1. Running matlab m-files from the terminal

In order to understand how to run m-files on the cluster, we will first show how to run a small matlab job on the head of the cluster.

However, please note that you are not allowed to run *big* jobs on the head node, since the head node of the cluster should only be used to *submit* jobs on the execution hosts.

Method 1. Run m-files by reading and executing them line by line

Use qsub to submit a matlab session were each line of a certain m-file is run sequentially

Assume we would like to run the following m-file called test.m:

```
a = 1
b = 10
c = a + b
```

Let's first check how matlab would run this if we execute the command directly on the head:

```
/usr/local/matlabR2010A/bin/matlab -nodisplay -nojvm  <
         "/home/henk/scripts/test.m"
```

The results of the equations will be shown in the terminal window. Note that the < operator here reads the file that is mentioned after this operator. This file runs through matlab *line by line*.

Method 2. Run m-files that are functions

To run m-files that are functions, we need to call the –r parameter in matlab.

Assume we would like to run the following function called testpar.m with the value 3 to be passed as the parameter:

```
function testpar(t)

a = t
b = 10
c = a + b

end
```

To do this, we can't simply refer to the testpar.m file directly, because it is not possible to define parameters within the filepath!

Instead, we need to run matlab in –r mode and refer to the name of the function, that is the name of the m-file, without the extension .m, like this

```
/usr/local/matlabR2010A/bin/matlab -nodisplay -nojvm -r "testpar(3)"
```

And this will show

```
< M A T L A B (R) >
                  Copyright 1984-2010 The MathWorks, Inc.
                Version 7.10.0.499 (R2010a) 64-bit (glnxa64)
                             February 5, 2010


 pathdef UITGEVOERD
 pathdef UITGEVOERD


  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
a =

     3


b =

    10


c =

    13

>>
```

Note that there is still a blinking matlab cursor at the bottom of the screen. Because we runned matlab in –r mode, it is not closed automatically. In order to exit matlab and return to the terminal prompt, type

```
exit
```

Be careful! If you did not run the command from the correct path, the function will not be found and you will get the following error within matlab

```
>> ??? Undefined function or method 'testpar' for input arguments of type
'double'.
```

So before running the matlab command, make sure to 1) **cd** to the correct working directory, or 2) to have added the path were the m-file is located in the pathdef.m (use **edit pathdef.m** in matlab).

2. Running matlab m-files on the cluster from the terminal

Let's now demonstrate how to run the same commands by submitting them to the SGE cluster.

What we need to do is call the matlab command via qsub. We also need to add some parameters relevant for the job that is going to be submitted.

Method 1. Run m-files by reading and executing them line by line

We are going to run the same matlab job, but this command is now echo'ed and pipelined to the qsub command (as the final parameter) using the pipeline operator "|".

If you run this

```
echo '/usr/local/matlabR2010A/bin/matlab -nodisplay -nojvm <
        "/home/henk/scripts/test.m"' | qsub -cwd -S /bin/bash
```

you will receive the following message:

```
Your job <job number> ("<job name>") has been submitted
```

So, you now have submitted your matlab job to the cluster.

Before running the command, make sure your terminal screen is wide enough to show the command on a single line. Otherwise, the echo might be interpreted differently and the job might not be submitted correctly. If the job is still not run correctly, try to open a fresh terminal session.

Note that the qsub command includes the parameters –cwd and –S /bin/bash.

-cwd sets the current directory (so the output and error files will be written in the same path as where you have run the command)

-S sets the default shell to bash

Feel fry to add other qsub parameters as well such as –q to set a preferred queue, -N to give the job a name, or –o and –e to redirect the output and error files to a different location. See the qsub documentation.

Given that the job is run on the cluster, you will not receive any visual feedback. After the job has been scheduled and is finished (in order to check that: use qstat), you may have a look at the output and error files created by the cluster. For instance, if you want to view the content of the output created by matlab, use this command

```
cat *.o<job number>
```

Alternatively, you can use a standard text editor like **gedit**.

Method 2. Run m-files that are functions

We can use the same pipelining method to run a matlab function (with parameters):

```
echo '/usr/local/matlabR2010A/bin/matlab -nodisplay -nojvm -r "testpar(3)"'
        | qsub -cwd -S /bin/bash
```

Again, before running the matlab command, make sure to 1) **cd** to the correct working directory, or 2) to have added the path were the m-file is located in the pathdef.m (use **edit pathdef.m** in matlab).

3. Running Batch jobs of matlab functions within matlab

In order to run system commands in matlab, you can use the matlab function **system**.

E.g., to run the command **qstat** in the matlab GUI, type:

```
system('qstat');
```

Similarly, we can now run a matlab function on the cluster, by using the submit command described in the previous section and nesting this into the system function, like:

```
system('echo ''/usr/local/matlabR2010A/bin/matlab -nodisplay -nojvm -r
        "testpar(3)"'' | qsub -cwd -S /bin/bash')
```

Note that, because matlab reserves the single quote to delineate the start and end of a string, we had to replace the single quote by two single quote characters in order to be interpreted correctly.

Ok, the next step is to create a loop around this command.

Assume, we have created the following vector which should be passed through testpar as parameters.

```
A = {'3','5','10','320'}
```

This results in

```
A =

    '3'    '5'    '10'    '320'
```

Now, we will add a loop that reads each single element of A (starting with element number 1 until the end of A has been reached) and submit the testpar function to the cluster using that element's value (A{i}) as parameter:

```
for i = 1:length(A)

    commandline = sprintf('echo ''/usr/local/matlabR2010A/bin/matlab -
         nodisplay -nojvm -r "testpar(%s)"'' | qsub -cwd -S /bin/bash',
         A{i});

    system(commandline);

end
```

Note that we use the sprintf function to replace the original value 3 by the value of A{i}. %s is used as the placeholder.