How to run Matlab jobs

What is Matlab?:
Matlab ( http://www.mathworks.com ) is a high-level language and interactive environment to analyse and visualize data, and to simulate models.

Execution modes :
Matlab can be run in different ways, supporting a variety of workflows:

- **Interactive mode**: this is the typical mode you are probably familiar with when using Matlab on your workstation or laptop. You can open Matlab ( either with a text mode interface or with the standard working environment GUI ) and execute your instructions in an interactive way one by one.
- **Batch mode**: you can invoke Matlab and have it execute a script that does not require your interaction.
- **Packaged Matlab executables**: you can compile your code with the Matlab compiler ‘mcc’ and create a standalone application that can be run anywhere without needing a Matlab license at runtime. Please note that, although the tool is called Matlab ‘compiler’, it does not really mean that your code gets optimized in any sort of real compilation to machine code. The tool is actually a ‘packaging tool’ to allow you to create a package that to be executed will only need the Matlab runtime environment and will not an active license at runtime. So it’s very useful if you would like to run your application on another computing platform where you do not have access to licenses (for example on one of the XSEDE clusters), but do not expect any performance benefit.

Parallelization in Matlab :
Matlab supports **serial calculation** - both as single threaded and multi-threaded runs - and also has a toolbox that allows to control explicitly a parallel execution within a single server (Parallel Computing Toolbox).

Note that there is also an additional tool (distributed computing server) that allows to distribute calculations across multiple servers in a computing cluster or a computing grid; however this requires an extremely expensive license, which we currently don’t have.

- **Serial calculation, with automatic multi-threading**: even if in your Matlab program you don’t use any explicit parallel algorithm, Matlab will start a number of workers threads, and many of the underlying instructions will be executed taking advantage of this multiple threads. This is the default mode. It is the recommended mode if you use Matlab on your workstation, or if you use it on the cluster ONLY when you allocate in your qsub request a full node.
- **Serial calculation, with a single computational thread**: using the option `-singleCompThread` Matlab will start as a single threaded application. This is the recommended mode when you run on the cluster (both in batch or interactive mode) requesting a single slot.
- **Parallel calculation using the Parallel computing toolbox**: with the parallel computing toolbox, you can explicitly control the parallel execution of your code by starting a pool of multiple workers and distribute the work among them. The work can be distributed only within the same physical server. You can also use the toolbox to offload some of the calculation to Graphical Processing Units.

SOME EXAMPLES

1- How to run a **single-thread serial job** in interactive mode ( in text mode ) :
This is probably useful when testing line by line parts of your script or when developing your code.
## first open an interactive session with qrsh. ( you can add more option if needed, e.g. memory allocation, ...) 

```
login-node-0-0 >$ qrsh -q your_favourite_queue.q
```

## once you are presented a terminal on the compute node, first load the Matlab version you need 

```
compute-node-0-0 >$ module load MATLAB/R2015a
```

## and then start matlab in single thread mode and specifying you don't want the GUI. 

```
compute-node-0-0 >$ matlab -nodesktop -nosplash -nojvm -singleCompThread
```

2- How to run a **single-thread serial job in batch mode**:

This is the typical way you would run a serial calculation. 

In this case there are two ways you can start your batch calculation with matlab:

a) If you have a **script** ( does not need to be a function ) that you want to run, you can indirect that as input using "<" for example:

```
## In this case the submission of the shell script will be done with qsub as we want to run a batch job
login-node-0-0 >$ qsub -q your_favourite_queue.q  my_matlab_job.txt

## where the content of the job script will be something like this
login-node-0-0 >$ cat  my_matlab_job.txt
# any other qsub option you might want ( name, memory , ... )
#$  -cwd
#$  -ckpt reloc
module load MATLAB/R2015a
matlab -nodesktop -nosplash -nojvm -singleCompThread < myscript.m
```

b) if instead your calculation can be **started executing a function** - e.g. you have a mymain.m that contains a mymain function that does your job - you can execute the function calling it with the option "-r" for example:

```
## In this case the submission of the shell script will be done with qsub as we want to run a batch job
login-node-0-0 >$ qsub -q your_favourite_queue.q  my_matlab_job.txt

## where the content of the job script will be something like this
login-node-0-0 >$ cat  my_matlab_job.txt
# any other qsub option you might want ( name, memory , ... )
#$  -cwd
#$  -ckpt reloc
module load MATLAB/R2015a
matlab -nodesktop -nosplash -nojvm -singleCompThread -r "mymain;quit"
```
3- How to run many single-thread serial jobs in batch mode at the same time:

Often you might want to run many very similar calculations, for example if you are doing a parameter sweep, or running a simulation sampling the phase space starting from many different conditions.

In this case you can probably leverage on SGE job arrays to differentiate the runs.

Remember that this is nothing like the parallel computing toolbox. The tasks will be executed as independent jobs (same as you would have really submitted them one by one with qsub).

When using job arrays, for example, if you use a function to start your calculation (like in example 2b), you can pass the value of the environmental variable SGE_TASK_ID as parameter to the function and then inside your function use that value to differentiate the work done.

Or, both if you start your calculation with a function or a script, inside your code you can read at runtime the value of that variable with the matlab function “getenv”

mytaskid = str2num(getenv("SGE_TASK_ID")) and then you can use the value of mytaskid to differentiate what each of the task will do. (getenv returns a string so you should use str2num if you need to covert to a number)
## Using the qsub option

- Using the `qsub` option `-t 1-10` we will start 10 tasks with values of `SGE_TASK_ID` ranging from 1 to 10.

```bash
login-node-0-0 > $ qsub -q your_favourite_queue.q -t 1-100 my_matlab_job.txt
```

## We can use the value of `SGE_TASK_ID` as argument of the function:

```bash
login-node-0-0 > $ cat my_matlab_job.txt
#$ -t 1-10
#$ -cwd
#$ -N mja ## matlab job array

module load MATLAB/R2015a
matlab -nodesktop -nosplash -nojvm -singleCompThread -r "mymain($SGE_TASK_ID);quit"

## and in this case mymain is a very silly function that does 2^x where x is the argument passed to the function.

```bash
login-node-0-0 > $ cat mymain.m
function mymain(x)
a = 2
fprintf('the answer for this task is %d\n',a.^x);
```

```
## we can check that each task computed a different result.
```

```bash
grep "the answer for this task" test1.job* | sort -n -k 7
mja.o10.1:the answer for this task is 2
mja.o10.2:the answer for this task is 4
mja.o10.3:the answer for this task is 8
mja.o10.4:the answer for this task is 16
mja.o10.5:the answer for this task is 32
mja.o10.6:the answer for this task is 64
mja.o10.7:the answer for this task is 128
mja.o10.8:the answer for this task is 256
mja.o10.9:the answer for this task is 512
mja.o10.10:the answer for this task is 1024
```

```bash
### or we could have used alternatively this version of the function (or of a script) reading with getenv the value of the variable SGE_TASK_ID.
```

```bash
login-node-0-0 > $ cat mymain.m
function mymain()
a = 2
x = str2num(getenv('SGE_TASK_ID'))
fprintf('the answer for this task is %d\n',a.^x);
```

```
## we can check that each task computed a different result.
```

```bash
grep "the answer for this task" test1.job* | sort -n -k 7
mja.o10.1:the answer for this task is 2
mja.o10.2:the answer for this task is 4
mja.o10.3:the answer for this task is 8
mja.o10.4:the answer for this task is 16
mja.o10.5:the answer for this task is 32
mja.o10.6:the answer for this task is 64
mja.o10.7:the answer for this task is 128
mja.o10.8:the answer for this task is 256
mja.o10.9:the answer for this task is 512
mja.o10.10:the answer for this task is 1024
```

```
### or we could have used alternatively this version of the function (or of a script) reading with getenv the value of the variable SGE_TASK_ID.
```

```bash
login-node-0-0 > $ cat mymain.m
function mymain()   
a = 2
x = str2num(getenv('SGE_TASK_ID'))
fprintf('the answer for this task is %d\n',a.^x);
```

```
## we can check that each task computed a different result.
```

```bash
grep "the answer for this task" test1.job* | sort -n -k 7
mja.o10.1:the answer for this task is 2
mja.o10.2:the answer for this task is 4
mja.o10.3:the answer for this task is 8
mja.o10.4:the answer for this task is 16
mja.o10.5:the answer for this task is 32
mja.o10.6:the answer for this task is 64
mja.o10.7:the answer for this task is 128
mja.o10.8:the answer for this task is 256
mja.o10.9:the answer for this task is 512
mja.o10.10:the answer for this task is 1024
```

```
### or we could have used alternatively this version of the function (or of a script) reading with getenv the value of the variable SGE_TASK_ID.
```

```bash
login-node-0-0 > $ cat mymain.m
function mymain()
   a = 2
   x = str2num(getenv('SGE_TASK_ID'))
   fprintf('the answer for this task is %d\n',a.^x);
```

```
## and in this case the script would have simply looked like this.
```

```bash
login-node-0-0 > $ cat my_matlab_job.txt
#$ -t 1-10
#$ -cwd
#$ -q all.q

module load MATLAB/R2015a
matlab -nodesktop -nosplash -nojvm -singleCompThread -r "mymain;quit"
```
4- How to run a parallel job with the Parallel Computing Toolbox:

When using the parallel computing toolbox you can start a parallel pool of workers that you can use to distribute your calculations.

It's IMPORTANT that you allocate a number of cores (for example using -pe orte N) == to the number of workers you will be initializing in the parallel pool. (see function parpool https://www.mathworks.com/help/distcomp/parpool.html)

In that case it does not matter if you are starting it with singleCompThread or not, as the workers will be anyway started as single threaded subprocesses.

XXX I will add a page with examples for different use cases later XXX

5- How to run an interactive job with the GUI

This is your typical use mode on your workstation, but on the cluster it probably makes senses only when you want to run some data analysis or need some interactive graphics.

For this we have a visualization node which is setup to leverage on the remote graphic acceleration provided by 2 Nvidia GPU.

NOTE: at the moment we only have 1 node on the visualization.q so please remember to logout when you are done.

Also, to ensure a good turnover we enforce a 6hrs walltime limit.

The recommended workflow is the following:
A- Connect to the cluster using a remote desktop solution. Currently we support both x2go and fastx, but x2go will be deprecated soon. See here how to access the cluster directly in your browser with fastx
B- open a terminal and start an interactive session with qrsh on the visualization queue.

C- load the module and start matlab using the vglrun command to enable the hardware accelerated openGL

```
## first open an interactive session with qrsh.
login-node-0-0 >$ qrsh -q visualization.q

## once you are presented a terminal on the visualization node, first load the Matlab version you need
>$ module load MATLAB/R2015a

## and then start matlab using vglrun
>$ vglrun -c proxy `which matlab`
```

FINAL NOTES:

summary of some of the other options used above (type matlab -help for a list of all the options):

-nodesktop -nosplash -nojvm: these are to run in batch mode without pulling up the GUI and the jvm.

If you use some toolboxes, like for example the parallel computing toolbox, you DO need the jvm. in that case just use "-nodesktop -nosplash"

-singlCompThread tells matlab not to start in multithreaded mode. If you are running many jobs, each of them allocating a single core, you definitely want to use that option.

If instead you want to leave the multithreading activated, don't use that option, but make sure to allocate a full node (for example 12 cores using the -pe orte 12 option of qsub)

Note that besides the case described in 5), using a multithreaded execution on the cluster does not really make much sense.