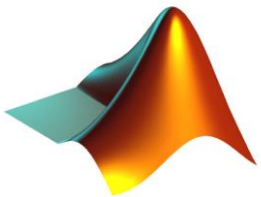
```
function [ret1, ret2, …] = nameF(input1, input2, …)
```

- The .m file containing the function must be named *nameF.m*

- Dynamic management of input/output:
  - `nargin, nargout`
  - `varargin, varargout`

Return number of inputs and outputs

Allow number of inputs and outputs to be determined by the function call



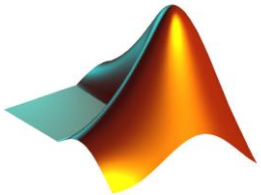
Functions – Examples

Example 1 – file *circ.m*

```
function [diam, area] = circ(radius)

diam = radius*2;
area = pi*(radius^2);
```

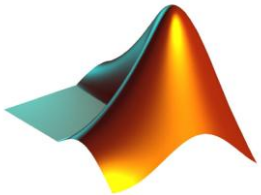
Example 2 – file *circ2.m*

```
function [varargout] = circ2(varargin)

r= zeros(nargin,1);
for in=1:nargin
    r(in) = varargin{in};
end

diam = r*2;
area = pi*(r.^2);
varargout = {diam, area};
```



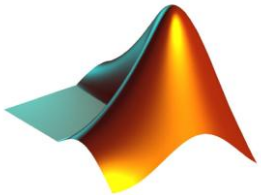
Functions

- We can define multiple Functions in the same *fun.m* file, as long as:
  - *fun.m* is a function file, not a simple script
  - the functions are called only by the main function of *fun.m*, which is *fun()*



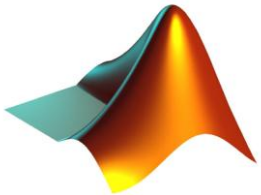
Functions – Exercise

- Write 2 functions for time conversion
  - **hms2secs(vec)** which takes as input a [1x3] vector *vec* containing values of hour, minutes and seconds and returns a scalar with the total number of seconds
  - **secs2hms(s)** which takes as input a scalar *s* with a number of seconds and converts it into a [1x3] vector containing values of hour, minutes and seconds



Functions – Exercise

- Write 2 functions for time conversion
  - **hms2secs(vec)**
    - `function [s] = hms2secs(vec)`
    - `s = vec(1)*3600 + vec(2)*60 + vec(3);`
  - **secs2hms(s)**
    - `function [vec] = secs2hms(s)`
    - `vec(1) = floor(s/3600);`
    - `vec(2) = floor((s-vec(1)*3600)/60);`
    - `vec(3) = s - vec(1)*3600 - vec(2)*60;`



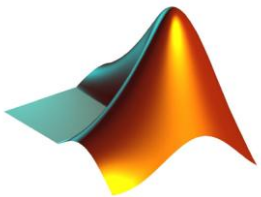
Exercises in class !

Review

- Exercise 1
  - Consider the series

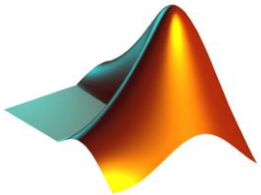
$$\sum_{i=0}^{n} x^n \rightarrow \frac{1}{1-x} \quad \text{if } |x| < 1$$

  - Compute it for x = 0.63 and n=10,20,100
  - Compare the values obtained with the limit value



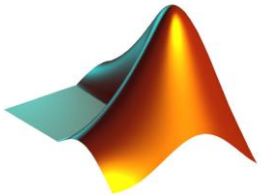
Review

- Exercise 2
  - x has 500 equally spaced elements in the range [-2:2]
  - Plot the function $y=e^x-x-1.5$, with red line and diamond shaped blue markers
  - Put title and axes labels in the figure
  - Find the value of x at the global minimum of the function



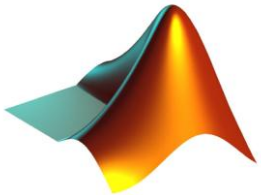
Review

- Exercise 3
  - Generate a vector *vec* with elements in increasing order from 1 to 10, at intervals of 2
  - Write a loop in which at every iteration a [7x1] matrix *A* is initialized with random numbers between 0 and 30
  - Keep iterating until you can access the element of *vec* located in the position corresponding to the value of the third element of *A*
  - Keep track of the number of failed attempts by printing a comment to command window at each iteration



Review

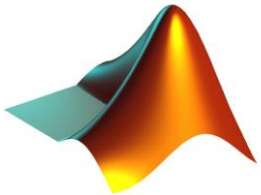
- Exercise 4
  - Load the *peaks* built in MATLAB variable into the variable Z
  - Plot its surface in the x interval [20 30] and y interval [30 50]
  - Find the peak in the interval with the MATLAB interface tool, without writing code
  - Plot in the same graph a surface parallel to the xy axes and passing through the peak



Review

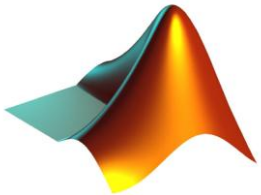
- Exercise 1

```
x = 0.63;
limit = 1/(1-x);
for n=[10 20 100]
    expTerm = [0:n];
    s = sum(x.^expTerm);
    disp(abs(limit - s))
end
```



Review

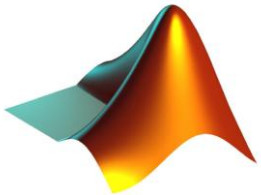
- Exercise 2
  - `x = linspace(-2,2,500);`
  - `y = exp(x)-x-1.5;`
  - `plot(x,y,'rdb');`
  - `title('y'); xlabel('x'); ylabel('f(x)');`
  - `[minVal minLoc ] = min(y);`
  - `disp(x(minLoc))`



Review

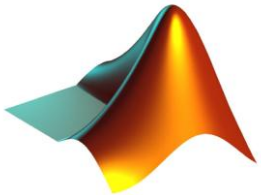
- Exercise 3

```
c=1;
vec = [1:10];
while(c)
  A = floor(30*rand(7,1));
  try
    t = vec(A(3));
    s = c;
    c = 0;
  catch
    fprintf('Attempt number %d failed\n',c);
    c = c+1;
  end
end
fprintf('Attempt number %d succeeded\n',s);
```



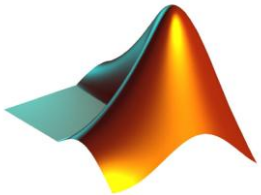
Review

- Exercise 4
  - `Z = peaks;`
  - `surf(Z);`
  - `xlim([20 30])`
  - `ylim([30 50])`
  - `hold on`
  - `Z2 = 8.075 * ones(size(Z));`
  - `mesh(Z2)`
  - `hold off`



Quiz

- In class, in front of computer
- 40 minutes
- Arguments: everything done so far
- Closed notes, closed web, closed phones
- Only MATLAB (with MATLAB help)
- Upload code to CourseWorks when done



Homeworks policy

- Due at beginning of class, no exceptions
- Put your code (.m files) and additional files in a single folder, name it *youruni*_hw_*X* and zip it
- Upload the zipped folder to CourseWorks
- Bring a printout of your code to class
- Good Luck and have fun !