Lab – Introduction to GNU/Octave

GNU/octave is a high-level scientific calculation language for developing algorithms, visualizing and analyzing date or carrying out numerical calculations.

INSTALLATION

GNU/Octave is freely distributed under the terms of the GNU General Public License, it is a multi-platform software: https://www.gnu.org/software/octave/

To install GNU/octave with Linux, do: *$ sudo apt-get install octave*

With Mac OS, you can install a virtual machine in order to host Linux on your PC, VirtualBox is then a good solution, see: https://www.virtualbox.org
Another solution on Mac OS is to use *Brew* that permits to handle (install, delete, etc.) packages, see: https://brew.sh/index_fr
Then to install Octave, type in a terminal: *$ brew install octave*

You start GNU/octave by default in a terminal: *$ octave*

After installing Octave, you will need to install other specialized octave packages like *image* and *signal*:
octave:1> pkg install -forge signal
octave:1> pkg install -forge image
Next you will need to load these packages in the current octave session:
octave:1> pkg load signal
octave:1> pkg load image

Note 1: It exists an online version of GNU/octave (but the disposable memory size is very limited): https://octave-online.net

Note 2: You can personalize your octave session by writing (in your home directory) an *.octaverc* file, in order to launch automatically the loading of the packages, to indicate your working directory, and so on.

Note 3: The *pwd, cd, ls* Unix in line commands are available on Octave.

Exercise 1 – Operators

Between each exercise, we recommend that you use the ***clear*** and ***close all*** commands so that you can empty the workspace and close all the figures.

1.1 – Enter the command *a=[1 4 5 ;4 8 9]*, what does this command correspond to?
In the console, enter the *a=rand(5)* command. Then enter the ***help rand*** command to get a description of the operation carried out by the ***rand*** function. Finally, enter the command: *a=rand(5)* followed by a semi-colon « *;* »
What difference can we observe in the console with the *a=rand(5)* command? Deduce from this the role of the « ; »in GNU/octave.

1.2 – In GNU/octave, the « : » operator is very useful. It can be used, among other things, to swap elements from a row or a column of a matrix.

Note: the index 0 does not exist in GNU/octave. The first element of a matrix is accessed by an index of 1. For example *array(1,1)* for images accesses the value of the pixel (1st row, 1st column). The first index is for the rows, the second index for the columns.
To fully understand these concepts, try the commands:

- *a(2,1)*
- *a(1:3,1)*
- *a(:,2)*
- *a(1, :)*
- *a([1 4], :)*
- *a(1 :2 :5, :)* (the *1:2:5* command sweeps the interval [1,5] by steps of 2)

Be careful however not to put a « ; » at the end of a row to visualize the results obtained in the console.

1.3 – GNU/octave is an interesting tool for matrix calculations. The various classical matrix operations (addition, multiplication, eigenvalues etc.) are there of course but there are also element by element operations that are very useful in image processing; these are available through a specific operator (for example: « **. *** », « **. /** »).
Enter the following commands (without « ; » at the end of the line to see the results):

- *a= [0 0 0; 1 1 1; 2 2 2]*
- *b=a+4*
- *c=a*b*
- *e=a.*b*

Explain the difference between « ***c*** » and « ***e*** ».

ACTION: Create a matrix *A* sized 4×4 using the method of your choice (use ***rand*** or enter the items one by one). How can you access:

- the first row of *A*?
- the fourth column of *A*?
- the first three items of the fourth row in *A*?

Exercise 2 – Display

GNU/octave also offers a wide range of diverse and practical display possibilities. For example, you can plot function on a logarithmic scale or view the values of a matrix as an image.

2.1 – A vector can be displayed using the ***plot*** command. Enter the ***help plot*** command to obtain more information about this function and on functions with similar properties (***subplot***, ***semilogx***, etc.).
Try out the following commands:

- *x=1:3:10*
- *plot(x)* then *plot(x,'r')*
- *y=rand(4,1)*, then *plot(y)*, then *plot(x,y,'g')*. Interpret the difference between these two commands.

The ***plot*** function is very useful for obtaining for example the curves of different plane functions.

2.2 – Displaying a matrix on the other hand corresponds to displaying an image. Each item *(n, m)* in the matrix is considered as having the value of the pixel *(n, m)* with which it is associated. Check this by entering the following commands:

- *a=rand(10)*10 ;* (so that the elements are not limited to [0,1])
- *a=exp(a) ;* (to obtain larger spans between the items of vector a)
- *image(a)*

Images can be displayed with the ***image***, ***imagesc***, and ***imshow*** functions. Try zooming with the mouse; what kind of interpolation is used?

Exercise 3 – Writing scripts

The classic extension of a GNU/octave file is *.m*. We can find two types of *.m* files: function files and script files, the laters are a set of commands that can follow on from each other. Using script files is a way of saving your commands from one session to the next. To write a script file, use your favorite editor.

To run a script, run the ***file_name*** command in the command window (without the .m extension), making sure that the path list is consistent.

Note that when a user runs a command in the console, GNU/octave will go looking in the folders indicated by the path variable (see also the *path* command), if a function or a script matches that command, the first one found will be the one used (so take care with the folder order and the names of scripts and functions).

ACTION: Enter all the commands from the section 1.2 into a script and save it (for example: ***test.m***). Then run the script.

Exercise 4 – Data types

Here, we are going to use the ***image package*** that contains a large number of existing functions (do not hesitate to look in the help using the ***help*** command). In any case, when using a function from packages, you need to be careful with the data types. Classically and by default, everything is a matrix of ***double***, but most of the functions in the *image* package works with the ***uint8*** type to represent pixel values between 0 and 255. So when necessary for computation purpose, you will need to cast the ***uint8*** type in ***double***.

ACTION: From the image base, download a grey level image into your working folder. Read and display the image file respectively using the commands ***imread*** and ***imshow***:

im = imread('image.bmp');
imshow(im);

By consulting the workspace (by using the *whos* command), look at the size and data type. So as to display a sub-image that corresponds to the top left corner of 64×64 pixels, test this command (here we assume that the image size is larger than 64x64):

imshow(im(1:64,1:64));

The matrix *im* represents the image. Try adding this matrix directly 300 (type: *im+300*) or -300 (type: *im-300*). What is the result (matrix values?) and why?

Exercise 5 – Writing your functions

Use the function files as you would in any classical imperative programming situation. These are also *.m* files. To use a function, you must call it with call parameters either in a script or in another function.

ACTION: Use the ***template.m*** file to write your own max_min function that will find the difference between the largest and smallest element in a matrix. You can use the Octave *max* and *min* functions. Be careful to save the name of your function (e.g. ***max_min)*** in a file of the same name (example ***max_min.m***).

Contents of the ***template.m*** file:

```
function[out1, out2] = template(arg1, arg2)

%-------------------------------------------
%
%    Description:
%
%    Input:
%         arg1:
%         arg2:
%
%    Output:
%         out1:
%         out2:
%
%    Date:
%    Modif:
%--------------------------------------------
```

Solution to the exercise for the introduction to GNU/Octave

The first objective of this exercise is to make you familiar with Octave if you have never used it before, and to remind current users of its basic functions.

1 – Operators

1.2 –

The command *a=[1 4 5 ;4 8 9]* returns the matrix 2×3: $\begin{bmatrix} 1 & 4 & 5 \\ 1 & 8 & 9 \end{bmatrix}$.

The command *a=rand(5)* returns a matrix 5×5 made up of random values between 0 and 1.

Finally, the operator « ; » is used when you do not want to display a command's result in the console. This operator is useful, for example when you work with large matrices (such as images), which are sometimes long to display and often not very representative of the data.

1.3 –

Working with the operator « : ».

1.4 –

There are two types of matrix operations that use the « * » and « / » operators:

- matrix multiplication and division,
- element by element multiplication and division (joint use with the operator « . ».

The command *c=a*b* will perform a matrix multiplication of matrix « a » by matrix « b » : $c_{ij} = \sum_k a_{ik} b_{kj}$.

The command *e=a.*b* will perform an element by element multiplication of matrix « a » by matrix « b » : $e_{ij} = a_{ij} \cdot b_{ij}$. The matrices have to be the same size.

ACTION:

Create a matrix *A* with a size of 4×4 by directly entering these coefficients one by one: A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]

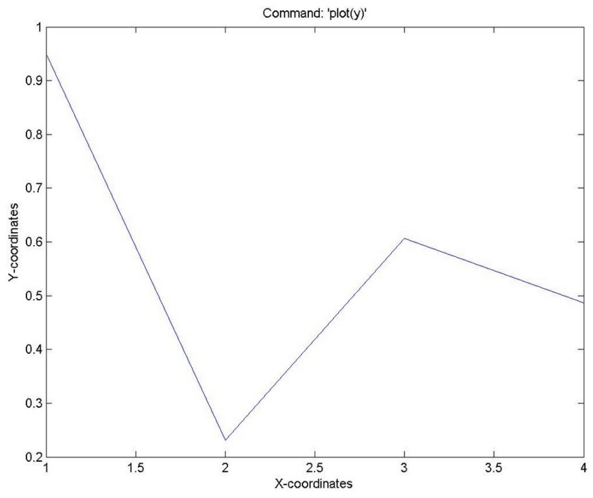
$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

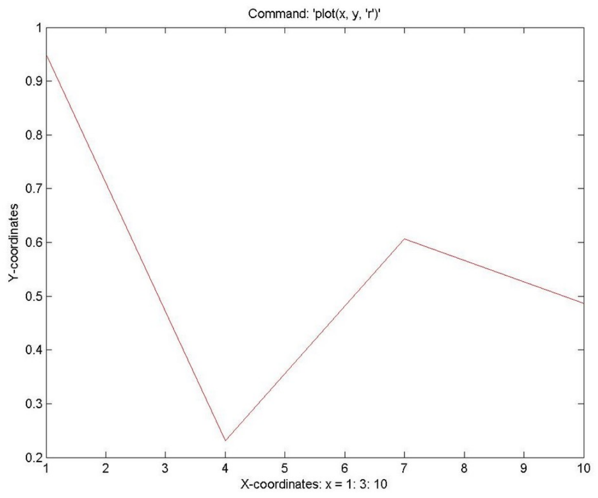
The 1st row of A is given by the command: *A(1,:)*.
The 4th column of A is given by the command: *A(:,4)*.
The first 3 elements of the 4th row of A are given by the command: *A(4,1:3)*.

2 – Display

2.1 –

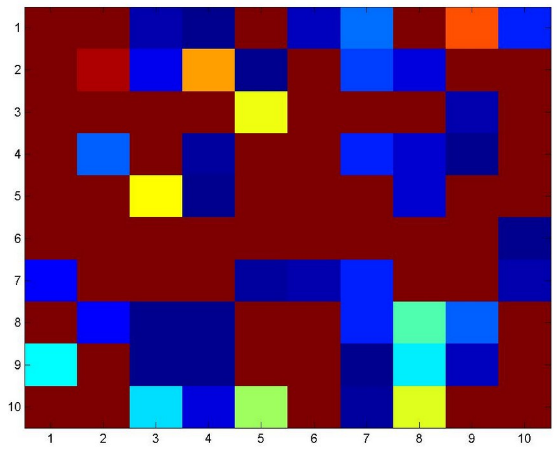
The command *y=rand(4,1)* returns a vector of 4 elements. By typing *plot(y)*, a curve appears. This curve represents the evolution of the vector « y » depending on the indices of the vector elements. However it is possible to modify the abscissas by creating an abscissa vector « x » in a given unit of the same size as the vector « y »: you plot y according to x by the command *plot(x,y)*.



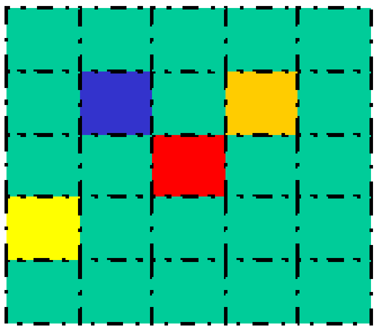


2.2 –

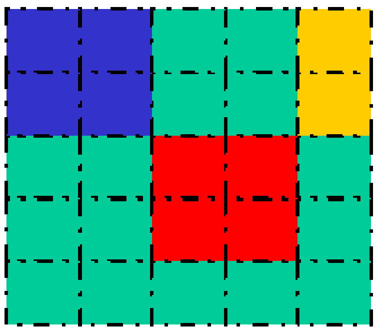
Here is an example of the result obtained by typing the commands indicated with a display by *image(a)*.



The 10×10 sized matrix is represented here by 10×10 square pixels of different colors. The colors correspond to the values of the various elements in the matrix.
The zoom uses a zero-order interpolation of a nearest neighbor type. This means that it is a simple imitation of an image pixel by several pixels on the display screen.
Let's consider for example a screen of 5×5 pixels, on which a 5×5 image is represented. The screen pixels correspond to the represented grid.



The figure below presents a ×2 zoom around the red pixel located in the center in the case of a nearest neighbor interpolation.



The red pixel is simply « duplicated » to twice its height and width on the screen pixels.

3 – Writing scripts

Working on a script.

4 – Data types

Working with uint8 type data.

5 – Writing your functions

The « ***max*** » function (and respectively the « ***min*** » function) given an M×N matrix as input, returns a vector of size N of which each element e_k is the maximum element (and for ***min*** the minimum element) of the matrix column k.

Here is the solution function:

```
function[out] = max_min(A)

%--------------------------------------------
%       Description: difference between the max and min
%       elements of a matrix A corresponding to a monochrome
%     image
%
%       Input:
%               A: the matrix on which the search occurs
%
%
%       Output:
%               out: the value of the difference
%---------------------------------------------

out = max(max(A))-min(min(A));
```