Running the BOUT++ SOL1D model using the GUI (27th July 15):

Installation:

This is an area that needs work. Currently all the files required to run the simulations are contained within the simulations folder on GitHub at https://github.com/joe1510/GUI-for-SOL1D/tree/master/simulation. Download these as well as the config.ini file to the same folder.

Change the file paths in the config file to an archive in which folders for simulations can be kept and the exe being where the SOL1D simulation files are kept.

```
[exe]
path = /hwdisks/home/jh1479/BOUT-dev/examples/bout-sol1d/sol1d

[archive]
path = /hwdisks/home/jh1479/python/Archive/
```

Currently file paths in runbout, scanbout and the main file also need changing too, this will hopefully soon no longer be the case, in runbout and scanbout needs file path of the config file and in the mainfile the file paths of the runbout and scanbout code.

It should be noted that an empty defaults folder is required in the archive and a config folder containing three files: BOUT.inp (the control file for simulations) and also blank .ini files called 'usernotes.ini' and 'record.ini'. The creation of all these files will be made automatic in the near future.

Running the GUI:

The Load Tab:

Load useforruns.py using ipython or otherwise. This will open up the graphical interface, an example is shown below:

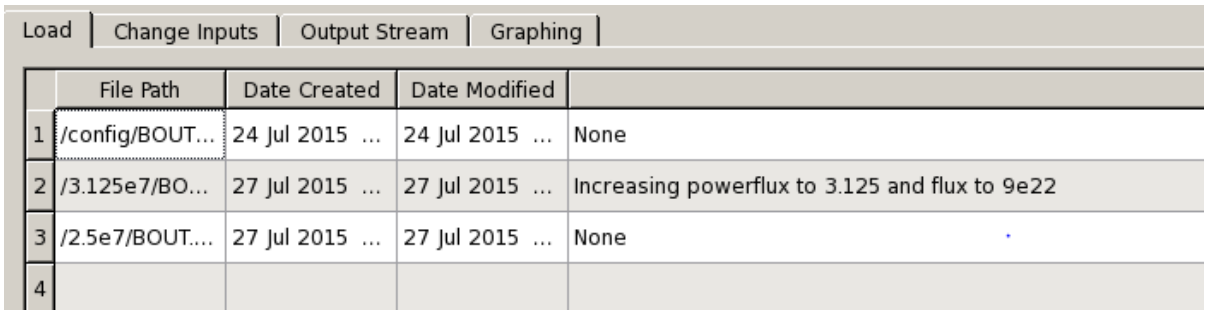

Figure 1 Example load page

This lists all of the BOUT.inp control files contained anywhere within the specified Archive along with some corresponding information such as user added comments. Double click to load a file or click the load button. The table can be sorted alphabetically by each category.

Loading a file automatically takes you to the change inputs tab.

The Change Inputs Tab:

Here the values for all the inputs are shown consistent with the values contained within the BOUT.inp control file that was loaded. These values can then be changed as the user requires. Below is an example screen:

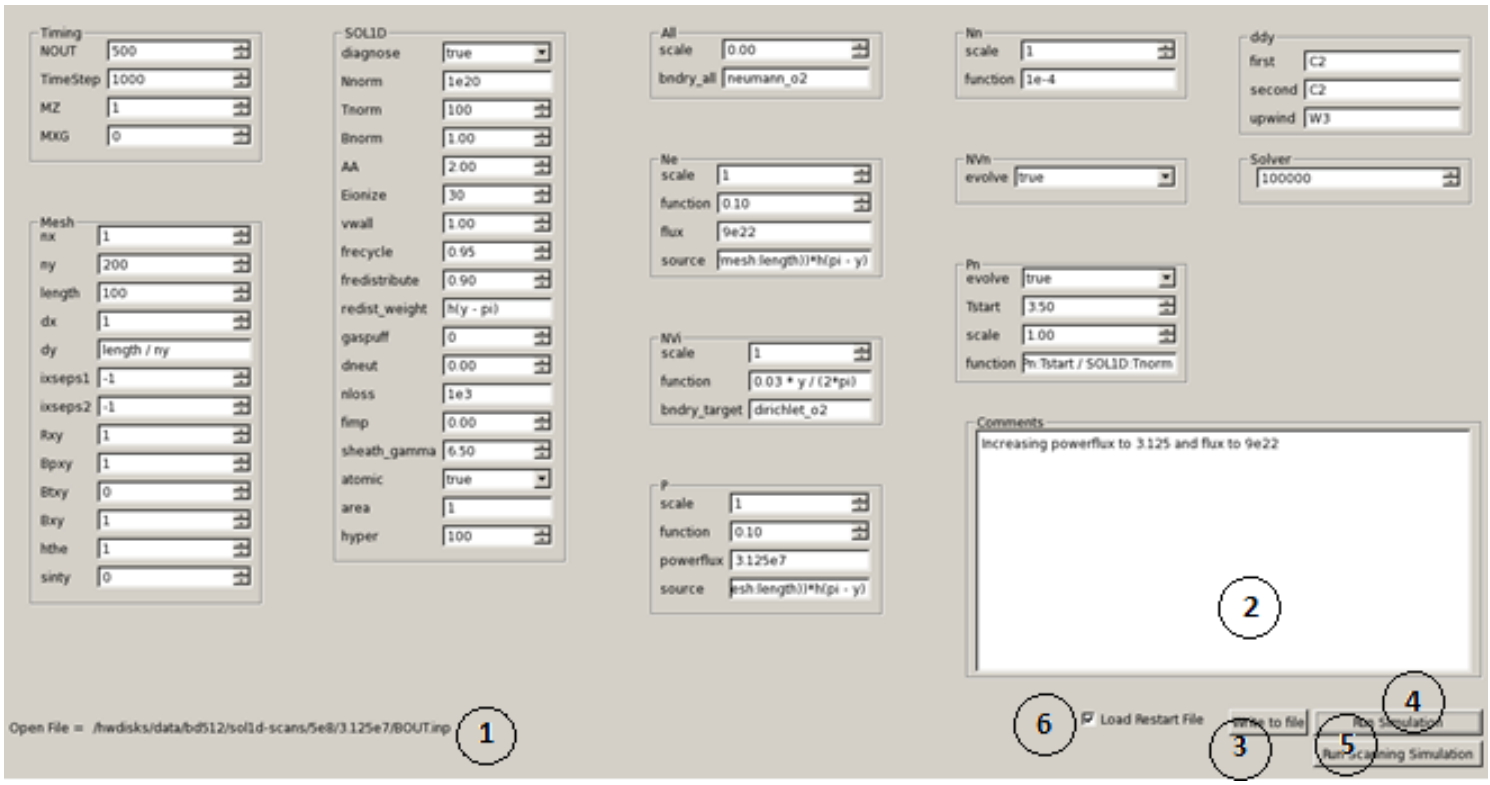

Figure 2 Example change inputs page

1. This shows the loaded config file that is currently being edited.

2. A box in which comments about what changes have been made can be added as the user wishes.

3. ‘Write to File’ – this simply saves any changes and does nothing else.

4. ‘Run simulation’ – this will load a dialog box requesting a folder path where all the data from the simulation will be stored. The automatic value for this is the same as the path that the data was loaded from so overwriting the old. This is useful for continuing previous data when steady state is trying to be reached but care should be taken if variables are changed as the user generally won’t want to overwrite the data in this case, then copies the config to a temporary folder to run the SOL1D simulation from.

5. ‘Run scanning simulation’ – similar to the above except the user can run more than one run automatically by giving a start and end value for a parameter and the increment that it is increased each time. Care should be taken here to ensure enough time steps are used to ensure steady state is reached as there are currently no inbuilt checks.

6. If the restart box is checked then the simulation will run from the previous timestep provided that restart files exist in the folder in which the config file was loaded from.

File information:

It is possible to view the file history of any loaded config file by using file -> history. This will load a list of folders and times when the config file existed in those folders so you know the history of the simulation.

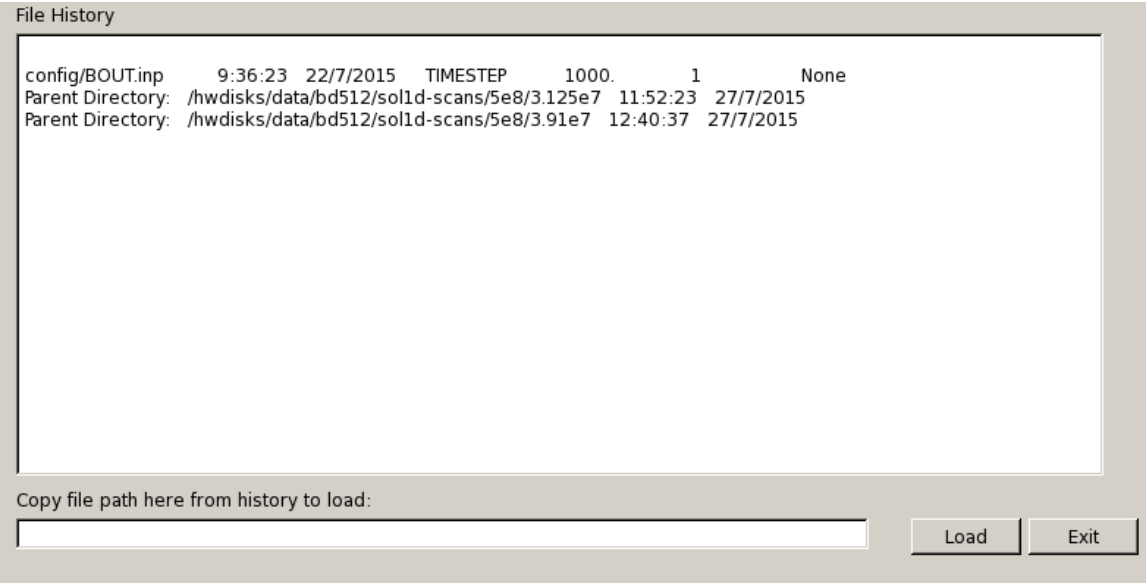

Figure 3 File History example

It is then possible to copy the file path from the parent directory as listed into the text line and click load to load and view/ rerun the old data.

Another useful feature is the compare function. Clicking file -> compare brings the user back to the initial load tab, with all the same files in it but this time selecting a file brings up a dialog box displaying all the differences between the selected file and the previously loaded files. So if NOUT is reduced to 1500 from 1700 and flux is increased to 1.125e23 we get the following list of changes.

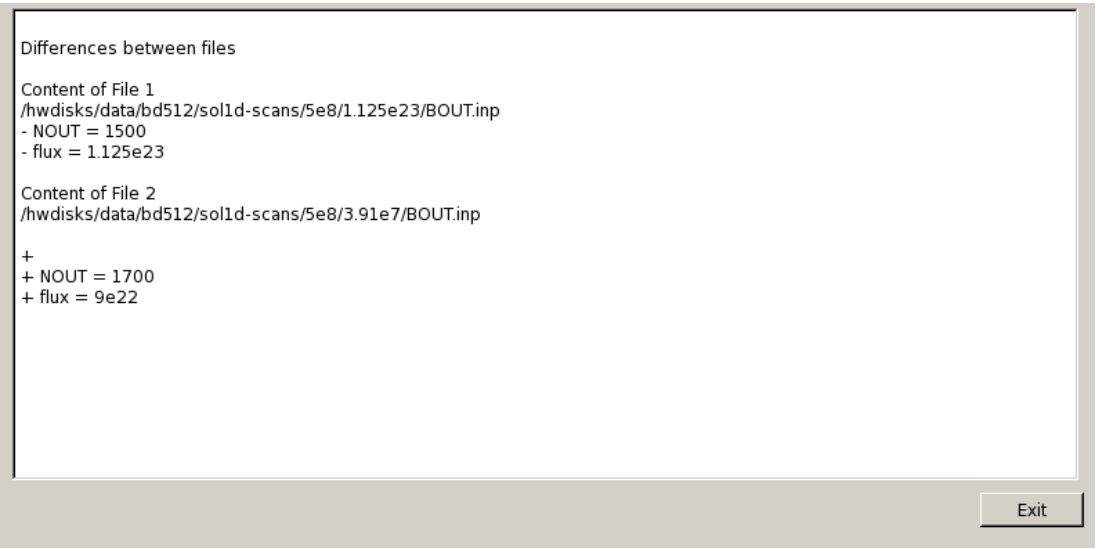

Figure 4 Compare example

Note: the compare will take notice of all little changes to the file regardless of whether they affect the simulation so the additional + in the contents of file 2 can be ignored as addition of white space doesn't effect SOL1D.

The Output Stream Tab:

While a simulation is running this text viewer displays the output from the BOUT simulation. This is identical information to what would have been printed to the screen when running simulations on the console. This information is for interest only to give an idea of how the simulation is progressing generally. It cannot be edited. It is not necessary to keep the output stream tab open while simulations are being run. Other files may be loaded in the background and data analysis of other data may also be undertaken however should the GUI as a whole close then the simulation will crash and the data in the temporary file will not be saved properly.

The stop simulation button will at some point be used to act in the same way as ctrl + c does whilst a simulation is run in the command line, however, it does not currently work.

The Graphing Tab:

Once a simulation has finished it will default to shifting to this graphing tab to enable the user to start analysing the data. It is important to note however that any data set can be loaded and analysed at any time using the load tab. Before any data is analysed it has to be collected using the automatic collect button. An example is shown below in figure 7.

1. This is the important collect data button. It must be clicked for the data files that are currently loaded to be analysed. So after a simulation if it is clicked it will 'collect' the data that has just been created from the .dmp files in that folder. If a different folder is loaded in the load tab then the collect path will equal the load path as displayed in the change inputs tab. Collect by default collects all possible variables in the SOL1D model unlike in the command line where each variable has to collected individually.
2. These are user inputs for creating graphs using the graphical side of the GUI. If all of one variable is to be used then the checkbox under that box should be clicked. Here is an example comparing how to plot data in the GUI compared to in the command line.

Figure 6 Example of how to input variables for a graph

   This is equivalent to writing plotdata(ne[:,0,-1,0]).
3. Once the correct variables are selected click 'create graph' to plot a graph in the plotting area to the right.
4. The 'value at point' button and 'divide by' are debatably a little redundant however they can be a quick way to find the value of the data at a specific point. Note that the 'all' boxes are ignored so figure 6 would give the same output as typing ne[0,0,-1,0] into the command line.

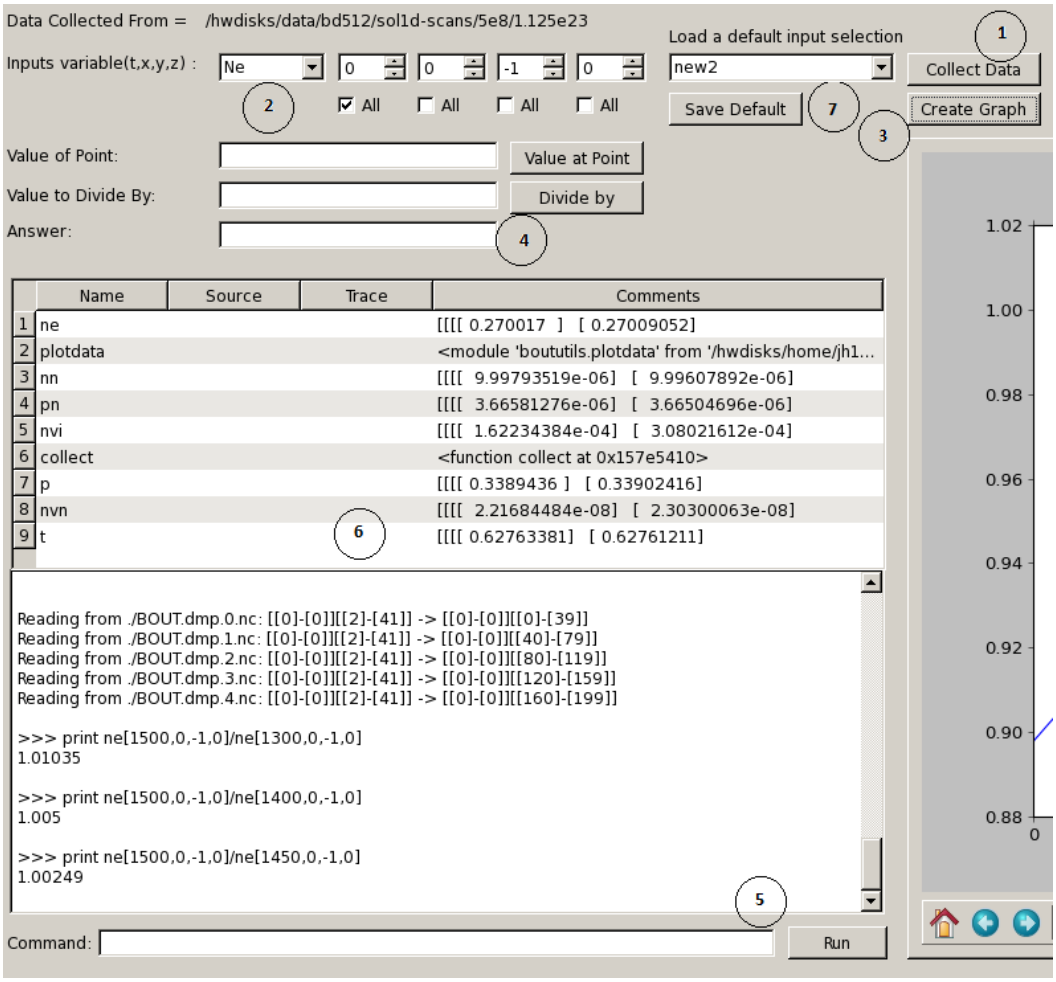

Figure 7 Example of Graphing Tab

5. These are default input selections. A default is created by selecting a variable setup that is commonly used and then clicking ‘save default’. A prompt to name the default then appears. Defaults are loaded by being selected from the dropdown box. Defaults help make it really quick to view some common graphs so are useful to help find steady state quickly.

6. The graphical input has its uses for quick checks of certain graphs but when analysis becomes more involved and complex it is easier to use this command line like shell. For example it is not possible in the GUI to plot ne[10:73, 0, -1, 0] as the input is either all points or a single point. For these cases the command line is the obvious choice. This command line exhibits all the usual properties of typing into ipython as running simulations in the command line. Modules can be imported, variables assigned etc etc. There is no need to worry about importing collect and plotdata however as this is taken care of by the collect data button. It is possible to hardwire some shorter commands into the command line. Currently only ‘plot’ is used such that: plot(ne[:,0,-1,0]) == plotdata(ne[:,0,-1,0]). If the user wants to add their own commands they can find the function ‘runCommand’ in the ‘useforruns.py’ file and write the shortcut their as has been done for plot.

```
def runCommand(self, cmd):
    # Output the command
    self.write(">>> " + cmd)

    glob = globals()
    glob['plot'] = self.DataPlot.plotdata
    # Evaluate the command, catching any exceptions
    # Local scope is set to self.data to allow access to user data
    self.runSandboxed(self._runExec, args=(cmd, glob, self.data))
    self.updateDataTable()
```

Please not that to get an output to show please use print. So '**print** ne[0,0,-1,0]'.

7. The table here shows the current memory of the python command line i.e. which variables

The Graphing Area:

This area to the right command line is where graphs are displayed. They are standard matplotlib graphs so have all the usual attributes such as zoom and save and should be of publishable standard, see matplotlib documentation for more information. Clicking on the green tick it is possible to edit some of the settings of the graph area, and for 1D graphs to add titles and x labels. If a 2D graph is input then it will be plotted with a colour bar to help indicate what each colour means. 3D plots are not possible. This all uses the same plotdata code as was in place for previous plotting in the console through ipython.

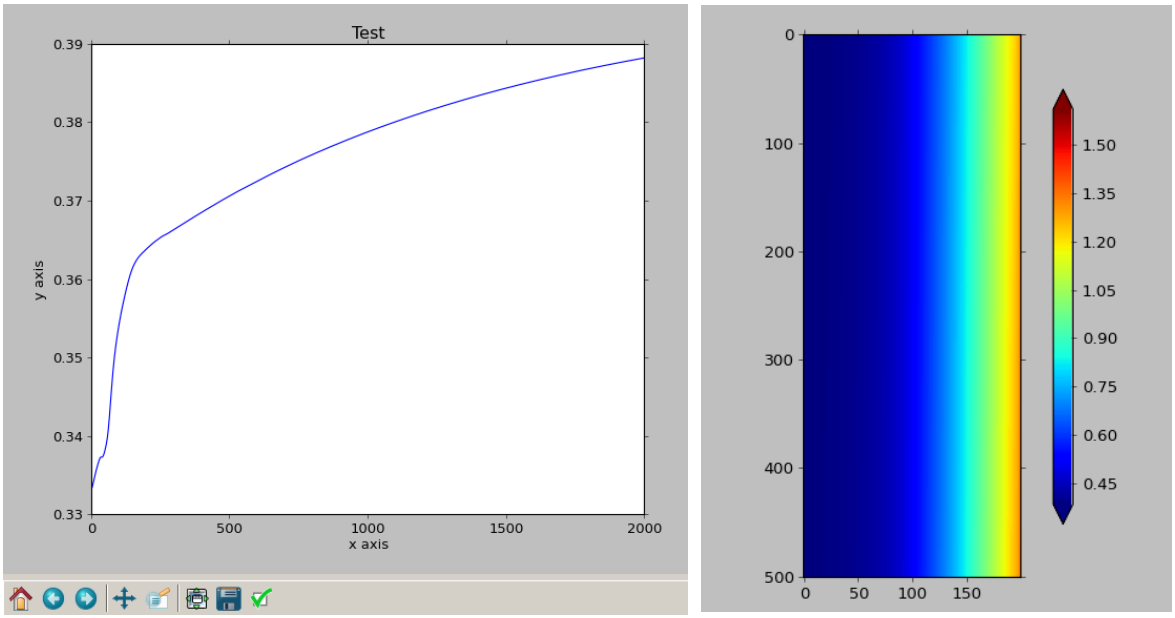

Figure 5 Example of 1d and 2d plot