SPM Batch Toolbox

I. Overview

The batch toolbox was developed to make easier SPM5 batch jobs creation and submission without the need of the SPM5 Graphical User Interface. In addition, a set of functions were also developed to allow remote job submission in the CC-IN2P3 farm.

The batch toolbox allows most of SPM5 tasks (temporal, spatial, statistic and utility[*]) to be run in a flexible way. Jobs created by the batch toolbox have the same structure as SPM5 jobs and therefore can be loaded for inspection/modification from within the SPM5 GUI.

The location of functional/anatomical images and the parametrization of selected tasks are defined by the user inside two Matlab .m scripts: *data.m* and *params.m*, respectively (see **Creating *data.m* script** and **Creating *params.m* script** below). After *data.m* and *params.m* creation, the *batch_launcher* function can be called to automatically create and submit the jobs (see **Launching a job** below).

Examples of *data.m* and *params.m* scripts are available in the ./examples folder. These scripts are supposed to be used as starting points for new batches creation. Eventually, a third .m script may also be created to facilitate managing and launching several subjects/sessions' data simultaneously (see ./examples/launcher_example.m).

The same *data.m* and *params.m* scripts can be used either for local or remote job submission without any additional modification. The piece of information that specifies if a job will be submitted locally or remotely is a string variable passed as input argument to the *batch_launcher* function (see **Lauching a job** below). The remote job submission functions were specifically written for the CC-IN2P3 facilities. However, they can be easily adapted for any other distributed analysis system.

II. Package content

./batch_launcher.m	– function called by the user for launching batch jobs;
./batch_builder.m	– builds/creates batch jobs;
./batch_remote.m	– remote job submission manager;
./batch_remote_shell.m	– creates shell scripts parsed by BQS (Batch Queuing System);
./batch_remote_node.m	– remote node specific routines.
./change_spm_path.m	– change SPM.mat images' path.
./README.pdf	– this file.
./examples	– example files' folder;
./examples/launcher_example.m	– script for assembling several *data.m* and *param.m* files and launching jobs easier;
./examples/params_example.m	– *params.m* script;
./examples/data_example_s*X*.m	– data.*m* script for a single subject/session *X*;
./examples/data_example_all.m	– data.*m* script for a group of subjects/sessions.

[*] At the moment, only the 'image calculator' utility task is available for local submission. Utility tasks are not available for remote submission.

III.Launching a job

1) Run Matlab;

2) Add the batch toolbox to your Matlab path, if it doesn't exist;

3) Create my_*data.m* and my_*param.m* scripts (see **Creating *data.m* script** and **Creating *params.m* script** below);

4) Go to my_*data.m* directory and run *my_data* (*data* structure will be loaded into matlab working space):
   >> my_data

5) Go to *my_param.m* directory and run *my_param* (*param* structure will be loaded into matlab working space):
   >> my_param

6) Run *batch_launcher*

   >> batch_launcher(data, param, *runmode*)

   runmode is one of the following string constants:

• ' local_submit'	- create the job structure and submit it locally;
• ' local_build'	- create the job structure only. The created job structure can be inspected from within the SPM5 GUI, before submission. You can use this mode for testing the example files;
• ' remote_submit'	- create the job structure and submit it remotely;
• ' remote_build'	- create the job structure only. The whole directory tree, which would be created remotely at each node, will be created locally for inspection.

   NOTE 1: A more efficient way of launching batch jobs is to create a new Matlab script (.m) that automatically executes most of the steps described above (see ./examples/launcher_example.m).

   NOTE 2: By default, the jobs structure will be saved in a .mat file named *bb_yyyy-mm-dd-time*. Use SPM5:Tasks->Batch->Load GUI to inspect if the jobs structure have been created correctly.

IV.Creating *data.m* script

The *data.m* script will hold the path name of every functional and anatomical images, as well as the path name of every statistics related file (e.g. SPM.mat, log files, etc), required to run the selected SPM5 pre-processing (temporal and spatial), stats and util tasks. Note that either full path or relative path names are accepted.

The *data.m* script will contain a single data structure named *data* with three main fields: *preproc*, *model and util.* The *preproc* field will hold the path to every image required by the pre-processing taks. The *model* field will hold the path to every image and file required by the stats tasks. And the *util* field will hold the path to every image and file required by the util tasks. See ./examples/data_example_all.m and ./examples/data_example_s*n*.m for a full description of the *data* structure fields.

To create your own *data.m s*cript, you should edit one of the *data_example_*.m* files available in ./examples and change it according to your data path.

V. Creating *param.m* script

The *param.m* script will hold the values of every parameter required to run the selected SPM5 pre-processing (temporal and spatial), stats and util tasks. If a parameter is not defined in *param.m*, the default value defined in SPM5 *spm_defaults.m* will be used.

The *param.m* script will contain some general purpose fields (e.g. *nslices, TR, slice_order*) and three main fields named *preproc, model* and *util.* The *preproc* field will hold values required only by the pre-processing tasks. The *model* field will hold values required only by the stats tasks. And the util field will hold values required only by the util tasks. See ./examples/param.m for a full description of the *param* structure fields.

To create your own *param.m s*cript, you should edit the *param_example.m* file available in ./examples and change the values according to the analysis you want to conduct. Usually, the same analysis is conducted on every subject/session, so a single *param.m* script may be enough for each experiment.

Three specially important fields in the *param* structure are the *param.preproc.order,* the *param.model.order* and the *param.util.order* fields. You will define in these fields the pre-processing, statistics and utility tasks that you want to run, respectively. The tasks will be run sequentially in the order they have been specified. For instance, if you want to run a slice timing correction, a realignment (estimate & write) and a spatial smoothing, in this order, you should enter 'param.preproc.order = {'slice_timing' 'realign_estimate&reslice' 'smooth'};'. Note that the pre-processing tasks always precedes the statistics tasks, and the statistics tasks always precedes the utilities tasks.

Even though specific pre-processing, statistics and util tasks parameters may have been defined in the *param* structure previously, only the tasks specified in the *param.preproc.order, param.model.order* and *param.util.order f*ields will be really launched. If you don't want to run any pre-processing task, just leave the *param.preproc.order* field empty (i.e. ' param.preproc.order = {};'). Similarly, if you don't want to run any statistics task, just leave the *param.model.order* field empty (i.e. ' param.model.order = {};'). Or, if you don't want to run any utility task, just leave the *param.util.order* field empty (i.e. ' param.util.order = {};').

VI.Acknowledgments

Sylvain Maurin, Philippe Domenech and Anne Cheylus for the initial scripts and the parameters for remote job submission in CC-IN2P3.

Written by Mateus Joffily (joffily@isc.cnrs.fr) – CNC/CNRS – August/2007