Introduction to MATLAB

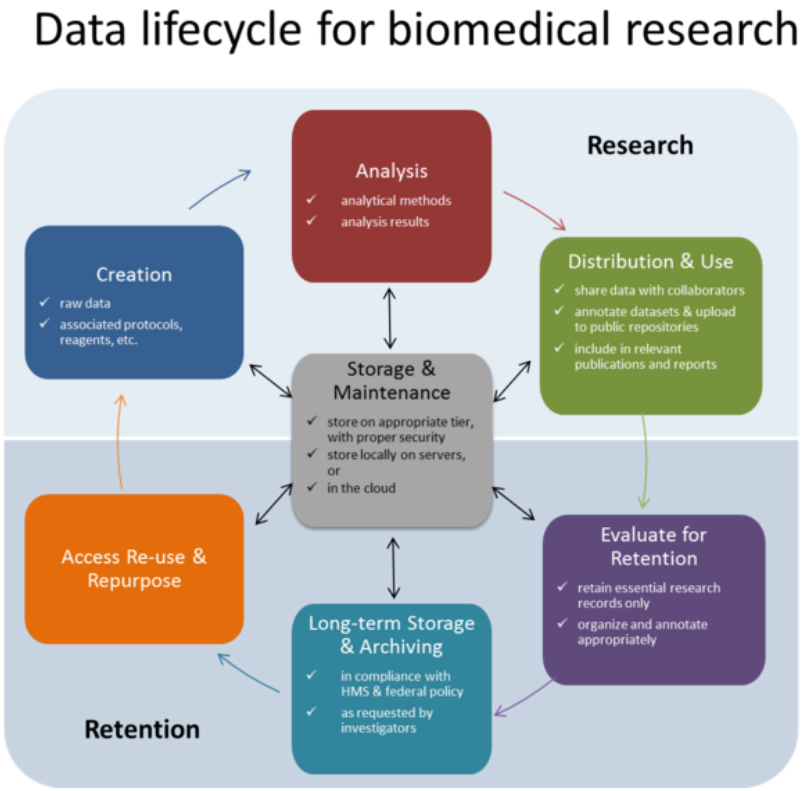
Raffaele Potami

Data and Script Management

2019

Data Management

As you run more jobs, you'll probably end up creating a whole bunch of files. In the same way it's important to plan bench projects beforehand, it’s good to think early about how you should manage and organize all those files you'll be making. *Note: be sure to ask your PI and your department about standard practices in your field!*



Harvard Biomedical Data Management Website: https://datamanagement.hms.harvard.edu

Resources & Information: https://datamanagement.hms.harvard.edu/overview

Top Data Management Best Practices

- **Planning**: Document the activities for the entire lifecycle. Create a Data Management Plan including sponsorship requirements, realistic budget, assigned responsibilities, all the data to be collected, *and each of these topics*! https://datamanagement.hms.harvard.edu/planning-overview

- **Organization**: Define how the data will be organized, including what is your folder hierarchy and how did you get from raw data to the final product? Consider versioning control for changes for both software and data products. https://datamanagement.hms.harvard.edu/versioning-1

- **Documentation**: Explain how the data will be documented such as naming conventions, acronyms, data fields and units. Determine whether there is a community-based metadata standard that can be adopted. Create a README file to record the metadata that will be associated with data. https://datamanagement.hms.harvard.edu/readme-files

Top Data Management Best Practices

- **Storage**: Your storage plan is integral to data management. Consider how the data will be stored and protected over the duration of the project. Identify short-term and long-term storage options. Remember to link accompanying metadata and related code and algorithms. https://datamanagement.hms.harvard.edu/storage-overview

- **Sharing**: Describe what data will be disseminated, to who, when, and where. Identify sharing tools to work with collaborators during the project and publish data in an open repository. Be sure to use standard, nonproprietary approaches and provide accompanying metadata & associated code. https://datamanagement.hms.harvard.edu/data-sharing

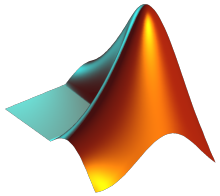
- **Retention**: Think about your preservation strategy from the start & adhere to your lab's standard practices. Research records should generally be retained no fewer than seven (7) years after the end of a research project or activity. https://datamanagement.hms.harvard.edu/data-retention

Material for this class

Download class material from:

https://rc.hms.harvard.edu/training/Intro_Matlab/

What is MatLab:



MATLAB = Matrix Laboratory

"MATLAB® is the high-level language and interactive environment used by millions of engineers and scientists worldwide. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, communications, control systems, and computational finance."

MATLAB is a high-performance language for scientific computing

MATLAB is an *interpreted* language (such as python, perl, java…and many others), no compilation is required !
Commands and instructions can be executed one at a time in an interactive way or as a collection via scripts.

Why you should use MATLAB:

- **Easy to code**: the language is quite intuitive
- **Easy to edit and debug**: the editor features include auto completion and error detection
- **Built-in function:** MATLAB comes with many built-in functions to solve complex problems
- **Visualization tools**: it is easy to plot data. Several options available to display results.

Why you should use MATLAB:

- **Powerful Toolboxes**: there are many available toolboxes to solve a wide range of problems (image, math, statistics, parallel, signal, biology, …)
- **Implicit Parallelization**: in MATLAB many built-in functions are automatically executed in a multithreading mode.
- **Explicit Parallelization**: users can explicitly parallelize parts of the code (shared and distributed memory parallelization, GPU)
- **Excellent Documentation**: easy to follow manual are readily available for each MATLAB command.

Start MATLAB on O2:

If you don't have MATLAB installed locally on your own laptop you can use MATLAB directly on O2. To do so:

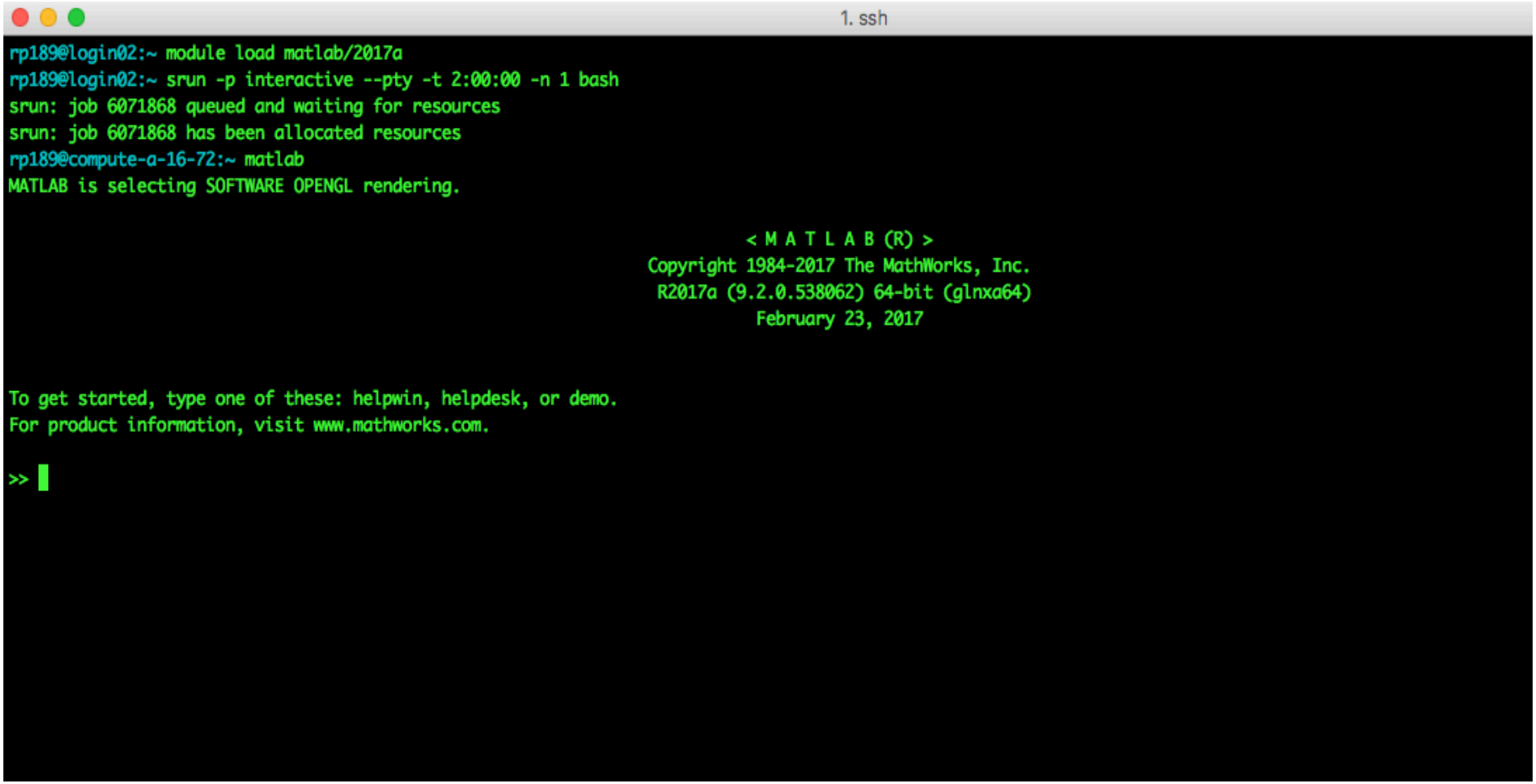
1) connect to the O2cluster (if you want to use the graphical interface make sure to connect using X11 forwarding)

2) load one of the MATLAB modules, for example *module load matlab/2018a*

3) start an interactive job session *srun -p interactive --pty -t 2:00:00 -n 1 bash*

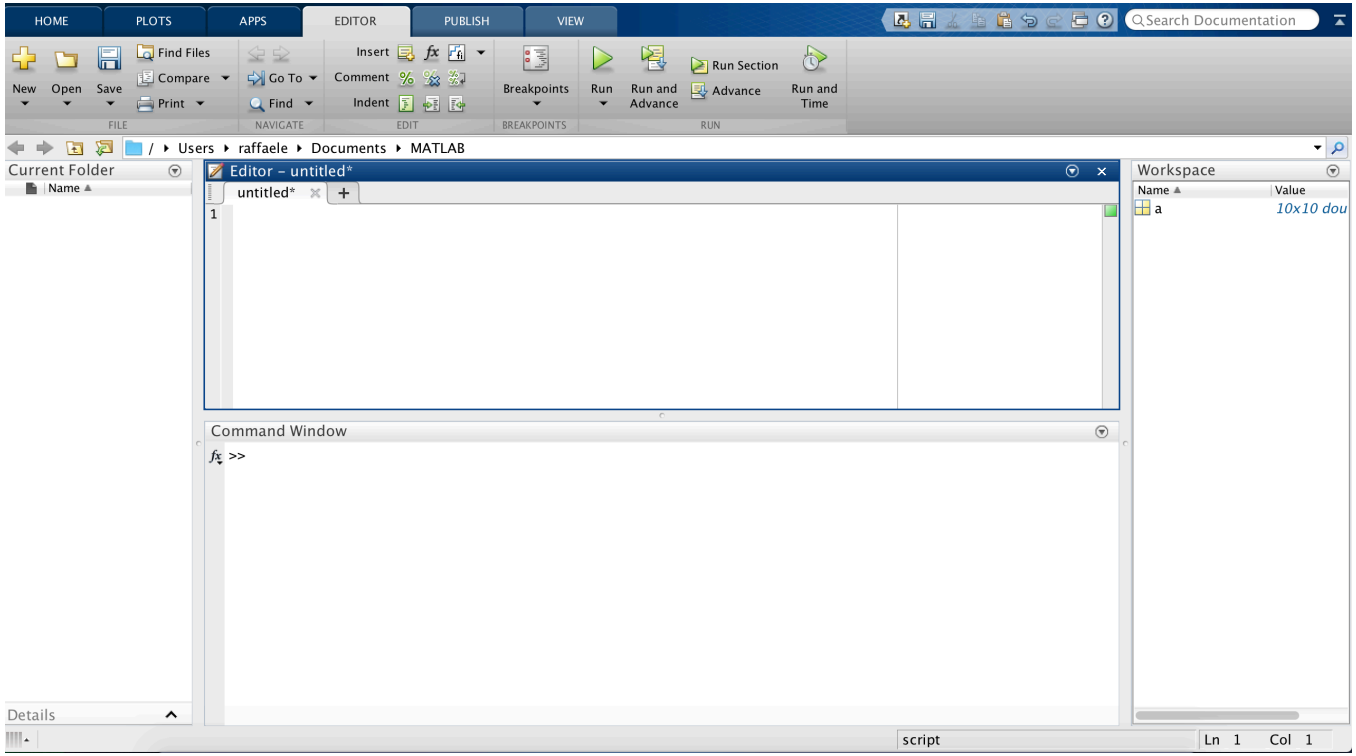
4) start the MATLAB desktop with the command *matlab*

Start MATLAB on Orchestra:



Start using MATLAB:

- Start a MATLAB section / Quit MATLAB
- MATLAB graphical user interface VS MATLAB command line
- Command window, workspace and script editor



- **Using Variables:** automatically assign type; functions to convert across different types (int8, int16, double, single, etc.). MATLAB is case sensitive !

- **Script Editor:** build up your first script and run it

- **Some of the basic commands and functions:** clear, clc, pwd, dir, ls, exit, quit, who, whos, save, load, plot, disp, format, length, size, +, -, *, /, ^, rand, max, min, sum, prod, sin, cos, ceil, floor, round, rem …

- **Getting help:** help, doc and lookfor (and Google !!!)

- **Using Vector and Matrices:** when possible try to use vector and matrices to store your data and perform operations. In MATLAB vectorization is the key to performance. Vectors are n x 1 matrices

Create a Vector:

manually with *a=[1 2 3 .. 10]* or
with an implicit loops *a=1:10*
using *linspace(1,10,10)*
using built-in functions *a=ones(1,5) a=zeros(1,5) a=rand(1,5)*

Slice a Vector:

b=a(1:3)
a(3:length(a))
b=a(3:end)
b=a(:)

Operations with Vector:

multiply or divide, add or subtract a single value to the vector
b=a*3 , b=a/3, b=a+3 … but with power exponent b=a.^2

multiply, divide, add, subtract, … each vector component to another vector component c=a.*b, c=a./b but c=a+b, c=a-b

Create a Matrix:

manually a=[1 2 3;4 5 6;7 8 9]
built-in function a=zeros(4), b=ones(4), c=rand(4), d=eye(4), e=diag([-2 -1 0 1 2 3])

Slice a Matrix:

a(3,5), a(2,:), a(:,3), a(:,3:4), a(1:2,1:2)

Operation with matrices:

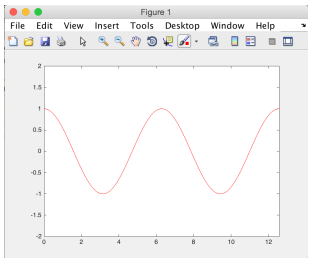
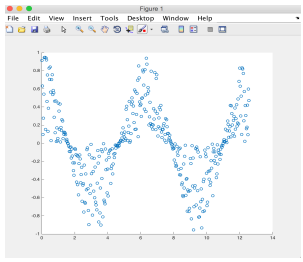
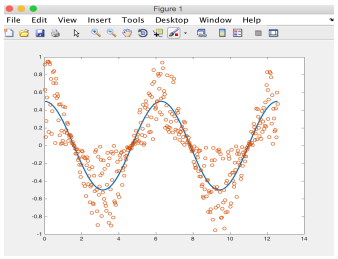
Element-wise operations and functions +, -, .*, ./, .^, sin(), cos()..

Matrices operation * / + -

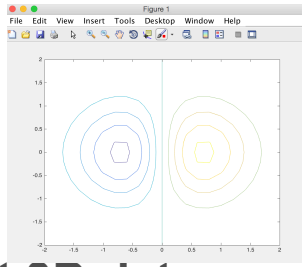
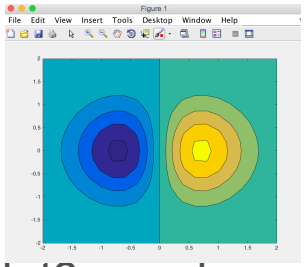
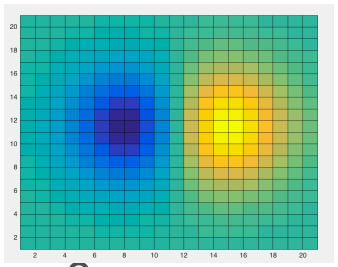
Functions max(), min(), sum(), diag(), inv(), find(),…

Data visualization in MATLAB

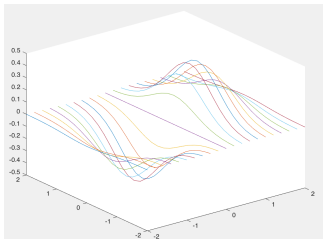
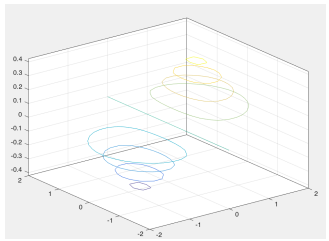
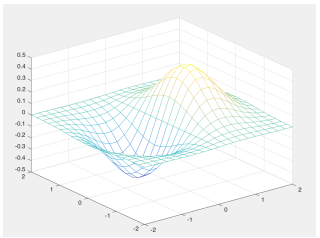
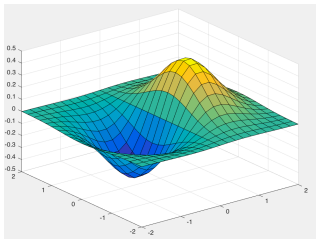
Plot 1D data commands: plot, scatter, loglog, semilogx, semilogy, plotyy, figure, axis, title, legend, xlabel,…

Plot 2D data commands: contour, contourf, pcolor, figure,…

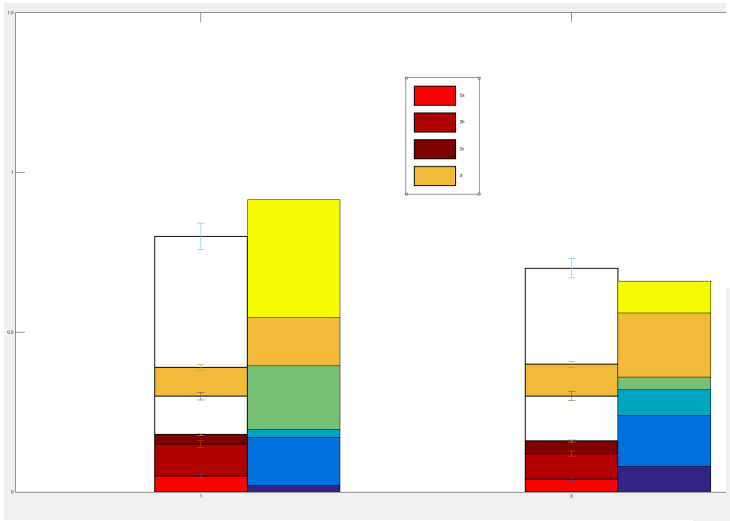




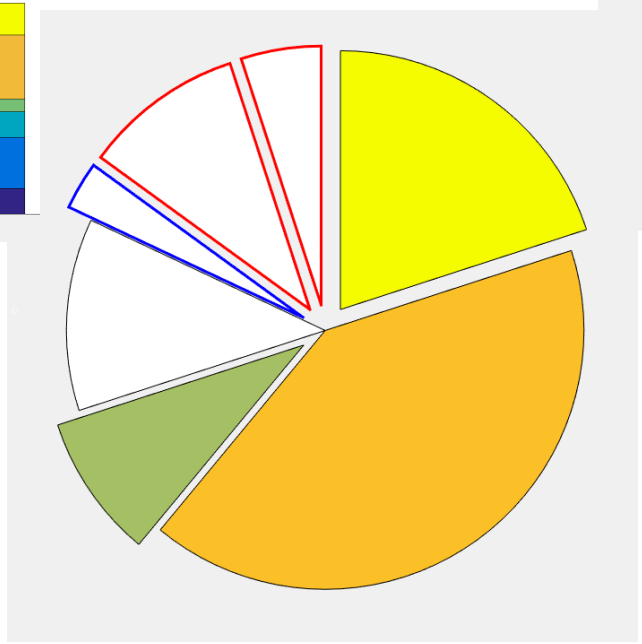
Plot 3D data commands: plot3, mesh, surf, contour3, …

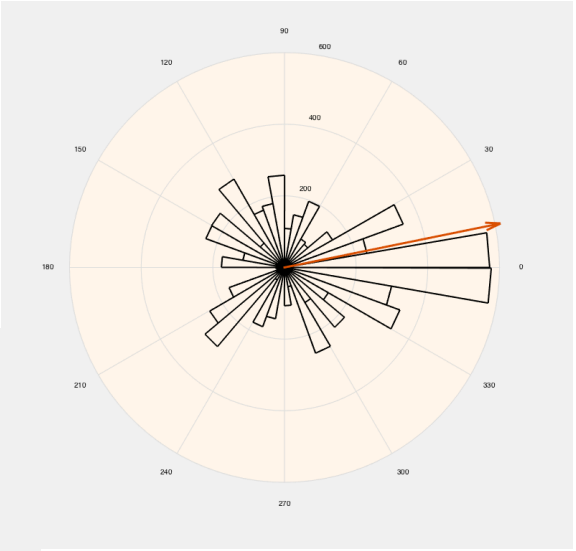





Data visualization in MATLAB

Plot Bar, Pie and Polar Charts

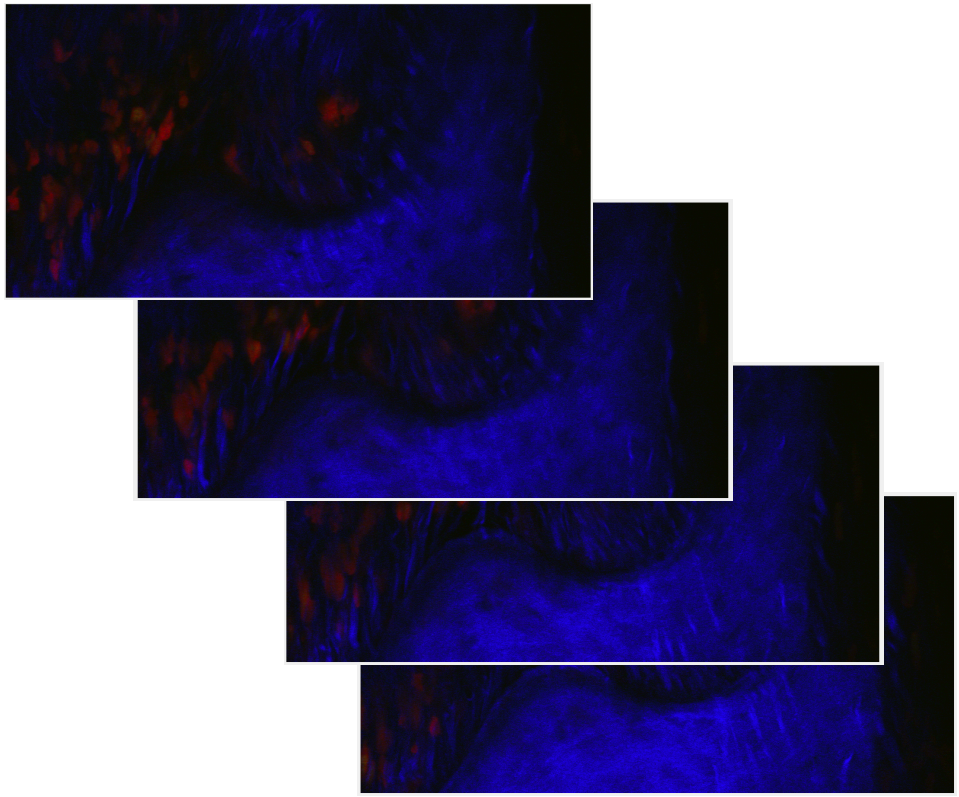


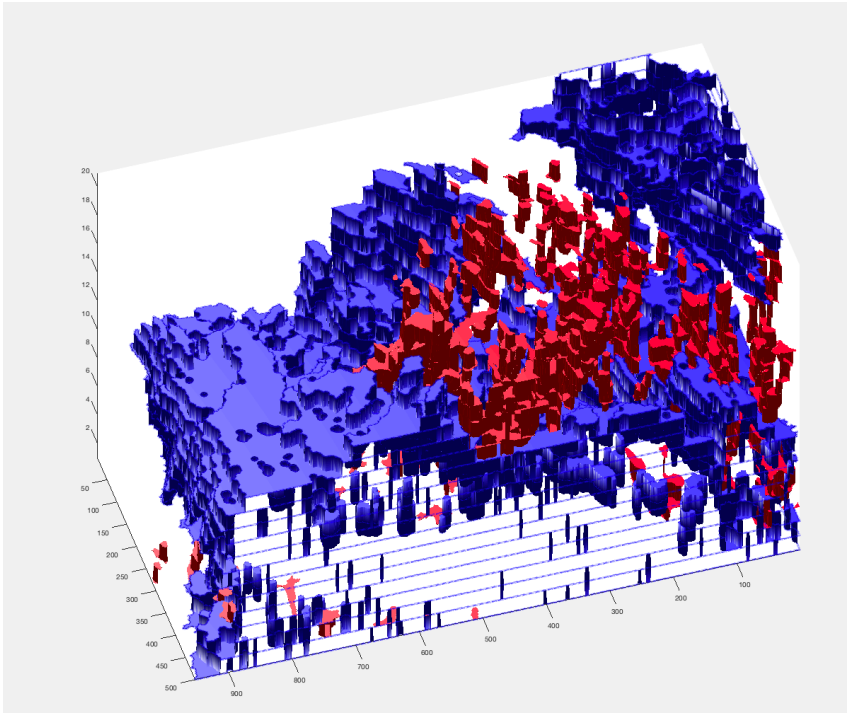




Data visualization in MATLAB

imshow, volumeViewer, isosurface, patch, etc





Programming in MATLAB: Scripts and Functions

Scripts: are collections of MATLAB commands that can be executed as a single jobs.

Functions: are collections of MATLAB commands used to simplify scripts by generalizing parts with same functionalities.

There are few major differences between functions and scripts:

SCRIPTS	FUNCTIONS
- Do not accept input arguments or return output arguments. - Store variables in a workspace that is shared with other scripts - Are useful for automating a series of commands	- Can accept input arguments and return output arguments. - Store variables in a workspace internal to the function. - Are useful for extending the MATLAB language for your application

Use MATLAB editor: commands completion and error auto detection

Functions in MATLAB

Functions are declared with the command ***function*** and can take multiple inputs and return multiple outputs

```
function [out1,out2,…] = my_function_name ( input1,input2,…)
   do something
end
```

you can then call the function from the command window or the script as

```
[val1,val2,…] = my_function_name (in1, in2, …)
```

If a function is saved as a stand alone file it is important to save the file with the same name as the function.
Matlab automatically searches local path and other specified paths for user-made functions and scripts.

The example above should be saved as *my_function_name.m* file

Useful tools to build scripts and functions

Control Flow: IF ELSE ELSEIF END

```
if expression
   statements
elseif expression
   statements
else
   statements
end
```

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
|	OR operator
~	NOT operator

Control Flow: SWITCH CASE END

```
switch switch_expression
  case case_expression
    statements
  case case_expression
    statements
   ...
  otherwise
    statements
end
```

Useful tools to build scripts and functions

Loop Control : FOR END

```
for index = values
  statements
end
```

```
s = 10;
H = zeros(s);

for c = 1:s
    for r = 1:s
        H(r,c) = r+c;
    end
end
```

Loop Control : WHILE END

```
while expression
   statements
end
```

```
n = 10;
f = n;
while n > 1
    n = n-1;
    f = f*n;
end
disp(['n! = ' num2str(f)])
```

Useful tools to build scripts and functions

return:

return forces MATLAB to return control to the invoking function/script before it reaches the end of the current function/script

continue:

continue passes control to the next iteration of a for or while loop. It skips any remaining statements in the body of the loop for the current iteration

break:

break terminates the execution of a for or while loop

exit:

exit terminates the current section of MATLAB

MATLAB save and load data

save: saves workspace variable to file.

- save myjob: saves all variables to the binary matlab file format myjob.mat
- save myjob var1 var2: saves var1 and var2 to the binary matlab file format myjob
- save myjob.txt var1 var2 -ascii: saves variables in ascii format file

csvwrite: writes a comma-separated value file

load: loads workspace variable to file.

- load myjob: loads all variables from the binary matlab file format myjob.mat
- load myjob var1 var2: loads var1 and var2 from the binary matlab file format myjob.mat
- load myjob.txt: loads variables in ascii format from file

csvread: reads a comma-separated value file

MATLAB no Desktop

It is possible to start an interactive section of MATLAB without the support of the GUI. This is often useful if working remotely on orchestra with a limited connectivity. To start MATLAB without GUI use

matlab –nodesktop

Often is also necessary to run long jobs/scripts, in this case it is possible to start the job/script with the command

sbatch jobscript

where jobscript is the file containing the SLURM flags and the Matlab script to be executed, for example:

```
#-----------------jobscript----------------------------------
#SBATCH -p short
#SBATCH  -t 1:00:00
#SBATCH --mem=8000
#SBATCH -n 1
module load matlab/2019a
matlab -batch "myfunction(my_inputs)"
#------------------------------------------------
```

It is also possible to include the flag *-singleCompThread* to disable automatic multithreading for built-in functions. Note that this flag does not disable explicit parallelization.

MATLAB Explicit Parallelization

parpool:

Create a parallel pool of workers on a cluster and return a pool object

parfor:

Execute for loop in parallel on workers in parallel pool

spmd:

Executes the code within the spmd body on several workers simultaneously

How to submit Matlab parallel jobs to O2:

Use the "Local Cluster" profile:

The parpool command creates a parallel pool of workers "locally" on the same node where the main Matlab code is executed.
To submit a parallel Matlab job using 8 workers (parpool(8)) use

sbatch parallel_jobscript

```
#-----------------parallel_jobscript---------------------------------
#SBATCH -p short
#SBATCH  -t 1:00:00
#SBATCH --mem=8000
#SBATCH -n 8
module load matlab/2019a
matlab -batch "myfunction(my_inputs)"
#------------------------------------------------
```

How to submit Matlab parallel jobs to O2:

Use the "O2 Cluster" profile:

MATLAB keeps track of parallel jobs with its own indexing (typically under ~/.matlab) and this could create a conflict if two or more jobs are dispatched by the SLURM scheduler at the very same time using the "Local cluster profile"

If you are submitting multiple MATLAB parallel jobs it is recommended to use the "*O2 MATLAB cluster profile"* and submit those jobs directly from within MATLAB

```
>> c=parcluster;
>> c.AdditionalProperties.WallTime = '48:00:00';
>> c.AdditionalProperties.QueueName = 'mpi';
>> c.AdditionalProperties.AdditionalSubmitArgs = '--mem-per-cpu=2G'
>> c.saveProfile
% submit a parallel job with 8 cores
>> job=c.batch(@my_parallalel_function, Nout, {Inputs},'Pool',8);
```

Please check our O2 wiki page for additional information.

MATLAB Vectorization key to performance

Whenever possible make sure to use built-in functions for vectors and array.

Test Problem: Given a random matrix find the sum of all elements > 0.5

```
clc
clear;
L=10000;
a=rand(L);
tic
total=0;
for m=1:L
    for n=1:L
        if a(m,n) > 0.5
            total=total+a(m,n);
        end
    end
end
total
toc
```

```
clc
clear;
L=10000;
a=rand(L);
tic
L=10000;
a=rand(L);
b=a>0.5;
c=a.*b;
total2=sum(sum(c));
total2
toc
```

Useful online resources:

How to run Matlab on O2 cluster: https://wiki.rc.hms.harvard.edu:8443/display/O2/Using+MATLAB

Matlab parallel computing on O2: https://wiki.rc.hms.harvard.edu:8443/display/O2/Matlab+Parallel+jobs+using+the+custom+O2+cluster+profile

Matlab installation notes: https://wiki.med.harvard.edu/Software/MatlabInstallationNotes