Automatic fine-grid determination for CASTEP OTF pseudopotentials

UKCP Summer Project

Author:
Alexander Liptk

Supervisor:
Prof. Keith Refson

April 27, 2020

Abstract:

This document summarises the objectives and results of the 2019 UKCP Summer Project carried out at Royal Holloway University of London, and also acts as a documentation and user guide to the software that was developed. The project involved the development of a Python script to automate the creation, submission and analysis of CASTEP jobs in order to determine the optimal `FINE_GRID_SIZE` values for all supported elements.

Contents

I. MOTIVATION

CASTEP is a leading academic and commercial software package which uses density functional theory to calculate the electronic properties of materials from first principles. One of CASTEP's unique selling points is its ease-of-use for scientists who are not experts in electronic structure theory or numerical methods, that is mainly achieved by intelligent automatic selection of default values for many of the convergence parameters as well as allowing for the selection of precision of most aspects of a calculation from a short list of discrete levels (from `COARSE` to `EXTREME`).

Despite this, there is no such intelligent selection available for setting the density of the fine FFT grid used for the on-the-fly (OTF) generation of pseudopotentials, and CASTEP currently defaults to a fine grid size identical to the standard grid used for calculating the Kohn-Sham orbitals. However, this default setting is inadequate for the default OTF pseudopotentials describing a majority of the elements in the periodic table, which require a much finer grid to represent an atom's pseudized core and ultrasoft augmentation charge densities. Additionally, the fine grid density is one of the most difficult parameters to test for convergence due to its strong correlation with plane-wave cutoff, and as a result, many CASTEP calculations are either under-converged (compromising accuracy) or over-converged (wasting CPU time).

This project aimed to remedy this shortcoming by developing and testing scripts for determining a set of `FINE_GMAX` values corresponding to `FINE_GRID_SIZE` precision levels from COARSE to EXTREME for a number of supported elements. This database of FINE_GMAX values would be generated for the C19, QC5 and NCP19 pseudopotential libraries and will be implemented into CASTEP using a lookup table.

The code that has been developed is described in detail in the following section, including a discussion of any prerequisites, troubleshooting, customisation and frequent use-cases.

II. TESTGRIDS.PY

`testgrids.py` is a Python 3 script for investigating the behaviour of CASTEP simulations with a varying FINE_GRID_SIZE parameter for all supported elements. The script takes care of generating test data, submitting it to the SLURM Workload Manager (SLURM), monitoring the SLURM queue, extracting and plotting the relevant data, as well as saving all relevant data to both machine- and human-readable files.

When running the script with no arguments (`python testgrids.py`), the default values for all possible arguments are assumed. The execution begins by checking if a temporary data folder from a previous run with the same name exists; this is `tmp` by default, but can be changed with the `--name` argument. If this folder exists, it is emptied to prepare it for the data that will be generated during this run. Additionally, a script that deletes any unused semaphores is executed prior to any work being submitted to SLURM.

Next, the job data necessary for this run is generated. By default, a total of 4473 folders is created: 9 different fine grid size values between 2.0 and 6.0, each with 7 different volumes between 94% and 106% (as specified by the Δ-project [1]), for all 71 supported elements that will be simulated. Each folder, named with the corresponding element symbol and a numerical suffix, will contain one `.cell` file that details the geometry and parameters of the cell being simulated, and one `.param` file that will contain all of the control parameters required to run a CASTEP job. The parent data folder will also contains one `.job` file per element, which contains the necessary information for SLURM to treat all jobs corresponding to that element as a single job array and submit it to the queue.

Following the preparation of job data, the script then submits each element as a job array to the SLURM queue, 10 elements at a time by default as to not pollute the queue with too many jobs. At this stage, the progress of job execution is monitored by polling the SLURM queue (every second by default) and reported to the user.

After the completion of all job arrays, the useful data generated by CASTEP needs to be collected and analysed; the reported cell volume and stress, final converged energy, number of iterations that were required for its convergence, total simulation time, fine grid dimensions and real lattice vectors are collected from the new `.castep` files and stored in an array. Any warnings or errors during this stage are also reported. The collected volumes and energies are sent to a Birch-Murnaghan equation-of-state (EOS) fitting function that returns the reference volume, bulk modulus and its derivative w.r.t. pressure. On occasions that a good fit cannot be found for any group of volumes, the user is warned and any data for that group is discarded. For all successful fits, Δ-values [1] are calculated from the EOS data by reference to the EOS data for the largest tested fine grid size of that element. At the end of this stage of execution, all collected values are saved in a human-readable file `tmp.table.txt`, all Δ-values are

summarised in a machine-readable file `tmp.deltas.txt` and the array of collected values is also pickled for further use in `tmp.pickle`.

Finally, a 2 by 2 grid of plots is drawn for every element and either shown on screen or saved to a `.png` file. The plots show the coarse and fine view of the relationship between the calculated Δ-values and the `FINE_GRID_SIZE` or corresponding `FINE_GMAX` parameters. When this is completed, the semaphore cleaning script that was used at the beginning of the execution of this script is sent to SLURM again in order to clean up after any potential failures or crashes. If requested (no by default), the `tmp` data directory is also deleted, as all useful data has already been saved elsewhere.

```
*****************************************************
*              == testgrids.py ==                   *
*****************************************************
* Job name:                                    tmp *
* Element(s):                                  ALL *
* Fine grid [start, stop, step]:   [2.0, 6.0, 9] *
* Type:                  Ground state structure *
* Debug level:                                   0 *
*****************************************************
* PARAM settings:                                  *
*     BASIS_PRECISION:                        FINE *
*     CALCULATE_STRESS:                      False *
*     ELEC_ENERGY_TOL:                   1e-05 eV *
*     GRID_SCALE:                             1.75 *
*     ILEVEL:                                    2 *
*     MAX_SCF_CYCLES:                          100 *
*     MIXING_SCHEME:                         PULAY *
*     MIX_CHARGE_AMP:                          0.5 *
*     MIX_SPIN_AMP:                            0.5 *
*     PERC_EXTRA_BANDS:                      100.0 *
*     XC_FUNCTIONAL:                           PBE *
*****************************************************
* CELL settings:                                   *
*     K-point spacing:                         0.2 *
*     Pseudopotential library:                 C19 *
*     Volume scale:                            1.0 *
*****************************************************
* SLURM settings:                                  *
*     Nodes per job:                             1 *
*     Poll time:                               1 s *
*     Queue size:                         10 jobs *
*     Reservation string:                     None *
*     Tasks per node:                           20 *
*****************************************************
```

Figure 1: Summary of all the customisable parameters and their values during a default run. A summary card like this is printed at the beginning of every run, and should be especially useful if the execution output is being logged.

A. Installation and Requirements

The `testgrids.py` script does not require any installation or compilation, however, there are a number of mandatory (and optional) packages that are required for the script to work. The list below summarised these requirements, with all quoted version numbers representing the versions that were used during the development and testing of the code. The usage of "`X.X` or higher" here assumes that no changes to the packages will be introduced in the future that will break compatibility with the current requirements. Similarly, it is likely that any package with the same major version will also work, therefore the version numbers in following list can be interpreted as a guidance more than an absolute requirement:

- [Mandatory] `Environment Modules 3.2.10` or higher [2]

- [Mandatory] `CASTEP 18.1.2` or higher [3]
- [Mandatory] `Python 3.6.4` or higher [4]
- [Mandatory] `NumPy 1.14.0` or higher [5]
- [Mandatory] `Matplotlib 2.1.2` or higher [6]
- [Mandatory] any of the following Matplotlib non-interactive backends: `agg`, `cairo`, `gdk`, `pdf`, `pgf`, `ps`, `svg`
- [Optional] any of the following Matplotlib interactive backends: `GTK`, `GTKAgg`, `GTKCairo`, `MacOSX`, `Qt4Agg`, `Qt5Agg`, `TkAgg`, `WX`, `WXAgg`, `GTK3Cairo`, `GTK3Agg`, `WebAgg`, `nbAgg`

It should be noted that with proper usage of the Environment Modules package, some of the requirements will have to be loaded using the `module load` statement before they can be used. On the system where the script was tested, only Python and Matplotlib needed to be loaded, as the script itself takes care of loading CASTEP, and NumPy as well as the required Matplotlib backends were available with Python. If Matplotlib is available as a module separate from Python, as was the case with this system, loading the Matplotlib module will also load the required Python module as a dependency, and therefore the loading of all required modules can be accomplished with a single statement. It should be also be noted that, if the user encounters a syntax error during the execution of the script, this means that Python 2 has been loaded instead of the required Python 3.

Below is also a directory tree showing all the auxiliary files that are included with `testgrids.py` along with a short description of their purpose. In order for the script to be able to access these files, its working directory has to be set to its root folder. Files and directories marked with `*` are obtained from the Δ calculation package directly (without any modifications) [1], while those marked with `**` are also obtained from the same package but have been modified for use with this script.

- `testgrids/` (root directory)
  - `* CIFs/` (directory with CIF files of 71 elemental crystals in their ground state structure)
    - `* Ag.cif`
    - `...`
    - `* Zr.cif`
  - `* primCIFs/` (directory with CIF files of 71 elemental crystals in a primitive unit cell)
    - `* Ag.cif`
    - `...`
    - `* Zr.cif`
  - `* calcDelta.py` (Python script that calculates the Δ-factor between two data sets)
  - `** eosfit.py` (Python script that fits a single set of E(V) data to the Birch-Murnaghan EOS)
  - `ipcclean` (shell script that cleans up unused semaphores from previous SLRUM jobs)
  - `template.job` (template shell script for submitting job arrays to SLURM)
  - `testgrids.py` (main Python script)

B. Usage examples

This section provides examples of common use case scenarios of `testgrids.py`, ranging from the most basic (that require only passing a few arguments) to fairly advanced (that may require some editing of code). A full list of all supported arguments, their accepted values and any other relevant information about them is given in a table at the end of this section. Depending on the argument, some text and values have been obtained from the CASTEP [7] or SLURM [8] documentation pages.

QUESTION:	I want to do a default run and save the results with the name "default". I don't care about any temporary files, these can be deleted.
SOLUTION:	`python testgrids.py --name default --clean true`
CONCISE:	`python testgrids.py -na default -cl y`
NOTE:	It is always a good idea to give your run a name, as this is the name that will also be given to all relevant files and plots.

QUESTION:	I want to do a default run but only for silicon. I also want to open the generated plot in interactive view.
SOLUTION:	`python testgrids.py --element Si --save false`
CONCISE:	`python testgrids.py -el Si -sa n`
NOTE:	Using `--save false` is not recommended for when a run involves more than a few elements. Interactive plotting windows are opened sequentially (next window opens after the previous one is closed). For all 71 elements, this would represent 71 separate plots.

QUESTION:	I want to do a default run but over a higher range (1.5 to 6.5) and with a larger density (0.1) of `FINE_GRID_SIZE` values.
SOLUTION:	`python testgrids.py --fine_grid_min 1.5 --fine_grid_max 6.5 ...` `... --fine_grid_step 51 --name long_run`
CONCISE:	`python testgrids.py -f0 1.5 -f1 6.5 -fs 51 -na long_run`
NOTE:	A run with 51 `FINE_GRID_SIZE` steps and the default number of elements will require $51 * 7 * 71 = 25,347$ total jobs, which will take a significant time to complete. When starting a long run, it is recommended to verify that the run behaves and gives results as expected, usually by doing a preliminary run with a single element or a smaller number of steps.

QUESTION:	I have a lot of nodes with 64 cores available to me under the reservation "El_Psy_Kongroo". As my run involves large jobs, I want each job to run on 2 nodes.
SOLUTION:	`python testgrids.py --reserve El_Psy_Kongroo --tasks 64 --nodes 2 ...` `... --queue 100`
CONCISE:	`python testgrids.py -r El_Psy_Kongroo -t 64 -no 2 -q 100`
NOTE:	The semaphore cleanup script is not available when using `--reserve`, so any necessary cleanup needs to be done before starting the run. When using `--tasks` and `--nodes`, it is important to know the number of cores available per node, as well as the performance implications of running a job across many cores and nodes. Setting a large value for `--queue` will submit an equally large number of job arrays to the SLURM queue (or all jobs waiting to be submitted, whichever is smaller), therefore it is important to be mindful of this if you aren't using reserved nodes.

QUESTION:	I am testing the behaviour of a new set of values for some parameters, and would like the script to be as verbose as possible. As I am logging the output, I do need frequent infromation about job progress in the SLURM queue.
SOLUTION:	`python testgrids.py --name test --debug 1 --poll_wait 60 >> test.log`
CONCISE:	`python testgrids.py -na test -d 1 -po 60 >> test.log`
NOTE:	Using `--debug 1` will enable debugging mode that, at level 1, only enables verbose output. At higher levels, debug mode is used to control the flow of code (this is discussed further in the next section) and should not be used unless you understand the implications of the value you have chosen. Setting `--poll_wait` to higher values increases the time between successive checks of the SLURM queue, that in turn decreases the amount of repetitive output. However, jobs that are pending to be submitted to the queue can only be submitted after the queue reports that it is empty, therefore very high values of `--poll_wait` are likely to waste time by leaving the SLURM queue empty for longer.

QUESTION:	I have completed a run previously and have the relevant `.pickle` file. I would like to open these plots in interactive view again.
SOLUTION:	`python testgrids.py --load old_run.pickle --save false`
CONCISE:	`python testgrids.py -l old_run.pickle -sa n`
NOTE:	When using `--load`, the built-in argument parser will ignore all arguments besides `--save`. This way, it is possible to either revisit old data using interactive plots, or re-plot old data and save the figures when the previous ones are no longer available or were never plotted. This is especially useful for runs with a reused `--name` argument that did not save any new figures (`testgrids.py` will not overwrite an existing directory in `./figs`). Similar functionality can be achieved by repeating your run with identical arguments and adding `--debug 4`, however, the former method is preferred.

QUESTION:	I would like to do a run involving the calculation of a stress tensor on elemental crystals in their primitive unit cell.
SOLUTION:	`python testgrids.py --primitive true --stress true`
CONCISE:	`python testgrids.py -pr y -st y`
NOTE:	Choosing to run calculations on elemental crystals in their primitive unit cell instead of the default ground state structure will allow for faster calculations, however, the results will no longer be interpretable in the same way as those quoted on the Δ-project website. Enabling calculations of the stress tensor will also increase the total calculation time. The pressure (in GPa) will be visible in the `name.table.txt` file, however, this setting should have no effect on the generated plots.

QUESTION:	I would like to tweak some arguments relating to the density mixing procedure.
SOLUTION:	`python testgrids.py --mix_scheme broyden --mix_charge_amp 0.8 ...` `... --mix_spin_amp 1.75`
CONCISE:	`python testgrids.py -mi broyden -mc 0.8 -ms 1.75`
NOTE:	When experiencing SCF convergence failures, it is often helpful to switch `--mix_scheme` between Pulay and Broyden, as well as decreasing the mixing amplitude for the charge and spin density arguments.

QUESTION:	I would like to do a run involving a metal or finite-temperature insulator. I would also like to also increase the precision of this run in all available aspects.

SOLUTION:	`python testgrids.py --extra_bands 120 --basis_prec precise ...`
	`... --elec_enrg_tol 1e-8 --grid 2.5 --kpoint_spacing 0.15 ...`
	`... --max_scf_cycles 200`

CONCISE:	`python testgrids.py -ba 120 -bp precise -ee 1e-8 -g 2.5 -k 0.15 -sc 200`
NOTE:	When dealing with metals or finite-temperature insulators, it can be useful to increase the number of available bands using `--extra_bands`. Although the default value of `--extra_bands 100` should be enough for most scenarios, no significant performance losses were measured with using higher values. It should be noted that the above combination of parameters does not necessarily correspond to a real-life scenario, and is instead used for demonstration of the available arguments. Good understanding of the advantages and drawbacks of the above parameters and their combinations is essential in obtaining meaningful results from a run. Setting `--basis_prec` to a higher level increases the precision of convergence of atomic energies with respect to the plane wave cutoff energy. Decreasing `--elec_enrg_tol` decreases the tolerance for accepting convergence of the total energy in an electronic minimisation. Increasing `--grid` increases the size of the standard grid, relative to the diameter of the cutoff sphere (it is important to be aware that changes to this argument also impacts `FINE_GRID_SIZE`, which is defined relative to `GRID_SIZE`). Choosing a smaller `--kpoint_spacing` increases the k-point density in reciprocal space, in turn increasing the smoothness of the ground state energy well. Increasing `--max_scf_cycles` increases the maximum number of SCF cycles performed in an electronic minimisation, therefore reducing the likelihood of failed convergence (although frequent SCF convergence failures often point to a different problem). A well thought-out combination of some or all of these arguments backed by a number of preliminary runs is often required. It should also be obvious that tweaking most of these parameters will have a noticeable impact on the performance of a run.

QUESTION:	I would like to use the LDA exchange-correlation functional for this run. Since the LDA description tends to overbind atoms and underestimate cell volume, I would like to apply a +3% cell volume correction before Δ calculations take place.
SOLUTION:	`python testgrids.py --xc_functional LDA --volume_scale 1.03`
CONCISE:	`python testgrids.py -x LDA -v 1.03`
NOTE:	Cell volume scaling is applied before any Δ-project relevant calculations take place; the `--volume_scale` argument can therefore be useful if you encounter a lot of fitting errors from the Birch-Murnaghan EOS fitting function. The `--xc_functional` argument currently supports a large number of functionals, including all the functionals documented in CASTEP 18.1. If you know that the CASTEP version you are using includes support for a functional that is not supported by `testgrids.py`, you can add it yourself: open `testgrids.py` in your preferred editor, scroll down to the `parse_args(...)` function, look for a statement with `args.xc_functional` and a list of functionals, and add your functional to the list. If a functional that is not currently supported becomes popular, I will endeavour to update the list.

QUESTION:	I would like to use a different version of CASTEP for this run, and because I'm doing some testing, I would like to increase the verbosity of the generated `.castep` files.
SOLUTION:	`python testgrids.py --castep CASTEP/19.11 --ilevel 3`
CONCISE:	`python testgrids.py -ca CASTEP/19.11 -i 3`
NOTE:	Although CASTEP's `IPRINT` argument accepts values of 1, 2 and 3, `--ilevel` does not support level 1 as this option does not produce a verbose enough output in the `.castep` files for scraping the required information. Level 2 is the default, and it can be increased up to level 3 for additional verbosity. If a different CASTEP version is to be used for all runs, it is fairly simple to edit the default CASTEP module used by `testgrids.py` to avoid having to use the `--castep` argument with every run: open `testgrids.py` in your preferred editor, scroll down to the `parse_args(...)` function, look for a statement with `args.castep = "CASTEP/19.1.1-foss-2018a"` and edit the string to match the name of your CASTEP module.

QUESTION:	I have a directory containing `.cif` files which I would like to use for my run.
SOLUTION:	`python testgrids.py --cif_dir /path/to/dir`
CONCISE:	`python testgrids.py -cf /path/to/dir`
NOTE:	When `--cif_dir` is specified, the `--primitive` argument is ignored. This argument is especially useful for doing a run involving a subset of elements from `.cif` files already provided with `testgrids.py`: simply copy the elements of your choice into a new directory, and supply the path to that directory (either relative or absolute) to the `--cif_dir` argument. When using custom `.cif` files, it is essential that the files contain enough information for CASTEP's `cif2cell` script to generate a `.cell` file containing `BLOCK LATTICE_CART`, `BLOCK POSITIONS_FRAC`, `BLOCK SPECIES_POT` and `BLOCK SYMMETRY_OPS`.

QUESTION:	I would like to use specific pseudopotential library for my run.
SOLUTION:	`python testgrids.py --pseudo_pot NCP19`
CONCISE:	`python testgrids.py -ps NCP19`
NOTE:	The list of possible pseudopotential libraries that can be officially used with `testgrids.py` is limited to only C19, QC5 and NCP19. Although this script was not intended for use with other libraries that are available with CASTEP, adding your own pseudopotential to the list of supported ones should not cause any issues: open `testgrids.py` in your preferred editor, scroll down to the `parse_args(...)` function, look for a statement with `args.pseudo_pot` and a list of pseudopotential libraries, and add your library to the list. Furthermore, the list of libraries is not necessarily limited to libraries: you may add `.otfglib`, `.usp`, `.uspcc`, `.uspso`, `.recpot`, `.upf`, `.dat` or `.data` files (with their file extension), as well as a valid OTFG pseudopotential string to the list. If your usage of `testgrids.py` will involve a number unsupported pseudopotentials, you may simply comment out the `if` block containing `args.pseudo_pot` to suppress the check carried out by the argument parser.

Argument	Abbr.	Type	Default	Accepted values	Notes
`--extra_bands`	`-ba`	`float`	`100.0`	any ≥ 0	Controls the percentage of extra bands in addition to the number of occupied bands (necessary for metals or finite temperature insulators). Corresponds to the CASTEP `.param` file keyword `PERC_EXTRA_BANDS`.
`--basis_prec`	`-bp`	`str`	`"FINE"`	any one of `COARSE`, `MEDIUM`, `FINE`, `PRECISE` or `EXTREME`	Specifies the precision of the basis set by choosing the level of convergence of atomic energies with respect to the plane wave cutoff energy. Corresponds to the CASTEP `.param` file keyword `BASIS_PRECISION`.
`--castep`	`-ca`	`str`	`""`	any string corresponding to a CASTEP module that can be loaded	Specifies the CASTEP module that will be used during the exacution of the script. Validated by attempting to load the specified module in a clean environment. If no value provided, argument defaults to `CASTEP/19.1.1-foss-2018a`
`--clean`	`-cl`	`bool`	`False`	any string than can be interpreted as a boolean	Specifies whether the temporary data folder should be deleted at the end of the execution of the script.
`--cif_dir`	`-cf`	`str`	`""`	any string corresponding to a valid path to a directory	Specifies the directory containing the `.cif` files to be used. Validated by checking the existence of the directory. If no value provided, argument defaults to either `./CIFs` or `./primCIFs` depending on the value of the `--primitive` argument
`--debug`	`-d`	`int`	`0`	any ≥ 0	Specifies the debug level, increasing script verbosity and controlling code flow.
`--elec_enrg_tol`	`-ee`	`float`	`1e-5`	any > 0	Controls the tolerance for accepting convergence of the total energy in an electronic minimisation. Corresponds to the CASTEP `.param` file keyword `ELEC_ENERGY_TOL`.
`--element`	`-el`	`str`	`""`	any string corresponding to a symbol of an element present in the CIFs directory	Specifies which single element to simulate, validated by verifying the existence of a corresponding .cif file in the directory specified by `--cif_dir`. If no value provided, argument defaults to all elements in the CIFs folder
`--fine_grid_min`	`-f0`	`float`	`2.0`	$1 \leq$ any $\leq$ `-f1`	Specifies the lowest value of the CASTEP `.param` file keyword `FINE_GRID_SCALE`.
`--fine_grid_max`	`-f1`	`float`	`6.0`	any $\geq$ `-f0`	Specifies the highest value of the CASTEP `.param` file keyword `FINE_GRID_SCALE`.
`--fine_grid_step`	`-fs`	`int`	`9`	any ≥ 2	Specifies the number of samples between `--fine_grid_min` and `--fine_grid_max` (inclusive).
`--grid`	`-g`	`float`	`1.75`	any ≥ 1	Specifies the fixed value of the CASTEP `.param` file keyword `GRID_SCALE`.
`--ilevel`	`-i`	`int`	`2`	2 or 3	Specifies the level of verbosity of `.castep` files. Corresponds to the CASTEP `.param` file keyword `IPRINT`.
`--kpoint_spacing`	`-k`	`float`	`0.2`	any > 0	Specifies the k-point density of a Monkhorst-Pack grid (units of inverse length). Corresponds to the CASTEP `.cell` file keyword `KPOINT_MP_SPACING`.
`--load`	`-l`	`str`	`""`	Any string corresponding to a valid path to a file	Specifies the path to a `.pickle` file that will be loaded instead of running any simulations. Validated by verifying the existence of the specified file.

Table 1: Summary of optional arguments that can be passed to the `testgrids.py` (Part 1: [a-l])

Argument	Abbr.	Type	Default	Accepted values	Notes
`--mix_charge_amp`	`-mc`	`float`	`0.5`	any > 0	Determines the mixing amplitude for the charge density in the density mixing procedure. Corresponds to the CASTEP `.param` file keyword `MIX_CHARGE_AMP`.
`--mix_scheme`	`-mi`	`str`	`"PULAY"`	any one of `BROYDEN`, `KERKER`, `LINEAR` or `PULAY`	Determines which mixing scheme will be used in the density mixing procedure. Corresponds to the CASTEP `.param` file keyword `MIXING_SCHEME`.
`--mix_spin_amp`	`-ms`	`float`	`0.5`	any > 0	Determines the mixing amplitude for the spin density in the density mixing procedure. Corresponds to the CASTEP `.param` file keyword `MIX_SPIN_AMP`.
`--name`	`-na`	`str`	`"tmp"`	any string	Specifies the name of this run, corresponding to the filenames of the generated files and the temporary simulation data directory.
`--nodes`	`-no`	`int`	`1`	any ≥ 1	Corresponds to the number of SLURM nodes that will be assigned to each job array.
`--poll_wait`	`-po`	`int`	`1`	any ≥ 1	Specifies the time in seconds to wait between polling the SLURM `squeue` (used for tracking job progress). Higher values may waste time by not scheduling new jobs in time, while lower values may slow down the machine.
`--primitive`	`-pr`	`bool`	`False`	any string that can be interpreted as a boolean	Specifies whether the primitive unit cell .cif files should be used instead of ground-state structure .cif files. Setting to `True` will reduce job time but may yield inaccurate results.
`--pseudo_pot`	`-ps`	`str`	`"C19"`	any one of `C19`, `QC5`, or `"NCP19"`	Specifies the pseudopotential library that will be used during this run. Corresponds to the CASTEP `.cell` file keyword `SPECIES_POT`.
`--queue`	`-q`	`int`	`10`	any ≥ 1	Specifies the maximum number of job arrays that will be submitted to the SLURM queue at a time.
`--reserve`	`-r`	`str`	`""`	any string	Specifies the reservation string used when submitting a job array to the SLURM queue.
`--save`	`-sa`	`bool`	`True`	any string that can be interpreted as a boolean	Specifies whether plots should be saved to `.png` files in the `./figs` directory, or drawn on screen.
`--max_scf_cycles`	`-sc`	`int`	`100`	any ≥ 1	Determines the maximum number of SCF cycles performed in an electronic minimisation. Corresponds to the CASTEP `.param` file keyword `MAX_SCF_CYCLES`.
`--stress`	`-st`	`bool`	`False`	any string that can be interpreted as a boolean	Controls whether or not a stress calculation will be performed. Setting to `True` will increase job time.
`--tasks`	`-t`	`int`	20	any ≥ 1	Specifies the number of SLURM tasks that a single job should use per node.
`--volume_scale`	`-v`	`float`	`1.0`	any > 0	Specifies the cell volume scale factor that will be applied on top of Δ project-relevant volume adjustments.

Table 2: Summary of optional arguments that can be passed to the `testgrids.py` (Part 2: [m-v])

Argument	Abbr.	Type	Default	Accepted values	Notes
`--xc_functional`	`-x`	`str`	`"PBE"`	any one of `"LDA"`, `"PW91"`, `"PBE"`, `"PBESOL"`, `"RPBE"`, `"WC"`, `"BLYP"`, `"LDA-C2"`, `"LDA-X"`, `"ZERO"`, `"HF"`, `"PBE0"`, `"B3LYP"`, `"HSE03"`, `"HSE06"`, `"EXX-X"`, `"HF-LDA"`, `"EXX"`, `"EXX-LDA"`, `"SHF"`, `"SX"`, `"SHF-LDA"`, `"SX-LDA"`, `"WDA"`, `"SEX"` or `"SEX-LDA"`	Controls which functional is used to calculate the exchange-correlation potential. Corresponds to the CASTEP `.param` file keyword `XC_FUNCTIONAL`.

Table 3: Summary of optional arguments that can be passed to the `testgrids.py` (Part 3: [x])

C. Troubleshooting

Although care has been taken to select a set of default parameters that minimise the rate of errors that can occur during the execution of the code, some errors with specific combinations of elements and corresponding parameters still remain. This section summarises all the possible ways in which code can fail; this includes non-critical errors (runtime errors in the SLURM queue, CASTEP jobs or EOS fitting function) as well as critical errors (often missing modules, permission issues, missing files, misspelled paths, or low disk space), and how to remedy them.

Error message	Level	Notes
`[WARNING] found .err file in dir ...`	NCE	This non-critical error is reported when the presence of an `.err` file is detected in any temporary job directory. The file is created by CASTEP when it encounters a runtime error such as a convergence failure or a non-variational total energy, and the contents of file should provide enough information regarding the error and how to remedy it. The presence of this error also means that a Δ value will not be calculated for that corresponding `FINE_GRID_SIZE` value of the element.
`[WARNING] job ... has encountered an error or been killed`	NCE	This non-critical error is reported when the presence the words "error" or "killed" is detected in an `.out` file. The error is reported by SLURM when a job exits with a non-zero return code (it failed in some way), or if the job has been killed (it requested more memory than is available, or exceeded the maximum allocated time); the contents of the file usually provides enough information regarding this error. The presence of this error also means that a Δ value will not be calculated for that corresponding `FINE_GRID_SIZE` value of the element.
`[WARNING] over 5 warnings detected in file ...`	W	This warning is reported when more than 5 occurrences of the word "warning" are detected in the `.castep` file. Although some warnings during the first few steps of convergence are common, a large number of warnings may point to an issue that needs investigating.
`[DEBUG] discarded as EOS fitting function returned a complex or None type (not float)`	NCE	This non-critical error is reported when the Birch-Murnaghan EOS fitting function returns a complex number or `None` as a value for at least one EOS parameter. The error is a result of an inadequate or failed fit; inspecting the `name.table.txt` file to deduce whether the relevant E-V curve has a minimum, and adjusting parameters such as `--volume_scale`, `--kpoint_spacing`, `--elec_enrg_tol` or `--basis_prec` should remedy this. The presence of this error also means that a Δ value will not be calculated for that corresponding `FINE_GRID_SIZE` value of the element.
`[DEBUG] discarded as EOS fitting function returned a negative bulk modulus derivative`	NCE	This non-critical error is simmilar to the previous one: it results from an inadequate fit, can be remedied in the same ways as the previous error, and also results in a Δ value being omitted. The difference between them is that this error is triggered when the EOS fitting function returns a negative bulk modulus derivative - an unphysical result.

Table 4: Summary of all possible non-critical errors and warnings, presented in order of the execution stage during which they can occur; [NCE] = Non-critical error, [W] = Warning

Return code	Corresponding error	Notes
00	None	Execution finished without encountering errors
01	ImportError when importing modules	Check all required modules are installed
02	arg directory does not exist or is inaccessible	Check spelling and permissions of directory
03	arg file does not exist or is inaccessible	Check spelling and permission of file
04	TypeError when parsing args	Check the required arg types in Tables 1, 2, 3
05	arg out of bounds	Check the arg accepted values in Tables 1, 2, 3
06	arg not one of possible options	Check the arg accepted values in Tables 1, 2, 3
07	CalledProcessError when loading CASTEP	Use --castep for different CASTEP module
10	OSError while generating job data	Check write permissions and remaining space
11	CalledProcessError while generating data	Verify that cif2cell is available in PATH
19	uncaught exception while generating job data	Inspect error message for more details
20	CalledProcessError while running jobs	Verify that sbatch and squeue are available
29	uncaught exception while running jobs	Inspect error message for more details
30	OSError while reading job output data	Check testgrids.py has read permissions
31	PickleError when saving array data to disk	Check write permissions and available space
39	uncaught exception while read job output data	Inspect error message for more details
40	OSError while cleaning up	Check write permissions or close opened files
49	uncaught exception while cleaning up	Inspect error message for more details
50	OSError while opening pickle file handle	Check read permissions and path spelling
51	PickleError when loading pickle data	Verify pickle file is valid and not corrupted
59	uncaught exception while loading data arrays	Inspect error message for more details
69	uncaught exception while deleting semaphores	Inspect error message for more details

Table 5: Summary of all possible critical errors. As all of these errors cease the execution of the script, the corresponding return codes are also provided.

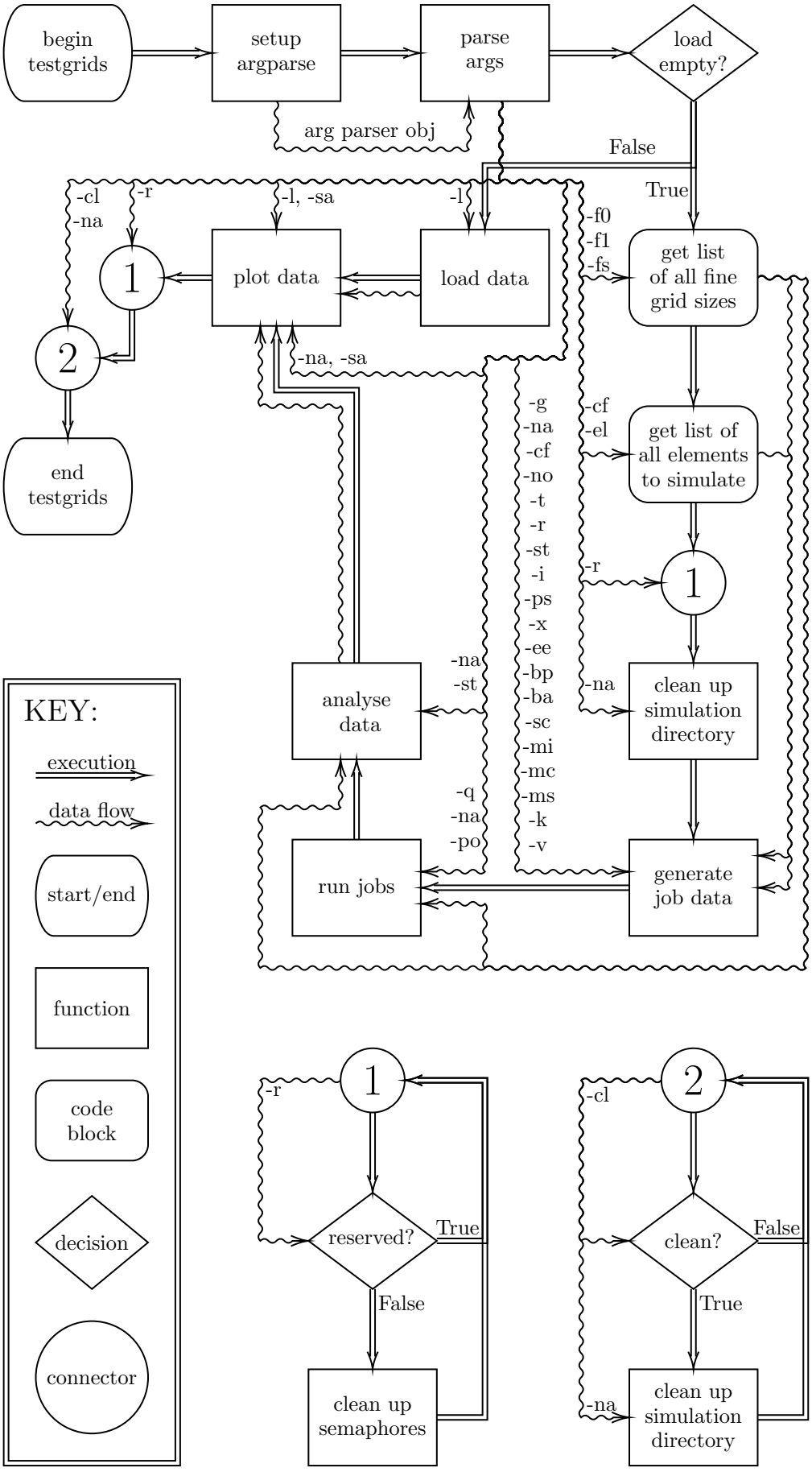
When debugging, it is possible to use the --debug flag to skip to a various execution points of the code. This is useful when an uncaught exception occurs in a part of the code; when adequate changes are made, the code execution can resume at around that point (assuming all prerequisites for code execution at that point are met). If "resuming" code execution this way, the same parameters must be used as those that were used previously, and for which runtime-related files have already been created. The table below gives a full list of allowed --debug values:

Debug level	Notes
0 (default)	Code runs from beginning to end with minimal verbosity
1	Code runs from beginning to end with increased verbosity
2	Code skips initial cleaning of semaphores and cleaning of working directory; execution starts at generation of job data
3	Code skips everything mentioned above; execution starts at sending jobs to SLURM queue
4	Code skips everything mentioned above; execution starts at analysis and plotting of data from completed simulations.
5	Code skips everything mentioned above; execution starts at final semaphore cleanup and removal of working directory (if --clean is True)

Table 6: List of allowed debug levels and their effect.

D. Overview of code

This section explores the code in more detail; the inputs, outputs and purpose of all functions is discussed in order of code execution. A flowchart summarising this information is also included below. As the signatures of all functions are described here, this section can be helpful as a reference when modifying or upgrading the code, or implementing existing functions into your own code.


begin testgrids
setup argparse
parse args
load empty?
arg parser obj
False
True
-cl
-na
-r
-l, -sa
-l
-f0
-f1
-fs
1
plot data
load data
get list of all fine grid sizes
2
-na, -sa
-g
-na
-cf
-no
-t
-r
-st
-i
-ps
-x
-ee
-bp
-ba
-sc
-mi
-mc
-ms
-k
-v
-cf
-el
get list of all elements to simulate
end testgrids
-r
1
-na
-st
analyse data
-na
clean up simulation directory
-q
-na
-po
run jobs
generate job data
KEY:
execution
data flow
start/end
function
code block
decision
connector
1
-r
reserved?
True
False
clean up semaphores
2
-cl
clean?
False
True
-na
clean up simulation directory

FUNCTION:	`setup_argparse`
INPUT:	`None`
OUTPUT:	`argparse.ArgumentParser` object
PURPOSE:	Sets up parser for command-line arguments; this includes setting argument names, their abbreviated form, allowed types, default values and description of each argument as well as the script as a whole to be shown when `--help` is used. A custom type `_str2bool` has been implemented via a private function that attempts to infer boolean values from their common textual representations. Besides settings up the parser, the function is useful for entry-level parsing checks (argument type check, missing required argument), as execution does not continue and the user is informed of the argument requirements if these checks are not satisfied. The parser is then returned.

FUNCTION:	`parse_args`
INPUT:	`parser` of type `argparse.ArgumentParser`
OUTPUT:	`argparse.Namespace` object
PURPOSE:	Uses previously setup parser to parse arguments passed to `sys.argv`, and verifies whether the arguments are valid. If `--load` is specified, only this argument is processed, and a check is performed whether the argument points to an existing file. If not loading data, the rest of the arguments are processed, terminating the execution with an appropriate message if a check fails. The module specified in `--castep` is temporarily loaded to test whether it exists. If `--cif_dir` is specified, check whether it points to a valid directory, if not, set up directory to one of the internal directories depending on whether the `--primitive` flag is used or not. If `--element` is not specified, the previously determined directory is scanned for all available elements. `--basis_prec`, `--mixing_scheme`, `--pseudo_pot`, and `--xc_functional` are compared against their respective available options. The rest of the arguments are then checked whether they are not out of their respective bounds. The `Namespace` object containing all arguments and their values is then returned.

FUNCTION:	`load_data`
INPUT:	`pickle_file` of type `str`
OUTPUT:	`dict` of element names as keys, and a `dict` of array and array-name pairs as values for each element key
PURPOSE:	Function simply opens a file handle to the file specified in `pickle_file` for binary reading only, and the pickled dictionary is loaded and returned.

FUNCTION:	`cleanup_wkdir`
INPUT:	`job_name` of type `str`
OUTPUT:	`None`
PURPOSE:	Function recursively deletes any directory with the name/path `job_name` if it exists. This is either carried out in preparation of a new job with the same name, or to clean up after a completed job is `--clean` is specified.

E. Analysis of Output

References

[1] Center for Molecular Modelling, *Comparing Solid State DFT Codes, Basis Sets and Potentials*, `molmod.ugent.be/deltacodesdft`, [Online; accessed 29-September-2019].

[2] X. Delaruelle, R. Owen, and K. Mein, *Environment Modules*, `modules.sourceforge.net`, [Online; accessed 03-October-2019].

[3] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. J. Probert, K. Refson, and M. C. Payne, Zeitschrift fur Kristallographie - Crystalline Materials **220**, 567 (2005).

[4] Python Software Foundation, *Python 3.6*, `python.org`, [Online; accessed 03-October-2019].

[5] NumPy development community, *NumPy 1.14*, `numpy.org`, [Online; accessed 03-October-2019].

[6] J. Hunter, D. Dale, E. Firing, M. Droettboom, and the Matplotlib development team, *Matplotlib 2.1*, `matplotlib.org`, [Online; accessed 03-October-2019].

[7] Dassault Systemes, *CASTEP Documentation*, `www.tcm.phy.cam.ac.uk/castep/documentation/WebHelp/CASTEP.html`, [Online; accessed 04-October-2019].

[8] SLURM Development Team, *SLURM Documentation*, `slurm.schedmd.com/documentation.html`, [Online; accessed 04-October-2019].