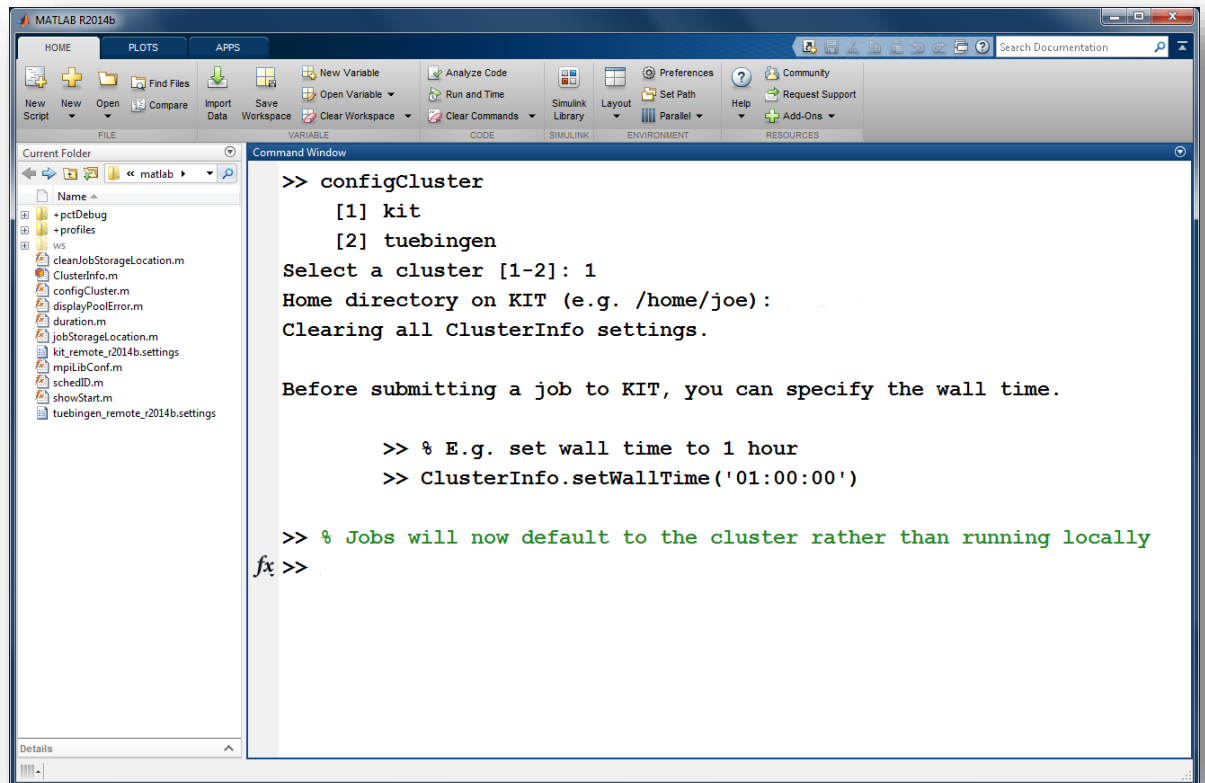


Getting Started with Serial and Parallel MATLAB on bwGRiD

CONFIGURATION

- Download either `bwgrid.remote.r2014b.zip` (Windows) or `bwgrid.remote.r2014b.tar` (Linux/Mac)
- For Windows users, unzip the download and place the contents into `%matlab%\toolbox\local`. If you don't have permissions, then place the contents in the folder returned by `userpath` (for example `My Documents\MATLAB` or `Documents\MATLAB`).
- For Linux users, untar the download and place the contents into `$matlab/toolbox/local`
- Start MATLAB. Configure MATLAB to run parallel jobs on the bwGRiD clusters by calling `configCluster`.
- For each cluster (e.g. `kit`, `tuebingen`), `configCluster` only needs to be called once per version of MATLAB (e.g. R2014b, R2015a)



```
>> configCluster
      [1] kit
      [2] tuebingen
Select a cluster [1-2]: 1
Home directory on KIT (e.g. /home/joe):
Clearing all ClusterInfo settings.

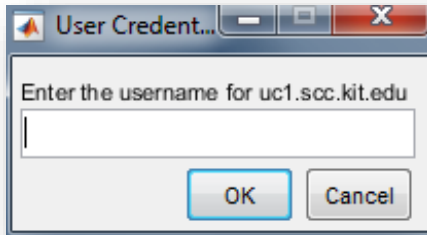
Before submitting a job to KIT, you can specify the wall time.

      >> % E.g. set wall time to 1 hour
      >> ClusterInfo.setWallTime('01:00:00')

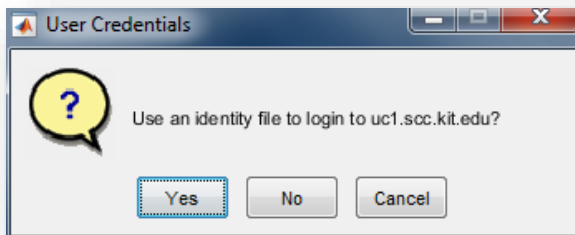
>> % Jobs will now default to the cluster rather than running locally
fx >>
```

CREDENTIALS

The first time a user submits a job to the cluster, the user will be prompted for their username

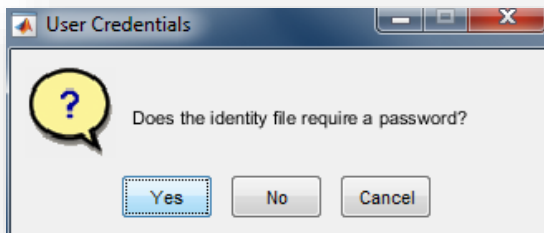


The user will then be prompted whether to supply a password or a private key



If the user chooses a private key, the user will be prompted for the location of the file. Both the username and private key are stored with MATLAB so that they are not prompted for it at a later time.

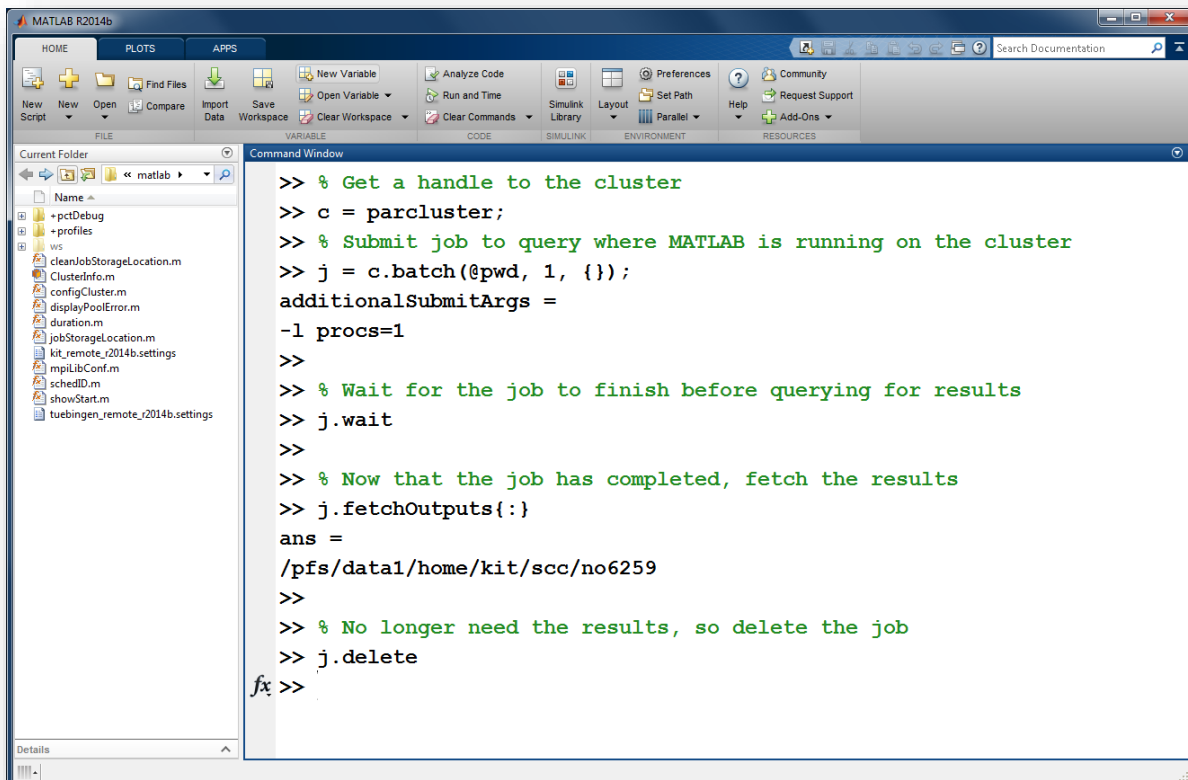
If using a private key, the user may also be prompted if the key requires a passphrase.



SERIAL JOBS

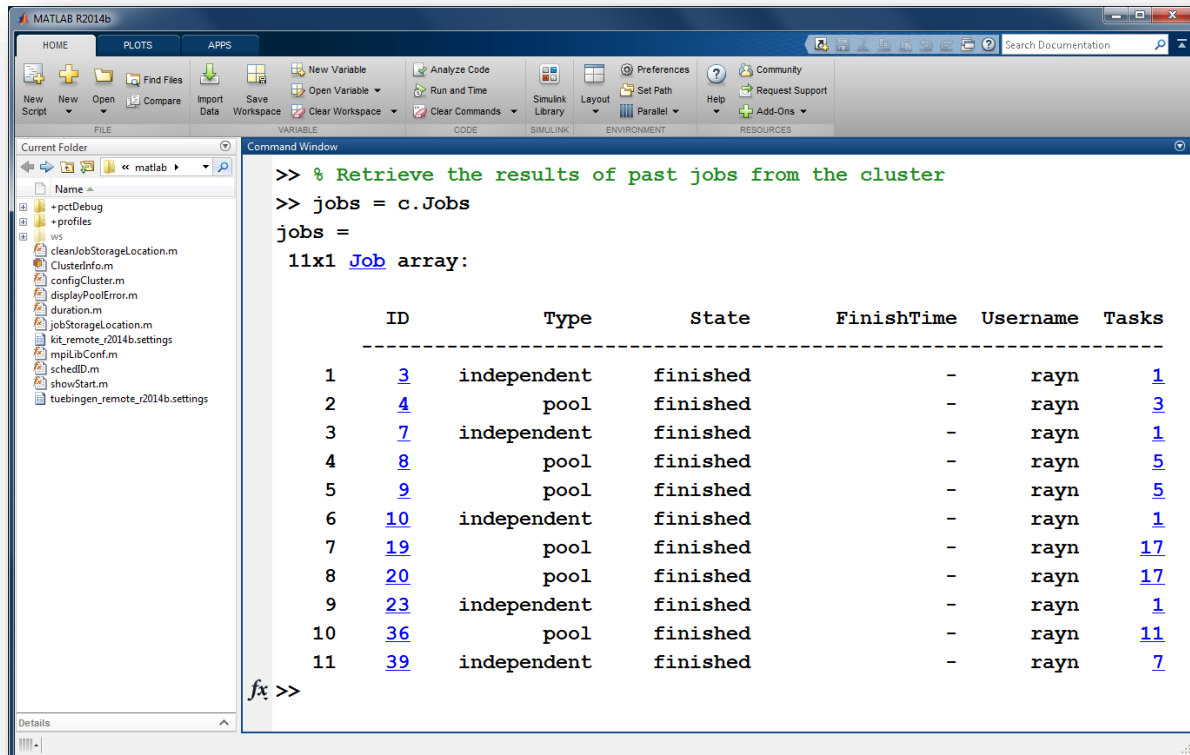
Use the `batch` command to submit asynchronous jobs to the cluster. The batch command will return a job object which is used to access the output of the submitted job. See the example below and see the MATLAB documentation for more help on `batch`.

Note: In the example below, `wait` is used to ensure that the job has completed before requesting results. In regular use, one would not use `wait`, since a job might take an elongated period of time, and the MATLAB session can be used for other work while the submitted job executes.



```
>> % Get a handle to the cluster
>> c = parcluster;
>> % Submit job to query where MATLAB is running on the cluster
>> j = c.batch(@pwd, 1, {});
additionalSubmitArgs =
-1 procs=1
>>
>> % Wait for the job to finish before querying for results
>> j.wait
>>
>> % Now that the job has completed, fetch the results
>> j.fetchOutputs{:}
ans =
/pfs/data1/home/kit/scc/no6259
>>
>> % No longer need the results, so delete the job
>> j.delete
fx >>
```

To retrieve a list of currently running or completed jobs, call `parcluster` to retrieve the cluster object. The cluster object stores an array of jobs that were run, are running, or are queued to run. This allows us to fetch the results of completed jobs. Retrieve and view the list of jobs as shown below.



The image shows a screenshot of the MATLAB R2014b Command Window. The Command Window contains the following text:

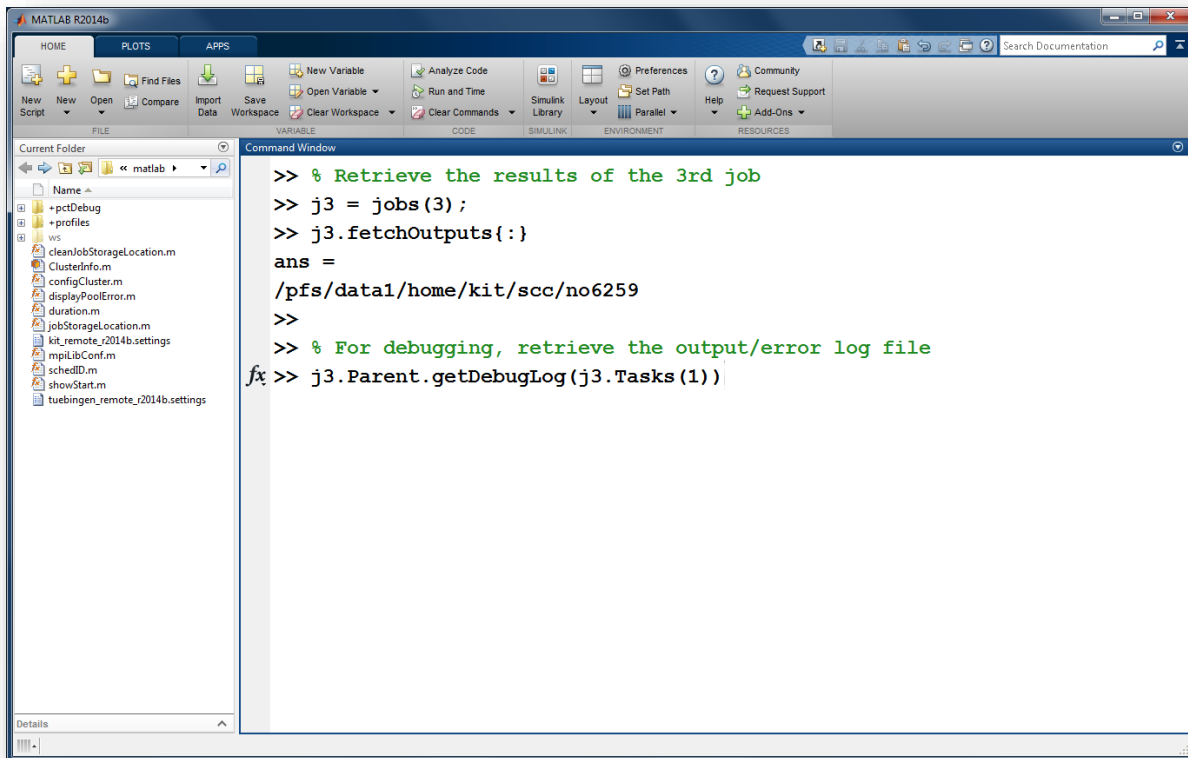
```
>> % Retrieve the results of past jobs from the cluster
>> jobs = c.Jobs
jobs =
11x1 Job array:
```

	ID	Type	State	FinishTime	Username	Tasks
1	3	independent	finished	-	rayn	1
2	4	pool	finished	-	rayn	3
3	7	independent	finished	-	rayn	1
4	8	pool	finished	-	rayn	5
5	9	pool	finished	-	rayn	5
6	10	independent	finished	-	rayn	1
7	19	pool	finished	-	rayn	17
8	20	pool	finished	-	rayn	17
9	23	independent	finished	-	rayn	1
10	36	pool	finished	-	rayn	11
11	39	independent	finished	-	rayn	7

fx >>

Once we've identified the job we want, we can retrieve the results as we've done previously. If the job produces an error, we can call the `getDebugLog` method to view the error log file. The error log can be lengthy and is not shown here. The example below will retrieve the results of job #3.

NOTE: `fetchOutputs` is used to retrieve function output arguments. Data that has been written to files on the cluster needs be retrieved directly from the file system.

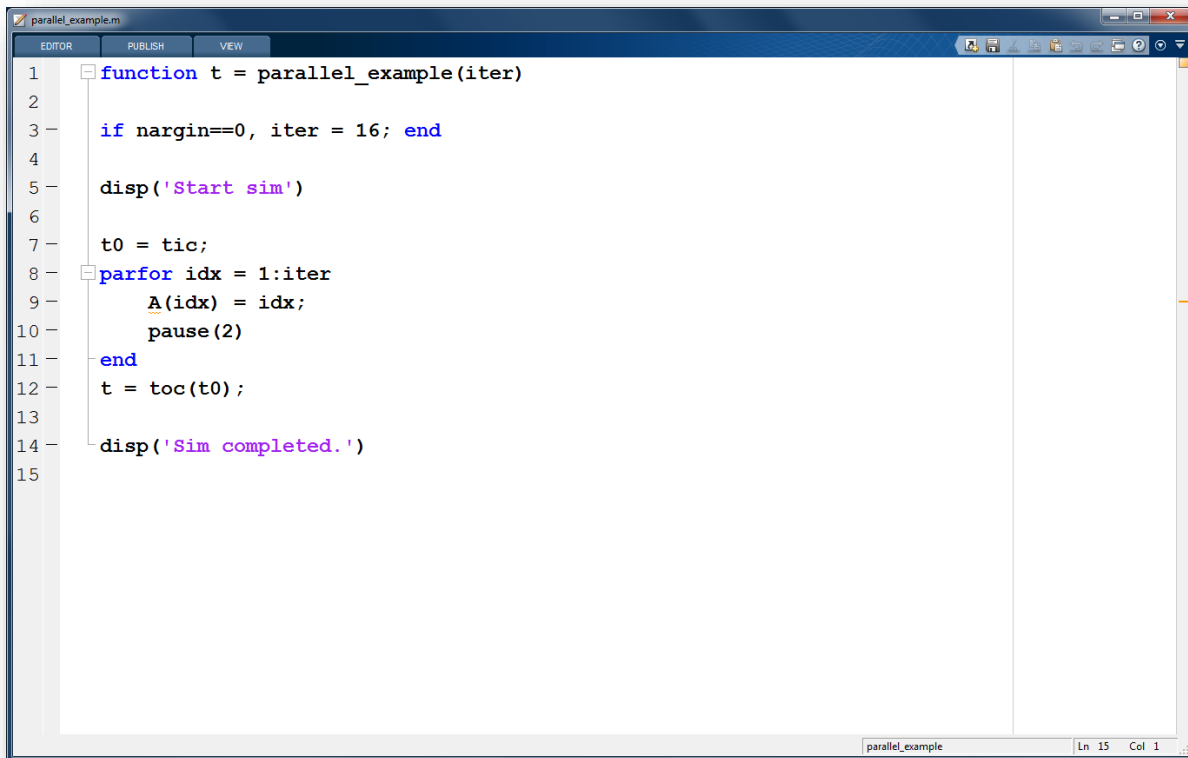


The image shows the MATLAB R2014b Command Window with the following code:

```
>> % Retrieve the results of the 3rd job
>> j3 = jobs(3);
>> j3.fetchOutputs{:}
ans =
/pfs/data1/home/kit/scc/no6259
>>
>> % For debugging, retrieve the output/error log file
fx >> j3.Parent.getDebugLog(j3.Tasks(1))
```

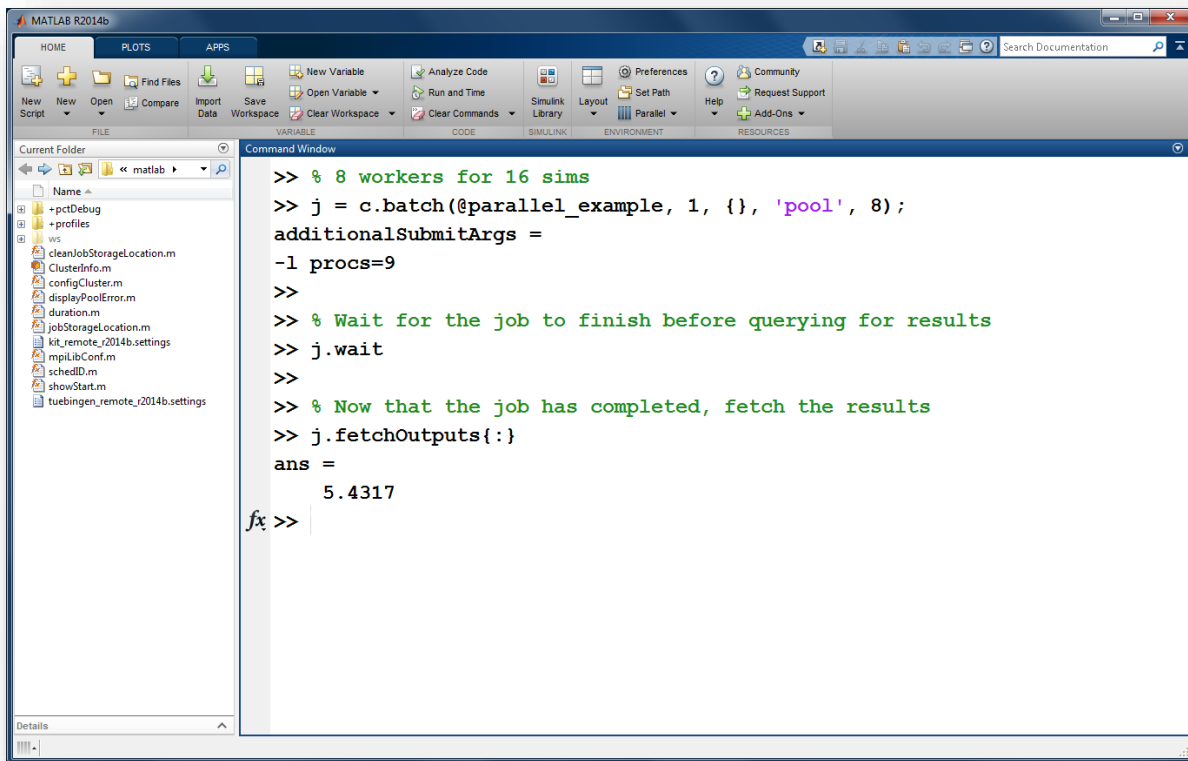
PARALLEL JOBS

Users can also submit parallel workflows with batch. Let's use the following example for a parallel job.



```
1 function t = parallel_example(iter)
2
3     if nargin==0, iter = 16; end
4
5     disp('Start sim')
6
7     t0 = tic;
8     parfor idx = 1:iter
9         A(idx) = idx;
10        pause(2)
11    end
12    t = toc(t0);
13
14    disp('Sim completed.')
15
```

We'll use the `batch` command again, but since we're running a parallel job, we'll also specify a MATLAB Pool.

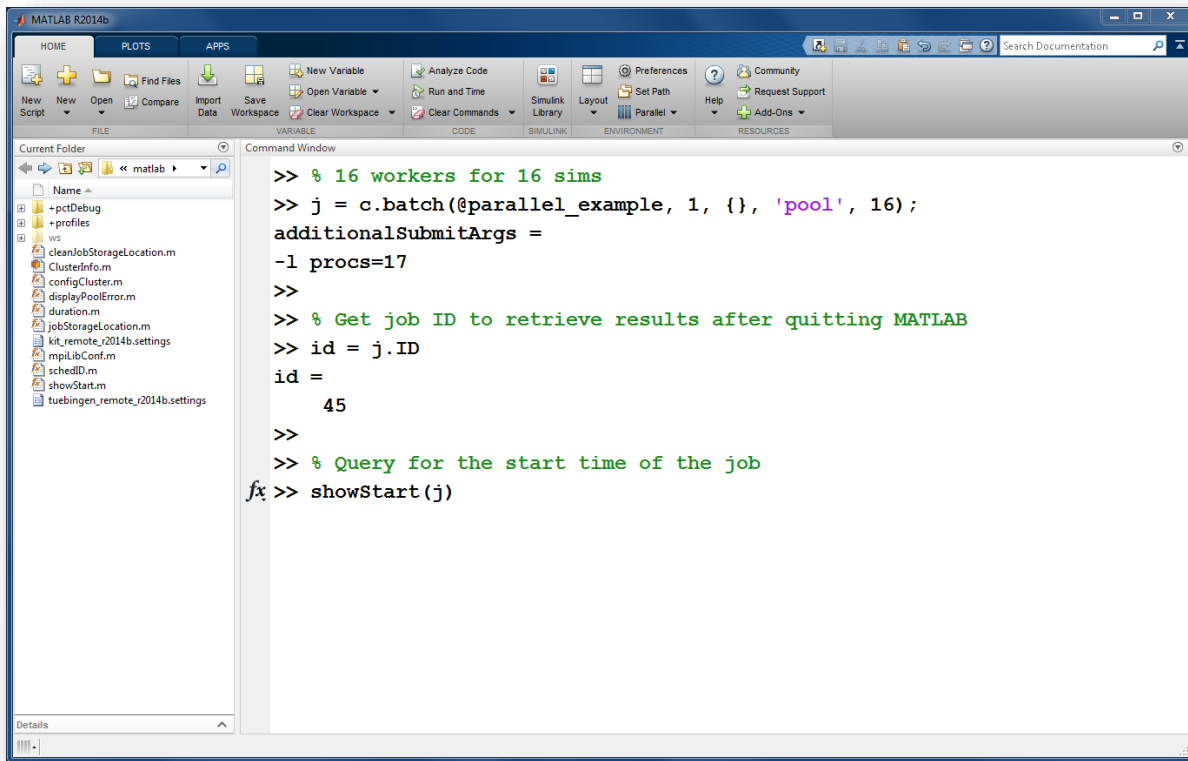


The image shows the MATLAB R2014b Command Window with the following code and output:

```
>> % 8 workers for 16 sims
>> j = c.batch(@parallel_example, 1, {}, 'pool', 8);
additionalSubmitArgs =
-1 procs=9
>>
>> % Wait for the job to finish before querying for results
>> j.wait
>>
>> % Now that the job has completed, fetch the results
>> j.fetchOutputs{:}
ans =
    5.4317
fx >>
```

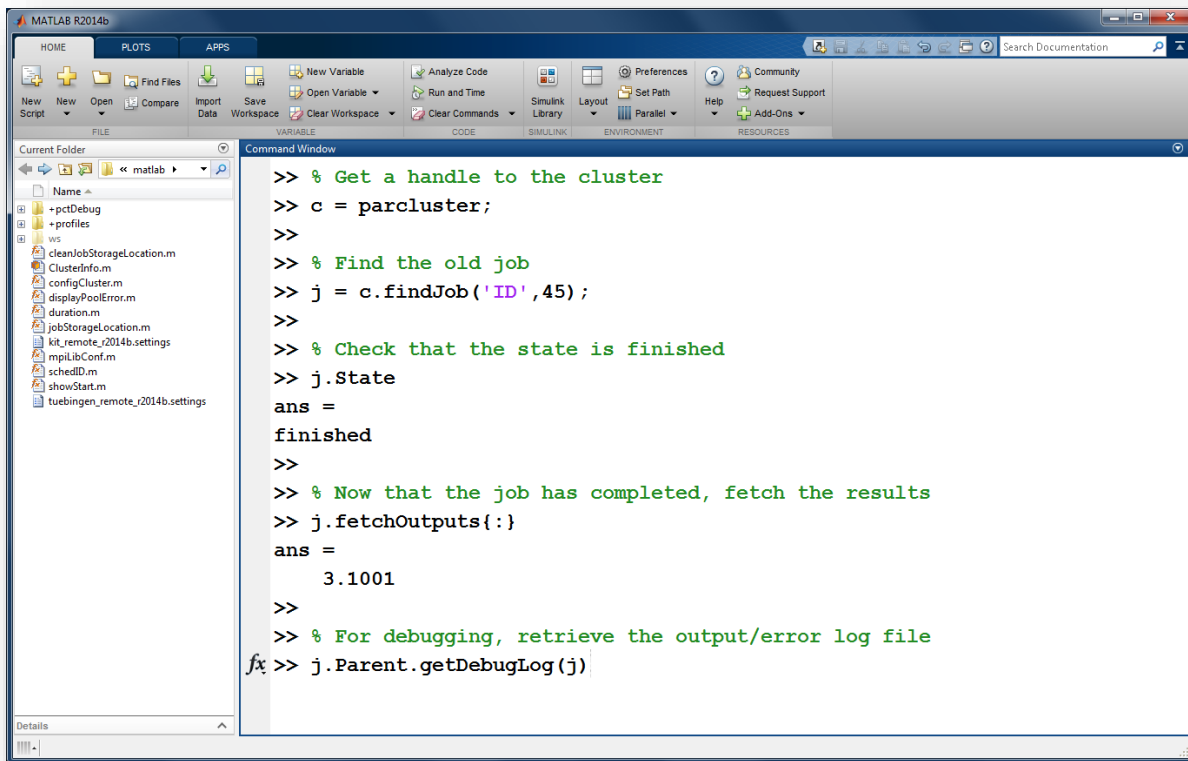
The job ran in 5.43 seconds using eight workers. Note that these jobs will always request N+1 CPU cores, since one worker is required to manage the batch job and pool of workers. For example, a job that needs eight workers will consume nine CPU cores.

We'll run the same simulation, but increase the Pool size. Note, for some applications, there will be a diminishing return when allocating too many workers. This time, to retrieve the results at a later time, we'll keep track of the job ID. We can also query for when the cluster may run the job by calling `showStart`.



```
>> % 16 workers for 16 sims
>> j = c.batch(@parallel_example, 1, {}, 'pool', 16);
additionalSubmitArgs =
-1 procs=17
>>
>> % Get job ID to retrieve results after quitting MATLAB
>> id = j.ID
id =
    45
>>
>> % Query for the start time of the job
fx >> showStart(j)
```

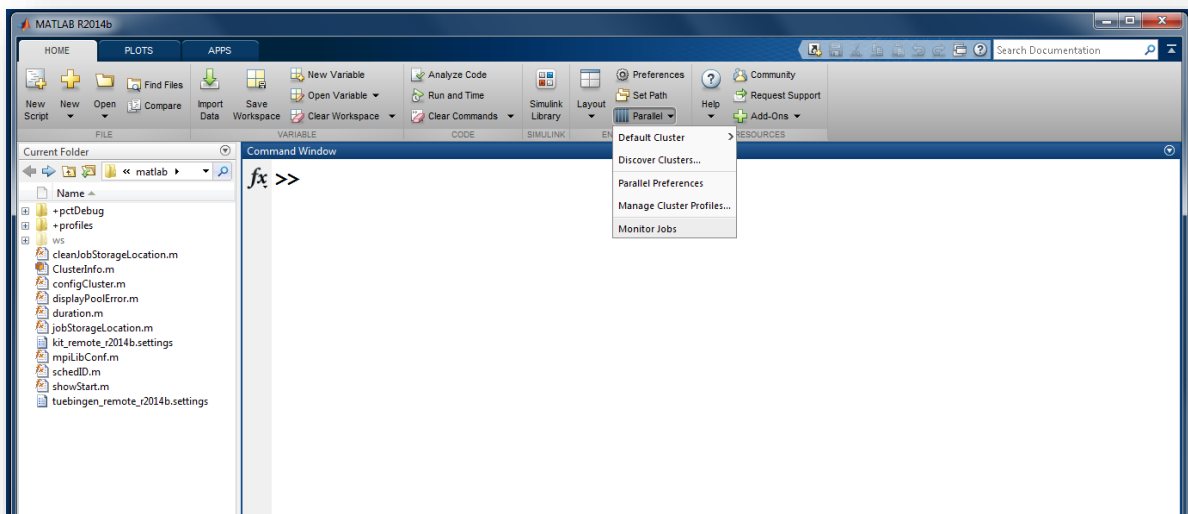
Once we have a handle to the cluster, we'll call the `findJob` method to search for the job with the specified job ID.



```
>> % Get a handle to the cluster
>> c = parcluster;
>>
>> % Find the old job
>> j = c.findJob('ID',45);
>>
>> % Check that the state is finished
>> j.State
ans =
finished
>>
>> % Now that the job has completed, fetch the results
>> j.fetchOutputs{:}
ans =
    3.1001
>>
>> % For debugging, retrieve the output/error log file
fx >> j.Parent.getDebugLog(j)
```

The job now runs in 3.1 seconds using 16 workers. Run code with different numbers of workers to determine the ideal number to use.

Alternatively, to retrieve job results via a graphical user interface, use the Job Monitor (Parallel > Monitor Jobs).

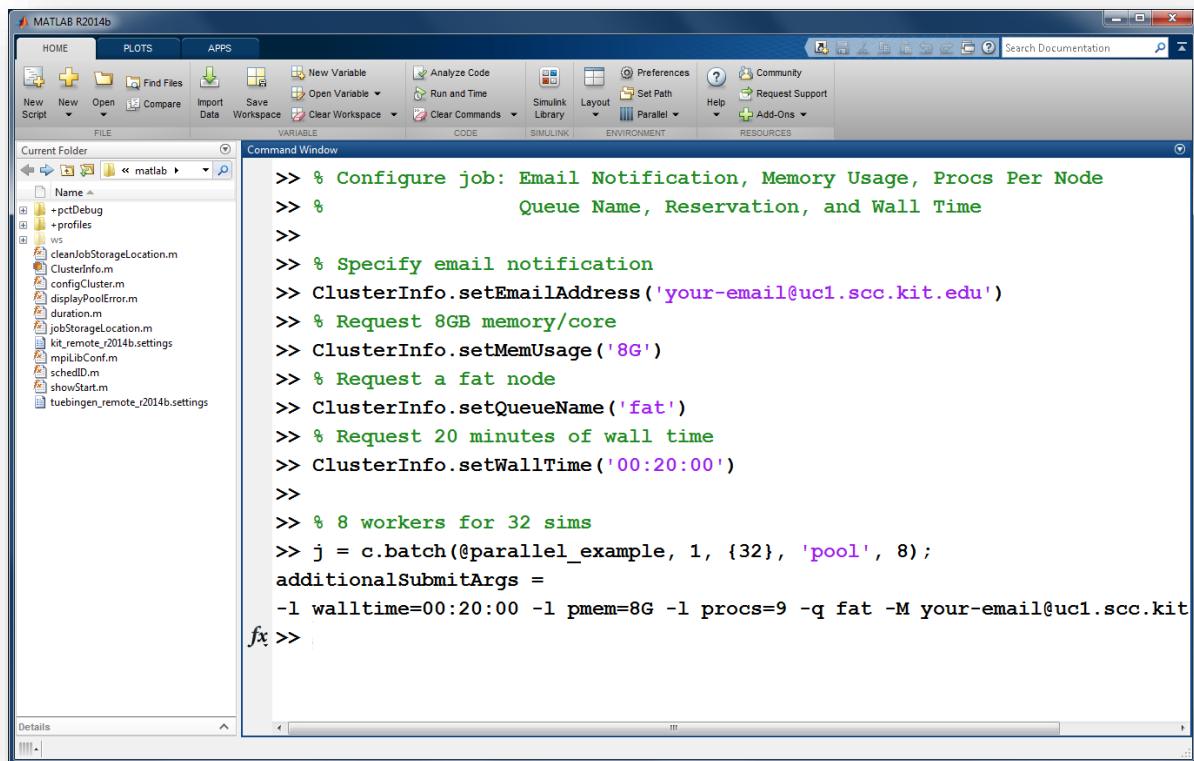


CONFIGURING JOBS on KIT

Prior to submitting the job, we can specify:

- Email Notification (when the job is running, exiting, or aborting)
- Memory Usage per Core,
- Procs Per Node
- Queue Name
- Reservation, and
- Wall time

Specification is done with `ClusterInfo`. The `ClusterInfo` class supports tab completion to ease recollection of method names.



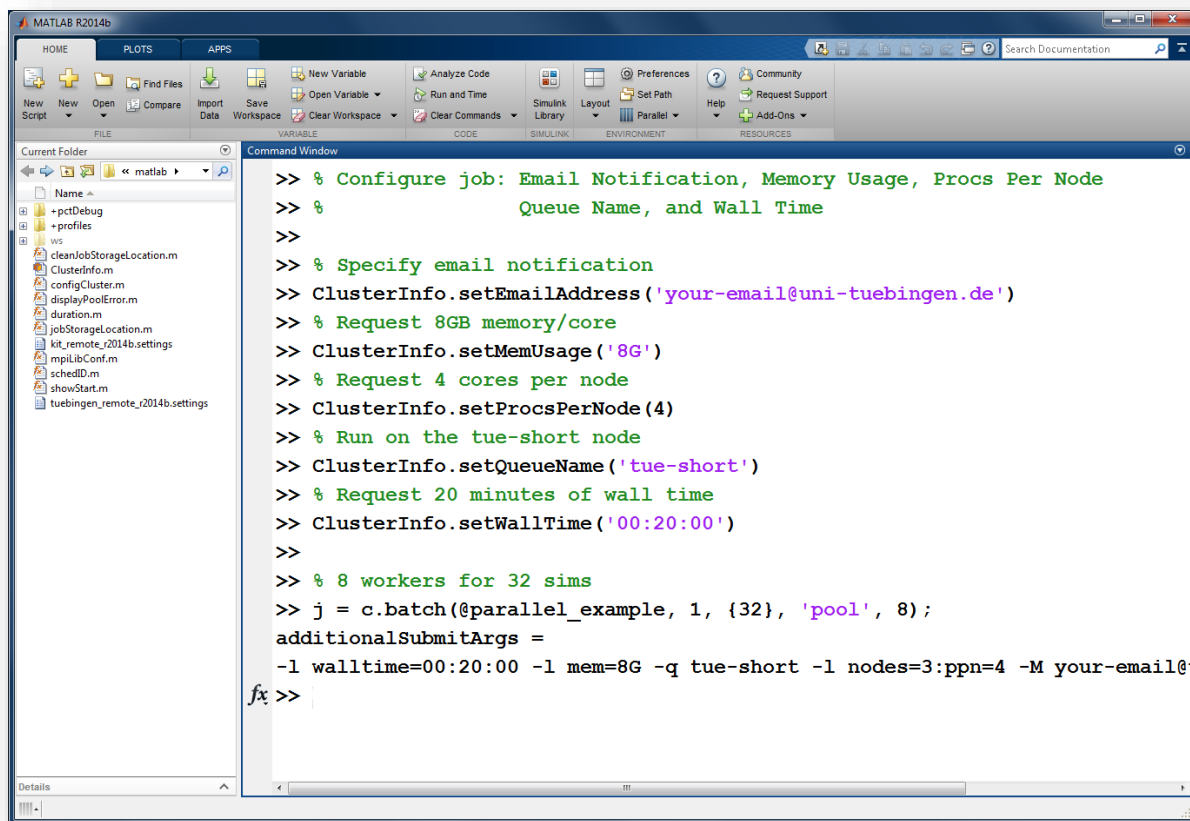
```
>> % Configure job: Email Notification, Memory Usage, Procs Per Node
>> %                               Queue Name, Reservation, and Wall Time
>>
>> % Specify email notification
>> ClusterInfo.setEmailAddress('your-email@uc1.scc.kit.edu')
>> % Request 8GB memory/core
>> ClusterInfo.setMemUsage('8G')
>> % Request a fat node
>> ClusterInfo.setQueueName('fat')
>> % Request 20 minutes of wall time
>> ClusterInfo.setWallTime('00:20:00')
>>
>> % 8 workers for 32 sims
>> j = c.batch(@parallel_example, 1, {32}, 'pool', 8);
additionalSubmitArgs =
-l walltime=00:20:00 -l pmem=8G -l procs=9 -q fat -M your-email@uc1.scc.kit
fx >>
```

CONFIGURING JOBS on TUEBINGEN

Prior to submitting the job, we can specify:

- Email Notification (when the job is running, exiting, or aborting)
- Memory Usage per Core,
- Procs Per Node
- Queue Name, and
- Wall time

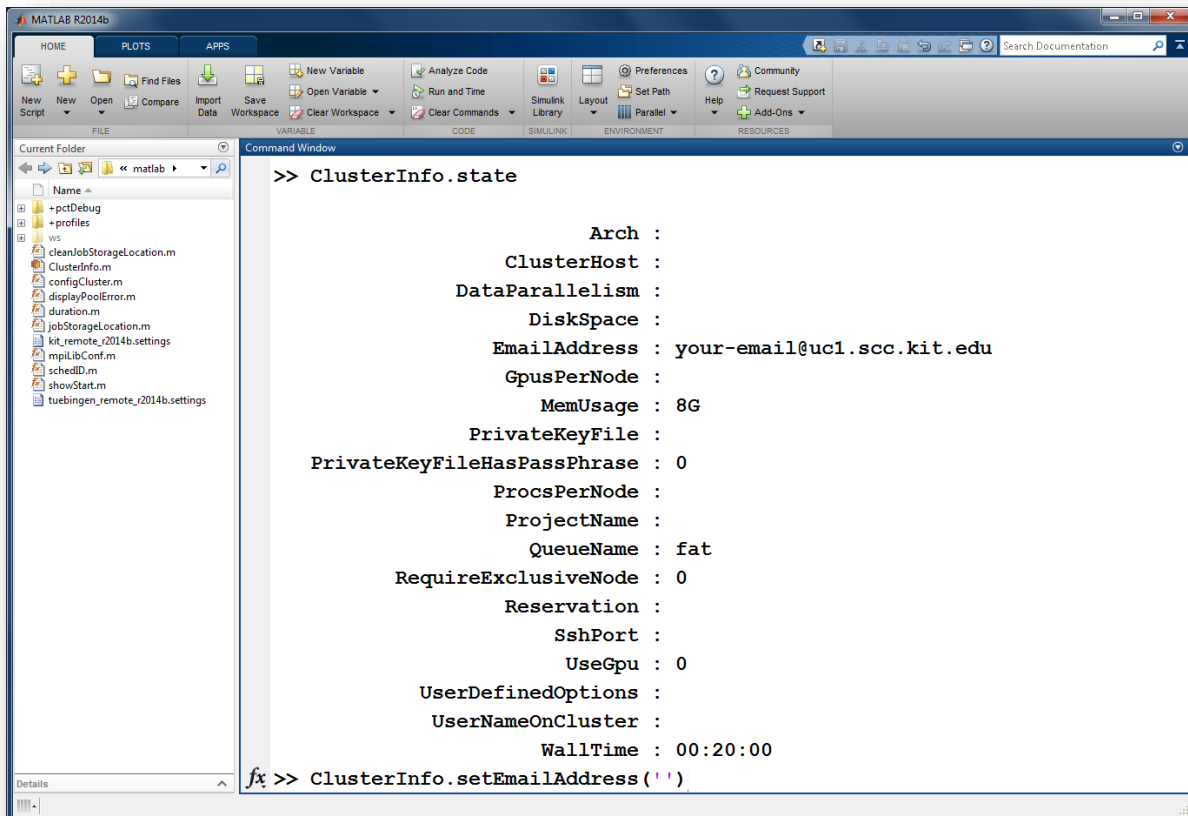
Specification is done with `ClusterInfo`. The `ClusterInfo` class supports tab completion to ease recollection of method names.



The image shows a screenshot of the MATLAB R2014b Command Window. The window title is 'MATLAB R2014b'. The Command Window contains the following code:

```
>> % Configure job: Email Notification, Memory Usage, Procs Per Node
>> %                               Queue Name, and Wall Time
>>
>> % Specify email notification
>> ClusterInfo.setEmailAddress('your-email@uni-tuebingen.de')
>> % Request 8GB memory/core
>> ClusterInfo.setMemUsage('8G')
>> % Request 4 cores per node
>> ClusterInfo.setProcsPerNode(4)
>> % Run on the tue-short node
>> ClusterInfo.setQueueName('tue-short')
>> % Request 20 minutes of wall time
>> ClusterInfo.setWallTime('00:20:00')
>>
>> % 8 workers for 32 sims
>> j = c.batch(@parallel_example, 1, {32}, 'pool', 8);
additionalSubmitArgs =
-l walltime=00:20:00 -l mem=8G -q tue-short -l nodes=3:ppn=4 -M your-email@
fx >>
```

Any parameters set with `ClusterInfo` will be persistent both between jobs and MATLAB sessions. To see the values of the current configuration options, call the `state` method. To clear a value, assign the property the appropriate empty value (`''`, `[]`, or `false`).



The image shows a screenshot of the MATLAB R2014b Command Window. The Command Window displays the output of the `ClusterInfo.state` command. The output lists various configuration parameters and their values. At the bottom of the Command Window, the command `ClusterInfo.setEmailAddresses('')` is entered.

```
>> ClusterInfo.state

    Arch :
    ClusterHost :
    DataParallelism :
    DiskSpace :
    EmailAddress : your-email@uc1.scc.kit.edu
    GpusPerNode :
    MemUsage : 8G
    PrivateKeyFile :
    PrivateKeyFileHasPassPhrase : 0
    ProcsPerNode :
    ProjectName :
    QueueName : fat
    RequireExclusiveNode : 0
    Reservation :
    SshPort :
    UseGpu : 0
    UserDefinedOptions :
    UserNameOnCluster :
    WallTime : 00:20:00

fx >> ClusterInfo.setEmailAddresses('')
```

TO LEARN MORE

To learn more about the MATLAB Parallel Computing Toolbox, check out these resources:

- [Parallel Computing Coding Examples](#)
- [Parallel Computing Documentation](#)
- [Parallel Computing Overview](#)
- [Parallel Computing Tutorials](#)
- [Parallel Computing Videos](#)
- [Parallel Computing Webinars](#)