CONQUEST Documentation

Release 1.0.2-pre

CONQUEST Developers

Mar 10, 2020

GETTING STARTED

CONQUEST is a local orbital density functional theory (DFT) code, capable of massively parallel operation with excellent scaling. It uses a local orbital basis to represent the Kohn-Sham eigenstates or the density matrix. CONQUEST can be applied to atoms, molecules, liquids and solids, but is particularly efficient for large systems. The code can find the ground state using exact diagonalisation of the Hamiltonian or via a linear scaling approach. The code has demonstrated scaling to over 2,000,000 atoms and 200,000 cores when using linear scaling, and over 3,400 atoms and 850 cores with exact diagonalisation. CONQUEST can perform structural relaxation (including unit cell optimisation) and molecular dynamics (in NVE, NVT and NPT ensembles with a variety of thermostats).

GETTING STARTED

1.1 Overview: Why CONQUEST?

There are already many DFT codes which are available under open-source licences. Here we give reasons why you might choose to use CONQUEST.

1.1.1 Large-scale simulations

CONQUEST is designed to scale to large systems, either using exact diagonalisation (with the multisite support function approach, we have demonstrated calculations on over 3,000 atoms) or with linear scaling (where calculations on over 2,000,000 atoms have been demonstrated). Moreover, the same code and basis sets can be used to model systems from 1 atom to more than 1,000,000 atoms.

1.1.2 Efficient parallelisation

CONQUEST is an inherently parallel code, with scaling to more than 800 cores demonstrated for exact diagonalisation, and nearly 200,000 cores with linear scaling. This scaling enables efficient use of HPC facilities. CONQUEST (in linear scaling mode, as well as to a certain extent for exact diagonalisation) scales best with weak scaling: fixing the number of atoms per core (or thread) and choosing a number of cores based on the number of atoms.

CONQUEST also offers some OpenMP parallelisation in linear scaling mode, with relatively low numbers of MPI threads per node, and further parallelisation performed with OpenMP.

1.1.3 Linear scaling

The ideas of linear scaling have been current for more than twenty years, but it has proven challenging to make efficient, accurate codes to implement these ideas. CONQUEST has demonstrated effective linear scaling (with excellent parallel scaling), though is still somewhat restricted in the basis sets that can be used. For calculations beyond 5,000-10,000 atoms with DFT, linear scaling is the only option.

1.1.4 Basis sets

CONQUEST expresses the Kohn-Sham eigenstates or the density matrix (which are equivalent) in terms of local orbitals called *support functions*. These support functions are made from one of two basis sets: pseudo-atomic orbitals (PAOs) or blip functions (B-splines); the main basis functions in use in CONQUEST are the PAOs. A PAO generation code is included with the CONQUEST distribution, with well-defined and reliable default basis sets for most elements.

The simplest choice is to use one PAO for each support function (typically this allows calculations up to 1,000 atoms). For diagonalisation beyond this system size, a composite basis is used, where PAOs from several are combined into a smaller set of support functions (multi-site support functions, or MSSF). With MSSF, calculations on 3,000+ atoms are possible on HPC platforms. For linear scaling, more care is required with basis sets (more details can be found *here*).

1.2 Frequently Asked Questions

1.2.1 When should I use CONQUEST?

You can use CONQUEST for any DFT simulations that you need to perform. It is efficient for small problems, though may not be as efficient as other codes (e.g. plane wave codes) because it has been designed for massively parallel operation, which brings some overhead. If you need to perform DFT calculations on large systems (several hundred atoms or beyond) or want to perform highly parallel calculations, you should definitely consider CONQUEST.

CONQUEST uses Hamann optimised norm-conserving Vanderbilt (ONCV) pseudopotentials, which can also be used by PWSCF and Abinit which allows direct comparisons between the codes.

1.2.2 When should I use linear scaling?

You should use linear scaling if you need to model systems with more than about 5,000 atoms, though gains are often found for smaller systems (from 1,000 atoms upwards).

Linear scaling calculations offer the prospect of scaling to significantly larger systems than traditional DFT calculations; however, they make approximations and require some care and characterisation. In particular, instead of solving for eigenvalues and eigenstates, linear scaling methods solve for the density matrix, so that energy-resolved information (e.g. DOS and band energies) are not available. To enable linear scaling, a range is also imposed on the density matrix and it is important to test the effect of this range.

1.2.3 Will you implement a specific feature for me?

We cannot guarantee to implement specific features, though we are always happy to take suggestions. We also welcome new developers: if there is something that you would like to see in the code, please do talk to us about joining the development effort.

1.2.4 How do I report a bug?

Please use the GitHub issues page. Include details of the compiler and libraries used, the version of CONQUEST, and the input and output files (if possible). We will do our best to check the bug and fix it, but cannot guarantee to help on any timescale.

1.2.5 How do I get help?

The Conquest mailing list (**details**) is the best place to get help. However, the developers cannot guarantee to answer any questions, though they will try. Bug reports should be made through the GitHub issues page.

1.3 Quick Overview

1.3.1 Setting up a calculation

CONQUEST requires three types of file for a calculation:

- A coordinate file
- Ion files (pseudopotentials)
- The input file (`Conquest_input`)

Coordinates

CONQUEST works with orthorhombic unit cells (i.e. with angles between lattice vectors at ninety degrees). The coordinate file is laid out simply: lattice vectors, number of atoms, atom coordinates (along with species and movement flags). Either fractional or Cartesian coordinates can be read (the default is fractional; Cartesian coordinates require a flag to be set in the input file). CONQUEST also reads and writes PDB format coordinate files for biomolecular simulations. More information can be found in *Coordinates*.

Ion files

The ion files contain the pseudopotentials and pseudo-atomic orbitals for the elements, and follow a format similar to the ion files from Siesta (CONQUEST can read Siesta ion files). A set of default inputs to generate ion files is available in the directory `pseudo-and-pao`. These contain pseudopotentials based on the PseudoDojo library, and allow ion files to be produced with the basis set generation code that is included with CONQUEST in the `tools/BasisGeneration` directory. Full details are found *here*.

Conquest_input

The `Conquest_input` file contains all of the input flags to control a CONQUEST run. At a minimum, the file must specify: the run type (e.g. `static` or `md`); the coordinate file name; and the number of species and the ion file names. For a well characterised calculation, further options must be given (for instance setting details for the calculation of the density matrix). Simple examples are given in *Example calculations* and full documentation of all options can be found in *Input tags*.

Go to *top*

1.3.2 Output from a calculation

The main output from CONQUEST is in a single file, named `Conquest_out` by default (this can be changed, and output can be written to `stdout` rather than a file). This file contains details of the calculation, energies, forces and stresses and the various electronic structure and atomic movement calculations performed. The most important files that are produced during a run are:

- `Conquest_out` The output file

- `Conquest_warnings` A list of any warnings issued by the code (also in `Conquest_out`)

- `coord_next.dat` The updated set of atomic positions

- `conquest.bib` References suggested for the calculation performed

- `input.log` A log of input options (both set by user and defaults)

Other files are produced by different run types, and are discussed elsewhere.

1.4 Installation

You will need to download and compile the code before you can use it; we do not supply binaries.

1.4.1 Downloading

CONQUEST is accessed from the GitHub repository; it can be cloned:

`git clone https://github.com/OrderN/CONQUEST-release destination-directory`

where `destination-directory` should be set by the user. Alternatively, it can be downloaded from GitHub as a zip file and unpacked:

https://github.com/OrderN/CONQUEST-release/archive/master.zip

1.4.2 Compiling

Once you have the distribution, you will need to compile the main Conquest code (found in the `src/` directory), along with the ion file generation code (found in the `tools/` directory). Conquest requires a working MPI installation including a Fortran90 compiler (often `mpif90` but this can vary), along with a few standard libraries:

- BLAS and LAPACK (normally provided by the system vendor)

- FFTW 3.x (more detail can be found at http://www.fftw.org/)

- ScaLAPACK (often provided as part of an HPC system; the source code can be obtained from the netlib repository if you need to compile it)

Additionally, Conquest can use LibXC if it is available (v2.x or later).

The library locations are set in the `system.make` file in the `src/` directory, along with other parameters needed for compilation.

- `FC` (typically `FC=mpif90` will be all that is required)

- `COMPFLAGS` (set these to specify compiler options such as optimisation)

- `BLAS` (specify the BLAS and LAPACK libraries)

- `SCALAPACK` (specify the ScaLAPACK library)

- `FFT_LIB` (must be left as FFTW)

- `XC_LIBRARY` (choose `XC_LIBRARY=CQ` for the internal Conquest library, otherwise `XC_LIBRARY=LibXC_v2` LibXC v2. or `XC_LIBRARY=LibXC` for LibXC v3.x or higher)

- Two further options need to be set for LibXC:

  - `XC_LIB` (specify the XC libraries)

  - `XC_COMPFLAGS` (specify the location of the LibXC include and module files, e.g. `-I/usr/local/include`)

Once these are set, you should make the executable using `make`.

The ion file generation code is compiled using the same options required for the main code.

1.5 Example calculations

All example calculations here use diagonalisation and PAO basis sets (with a simple one-to-one mapping between PAOs and support functions).

1.5.1 Static calculation

We will perform a self-consistent electronic structure calculation on bulk silicon. The coordinate file that is needed is:

```
10.36  0.00  0.00
 0.00 10.36  0.00
 0.00  0.00 10.36
8
  0.000 0.000 0.000  1 T T T
  0.500 0.500 0.000  1 T T T
  0.500 0.000 0.500  1 T T T
  0.000 0.500 0.500  1 T T T
  0.250 0.250 0.250  1 T T T
  0.750 0.750 0.250  1 T T T
  0.250 0.750 0.750  1 T T T
  0.750 0.250 0.750  1 T T T
```

You should save this in an appropriate file (e.g. `coords.dat`). The inputs for the ion file can be found in `pseudo-and-pao/PBE/Si` (for the PBE functional). Changing to that directory and running the `MakeIonFiles` utility (in `tools`) will generate the file `SiCQ.ion`, which should be copied to the run directory, and renamed to `Si.ion`. The `Conquest_input` file requires only a few simple lines at its most basic:

```
AtomMove.TypeOfRun static
IO.Coordinates coords.dat
Grid.GridCutoff  50
Diag.MPMesh   T
Diag.GammaCentred T
Diag.MPMeshX  2
Diag.MPMeshY  2
Diag.MPMeshZ  2
General.NumberOfSpecies  1
%block ChemicalSpeciesLabel
 1 28.086 Si
%endblock
```

The parameters above should be relatively self-explanatory; the grid cutoff (in Hartrees) sets the integration grid spacing, and can be compared to the *charge density* grid cutoff in a plane wave code (typically four times larger than the plane wave cutoff). The Monkhorst-Pack k-point mesh (`Diag.MPMeshX/Y/Z`) is a standard feature of solid state codes; note that the grid can be forced to be centred on the Gamma point.

The most important parameters set the number of species and give details of what the species are (`ChemicalSpeciesLabel`). For each species label (in this case `Si`) there should be a corresponding file with the extension `.ion` (again, in this case `Si.ion`). CONQUEST will read the necessary information from this file for default operation, so no further parameters are required. This block also allows the mass of the elements to be set (particularly important for molecular dynamics runs).

The output file starts with a summary of the calculation requested, including parameters set, and gives details of papers that are relevant to the particular calculation. After brief details of the self-consistency, the total energy, forces and stresses are printed, followed by an estimate of the memory and time required. For this calculation, these should be close to the following:

```
Harris-Foulkes Energy           :        -33.792210321858057 Ha

                Atom   X               Y               Z
                  1  -0.0000000000   0.0000000000    0.0000000000
                  2  -0.0000000000   0.0000000000    0.0000000000
                  3  -0.0000000000   0.0000000000   -0.0000000000
                  4   0.0000000000   0.0000000000    0.0000000000
                  5  -0.0000000000   0.0000000000   -0.0000000000
                  6   0.0000000000   0.0000000000    0.0000000000
                  7  -0.0000000000   0.0000000000    0.0000000000
                  8  -0.0000000000  -0.0000000000    0.0000000000
 Maximum force :      0.00000000(Ha/a0) on atom, component        2        3

                          X               Y               Z
 Total stress:      -0.01848219     -0.01848219    -0.01848219 Ha

 Total pressure:     0.48902573      0.48902573     0.48902573 GPa
```

The output file ends with an estimate of the total memory and time used.

You might like to experiment with the grid cutoff to see how the energy converges (note that the number of grid points is proportional to the square root of the energy, while the spacing is proportional to one over this, and that the computational effort will scale with the *cube* of the number of grid points); as with all DFT calculations, you should ensure that you test the convergence with respect to all parameters.

Go to *top*.

1.5.2 Relaxation

Atomic Positions

We will explore structural optimisation of the methane molecule (a very simple example). The coordinates required are:

```
20.000   0.000   0.000
 0.000  20.000   0.000
 0.000   0.000  20.000
5
0.500 0.500 0.500 1  F F F
0.386 0.500 0.500 2  T F F
0.539 0.607 0.500 2  T T F
0.537 0.446 0.593 2  T T T
0.537 0.446 0.407 2  T T T
```

The size of the simulation cell should, of course, be tested carefully to ensure that there are no interactions between images. We have fixed the central (carbon) atom, and restricted other atoms to prevent rotations or translations during optimisation.

The `Conquest_input` file changes only a little from before, as there is no need to specify a reciprocal space mesh (it defaults to gamma point only, which is appropriate for an isolated molecule). We have set the force tolerance (`AtomMove.MaxForceTol`) to a reasonable level (approximately 0.026 eV/A). Note that the ion files can be generated in the same way *as before*, and that we assume that the ion files are renamed to `C.ion` and `H.ion`.

```
IO.Coordinates CH4.in
Grid.GridCutoff 50

AtomMove.TypeOfRun lbfgs
AtomMove.MaxForceTol 0.0005

General.NumberOfSpecies  2
%block ChemicalSpeciesLabel
1 12.00 C
2 1.00 H
%endblock
```

The progress of the optimisation can be followed by searching for the string `Geom` (using `grep` or something similar). In this case, we find:

```
GeomOpt - Iter:    0 MaxF:   0.04828504 E:  -0.83676760E+01 dE:   0.00000000
GeomOpt - Iter:    1 MaxF:   0.03755566 E:  -0.83755762E+01 dE:   0.00790024
GeomOpt - Iter:    2 MaxF:   0.02691764 E:  -0.83804002E+01 dE:   0.00482404
GeomOpt - Iter:    3 MaxF:   0.00613271 E:  -0.83860469E+01 dE:   0.00564664
GeomOpt - Iter:    4 MaxF:   0.00126136 E:  -0.83862165E+01 dE:   0.00016958
GeomOpt - Iter:    5 MaxF:   0.00091560 E:  -0.83862228E+01 dE:   0.00000629
GeomOpt - Iter:    6 MaxF:   0.00081523 E:  -0.83862243E+01 dE:   0.00000154
GeomOpt - Iter:    7 MaxF:   0.00073403 E:  -0.83862303E+01 dE:   0.00000603
GeomOpt - Iter:    8 MaxF:   0.00084949 E:  -0.83862335E+01 dE:   0.00000316
GeomOpt - Iter:    9 MaxF:   0.00053666 E:  -0.83862353E+01 dE:   0.00000177
GeomOpt - Iter:   10 MaxF:   0.00033802 E:  -0.83862359E+01 dE:   0.00000177
```

The maximum force reduces smoothly, and the structure converges well. By adjusting the output level (using `IO.Iprint` for overall output, or `IO.Iprint_MD` for atomic movement) more information about the structural relaxation can be produced (for instance, the force residual and some details of the line minimisation will be printed for `IO.Iprint_MD 2`).

Go to *top*.

Cell Parameters

We will optimise the lattice constant of the bulk silicon cell that we studied for the static calculation. Here we need to change the type of run, and add one more line:

```
AtomMove.TypeOfRun cg
AtomMove.OptCell T
```

Adjust the simulation cell size to 10.26 Bohr radii in all three directions (to make it a little more challenging). If you run this calculation, you should find a final lattice constant of 10.372 after 3 iterations. The progress of the optimization can be followed in the same way as for structural relaxation, and gives:

```
GeomOpt - Iter:    0 MaxStr:   0.00011072 H:  -0.33790200E+02 dH:   0.00000000
GeomOpt - Iter:    1 MaxStr:   0.00000195 H:  -0.33792244E+02 dH:   0.00204424
GeomOpt - Iter:    2 MaxStr:   0.00000035 H:  -0.33792244E+02 dH:  -0.00000017
```

Go to *top*.

1.5.3 Simple Molecular Dynamics

We will perform NVE molecular dynamics for methane, CH4, as a simple example of how to do this kind of calculation. You should use the same coordinate file and ion files as you did for the structural relaxation, but change the atomic movement flags in the coordinate file to allow all atoms to move (the centre of mass is fixed during MD by default). Your coordinate file should look like this:

```
20.00000000000000      0.00000000000000      0.00000000000000
 0.00000000000000     20.00000000000000      0.00000000000000
 0.00000000000000      0.00000000000000     20.00000000000000
5
0.500 0.500 0.500 1  T T T
0.386 0.500 0.500 2  T T T
0.539 0.607 0.500 2  T T T
0.537 0.446 0.593 2  T T T
0.537 0.446 0.407 2  T T T
```

The input file should be:

```
IO.Coordinates CH4.in

AtomMove.TypeOfRun md
AtomMove.IonTemperature 300
AtomMove.NumSteps 100

General.NumberOfSpecies  2
%block ChemicalSpeciesLabel
1 12.00 C
2 1.00 H
%endblock
```

where the default timestep (0.5fs) is necessary for simulations involving light atoms like hydrogen. The file md. stats contains details of the simulation, while the trajectory is output to trajectory.xsf which can be read by VMD among other programs. In this simulation, the conserved quantity is the total energy (the sum of ionic kinetic energy and potential energy of the system) which is maintained to better than 0.1mHa in this instance. More

importantly, the variation in this quantity is much smaller than the variation in the potential energy. This can be seen in the plot below.
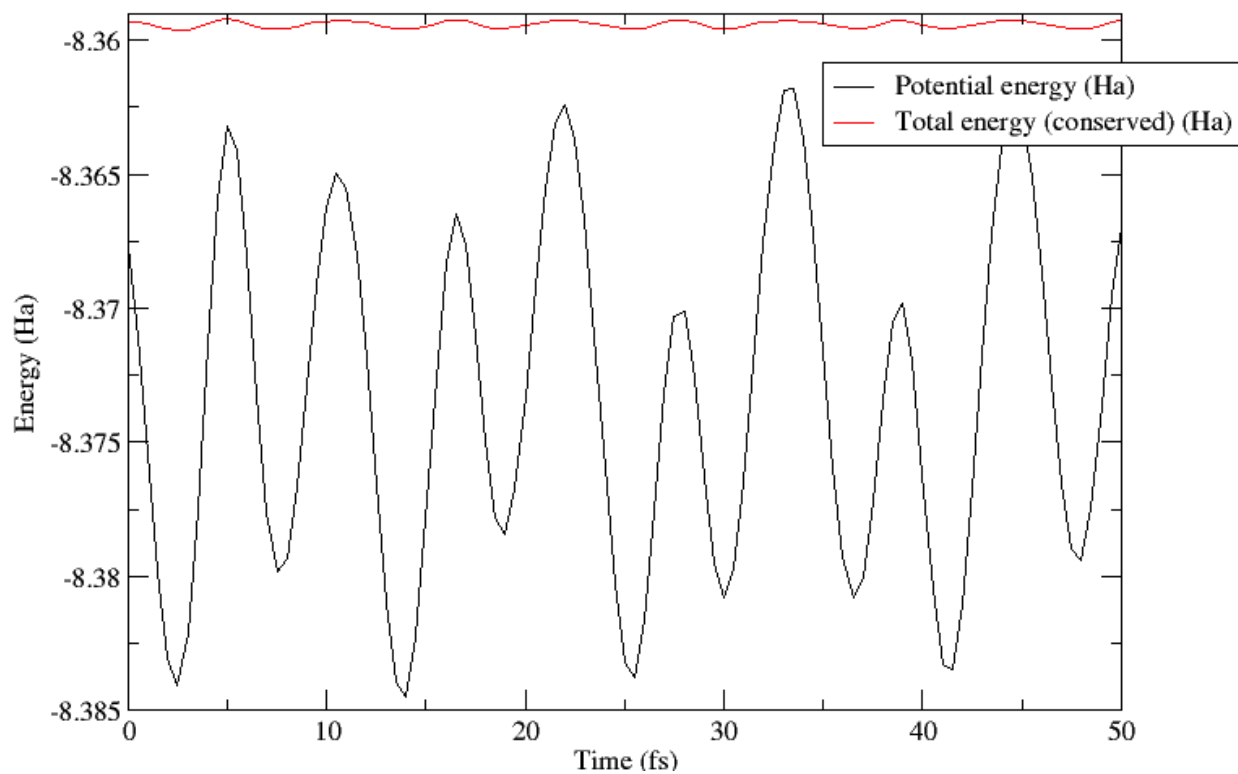
Go to *top*.

1.5.4 Tutorials

We recommend that you work through, in order, the tutorials included in the distribution in the `tutorials/` directory to become familiar with the modes of operation of the code.

NOTE In the initial pre-release of CONQUEST (January 2020) we have not included the tutorials; they will be added over the coming months.

Go to *top*.

1.5.5 Where next?

While the tutorials have covered the basic operations of Conquest, there are many more subtle questions and issues, which are given in the User Guide.

Go to *top*.

TWO

USER GUIDE

- *Input and output*
- *Finding the ground state*
- *Converging Parameters*
- *Basis sets*
- *Structural relaxation*
- *Molecular Dynamics*
- *External tools*
- *Error codes*
- *Input tags*

2.1 Input and output

2.1.1 Input files

Conquest_input

All necessary input parameters should be specified in the `Conquest_input` file, including the names of the coordinate file and the ion files. This file controls the run; there are many sensible default values for input parameters, but you should ensure that you understand what they mean. After a run, the full set of relevant input parameters (whether specified by the user, or default, are available in the file `input.log`).

The most common input tags are listed briefly here. Full documentation can be found in *Input tags*.

- `AtomMove.TypeOfRun` takes `static`, `md`, `lbfgs`, `cg`
- `IO.Coordinates` File name
- `DM.SolutionMethod` diagon
    - `Diag.MPMesh` T/F
    - `Diag.MPMeshX` (and `Y` and `Z`) N
    - `Diag.GammaCentred` T/F
- `General.NumberOfSpecies` N
- `%block ChemicalSpeciesLabel` Specifies element number, mass and ion file name
- `General.LoadDM` T/F (to reload the density matrix)

- `IO.FractionalAtomicCoords` T/F
- `Spin.SpinPolarised` T/F
    - `Spin.FixSpin` T/F
    - `Spin.Magn` Difference between spin channel occupations
- `Grid.GridCutoff` Energy in Ha
- `minE.SCTolerance` Fractional tolerance
- `AtomMove.NumSteps` N
- `AtomMove.MaxForceTol` in Ha/bohr
- `AtomMove.OptCell` T/F
- `SC.KerkerPreCondition` T/F (for Kerker preconditioning of SCF)
- `SC.MaxIters` N (maximum number of SCF iterations

Go to *top*

Ion files

The ion files contain data on the different species being modelled: valence charge, pseudopotentials, pseudo-atomic orbitals (PAOs) etc. Full details on how the PAOs are used as basis functions for CONQUEST can be found in the manual section on *basis sets*. A utility for generating these files is provided with CONQUEST, but Siesta ion files can also be read. The CONQUEST utility uses the pseudopotentials generated by the ONCVPSP code (though note that to generate new files for CONQUEST, you will need a small patch).

A set of input files for all elements in the PseudoDojo library for both LDA and PBE exchange-correlation functionals is provided in the directory `pseudo-and-pao`. This will allow you to generate ion files for these elements easily.

The utility for generating ion files is called *MakeIonFiles*, and its source code is found in the `tools/BasisGeneration` directory. It uses the same `system.make` file as CONQUEST, and following compilation the executable will be moved to the `bin` directory. The key parameters to be set in the `Conquest_input` file for the ion file generation are:

- `General.NumberOfSpecies` to specify number of species
- `%block SpeciesLabels` to specify what the species are
- In the species block (set with `%block XX` for species XX):
    - `Atom.PseudopotentialFile` to specify the input file for the ONCVPSP code
    - `Atom.VKBFile` to specify the file that CONQUEST needs to read (included in the library of inputs)
    - `Atom.BasisSize` to specify the size of the basis; at present this can take the values: `minimal`; `small`; `medium`; and `large`.

Further fine-grained control can be applied to the basis functions; this will be documented after the pre-release of CONQUEST.

Go to *top*

Coordinates

The coordinates are specified in a separate file with relatively simple format. The coordinates can be specified in fractional form (default) or cartesian (set the input tag `IO.FractionalAtomicCoords F`). Distance units can be Bohr radii (default) or Angstroms (set the input tag `General.DistanceUnits` to `Ang`). At present, CONQUEST only handles *orthorhombic* unit cells.

The coordinate file is formatted as follows:

```
a    0.0 0.0
0.0 b    0.0
0.0 0.0 c
NAtoms
x y z species MoveX MoveY MoveZ
.
.
.
```

Note that the flags `MoveX` etc take values T/F and indicate whether atoms are free to move in x, y and z, respectively. The flag `species` is an integer, and selects based on species defined in the *atomic specification* section of the `Conquest_input` file.

Go to *top*

2.1.2 Output files

Main output

By default, CONQUEST writes output to the `Conquest_out` file (though the filename can be set with the parameter `IO.OutputFile`, and the flag `IO.WriteOutToFile` (T/F) selects output to file or `stdout`). This file contains all details of the calculation, including energies, forces and information on the different stages of the calculation. The output verbosity is controlled by the `IO.Iprint` family of parameters, which allows different levels of output detail to be set for different areas of the code.

Note that, during the pre-release period of CONQUEST, we will be tidying the output and making it more consistent.

Go to *top*

Electronic structure

Different electronic structure outputs are available; in each case, the key output flag is given. Further output flags are described in *Input tags*.

- Charge density

- Band-resolved charge density (`IO.outputWF`)

- Density of states (`IO.writeDOS`)

- Atom-projected density of states (`IO.write_proj_DOS`)

- Atomic charges, using the Mulliken approach (`IO.AtomChargeOutput`)

The Kohn-Sham eigenvalues are output in the `Conquest_out` file. The charge densities need post-processing to convert from the standard output format to a file compatible with visualisation (current supported formats include Gaussian CUBE file and OpenDX files).

Note that Becke charges can be calculated if the following parameters are set:

:: SC.BeckeWeights T SC.BeckeAtomicRadii T IO.Iprint_SC 3

This method of output will be refined soon.

Go to *top*

Atomic structure

During structural relaxation and molecular dynamics, the atomic structure at the end of each step is saved in the output file coord_next.dat. This is in the same format as the input.

Go to *top*

Molecular dynamics

A molecular dynamics run will generate a number of additional plain text output files:

- md.stats — summarises thermodynamic quantities at each steps
- md.frames — contains the complete physical state of the system (lattice parameters, atomic positions, velocities, forces, stress).
- md.checkpoint — data required for MD restart, namely atomic velocities and extended system variables.
- md.positions — Atomic coordinates saved at the moment of checkpointing
- trajectory.xsf — atomic coordinates save in .xsf format, which can be visualised using (for example) VMD, if AtomMove.WriteXSF is true..

Full details are available in *Molecular Dynamics*.

Go to *top*

2.2 Finding the ground state

Finding the electronic ground state is the heart of any DFT code. In CONQUEST, we need to consider several linked stages: the density matrix (found using *diagonalisation* or *linear scaling*); *self-consistency between charge and potential*; and the *support functions* (though these are not always optimised).

The basis functions in CONQUEST are *support functions* (localised functions centred on the atoms), written as $\phi_{i\alpha}(\mathbf{r})$ where i indexes an atom and α a support function on the atom. The support functions are used as basis functions for the density matrix and the Kohn-Sham eigenstates:

$$\psi_{n\mathbf{k}}(\mathbf{r}) = \sum_{i\alpha} c_{i\alpha}^{n\mathbf{k}} \phi_{i\alpha}(\mathbf{r})$$

$$\rho(\mathbf{r}, \mathbf{r}') = \sum_{i\alpha j\beta} \phi_{i\alpha}(\mathbf{r}) K_{i\alpha,j\beta} \phi_{j\beta}(\mathbf{r}')$$

where n is an eigenstate index and $\mathbf{k}$ is a point in the Brillouin zone (see *here* for more on this). The total energy can be written in terms of the density matrix, as:

$$E_{KS} = \text{Tr}[HK] + \Delta E_{Har} + \Delta E_{XC}$$

for the Hamiltonian matrix H in the basis of support functions, with the last two terms the standard Harris-Foulkes [G1][G2] correction terms.

For diagonalisation, the density matrix is made from the coefficients of the Kohn-Sham eigenstates, $c_{i\alpha}^{n\mathbf{k}}$, while for *linear scaling* it is found directly during the variational optimisation of the energy.

The question of whether to find the density matrix via diagonalisation or linear scaling is a complex one, depending on the system size, the accuracy required and the computational resources available. The simplest approach is to test diagonalisation before linear scaling.

2.2.1 Diagonalisation

Exact diagonalisation in CONQUEST uses the ScaLAPACK library which scales reasonably well in parallel, but becomes less efficient with large numbers of processes. The computational time will scale as N^3 with the number of atoms N, but will probably be more efficient than linear scaling for systems up to a few thousand atoms. (Going beyond a thousand atoms with diagonalisation is likely to require the *multi-site support function* technique.)

To choose diagonalisation, the following flag should be set:

```
DM.SolutionMethod diagon
```

It is also essential to test relevant parameters, as described below: the k-point grid in reciprocal space (to sample the Brillouin zone efficiently); the occupation smearing approach; and the parallelisation of k-points.

Go to *top*

Brillouin zone sampling

We need to specify a set of discrete points in reciprocal space to approximate integrals over the Brillouin zone. The simplest approach is to use the Monkhorst-Pack approach [G3], where a grid of points is specified in all directions:

```
Diag.MPMesh T
Diag.MPMeshX 2
Diag.MPMeshY 2
Diag.MPMeshZ 2
```

This grid can be forced to be centred on the gamma point (often an important point) using the parameter `Diag.GammaCentred T`. The origin of the Monkhorst-Pack grid may also be offset by an arbitrary vector from the origin of the Brillouin zone, by specifying:

```
Diag.MPShiftX 0.0
Diag.MPShiftY 0.0
Diag.MPShiftZ 0.0
```

Alternatively, the points in reciprocal space can be specified explicitly by giving a number of points and their locations and weights:

```
Diag.NumKpts 1

%block Diag.Kpoints
0.00 0.00 0.00 1.00
%endblock Diag.Kpoints
```

where there must be as many lines in the block as there are k-points. It is important to note that CONQUEST does not consider space group symmetry when integrating over the Brillouin zone.

Go to *top*.

K-point parallelization

It is possible to parallelise over k-points: to split the processes into sub-groups, each of which is responsible for a sub-set of the k-points. This can be very efficient, and is specified by the parameter `Diag.KProcGroups N`, where it is important that the number of processes is an integer multiple of the number of groups `N`. It will be most efficient when the number of k-points is an integer multiple of the number of groups.

Go to *top*.

Electronic occupation smearing

The occupation numbers of the eigenstates are slightly smeared near the Fermi level, following common practice. The default smearing type is Fermi-Dirac smearing with a temperature (in Hartrees) set with the flag `Diag.kT` which defaults to 0.001Ha.

The Methfessel-Paxton approach [G4] to occupations allows much higher smearing temperatures with minimal effect on the free energy (and hence accuracy) of the energy. This generally gives a similar accuracy with fewer k-points, and is selected as:

```
Diag.SmearingType 1
Diag.MPOrder 0
```

where `Diag.MPOrder` specifies the order of the Methfessel-Paxton expansion. It is recommended to start with the lowest order and increase gradually, testing the effects.

Go to *top*.

2.2.2 Linear Scaling

A linear scaling calculation is selected by setting `DM.SolutionMethod ordern`. There are two essential parameters that must be set: the range of the density matrix, and the tolerance on the optimisation.

```
DM.L_range 16.0
minE.Ltolerance 1.0e-6
```

The tolerance is applied to the residual (the RMS value of the gradient of the energy with respect to the density matrix). The maximum number of iterations in the density matrix optimisation can be set with `DM.LVariations` (default 50).

At present, CONQUEST can only operate efficiently in linear scaling mode with a restricted number of support functions (though this is an area of active development). PAO basis sets of SZ and SZP size (minimal and small in the ion file generator) will run without restrictions. For larger PAO basis sets, the *OSSF* approach must be used, and is effective. With a blip basis there are no restrictions, though efficient optimisation is still under active development.

It is almost always more efficient to update the charge density while optimising the density matrix, avoiding the need for a separate self-consistency loop. This is set by choosing `minE.MixedLSelfConsistent T`.

An essential part of a linear scaling calculation is finding the approximate, sparse inverse of the overlap matrix. Normally this will happen automatically, but it may require some tests. The key parameters are the range for the inverse (see the *Atomic Specification* block, and specifically the *Atomic Specification* block) and the tolerance applied to the inversion.

```
Atom.InvSRange R
DM.InvSTolerance R
```

A tolerance of up to 0.2 can give convergence without significantly affecting the accuracy. The range should be similar to the radius of the support functions, though increasing it by one or two bohr can improve the inversion in most cases.

The input tags are mainly found in the *Density Matrix* section of the *Input tags* page.

Go to *top*.

2.2.3 Self-consistency

The normal mode of operation for CONQUEST involves an iterative search for self-consistency between the potential and the charge density. However, it is also possible to run in a non-self-consistent manner, either with a converged charge density for electronic structure analysis, or for dynamics, which will be considerably more efficient than a self-consistent calculation, but less accurate.

Self consistency is set via the following parameters:

```
minE.SelfConsistent  T
minE.SCTolerance     1E-7
SC.MaxIters          50
```

The tolerance is applied to the RMS value of the residual, $R(\mathbf{r}) = \rho^{out}(\mathbf{r}) - \rho^{in}(\mathbf{r})$, integrated over all space:

$$R_{RMS} = \sqrt{\Omega \sum_l \left(R(\mathbf{r}_l)\right)^2}$$

where $\mathbf{r}_l$ is a grid point and Ω is the grid point volume (integrals are performed on a grid explained in *Integration Grid*). The maximum number of self-consistency cycles is set with `SC.MaxIters`, defaulting to 50.

For non-self-consistent calculations, the main flag should be set as `minE.SelfConsistent F`. The charge density at each step will either be read from a file (if the flag `General.LoadRho T` is set), or constructed from a superposition of atomic densities. The Harris-Foulkes functional will be used to find the energy.

Go to *top*.

Advanced options

Instabilities during self-consistency are a well-known issue in electronic structure calculations. CONQUEST performs charge mixing using the Pulay approach, where the new charge density is prepared by combining the charge densities from a number of previous iterations. In general, we write:

$$rho_{n+1}^{in} = \sum_i \alpha_i \left[\rho_i^{in} + AR_i\right]$$

where R_i is the residual at iteration i, defined above. The fraction of the output charge density that is included is governed by the variable A, which is set by the parameter `SC.LinearMixingFactor` (default 0.5). If there is instability during the self consistency, reducing A can help (though will likely make convergence a little slower).

It is also advisable to apply Kerker preconditioning to the residual when the system is large in any dimension. This removes long wavelength components of the residual, reducing charge sloshing. This is controlled with the following parameters:

```
SC.KerkerPreCondition  T
SC.KerkerFactor        0.1
```

where the Kerker factor gives the wavevector at which preconditioning starts to reduce. The Kerker preconditioning is applied to the Fourier transform of the residual, $\tilde{R}$ as:

$$\tilde{R}\frac{q^2}{q^2 - q_0^2}$$

where q_0^2 is the square of the Kerker factor and q is a wavevector. You should test values of q_0 around π/a where a is the longest dimension of the simulation cell (or some important length scale in your system).

Go to *top*.

2.2.4 Support functions

Support functions in CONQUEST represent the density matrix, and can be simple (pseudo-atomic orbitals, or PAOs) or compound, made from simple functions (either PAOs or blips). If they are compound, made from other functions, then the search for the ground state involves the construction of this representation. Full details of how the support functions are built and represented can be found in the manual section on *basis sets*.

Go to *top*

2.2.5 Charged systems

CONQUEST uses periodic boundary conditions, which require overall charge neutrality. However, charged systems can be modelled: if an excess of electrons is specified by the user, a uniform positive background charge is added automatically to restore overall neutrality. At present, there are no correction schemes implemented, so it is important to test the convergence of the energy with unit cell size and shape. Electrons are added by setting the parameter `General.NetCharge`.

```
General.NetCharge 1.0
```

This gives the number of extra electrons to be added to the unit cell, beyond the valence electrons.

Go to *top*.

2.2.6 Spin polarisation

CONQUEST performs collinear spin calculations only. A spin-polarised calculation is performed by setting the parameter `Spin.SpinPolarised` to T.

Users need to specify *either* the total initial number of spin-up and spin-down electrons in the simulation cell (using the parameters `Spin.NeUP` and `Spin.NeDN`), *or* the difference between the number of spin-up and spin-down electrons (using the parameter `Spin.Magn`).

The number of electrons for each spin channel can be fixed during SCF calculations by setting the parameter `Spin.FixSpin` to T (default is F).

It is possible to specify the spin occupation in the atomic charge densities (i.e. the number of spin-up and spin-down electrons used to build the density). This is done in the *Atomic Specification* part of the `Conquest_input` file. Within the atom block for each species, the numbers of electrons should be set with `Atom.SpinNeUp` and `Atom.SpinNeDn`. Note that these numbers *must* sum to the number of valence electrons for the atom.

Go to *top*.

Examples: FM and AFM iron

A two atom ferromagnetic iron simulation might be set up using the parameters below. Note that the net spin here is S=1 μ_B (i.e. two more electrons in the up channel than in the down), and that the net spin is not constrained.

```
# example of ferro bcc Fe
Spin.SpinPolarised T
Spin.FixSpin  F
Spin.NeUP  9.0      # initial numbers of up- and down-spin electrons,
Spin.NeDN  7.0      # which will be optimised by a SCF calculation when Spin.FixSpin=F

%block ChemicalSpeciesLabel
1   55.845   Fe
%endblock ChemicalSpeciesLabel
```

An equivalent anti-ferromagnetic calculation could be set up as follows (though note that the initial specification of spin for the atoms does *not* guarantee convergence to an AFM ground state). By defining two species we can create spin-up and spin-down atoms (note that both species will require their own, appropriately labelled, ion file).

```
# example of anti-ferro bcc Fe
Spin.SpinPolarised T
Spin.FixSpin  F
Spin.NeUP  8.0      # initial numbers of up- and down-spin electrons in an unit cell
Spin.NeDN  8.0      # are set to be the same

%block ChemicalSpeciesLabel
1   55.845   Fe1
2   55.845   Fe2
%endblock ChemicalSpeciesLabel

%block Fe1           # up-spin Fe
Atom.SpinNeUp 5.00
Atom.SpinNeDn 3.00
%endblock Fe1
%block Fe2           # down-spin Fe
Atom.SpinNeUp 3.00
Atom.SpinNeDn 5.00
%endblock Fe2
```

When using multi-site or on-site support functions in spin-polarised calculations, the support functions can be made spin-dependent (different coefficients for each spin channel) or not by setting `Basis.SpinDependentSF` (T/F, default is T).

Go to *top*.

Go to *top*.

2.3 Converging Parameters

There are various important parameters in CONQUEST that affect the convergence of the total energy, and need to be tested. Integrals are calculated on a grid; the density matrix is found approximately; a self-consistent charge density is calculated; and support functions are, in some modes of operations, optimised. These parameters are described here.

2.3.1 Integration Grid

While many integrals are calculated analytically or on fine grids that move with the atoms, there are still some integrals that must be found numerically, and CONQUEST uses an orthorhombic, uniform grid to evaluate these integrals (this grid is also used for the Fourier transforms involved in finding the Hartree potential). The spacing of the grid will affect the accuracy of the calculation, and it is important to test the convergence of the total energy with the grid spacing.

The grid spacing can be set intuitively using an energy (which corresponds to the kinetic energy of the shortest wavelength wave that can be represented on the grid). In atomic units, $E = k^2/2$ with $k = \pi/\delta$ for grid spacing δ. The cutoff is set with the parameter:

```
Grid.GridCutoff E
```

where `E` is an energy in Hartrees. The grid spacing can also be set manually, by specifying the number of grid points in each direction:

```
Grid.PointsAlongX N
Grid.PointsAlongY N
Grid.PointsAlongZ N
```

If setting the grid in this manner, it is important to understand a little more about the internal workings of CONQUEST. The grid is divided up into *blocks* (the default size is 4 by 4 by 4), and the number of grid points in any direction must correspond to an integer multiple of the block size in that direction. The block size can be set by the user:

```
Grid.InBlockX N
Grid.InBlockY N
Grid.InBlockZ N
```

Note that the blocks play a role in parallelisation and memory use, so that large blocks may require larger memory per process; we recommend block sizes no larger than 8 grid points in each direction. There is also, at present, a restriction on the total number of grid points in anuy direction, that it must have prime factors of only 2, 3 and 5. This will be removed in a future release.

Go to *top*.

2.3.2 Finding the density matrix

As discussed in the section on *finding the ground state*, the density matrix is found either with exact diagonaliation, or the linear scaling approach. These two methods require different convergence tests, and are described separately.

Diagonalisation: Brillouin Zone Sampling

The sampling of the Brillouin zone must be tested for convergence, and the parameters are described *here*. The convergence of charge density will be faster than detailed electronic structure such as density of states (DOS), and it will be more accurate for these types of calculations to generate a converged charge density, and then run non self-consistently (see the section on *self consistency*) with appropriate k-point sampling.

Go to *top*.

Linear Scaling

The range applied to the density matrix (`DM.L_range`) determines the accuracy of the calculation, as well as the computational time required (as the number of non-zero elements will increase based on a sphere with the radius of the range, the time will increase roughly proportional to the cube of the range). In almost all circumstances, it is best to operate with a range which converges energy *differences* and forces, rather than the absolute energy. Testing for this convergence is an essential part of the preparation for production calculations.

The tolerance applied to the density matrix optimisation (`minE.Ltolerance`) must be chosen to give adequate convergence of the energy and forces. The tolerance is applied to the residual in the calculation, defined as:

$$R = \sqrt{\sum_{i\alpha j\beta} \partial E/\partial L_{i\alpha j\beta} \cdot \partial E/\partial L_{i\alpha j\beta}}$$

The dot product uses the inverse of the overlap matrix as the metric.

The approximate, sparse inversion of the overlap matrix is performed before the optimisation of the density matrix. The method used, Hotelling's method (a version of a Newton-Raphson approach) is iterative and terminates when the characteristic quantity Ω increases. On termination, if Ω is below the tolerance `DM.InvSTolerance` then the inverse is accepted; otherwise it is set to the identity (the density matrix optimisation will proceed in this case, but is likely to be inefficient). We define:

$$\Omega = (Tr[I - TS])^2$$

where T is the approximate inverse. The range for the inverse must be chosen (`Atom.InvSRange` in the species block); by default it is same as the support function range (which is then doubled to give the matrix range) but can be increased. The behaviour of the inversion with range is not simple, and must be carefully characterised if necessary.

Go to *top*.

2.3.3 Self-consistency

The standard self-consistency approach uses the Pulay RMM method, and should be robust in most cases. It can be monitored via the residual, which is currently defined as the standard RMS difference in charge density:

$$R = \sqrt{\int d\mathbf{r} \mid \rho^{out}(\mathbf{r}) - \rho^{in}(\mathbf{r}) \mid^2}$$

where ρ^{in} is the input charge density for an iteration, and ρ^{out} is the resulting output charge density. The SCF cycle is terminated when this residual is less than the parameter `minE.SCTolerance`. The maximum number of iterations is set with `SC.MaxIters` (defaults to 50).

There are various further approaches and parameters which can be used if the SCF cycle is proving hard to converge. As is standard, the input for a given iteration is made by combining the charge density from a certain number of previous steps (`SC.MaxPulay`, default 5). The balance between input and output charge densities from these previous steps is set with `SC.LinearMixingFactor` (default 0.5; N.B. for spin polarised calculations, `SC.LinearMixingFactor_SpinDown` can be set separately). Reducing this quantity may well improve stability, but slow down the rate of convergence.

Kerker-style preconditioning (damping long wavelength charge variations) can be selected using `SC.KerkerPreCondition T` (this is most useful in metallic and small gap systems). The preconditioning is a weighting applied in reciprocal space:

$$K = \frac{1}{1 + q_0^2/q^2}$$

where q_0 is set with `SC.KerkerFactor` (default 0.1). This is often very helpful with slow convergence or instability.

Go to *top*.

2.3.4 Support Functions

The parameters relevant to support functions depend on the basis set that is used. In the case of pseudo-atomic orbitals (PAOs), when support functions are primitive PAOs, the only relevant parameter is the basis set size, which is set when the ion files are generated. It is important to test the accuracy of a given basis set carefully for the problem that is to be modelled.

When using multi-site support functions (MSSF), the key parameter is the radius of the MSSF (`Atom.MultisiteRange` in the *atomic specification* block). As this is increased, the accuracy of the calculation will also increase, but with increased computational effort. Full details of the MSSF (and related OSSF) approach are given in the section on *multi-site support functions*.

For the blip basis functions, the spacing of the grid where the blips are defined is key (`Atom.SupportGridSpacing` in the *atomic specification* block), and is directly related to an equivalent plane wave cutoff (via $k_{bg} = \pi/\delta$ and $E_{PW} = k_{bg}^2/2$, where δ is the grid spacing in Bohr radii and E_{PW} is in Hartrees). For a particular grid spacing, the energy will converge monotonically with support function radius (`Atom.SupportFunctionRange` in the *atomic specification* block). A small support function radius will introduce some approximation to the result, but improve computational performance. It is vital to characterise both blip grid spacing and support function radius in any calculation. A full discussion of the blip function basis is found *here*.

Go to *top*.

2.4 Basis sets

As we have mentioned in *finding the ground state*, the density matrix is represented by *support functions*. These, in turn, are made up of basis functions, and the choice of basis set and how it represents the support functions affects both the accuracy and performance of a calculation.

There are two kinds of basis functions which are used in CONQUEST to represent the support functions: pseudo-atomic orbitals (keyword `PAOs`) which are the default; and b-splines (keyword `blips`) which allow systematic convergence at the expense of greater complexity.

The basis set is selected as follows:

```
Basis.BasisSet PAOs
```

When the basis set is taken to be PAOs, there are three different ways to construct the support functions, discussed below:

- Each support function is represented by a single PAO (primitive PAOs)

- Multi-site support functions, built from PAOs on several atoms

- on-site support functions, built from PAOs on one atom

Primitive PAOs are efficient for small systems. When using large PAO basis sets for systems containing more than several hundred atoms, multi-site support functions and on-site support functions will be more efficient than primitive PAOs.

Go to *top*.

2.4.1 Pseudo-atomic orbitals (PAOs)

PAOs are solutions of the Schrodinger equation for isolated atoms, using pseudopotentials, with some confinement applied. They consist of radial functions multiplied by spherical harmonics. For the valence orbitals, the radial functions are referred to as *zeta* (ζ), while for unoccupied orbitals, they are termed *polarisation*.

A minimal PAO basis set is *single-ζ (SZ)*, with one radial function for each angular momentum quantum number in the valence electrons. While the cost is significantly lower than for other basis sets, the accuracy will be rather low.

The accuracy of a calculation can be improved by adding polarisation functions and multiple radial functions for different angular momentum values, though systematic improvement is rather difficult to achieve (this is straightforward with a *blip function basis*). The PAO utility included with CONQUEST generates basis sets with differing sizes and accuracies; full details of the performance of these basis sets can be found elsewhere [B1].

- minimal (single zeta)

- small (single zeta and polarisation, SZP)

- medium (double zeta, single polarisation, DZP)

- large (triple zeta, double polarisation, TZDP)

Go to *top*.

Primitive PAOs as support functions

The easiest way to prepare support functions is to use primitive PAOs as the support functions without any modifications. In this case, the input parameters related to the support functions are automatically set by obtaining the information from the PAO files (`.ion` files) so long as they are generated by the CONQUEST `MakeIonFiles` utility, version 1.0.3 or later. No further input parameters need to be set in `Conquest_input`.

Go to *top*.

Multi-site support functions

Since the computational cost of Conquest scales cubically with the number of support functions, contracting the PAOs into a smaller set of support functions is an efficient way to reduce the computational cost when we use large multiple-ζ PAO basis sets. Multi-site support functions (MSSFs) [B2][B3] are constructed for each atom by taking linear combinations of the atom's PAOs and the PAOs from neighbouring atoms within a certain range (set with the parameter `Atom.MultisiteRange` in the *atom specification block*).

Multi-site support functions can be selected by setting the following parameters:

```
Basis.BasisSet PAOs
Basis.MultisiteSF T
```

Various other parameters need to be set in the *atom specification block*. The number of support functions for the atoms must be set, and is normally equivalent to a minimal (single zeta) basis; it is set with `Atom.NumberOfSupports`. (To use a number of support functions larger than this minimal number, the parameter `Multisite.nonminimal` needs to be set to `T`.) The range for the multi-site support functions (the PAOs of any atom within this distance of the atom will be included in the support functions) is set with `Atom.MultisiteRange`. The accuracy of the MSSF will improve as this range is increased, though the computational cost will also increase; careful tests must be made to find an appropriate range. For a minimal number of MSSF, the range must be large enough to include other atoms, though this restriction can be removed (see *on-site support functions* for more details).

As well as setting the range for the MSSFs, we need to specify an approach for finding the expansion coefficients. A reasonable set of MSSF coefficients can be found using the *local filter diagonalization (LFD)* method. For improved accuracy, this should be followed by variational *numerical optimisation*.

Go to *top*.

Local filter diagonalization (LFD)

In this method, which is selected by setting `Multisite.LFD T`, the MSSF coefficients are found by diagonalising the Hamiltonian in the primitive PAO basis, for a small cluster of atoms surrounding the target atom. The MSSF coefficients C are determined by projecting the sub-space molecular orbitals C_{sub} around each atom onto localized *trial* vectors t,

$$C = C_{sub}f(\varepsilon_{sub})C_{sub}^{T}S_{sub}t$$

The cluster for diagonalisation must be at least as large as the MSSF range, but larger clusters tend to give better MSSF coefficients (at the expense of an increased computational cost). The LFD sub-space region is determined for each atom by setting `Atom.LFDRange`.

An example set of parameters for an MSSF calculation for bulk Si would be:

```
Basis.BasisSet PAOs
Basis.MultisiteSF T
Multisite.LFD T

%block ChemicalSpeciesLabel
1 28.07 Si
%endblock

%block Si
Atom.NumberOfSupports 4
Atom.MultisiteRange 8.0
Atom.LFDRange 8.0
%endblock
```

When calculating binding energy curves or optimising cells, a change of lattice constant can suddenly bring a new set of atoms within the range of the support functions. In this case, a smearing can be applied at the edges of the range, by setting `Multisite.Smear T`. Further details are *given below*.

Some form of self-consistency between the MSSF and the charge density is required (as the MSSF will determine the Hamiltonian and hence the output charge density). At present, this is performed as a complete SCF cycle for each set of MSSF coefficients (though this is likely to be updated soon for improved efficiency). This is selected by default (but can be turned off by setting the parameter `Multisite.LFD.NonSCF T`).

This iterative process is not variational, but is terminated when the absolute energy change between iterations is less than `Multisite.LFD.Min.ThreshE`, or the residual (defined in *self-consistency*) is less than `Multisite.LFD.Min.ThreshD`.

An example input block for this process would be as follows:

```
Multisite.LFD T
Multisite.LFD.Min.ThreshE 1.0e-6
Multisite.LFD.Min.ThreshD 1.0e-6
```

Go to *top*.

Numerical optimisation

The MSSF coefficients can also be optimised by minimizing the DFT energy with respect to the coefficients, in a variational process. The threshold and the maximum iteration number of the numerical optimisation are specified by `minE.EnergyTolerance` and `minE.SupportVariations`. The optimisation is based on the conjugate gradient (CG) method, and the initial CG step size can be specified by `minE.InitStep_paomin` (default is 5.0).

```
minE.VaryBasis T
minE.EnergyTolerance 1.0e-6
minE.SupportVariations 30
```

The numerical optimisation provides more accurate coefficients than the LFD method but is usually more time consuming. Therefore, it is generally better to start from good initial values, for example, the coefficients calculated by LFD. When both `Multisite.LFD` and `minE.VaryBasis` are selected, the initial coefficients will be calculated by LFD and the coefficients will then be optimised.

```
Basis.MultisiteSF T
Multisite.LFD T
minE.VaryBasis T
```

If good initial coefficient values have been found in a previous calculation, reading these from files (the base name of these files is `SFcoeffmatrix2`) and performing only the numerical optimisation is also a good choice.

```
Basis.LoadCoeffs T
Basis.MultisiteSF T
Multisite.LFD F
minE.VaryBasis T
```

Go to *top*.

Advanced MSSF concepts

Smearing the edge of the support functions Here, we are concerned with changes of lattice constant which may bring new atoms inside the support function range.

We can set the smearing-function type `Multisite.Smear.FunctionType` (default=1:Fermi-Dirac, 2=Error function), the center position of the function `Multisite.Smear.Center` (default is equal to the range of the support functions), offset of the center position `Multisite.Smear.Shift` and the width of the Fermi-Dirac function `Multisite.Smear.Width` (default=0.1).

Selecting states from the sub-space Here, we consider how to create the MSSF themselves from the results of the sub-space diagonalisation.

The Fermi function f with ε_{sub} `Multisite.LFD.ChemP` and kT `Multisite.LFD.kT` in the equation removes the effects of the subspace molecular orbitals in higher energy region. In default, ε_{sub} is automatically set to the mean value of the subspace HOMO and LUMO energies for each subspace. If users want to modify this, set `Multisite.LFD.UseChemPsub F` and the ε_{sub} value with `Multisite.LFD.ChemP`.

For the LFD trial functions t, when `Atom.NumberOfSupports` is equal to the number of SZ or single-zeta plus polarization (SZP), the PAOs which have the widest radial functions for each spherical harmonic function are chosen as the trial vectors automatically in default. When `Atom.NumberOfSupports` is equal to the number of SZP and `Multisite.nonminimal.offset` is set, the other PAOs will have the weight in the trial vectors with the value of `Multisite.nonminimal.offset`. The users can also provide the trial vectors from the input file using the `LFDTrialVector` block

```
# Trial vectors of Au (element 1) and O (element 2) atoms.
# Au: 15 PAOs (DZP) -> 6 support functions, O: 13 PAOs (DZP) -> 4 support functions.
%block LFDTrialVector
# species sf npao   s   s   x   y   z  d1  d2  d3  d4  d5  d1  d2  d3  d4  d5 for Au
        1  1   15 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        1  2   15 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
        1  3   15 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
        1  4   15 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
        1  5   15 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
        1  6   15 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
        2  1   13 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        2  2   13 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        2  3   13 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        2  4   13 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
# species sf npao   s   s   x   y   z   x   y   z  d1  d2  d3  d4  d5 for O
%endblock LFDTrialVector
```

The first, second and third columns correspond to the indices of species, support functions for each species, and the number of PAOs for each species. The other columns provide the initial values of the trial vectors. For example, in the first line in the above example, the second *s* PAO is chosen as the trial vector for the first support function of Au.

Self-consistent LFD Two further conditions are applied to end the LFD self-consistency process. The maximum number of iterations is set with `Multisite.LFD.Min.MaxIteration`. It is also possible, as the process is not variational, that the energy can increase as well as decrease between iterations. If the energy *increase* is less than `Multisite.LFD.Min.ThreshEnergyRise` (which defaults to ten times `Multisite.LFD.Min.ThreshE`) then convergence is deemed to have been reached.

Go to *top*.

On-site support functions

On-site support functions (OSSF) are similar to multi-site support functions, but are linear combinations of PAOs only on the target atom. In this case, `Atom.MultisiteRange` should be small enough not to include any neighboring atoms (suggested values between 0.1 to 0.5). The number of support functions must be equivalent to the number of functions in an SZP basis (if polarisation functions are in the basis set) or an SZ basis (if there are no polarisation functions). The parameter `Multisite.nonminimal` should be set to true if polarisation functions are included.

The coefficients can be determined in the same was as for MSSF (with the LFD method and/or the numerical optimisation described above). It is likely that significant improvement in accuracy will be found with numerical optimisation. It is also important to test the effect of the parameter `Atom.LFDRange` which should be large enough to include several shells of neighbouring atoms.

The OSSF approach is most likely to be useful when linear scaling calculations with large basis sets are required. An example set of parmeters is found below.

```
Basis.BasisSet PAOs
Basis.MultisiteSF T
Multisite.LFD T
Multisite.nonminimal T

minE.VaryBasis T

# example of Si
%block Si
Atom.NumberOfSupports 9
Atom.MultisiteRange 0.1
```

```
Atom.LFDRange 8.0
%endblock
```

Go to *top*.

2.4.2 Blips

Blips (which are a type of piecewise continuous polynomial called a B-spline) [B4] are useful for very accurate calculations, since the basis set can be systematically improved, in the same way as a planewave basis set. However, the calculations can be expensive depending on the parameters, and the code for blip optimisation is under development. The following description, and possible keywords, may change during development.

The blips are defined on a blip grid, which is a regular cubic grid centred on the atoms, which also moves with the atoms. The basis set can be systematically improved, by increasing the support function radius and/or reducing the spacing of the blip grids. (The support grid spacing, which defines the grid for the blips, is equivalent to a plane wave cutoff; for a given support grid spacing the energy decreases variationally with support function radius.) For each species, we need to provide these two parameters, as well as the number of support functions, which should have a minimal basis size. (At present, the smallest blip-grid spacing is used for all species.)

For a given atom, we would set:

```
%block atom
Atom.NumberOfSupports                    4
Atom.SupportFunctionRange               6.0
Atom.SupportGridSpacing                 0.3
%endblock
```

For each atomic species, an ion file with a minimal (SZ) basis set is required for the charge density and to initialise the blips.

The blip-grid spacing is directly related to the cutoff energy of the wavefunctions in planewave calculations. For a given cutoff energy E_{cutoff} in Hartree, the blip-grid spacing should be $\frac{2\pi}{\sqrt{2E_{\text{cutoff}}}}$ in bohr. Note that the grid spacing of integration grids (or FFT grids for the charge density) should be half the spacing of the blip grid, or smaller.

It is essential to optimise the support functions (blip coefficients) in the case of blips. The tolerance and maximum number of iterations can be set with the following keywords:

```
minE.VaryBasis              T
minE.EnergyTolerance              0.10E-07
minE.SupportVariations             30
```

It is not recommended, but if memory problems are encountered for very accurate blip calculations, you may need to switch off the preconditioning procedure for length-scale ill conditioning by setting the parameter `minE.PreconditionBlips F`

Go to *top*.

2.4.3 Reading coefficients from files

The calculated linear-combination coefficients of the support functions are stored in `SFcoeffmatrix2` files for PAOs or `blip_coeffs` files for blips. Those files can be read by setting `Basis.LoadCoeffs T` in the subsequent calculations.

Go to *top*.

2.4.4 Basis Set Superposition Error

Basis set superposition error (BSSE) arises when the two monomer units come closer and the basis set localized on one unit can act as diffuse functions for the electrons from the other unit, and therefore could be responsible for the overestimation of the binding energy for the interacting systems. It is unlikely to affect blip basis calculations [B5].

To correct this BSSE, the Counterpoise (CP) correction method [B6] is used, where the artificial stabilization is controlled by enabling the atoms in monomer calculations to improve their basis sets by including the basis sets from other monomers (using so-called ghost atoms).

When systems A and B approach and make a new system AB, the typical interaction energy between A and B is calculated as:

$$E_{AB}^{int} = E_{AB}(AB) - E_A(A) - E_B(B).$$

where $E_{AB}(AB)$ is the energy of system AB and $E_A(A)$ and $E_B(B)$ are the energies of isolated A and B. The lowerscript and parentheses correspond to the system and its structure, respectively.

Now, the estimate for the amount of artificial stabilization of A coming from the extra basis functions from B is:

$$E_A^{BSSE} = E_{A\bar{B}}(AB) - E_A(A \text{ in } AB),$$

where $\bar{A}$ and $\bar{B}$ are the ghost atoms, which have basis functions, but no potential or charge density. $E_{A\bar{B}}(AB)$ is the energy of system A with the basis sets from ghost-atom system B in the AB structure. $E_A(A \text{ in } AB)$ is the energy of system A in the AB structure but without system B (neither basis functions nor atoms). Therefore, the subtraction corresponds to how much system A is stabilized by the basis function of B.

Similarly, for monomer B,

$$E_B^{BSSE} = E_{\bar{A}B}(AB) - E_B(B \text{ in } AB),$$

Subtracting the BSSE part of A and B units from the typical interaction energy mentioned above, the counterpoise corrected interaction energy without BSSE ($E_{AB}^{int,CP}$) will be:

$$E_{AB}^{int,CP} = E_{AB}^{int} - E_A^{BSSE} - E_B^{BSSE}.$$

Practically, to calculate $E_{A\bar{B}}(AB)$, the basis functions of B should be placed on atomic centers of B, however with zero nuclear charge and mass. This can be performed in CONQUEST by specifying negative masses for the ghost atoms in B in the `block ChemicalSpeciesLabel` of the input file:

```
%block ChemicalSpeciesLabel
  1   1.01  A
  2  -1.01  B
%endblock
```

Go to *top*.

Go to *top*.

2.5 Structural relaxation

This section describes how to find the zero-Kelvin equilibrium structure, given a starting structure with non-zero forces and/or stresses. CONQUEST can employ a variety of algorithms algorithm to minimise energy with respect to atomic positions, including: L-BFGS; conjugate gradients; and damped molecular dynamics (both MDMin and FIRE approaches). The minimisation of energy or enthalpy with respect to cell vectors is restricted to conjugate gradients at present, though L-BFGS will be implemented.

Setting `AtomMove.WriteXSF T` for all flavours of optimisation will dump the trajectory to the file `trajectory.xsf`, which can be visualised using VMD. Setting `AtomMove.AppendCoords T` will append the structure at each step to `UpdatedAtoms.dat` in the format of a CONQUEST structure input.

For the L-BFGS and conjugate gradients relaxations, the progress of the calculation can be monitored by searching for the word `GeomOpt`; grepping will print the following:

```
$ grep GeomOpt Conquest_out
GeomOpt - Iter:    0 MaxF:   0.00329282 H:   -0.14168571E+03 dH:    0.00000000
GeomOpt - Iter:    1 MaxF:   0.00331536 H:   -0.14168995E+03 dH:    0.00424155
GeomOpt - Iter:    2 MaxF:   0.00350781 H:   -0.14168997E+03 dH:    0.00001651
GeomOpt - Iter:    3 MaxF:   0.00504075 H:   -0.14169161E+03 dH:    0.00164389
GeomOpt - Iter:    4 MaxF:   0.00725611 H:   -0.14169172E+03 dH:    0.00010500
GeomOpt - Iter:    5 MaxF:   0.01134145 H:   -0.14169329E+03 dH:    0.00157361
GeomOpt - Iter:    6 MaxF:   0.01417229 H:   -0.14169385E+03 dH:    0.00056077
GeomOpt - Iter:    7 MaxF:   0.01434628 H:   -0.14169575E+03 dH:    0.00190304
GeomOpt - Iter:    8 MaxF:   0.01711197 H:   -0.14170001E+03 dH:    0.00425400
GeomOpt - Iter:    9 MaxF:   0.02040556 H:   -0.14170382E+03 dH:    0.00381110
GeomOpt - Iter:   10 MaxF:   0.01095167 H:   -0.14170752E+03 dH:    0.00370442
```

In this example, MaxF is the maximum single force component, H is the enthalpy and dH is the change in enthalpy.

Go to *top*.

2.5.1 Ionic relaxation

To optimise the ionic positions with respect to the DFT total energy, the following flags are essential:

```
AtomMove.TypeOfRun lbfgs
AtomMove.MaxForceTol 5e-4
AtomMove.ReuseDM T
```

The parameter `AtomMove.TypeOfRun` can take the value `lbfgs` or `cg` for iterative optimisation. Both algorithms are robust and relatively efficient in most instances; L-BFGS is preferred. The parameter `AtomMove.MaxForceTol` specifies the force convergence criterion in Ha/bohr, i.e. the calculation will terminate when the largest force component on any atom is below this value. The parameter `AtomMove.ReuseDM` specifies that the density matrix (the K-matrix for diagonalisation or L-matrix for O(N) calculations) from the previous step will be used as an initial guess for the SCF cycle after propagating the atoms; this should generally decrease the number of SCF cycles per ionic step.

If the self-consistency tolerance is too low, the optimisation may fail to converge with respect to the force tolerance; this may necessitate a tighter `minE.SCTolerance` for diagonalisation (also possibly `minE.LTolerance` for O(N) calculations). A grid which is too coarse can also cause problems with structural relaxation to high tolerances.

For problematic cases where the conjugate gradients algorithm fails to find a downhill search direction, it may be worth trying quenched molecular dyanamics, which propagates the equations of motion following a simple NVE approach, but resets the velocities to zero when the dot product of force and velocity is zero.

```
AtomMove.TypeOfRun md
AtomMove.QuenchedMD T
AtomMove.MaxForceTol 5e-4
AtomMove.ReuseDM T
```

The FIRE algorithm [SR1] is a variant of quenched MD that has been shown to outperform conjugate gradients in some circumstances.

```
AtomMove.TypeOfRun md
AtomMove.FIRE T
AtomMove.MaxForceTol 5e-4
AtomMove.ReuseDM T
```

Go to *top*.

2.5.2 Cell optimisation

The unit cell can be optimised with respect to enthalpy *with fixed fractional coordinates* (`AtomMove.OptCellMethod 1`) using the following input:

```
AtomMove.TypeOfRun cg
AtomMove.OptCell T
AtomMove.OptCellMethod 1
AtomMove.TargetPressure 1.0
AtomMove.ReuseL T
AtomMove.EnthalpyTolerance 1E-6
AtomMove.StressTolerance 0.01
```

Here, we specify the target pressure in GPa and two new tolerances, the enthalpy tolerance in Ha and the stress tolerance in GPa.

Go to *top*.

2.5.3 Combined optimisation

For simple crystals, the fractional ionic coordinates vary trivially with changes in the lattice vectors; however for more complicated systems such as molecular crystals and amorphous materials, it is necessary simultaneously relax the ionic positions and lattice vectors. This can be done by setting `AtomMove.OptCellMethod 3`

```
AtomMove.TypeOfRun cg
AtomMove.OptCell T
AtomMove.OptCellMethod 3
AtomMove.TargetPressure 1.0
AtomMove.ReuseL T
AtomMove.MaxForceTol 5e-4
AtomMove.EnthalpyTolerance 1E-6
AtomMove.StressTolerance 0.01
```

Note that the enthalpy will generally converge much more rapidly than the force and stress, and that it may be necessary to tighten `minE.SCTolerance` (diagonalisation) or `minE.LTolerance` (order(N)) to reach the force tolerance, if it is even possible.

Due to the nature of the complex partitioning system, large and sudden changes in volume may cause the calculation to crash, particlularly in the case of combined optimisation. In such cases, it may help to try `AtomMove.OptCellMethod 2`, which uses a simple but robust double-loop minimisation: a full ionic conjugate gradients

relaxation for the inner loop and a single cell steepest descent relaxation for the outer loop. This is considerably less efficient, but may help in particularly problematic cases.

Go to *top*.

Go to *top*.

2.6 Molecular Dynamics

CONQUEST can perform molecular dynamics both when the density matrix is computed using diagonalisation and O(N), the latter allowing dynamical simulations of (but not limited to) tens of thousands of atoms. The equations of motion are integrated using the velocity Verlet method in the case of the microcanonical ensemble (NVE), and modifications thereof for the canonical (NVT) and isobaric-isothermal (NPT) ensembles, the details of which can be found in *Molecular Dynamics: Theory*. In addition to converging the parameters for the electronic structure calculations, the following points must also be considered.

Go to *top*.

2.6.1 Self-consistency tolerance and XL-BOMD

The convergence of the electronic structure is important in MD, as insufficient convergence can be responsible for "drift" in the conserved quantity of the dynamics. Although the molecular dynamics integrators used in CONQUEST are time reversible, *the SCF procedure is not*. Therefore tight convergence (`minE.SCTolerance` for diagonalisation, `minE.LTolerance` for linear scaling) is necessary. In the case of diagonalisation, SCF tolerance of `1E-6` is typically enough to negate the drift. However, extended-Lagrangian Born-Oppenheimer MD (XL-BOMD) [MD1], currently only implemented for O(N), essentially makes the SCF component of the MD time-reversible by adding the electronic degrees of freedom to the Lagrangian, relaxing the constraint on `minE.LTolerance` — although it is still somewhat dependent on the ensemble. In the NVE and NVT ensembles, a L-tolerance of `1E-5` has been found to be sufficient to give good energy conservations, decreasing to `1E-6` in the NPT ensemble. The following flags are required for XL-BOMD:

```
DM.SolutionMethod ordern
AtomMove.ExtendedLagrangian T
```

Go to *top*.

2.6.2 Restarting

Assuming the calculation ended gracefully, it can easily be restarted by setting,

```
AtomMove.RestartRun T
```

This will do several things: it will read the atomic coordinates from `md.position` and read the `md.checkpoint` file, which contains the velocities and extended system (Nose-Hoover chain and cell) variables. Depending on the value of `DM.SolutionMethod`, it will read the K-matrix files (`diagon`) or the L-matrix files (`ordern`), and if XL-BOMD is being used, the X-matrix files. Finally, it will *append* new data to the `md.stats` and `md.frames` files, but it will overwrite all other files, including `Conquest_out`. Note that this flag is equivalent to setting the following:

```
General.LoadL T
SC.MakeInitialChargeFromK T
XL.LoadL T
```

In addition to the files mentioned above, CONQUEST will try to read the K-matrix from `Kmatrix2.i00.*` when using diagonalisation or the L-matrix from `Lmatrix2.i00.*` when using O(N), and `Xmatrix2.i0*.*` if the extended-Lagrangian formalism is used. Note that metadata for these files is stored in `InfoGlobal.i00.dat` which is also required when restarting. If the calculation ended by hitting the walltime limit, the writing of these matrix files may have been interrupted, rendering them unusable. In this case, the calculation can be restarted by setting the above flags to F *after* setting `AtomMove.RestartRun T`. Setting the flag `General.MaxTime` to some number of seconds less (say 30 minutes) than the calculation wall time limit will force the calculation to stop gracefully, preventing the aforementioned situation.

Go to *top*.

2.6.3 Visualising the trajectory

Setting the flag `AtomMove.WriteXSF T` dumps the coordinates to the file `trajectory.xsf` every `AtomMove.OutputFreq` steps. The .xsf file can be read using VMD. A small VMD script, `view.vmd` is included with the code, and can be invoked using,

```
vmd -e view.vmd
```

assuming the vmd executable is in your path.

Go to *top*.

2.6.4 TDEP output

CONQUEST molecular dynamics data can be used to perform lattice dyanmical calculations using the Temperature Dependent Effective Potential (TDEP) code. Setting the flag `MD.TDEP T` will make conquest dump configurations, forces and metadata in a format readable by TDEP.

Go to *top*.

2.6.5 Non-Hamiltonian dynamics

Canonical (NVT) ensemble

The thermostat is set using the `MD.Thermostat` flag, and can take the values `svr` (stochastic velocity rescaling) and `nhc` (Nose-Hoover chain). These thermostats generate the correct canonical ensemble phase space distribution, and both give a conserved quantity that allows the quality of the dynamics to be monitored.

1. Stochastic velocity rescaling

```
AtomMove.IonTemperature 300.0
MD.Ensemble nvt
MD.Thermostat svr
MD.tauT 10
```

While the NHC uses chaotic sensitivity to initial conditions to achieve better ergodicity, the SVR thermostat [MD2] uses a judiciously chosen stochastic force coupled to a weak scaling thermostat to correctly generate the canonical phase space distribution. The `MD.tauT` parameter gives the coupling timescale; the velocity scaling factor is modified by a factor $\Delta t/\tau$, so a larger τ results in a more slowly varying temperature. While some characterisation of the system is recommended, values of τ around 20–200fs are reasonable. To reproduce a simulation, the random number generator seed can be set with the `General.RNGSeed <integer>` flag.

2. Nose-Hoover chain

```
AtomMove.IonTemperature 300.0
MD.Ensemble nvt
MD.Thermostat nhc
MD.nNHC 5
MD.nYoshida 5
MD.tauT 30
```

When thermostatting using a Nose-Hoover chain [MD3][MD4][MD5], it may be necessary to set a couple more flags. `MD.nNHC` sets the number of thermostats in the chain (the default of 5 is generally sensible), and `MD.nYoshida` determines the order of Yoshida-Suzuki integration. This is essentially a higher level integration scheme that *can* improve energy conservation in cases when rapid changes in the Nose-Hoover thermostat velocity is causing integration errors. **Note that** `MD.tauT` **means something different to the SVR case**. A good guess is the time period of the highest frequency motion of the system in fs; however, in the NVT ensemble, the energy conservation is not very sensitive to this value. The NHC masses can also be set manually using the following block.

```
MD.CalculateXLMass F
MD.nNHC 5
%block MD.NHCmass
  5 1 1 1 1
%endblock
```

Go to *top*.

Isobaric-Isothermal (NPT) ensemble

There is one implemented barostat at present, the extended system, Parrinello-Rahman [MD6]. At present the barostat should be treated as a beta-version implementation, which will be fully characterised and made robust for the full release of the code.

1. Parrinello-Rahman

```
AtomMove.IonTemperature 300.0
AtomMove.TargetPressure 10.0
MD.Ensemble npt
MD.Thermostat nhc
MD.Barostat pr
MD.nNHC 5
MD.nYoshida 5
MD.tauT 100
MD.tauP 200
MD.PDrag 10.0
```

The Parrinello-Rahman barostat generates the correct ensemble, but can be subject to low frequency "ringing" fluctuations in the temperature and pressure that can destabilise the system or slow equilibration. Unlike in the NVT ensemble, this combination of barostat and thermostat is *very* sensitive to the choice of both `MD.tauT` and `MD.tauP`; note that their values are somewhat higher in this case, since integration errors in the NHC tend to be more severe due to coupling of the cell and atomic motions. They are dependent on the system, so it is advised that you find a combination of these parameters that gives the best energy conservation. The cell is thermostatted using a separate Nose-Hoover chain to the atoms by default, but they can be controlled with the same chain by setting `MD.CellNHC` F. An *ad hoc* drag factor specified by `MD.PDrag` reduces the thermostat and cell velocities at every timestep to damp out the ringing fluctuations. In this case, they are reduced by $10/200 \simeq 5\%$, which strictly speaking breaks the NPT dynamics, but not significantly, and the stability is significantly improved.

Note that the NPT ensemble can also be generated correctly by thermostatting using the SVR thermostat, although the meaning of the parameter `MD.tauT` is different in this case, as in NVT dynamics.

2.6.6 Postprocessing tools

Details of Python post-processing tools for CONQUEST can be found in *Molecular dynamics analysis*.

Go to *top*.

Go to *top*.

2.7 External tools

2.7.1 Molecular dynamics analysis

Several scripts that may be helpful with postprocessing molecular dynamics are included with CONQUEST. The can be found in the `tools` directory, and the executables are `plot_stats.py`, `md_analysis.py` and `heat_flux.py`. They have the following dependencies:

- Python 3
- Scipy/Numpy
- Matplotlib

If Python 3 is installed the modules can be added easily using `pip3 install scipy` etc.

These scripts should be run in the calculation directory, and will automatically parse the necessary files, namely `Conquest_input`, `input.log`, `md.stats` and `md.frames` assuming they have the default names. They will also read the CONQUEST input flags to determine, for example, what ensemble is used, and process the results accordingly.

Go to *top*.

Plotting statistics

```
usage: plot_stats.py [-h] [-c] [-d DIRS [DIRS ...]]
                     [--description DESC [DESC ...]] [--skip NSKIP]
                     [--stop NSTOP] [--equil NEQUIL] [--landscape]
                     [--mser MSER_VAR]

Plot statistics for a CONQUEST MD trajectory

optional arguments:
  -h, --help            show this help message and exit
  -c, --compare         Compare statistics of trajectories in directories
                        specified by -d (default: False)
  -d DIRS [DIRS ...], --dirs DIRS [DIRS ...]
                        Directories to compare (default: .)
  --description DESC [DESC ...]
                        Description of graph for legend (only if using
                        --compare) (default: )
  --skip NSKIP          Number of equilibration steps to skip (default: 0)
  --stop NSTOP          Number of last frame in analysis (default: -1)
  --equil NEQUIL        Number of equilibration steps (default: 0)
  --landscape          Generate plot with landscape orientation (default:
                        False)
  --mser MSER_VAR       Compute MSER for the given property (default: None)
```

Running `plot_stats.py --skip 200` in your calculation will generate a plot which should resemble the example below, skipping the first 200 steps. This example is a molecular dynamics simulation of 1000 atoms of bulk silicon in the NPT ensemble, at 300 K and 0.1 GPa.
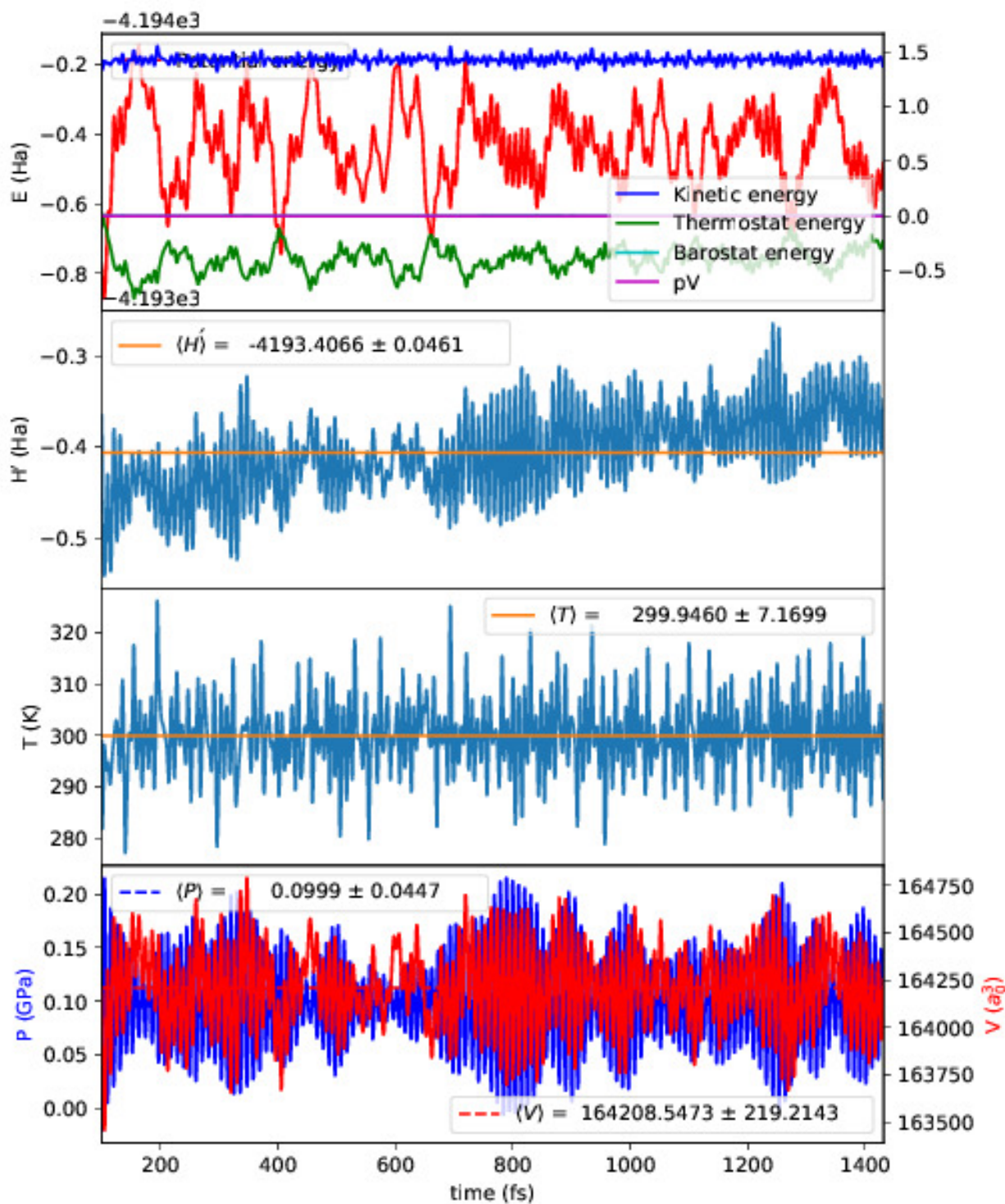
The four plots are respectively the breakdown of energy contributions, the conserved quantity, the temperature and the pressure, the last of which is only included for NPT molecular dynamics. Several calculations in different directories can be compared using `plot_stats.py --compare -d dir1 dir2 --description "dir1`

2.7. External tools

description" "dir2 description". The following example compares the effect of changing the L toler-ance in the above simulation. Note that the contents of the description field will be in the legend of the plot.
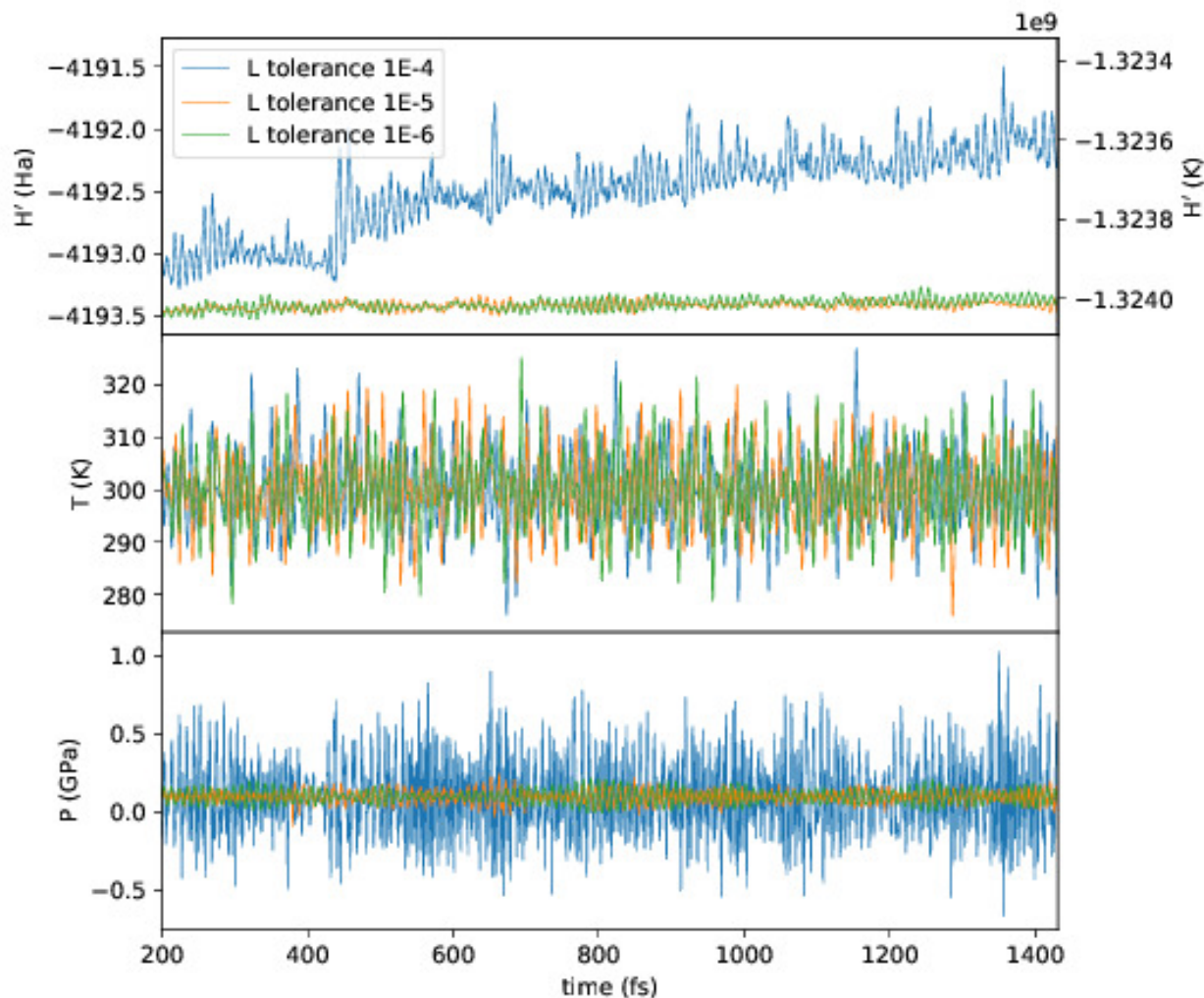
Go to *top*.

MD analysis

```
usage: md_analysis.py [-h] [-d DIRS [DIRS ...]] [--skip NSKIP]
                      [--stride STRIDE] [--snap SNAP] [--stop NSTOP]
                      [--equil NEQUIL] [--vacf] [--msd] [--rdf] [--stress]
                      [--nbins NBINS] [--rdfwidth RDFWIDTH] [--rdfcut RDFCUT]
                      [--window WINDOW] [--fitstart FITSTART] [--dump]

Analyse a CONQUEST MD trajectory

optional arguments:
  -h, --help            show this help message and exit
  -d DIRS [DIRS ...], --dirs DIRS [DIRS ...]
                        Directories to compare (default: .)
  --skip NSKIP          Number of equilibration steps to skip (default: 0)
  --stride STRIDE       Only analyse every nth step of frames file (default:
```

(continues on next page)

```
                            1)
  --snap SNAP               Analyse Frame of a single snapshot (default: -1)
  --stop NSTOP              Number of last frame in analysis (default: -1)
  --equil NEQUIL            Number of equilibration steps (default: 0)
  --vacf                    Plot velocity autocorrelation function (default:
                            False)
  --msd                     Plot mean squared deviation (default: False)
  --rdf                     Plot radial distribution function (default: False)
  --stress                  Plot stress (default: False)
  --nbins NBINS             Number of histogram bins (default: 100)
  --rdfwidth RDFWIDTH       RDF histogram bin width (A) (default: 0.05)
  --rdfcut RDFCUT           Distance cutoff for RDF in Angstrom (default: 8.0)
  --window WINDOW           Window for autocorrelation functions in fs (default:
                            1000.0)
  --fitstart FITSTART       Start time for curve fit (default: -1.0)
  --dump                    Dump secondary data used to generate plots (default:
                            False)
```

The script `md_analysis.py` script performs various analyses of the trajectory by parsing the *md.frames`* file. So far, these include the radial distribution function, the velocity autocorrelation function, the mean squared deviation, and plotting the stress. For example, the command,

```
md_analysis.py --rdf --stride 20 --rdfcut 8.0 --nbins 100 --dump --skip 200
--stop 400
```

computes the radial distribution function of the simulation in the first example from every 20th time step (every 10 fs in this case), stopping after 400 steps, with a cutoff of 8.0 A, and the histogram is divided into 100 bins.

Go to *top*.

2.7. External tools
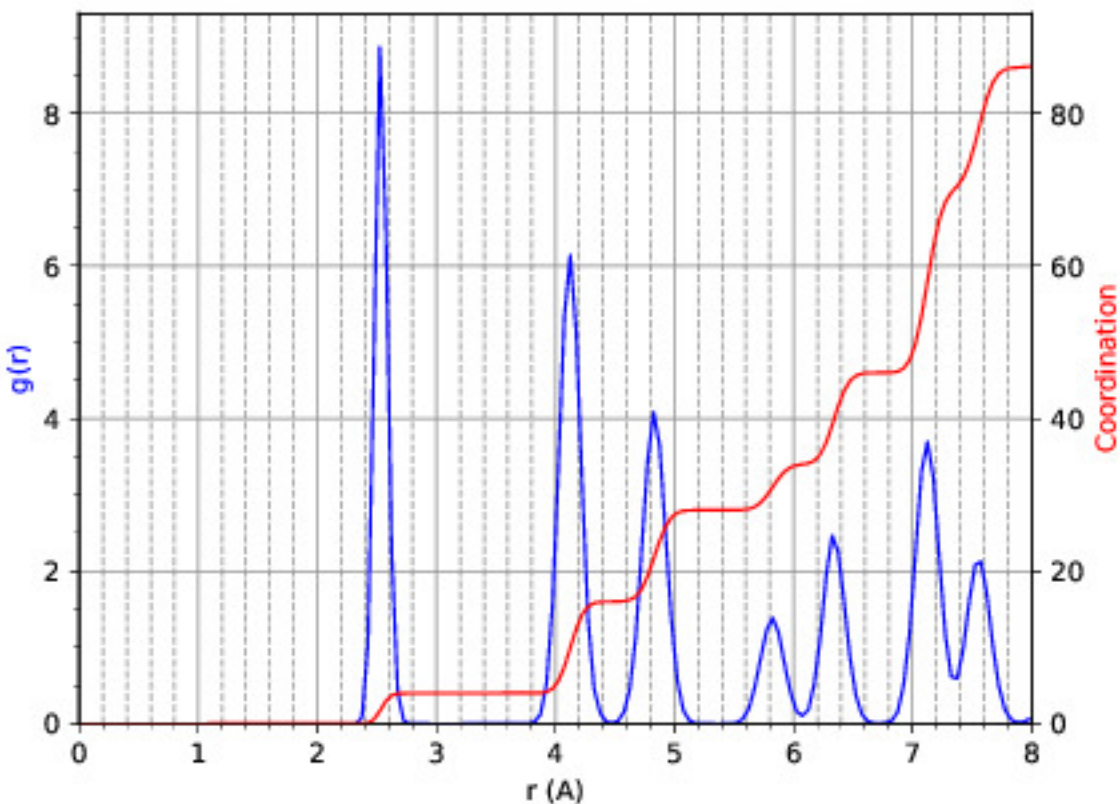
CONQUEST structure file analysis

```
usage: structure.py [-h] [-i INFILE] [--bonds] [--density] [--nbins NBINS]
                    [-c CUTOFF [CUTOFF ...]] [--printall]

Analyse a CONQUEST-formatted structure

optional arguments:
  -h, --help            show this help message and exit
  -i INFILE, --infile INFILE
                        CONQUEST format structure file (default:
                        coord_next.dat)
  --bonds               Compute average and minimum bond lengths (default:
                        False)
  --density             Compute density (default: False)
  --nbins NBINS         Number of histogram bins (default: 100)
  -c CUTOFF [CUTOFF ...], --cutoff CUTOFF [CUTOFF ...]
                        Bond length cutoff matrix (upper triangular part, in
                        rows (default: None)
  --printall            Print all bond lengths (default: False)
```

The script `structure.py` can be used to analyse a CONQUEST-formatted structure file. This is useful to sanity-check the bond lengths or density, since an unphysical structure is so often the cause of a crash. For example, the bond lengths can be computed with

```
structure.py --bonds -c 2.0 3.0 3.0
```

where the `-c` flag specifies the bond cutoffs for the bonds 1-1, 1-2 and 2-2, where 1 is species 1 as specified in `Conquest_input` and 2 is species 2. The output will look something like this:

```
Mean bond lengths:
O-Si:    1.6535 +/-    0.0041 (24 bonds)
Minimum bond lengths:
O-Si:    1.6493
```

Go to *top*.

2.7.2 Atomic Simulation Environment (ASE)

ASE is a set of Python tools for setting up, manipulating, running, visualizing and analyzing atomistic simulations. ASE contains a CONQUEST interface, so that it can be used to calculate energies, forces and stresses for calculations that CONQUEST can't do (yet). Detailed instructions on how to install and invoke it can be found on its website, but we provide some details and examples for the CONQUEST interface here.

Note that the script will need to set environmental variables specifying the locations of the CONQUEST executable `Conquest`, and if required, the basis set generation executable `MakeIonFiles` and pseudopotential database.

```python
import os

# The command to run CONQUEST in parallel
os.environ["ASE_CONQUEST_COMMAND"] = "mpirun -np 4 /path/to/Conquest_master"
# Path to a database of pseudopotentials (for basis generation tool)
os.environ["CQ_PP_PATH"] = "~/Conquest/PPDB/"
# Path to the basis generation tool executable
os.environ["CQ_GEN_BASIS_CMD"] = "/path/to/MakeIonFiles"
```

Go to *top*.

Keywords for generating the Conquest_input file

The calculator object contains a dictionray containing a small number of mandatory keywords, listed below:

```
default_parameters = {
    'grid_cutoff'   : 100,      # DFT defaults
    'kpts'          : None,
    'xc'            : 'PBE',
    'scf_tolerance' : 1.0e-6,
    'nspin'         : 1,
    'general.pseudopotentialtype' : 'Hamann', # CONQUEST defaults
    'basis.basisset'            : 'PAOs',
    'io.iprint'                 : 2,
    'io.fractionalatomiccoords' : True,
    'mine.selfconsistent'       : True,
    'sc.maxiters'               : 50,
    'atommove.typeofrun'        : 'static',
    'dm.solutionmethod'         : 'diagon'}
```

The first five key/value pairs are special DFT parameters, the grid cutoff, the k-point mesh, the exchange-correlation functional, the SCF tolerance and the number of spins respectively. The rest are CONQUEST-specific input flags.

The atomic species blocks are handled slightly differently, with a dictionary of their own. If the .ion files are present in the calculation directory, they can be specified as follows:

```
basis = {"H": {"valence_charge": 1.0,
               "number_of_supports": 1,
               "support_fn_range": 6.9},
         "O": {"valence_charge": 6.0,
               "number_of_supports": 4,
               "support_fn_range": 6.9}}
```

If the basis set .ion files are present in the directory containing the ASE script are pressent and are named `element.ion`, then the relevant parameters will be parsed from the .ion files and included when the input file is written and this dictionary can be omitted. It is more important when, for example, setting up a multisite calculation, when the number of contracted support functions is different from the number in the .ion file.

ASE can also invoke the CONQUEST basis set generation tool, although care should be taken when generating basis sets:

```
basis = {"H": {"basis_size": "minimal",
               "pseudopotential_type": hamann",
               "gen_basis": True},
         "O": {"basis_size": "minimal",
               "pseudopotential_type": hamann",
               "gen_basis": True}}
```

Finally, non-mandatory input flags can be defined in a new dictionary, and passed as an expanded set of keyword arguments.

```
conquest_flags = {'IO.Iprint'        : 1,          # CONQUEST keywords
                  'DM.SolutionMethod' : 'ordern',
                  'DM.L_range'        : 8.0,
                  'minE.LTolerance'   : 1.0e-6}
```

Here is an example, combining the above. We set up a cubic diamond cell containing 8 atoms, and perform a single point energy calculation using the order(N) method (the default is diagonalisation, so we must specify all of the

order(N) flags). We don't define a basis set, instead providing keywords that specify that a minimal basis set should be constructed using the MakeIonFiles basis generation tool.

```python
from ase.build import bulk
from ase.calculators.conquest import Conquest

os.environ["ASE_CONQUEST_COMMAND"] = "mpirun -np 4 Conquest_master"
os.environ["CQ_PP_PATH"] = "/Users/zamaan/Conquest/PPDB/"
os.environ["CQ_GEN_BASIS_CMD"] = "MakeIonFiles"

diamond = bulk('C', 'diamond', a=3.6, cubic=True)  # The atoms object
conquest_flags = {'IO.Iprint'        : 1,          # Conquest keywords
                  'DM.SolutionMethod' : 'ordern',
                  'DM.L_range'        : 8.0,
                  'minE.LTolerance'   : 1.0e-6}
basis = {'C': {"basis_size"          : 'minimal', # Generate a minimal basis
               "gen_basis"           : True,
               "pseudopotential_type" : "hamann"}}

calc = Conquest(grid_cutoff = 80,     # Set the calculator keywords
                xc="LDA",
                self_consistent=True,
                basis=basis,
                nspin=1,
                **conquest_flags)
diamond.set_calculator(calc)          # attach the calculator to the atoms object
energy = diamond.get_potential_energy()  # calculate the potential energy
```

Go to *top*.

Multisite support functions

Multisite support functions require a few additional keywords in the atomic species block, which can be specified as follows:

```python
basis = {'C': {"basis_size": 'medium',
               "gen_basis": True,
               "pseudopotential_type": "hamann",
               "Atom.NumberofSupports": 4,
               "Atom.MultisiteRange": 7.0,
               "Atom.LFDRange": 7.0}}
```

Note that we are constructing a DZP basis set (size medium) with 13 primitive support functions using MakeIonFiles, and contracting it to multisite basis of 4 support functions. The calculation requires a few more input flags, which are specified in the other_keywords dictionary:

```python
other_keywords = {"Basis.MultisiteSF": True,
                  "Multisite.LFD": True,
                  "Multisite.LFD.Min.ThreshE": 1.0e-7,
                  "Multisite.LFD.Min.ThreshD": 1.0e-7,
                  "Multisite.LFD.Min.MaxIteration": 150,
                  }
```

Go to *top*.

Loading the K/L matrix

Most calculation that involve incrementally moving atoms (molecular dynamics, geometry optimisation, equations of state, nudged elastic band etc.) can be made faster by using the K or L matrix from a previous calculation as the initial guess for a subsequent calculation in which that atoms have been moved slightly. This can be achieved by first performing a single point calculation to generate the first K/L matrix, then adding the following keywords to the calculator:

```
other_keywords = {"General.LoadL": True,
                  "SC.MakeInitialChargeFromK": True}
```

These keywords respectively cause the K or L matrix to be loaded from file(s) `Kmatrix.i**.p*****`, and the initial charge density to be constructed from this matrix. In all subsequent calculations, the K or L matrix will be written at the end of the calculation and used as the initial guess for the subsequent ionic step.

Go to *top*.

Equation of state

The following code computes the equation of state of diamond by doing single point calculations on a uniform grid of the `a` lattice parameter. It then interpolates the equation of state and uses `matplotlib` to generate a plot.

```python
import scipy as sp
from ase.build import bulk
from ase.io.trajectory import Trajectory
from ase.calculators.conquest import Conquest


# Construct a unit cell
diamond = bulk('C', 'diamond', a=3.6, cubic=True)

basis = {'C': {"basis_size": 'minimal',
               "gen_basis": True,
               "pseudopotential_type": "hamann"}}
calc = Conquest(grid_cutoff = 50,
                xc = "LDA",
                basis = basis,
                kpts = [4,4,4]}
diamond.set_calculator(calc)

cell = diamond.get_cell()
traj = Trajectory('diamond.traj', 'w') # save all results to trajectory

for x in sp.linspace(0.95, 1.05, 5):   # grid for equation of state
  diamond.set_cell(cell*x, scale_atoms=True)
  diamond.get_potential_energy()
  traj.write(diamond)

from ase.io import read
from ase.eos import EquationOfState

configs = read('diamond.traj@0:5')
volumes = [diamond.get_volume() for diamond in configs]
energies = [diamond.get_potential_energy() for diamond in configs]
eos = EquationOfState(volumes, energies)
v0, e0, B = eos.fit()
```

```python
import matplotlib
eos.plot('diamond-eos.pdf')      # Plot the equation of state
```

Go to *top*.

2.8 Error codes

Error codes generated by different CONQUEST routines will be collated here, along with explanations of the root cause and suggested fixes. This will form part of the full release, but is not implemented in the pre-release.

2.9 Input tags

We have broken down the input tags based on the areas of the code where they apply. For each tag, a default is given. Types of value are specified as: *integer*; *real*; *boolean*; or *string* (optA/optB are given for string options).

Areas

- *General*
- *Atomic Specification*
- *Input-Output General Tags*
- *Atomic Coordinates*
- *Levels of Output*
- *Integration Grid*
- *Minimising Energy*
- *Charge Self-Consistency*
- *Density Matrix*
- *Diagonalisation*
- *Moving Atoms*
- *Molecular Dynamics*
- *Spin Polarisation*
- *DeltaSCF*
- *Constrained DFT (cDFT)*
- *vdW-DF*
- *DFT-D2*
- *XL-BOMD*
- *Advanced and obscure tags*

2.9.1 General

General.Title (*string*) Title for the calculation

> *default*: none

General.NumberOfSpecies (*integer*) Number of species in cell

> *default*: none

General.PseudopotentialType (*string*) siesta/hamann Type of pseudopotential (in practice, this defines how the local part of the pseudopotential is handled)

> *default*: hamann (read from ion file)

General.NeutralAtom (*boolean*) Use neutral atom potential or not (removes need for Ewald sum)

> *default*: T

General.FunctionalType (*integer*) Selects the exchange-correlation functional. If the native CONQUEST XC implementation is used, there are three parameterisations of the LDA available, as well as three variants of the PBE GGA functional, with numbers given below.

> *default*: read from ion file (same as pseudopotentials)

Functional	Keyword	Ref
LDA Perdew-Zunger, no SIC	1	[E1]
LDA Goedecker-Teter-Hutter 96	2	[E2]
LSDA Perdew-Wang 92 (default)	3	[E3]
GGA Perdew-Burke-Ernzerhof 96 (PBE)	101	[E4]
GGA PBE + Zhang-Yang 98 (revPBE)	102	[E5]
GGA PBE + Hammer-Hansen-Norskov 99 (RPBE)	103	[E6]
GGA WC	104	[E7]

At the moment, only LSDA Perdew-Wang 92 and the three GGA Perdew-Burke-Ernzerhof functional variants can be used in spin polarised calculations.

Note that, if the code is compiled with LibXC, the full LibXC set of functionals is available, selected with a negative six digit number (-XXXCCC or -CCCXXX).

General.EnergyUnits (*string*) Ha/Ry/eV Output only Chooses units for energy

> *default*: Ha

General.DistanceUnits (*string*) a0/bohr/A Output only Chooses units for distance (Bohr: a0/bohr or Ångströms: A

> *default*: a0

General.MemoryUnits (*string*) kB/MB/GB Output only Chooses units for memory use

> *default*: MB

General.PartitionMethod (*string*) File/Hilbert Chooses method for partitioning (read from file or use dynamic partitioner based on Hilbert curve)

> *default*: Hilbert

> Options:

> - Hilbert (default) — Automatic partitioning using Hilbert curves; safe for initial use though optimum load balancing *not* guaranteed

> - File — Reads a file (NOT recommended)

General.LoadBalance (*string*) partitions/atoms Applies to Hilbert above; chooses whether to distribute atoms or partitions evenly between processors (you are *strongly* recommended to use atoms)

> *default*: atoms

General.ManyProcessors (*boolean*) Applies to Hilbert above; chooses method for parallelising Hilbert curve work; "many" processors here probably means more than two

> *default*: T

General.MaxAtomsPartition (*integer*) Applies to Hilbert above; specifies maximum number of atoms allowed in a partition; triggers extra level of recursion in partitioner

> *default*: 34

General.NPartitions[X/Y/Z] (*integer*) Allows the user to specify the number of partitions in x, y and z directions

> *default*: 0 (i.e. use Hilbert partitioning, above)

General.NewRun (*boolean*) Switches between new run and restart (N.B. restart has *not* been implemented yet)

> *default*: T

General.LoadL (*boolean*) Specifies whether to load a previous L matrix from files

> *default*: F

General.LoadRho (*boolean*) Specifies whether to load a previous charge density from files

> *default*: F

General.NetCharge (*real*) Specifies net charge on unit cell; implemented rather crudely with a neutralising background charge assumed. Note that a *positive* value indicates *excess* electrons

> *default*: 0.0

General.EwaldAccuracy (*real*) Accuracy for ewald sum (in Ha/atom)

> *default*: 10^{-10}

General.TimeThreshold (*real*) Minimum time for a timer to be printed (in seconds)

> *default*: 0.001

General.vdWDFT (*boolean*) Selects vdW DF

> *default*: F

General.DFT_D2 (*boolean*) Selects DFT-D2

> *default*: F

General.MaxTime (*real*) Maximum wall time for calculation in seconds. Conquest will exit gracefully on completion of an ionic relaxation/MD step

> *default*: 0.0

General.RNGSeed (*integer*) Seed for the random number generator. If less than 0, a random seed will be generated, otherwise the specified seed is used, and the same sequence of random numbers will be generated every time. Useful for reproducing MD runs.

> *default*: -1

Go to *top*.

2.9.2 Atomic Specification

ChemicalSpeciesLabel (*block*) Lists all atomic species used in the calculation. Format:

```
1 atomic_mass1 element_label_1
2 atomic_mass2 element_label_2
...
n atomic_mass_n_ element_label_n
```

(Note that the block must end with %endblock ChemicalSpeciesLabel.) 1--n are integer numbers used in the coordinate file to identify atomic species, as discussed in the *Coordinates* section. The atomic masses are only used for dynamics. The element labels should have a corresponding ion file element_label_x.ion and *may* have an accompanying atom specification block.

There can then be up to n atom specification blocks whose names should be element_label_x. When using *primitive PAOs* for support functions many of these are read from the ion file.

Atom.MultisiteRange (*real*) Range for multi-site support functions (the PAOs on all atoms within this range will be included in the support function)

default: 0.0

Atom.LFDRange (*real*) Range for local filter diagonalisation (the Hamiltonian and overlap matrix elements from all atoms within this range will be included in the cluster diagonalisation)

default: 0.0

Go to *top*.

2.9.3 Input-Output General Tags

IO.Title (*string*) Title for run

default: none

IO.Partitions (*string*) Name for file containing distribution of partitions over processors (generated by accompanying utilities)

default: make_prt.dat

IO.WriteOutToFile (*boolean*) Specifies whether the main output information is written to standard output or to a file

default: T

IO.OutputFile (*string*) Name for the main output file

default: Conquest_out

IO.DumpL (*boolean*) Whether to write the auxiliary matrices L to file at each self-consistent steps

default: T

IO.TimingOn (*boolean*) Whether time information will be measured and written to output

default: F

IO.TimeAllProcessors (*boolean*) Specifies whether time information will be written for all processors or just for the input/output process (the default)

default: F

IO.WriteTimeFile (*boolean*) Whether time files are written or not. This flag will be ignored if `IO.TimeAllProcessors` is true, in which case time files are always written.

> *default*: T

IO.TimeFileRoot (*string*) Root to be used in the time files, with an extension indicating the processor number, e.g. `.001`

> *default*: `time`

Go to *top*.

2.9.4 Atomic Coordinates

IO.Coordinates (*string*) Specifies the file with atomic coordinates. See *Coordinates* for details on the file format

> *default*: none

IO.FractionalAtomicCoords (*boolean*) Specifies whether fractional or absolute (Cartesian) coordinates are used in the coordinate file

> *default*: T

IO.PdbIn (*boolean*) Switches between the coordinate file format (F) and PDB format (T)

> *default*: F

Go to *top*.

2.9.5 Levels of Output

The overall level of output is controlled by **IO.Iprint** and can be fine-tuned with the other IO.Iprint keywords. These are by default set to the value of iprint, but that will be over-ridden if setting them explicitly. For instance, IO.Iprint could be set to 0, but IO.Iprint_MD could be set to 2 giving more extensive information about atomic movements but little other information.

N.B. At beta release, these levels of output are still being tuned; level 0 is reliable, and generally fairly minimal.

IO.Iprint (*integer*) The amount of information printed out to the output file The larger the value the more detailed the output is.

> 0 Basic information about the system and the run
> 1 Breakdown of energies, and details of the SCF cycle
> 2 Matrix range info, matrix multiplication details (covering set), partition details and general parallelisation info.
> 3 Subroutines called, messages upon entering/quitting subroutines
> 4 Details including internal variables of subroutines
> 5 Don't do this.

> *default*: 0

IO.Iprint_init (*integer*) The initialisation process

IO.Iprint_mat (*integer*) Matrix operations

IO.Iprint_ops (*integer*) Creation of operators H and S

IO.Iprint_DM (*integer*) Density matrix

IO.Iprint_SC (*integer*) Self-consistency

IO.Iprint_minE (*integer*) Energy minimisation

IO.Iprint_MD (*integer*) Molecular dynamics

IO.Iprint_index (*integer*) Indexing routines

IO.Iprint_gen (*integer*) General (not covered by other areas)

IO.Iprint_pseudo (*integer*) Pseudopotentials

IO.Iprint_basis (*integer*) Basis set

IO.Iprint_intgn (*integer*) Integration on the grid (not used at present)

IO.Iprint_time (*integer*) Timing information

Go to *top*.

2.9.6 Integration Grid

Grid.GridCutoff (*real*) An energy that defines the spacing of the *integration* grid (though for a blip calculation must be at least twice as fine as blip grid, and will be adjusted). Note that the value chosen will automatically be forced to be a factor of 3, 4 and 5 only (to fit with default FFT routines)

Default: 20 Ha.

Go to *top*.

2.9.7 Minimising Energy

minE.VaryBasis (*boolean*) Chooses whether or not basis coefficients should be varied to minimise the total energy

default: F

minE.SelfConsistent (*boolean*) Determines whether or not self-consistency cycles are imposed between charge density and potential

default: T

minE.MixedLSelfConsistent (*boolean*) Determines whether or not to perform self-consistent cycle at the same time as energy minimisation with respect to L

default: F

minE.EnergyTolerance (*real*) Fractional tolerance for energy on minimisation of support function coefficients

default: 1×10^{-5}

minE.LTolerance (*real*) Tolerance on *residual* in O(N) minimisation

default: 1×10^{-7}

minE.SCTolerance (*real*) Tolerance on *residual* in self-consistency

default: 1×10^{-6}

minE.SupportVariations (*integer*) Maximum number of support-function iterations

default: 20

minE.PreconditionBlips(*boolean*) Should blip variation be pre-conditioned? Pre-conditioning is (at present) more memory-intensive than it should be, but is efficient

> *default*: F

minE.GlobalTolerance (*boolean*) Are the convergence criteria applied to minimisation summed over the whole system, or per atom?

> *default*: T

Go to *top*.

2.9.8 Charge Self-Consistency

SC.LinearMixingSC (*boolean*) Should Pulay mixing be used? It is recommended that this is always used

> *default*: T

SC.LinearMixingFactor (*real*) Amount of output charge density which is mixed into new charge

> *default*: 0.5

SC.LinearMixingFactor_SpinDown (*real*) Amount of output charge density which is mixed into new charge for spin down channel.

> *default*: value of **SC.LinearMixingFactor**

SC.LinearMixingEnd (*real*) Tolerance for end of Pulay mixing

> *default*: self-consistency tolerance

SC.LateStageReset (*integer*) If using GR-Pulay, how often is residual calculated fully (rather than interpolated) ?

> *default*: 5

SC.MaxIters (*integer*) Maximum self-consistency iterations

> *default*: 50

SC.MaxEarly (*integer*) Maximum early-stage iterations

> *default*: 3

SC.MaxPulay (*integer*) Number of iterations stored and mixed during Pulay mixing

> *default*: 5

SC.ReadAtomicDensityFile (*string*) Filename for radial tables of atomic density (*rarely* used: normally generated from PAOs)

> default:

SC.AtomicDensityFlag (*string*) values: pao/read

> Flag determining how atomic densities should be found

> *default*: pao

SC.KerkerPreCondition (*boolean*) Flag determining if Kerker precondition is to be used.

> *default*: F

SC.KerkerFactor (*real*) Wave-vector magnitude used in Kerker preconditioning, it is q_0 from the factor $q^2 / \left(q^2 + q_0^2 \right)$

> *default*: 0.1

SC.WaveDependentMetric (*boolean*) Flag determining if wave-dependent metric is to be used in Pulay mixing.

> *default*: F

SC.MetricFactor (*real*) Wave-vector magnitude used by wave-dependent metric method, it is q_1 from the factor $\left(q^2 + q_1^2\right)/q^2$.

> *default*: 0.1

Go to *top*.

2.9.9 Density Matrix

DM.SolutionMethod (*string*) values: ordern/diagon

> Selects the method for finding the ground state density matrix. This can currently be either diagonalisation (diagon: minimising the energy with respect to the density matrix elements) or an O(N) method (ordern a combination of the techniques of Li et al. [E8] and Palser and Manolopoulos [E9].)

> *default*: ordern

DM.L_range (*real*) Cutoff applied to L matrix (total energy will converge with increasing range; suggested minimum for O(N) calculations is twice largest support function range; see *Linear Scaling* for more details)

> *default*: 1.0

DM.LVariations (*integer*) Maximum number of variations performed in search for ground-state density matrix

> *default*: 50

DM.MaxPulay (*integer*) Maximum number of iterations stored for Pulay minimisation

> *default*: 5

DM.MinPulayStepSize (*real*) Minimum allowed step size for Pulay minimisation in Energy minimisation stage of the calculation. Note that the actual step size is calculated by automatically, but will be constrained within the range defined by `DM.MinPulayStepSize` and `DM.MaxPulayStepSize`. Not to be confused with the Pulay mixing step size for charge self-consistency.

> *default*: 0.001

DM.MaxPulayStepSize (*real*) Maximum allowed step size for Pulay minimisation in Energy minimisation stage of the calculation. Not to be confused with the Pulay mixing step size for charge self-consistency.

> *default*: 0.1

DM.LinTol (*real*) Tolerance on linearity required before switching to Pulay minimisation

> *default*: 0.1

DM.InvSTolerance (*real*) Tolerance on iterative minimisation to find S^{-1}. If $\Omega = \mathrm{Tr}[(I - TS)^2]/N_{\mathrm{orbitals}}$ is above this, identity will be used

> *default*: 0.01

DM.InvSMaxSteps (*integer*) Sets the maximum number of iterations for finding S^{-1}

> *default*: 100

DM.InvSDeltaOmegaTolerance (*real*) Tolerance which determines when the iterative minimisation to find S^{-1} should finish. $\delta\Omega_n = N_{\mathrm{orbitals}}(\Omega_n - \Omega_{n-1})$, where Ω is defined in description for `DM.InvSTolerance`. This parameter differs from `DM.InvSTolerance` in that the iterative S^{-1} finder will end iteration when $\delta\Omega$ is less than or equal to `DM.InvSDeltaOmegaTolerance`, while `DM.InvSTolerance` determines whether to reset S^{-1} to identity (i.e. whether a satisfactory S^{-1} has been found) based on the final Ω produced from the iterative loop

default: 0.0001

DM.ConstantMu (*boolean*)

> **Switches between fixed Fermi level (T) and fixed number of electrons (F). You** are *strongly* recommended to leave at default

> *default*: F

DM.mu (*real*) Value of Fermi level for fixed Fermi level calculations

> *default*: 0.0

Go to *top*.

2.9.10 Diagonalisation

Diag.NumKpts (*integer*) Number of all k-points. No symmetry is applied.

> *default*:

Diag.Kpoints (*block*) Lists fractional coordinates and weights of all k-points: `x_fract y_fract z_fract weight` Generates the Monkhorst-Pack mesh, an equally spaced mesh of k-points.

> *default*:

Diag.MPMesh (*boolean*) Switches on/off the Monkhorst-Pack mesh. Note: if this keyword is present in the input file, the keyword **Diag.NumKpts** and the block **Kpoints** will be ignored.

> *default*:

Diag.MPMesh[X/Y/Z] (*integer*) Specifies the number n of k-points along the x(y,z) axis.

> *default*: 1

Diag.GammaCentred (*boolean*) Selects Monkhorst-Pack mesh centred on the Gamma point

> *default*: F

Diag.Proc[Rows/Cols] (*integer*) Number of rows in the processor grid for SCALAPACK within each k-point processor group

> *default*: Calculated by CONQUEST

Diag.BlockSize[R/C] (*integer*) R . . . rows, C . . . columns These are ScaLAPACK parameters, and can be set heuristically by the code. Blocks are sub-divisions of matrices, used to divide up the matrices between processors. The block sizes need to be factors of the square matrix size (i.e. $\sum_{\text{atoms}} \text{NSF}(\text{atom})$). A value of 64 is considered optimal by the ScaLAPACK user's guide. The rows and columns need to multiply together to be less than or equal to the number of processors. If ProcRows $\times$ ProcCols $<$ number of processors, some processors will be left idle.

> *default*:

Diag.MPShift[X/Y/Z] (*real*) Specifies the shift *s* of k-points along the x(y,z) axis, in fractional coordinates.

> *default*: 0.0

Diag.SmearingType (*integer*) Specifies the type of smearing used

0	Fermi-Dirac
1	Methfessel-Paxton

> *default*: 0

Diag.kT (*real*) Smearing temperature

> *default*: 0.001

Diag.MPOrder (*integer*) Order of Bessel function approximation to delta-function used in Methfessel-Paxton smearing

> *default*: 0

Diag.GaussianHeight (*real*) The height of Gaussian function used to determine the width of Methfessel-Paxton approximation to delta-function (see *Electronic occupation smearing*)

> *default*: 0.1

Diag.EfStepFiness (*real*) Parameter controlling the finness of the Fermi energy search step used in Methfessel-Paxton smearing method (see *Electronic occupation smearing*)

> *default*: 1.0

Diag.NElecLess (*Real*) The number of electrons to subtract from the total number of electrons in each spin channel, which gives the starting point for searching the lower bound for Fermi energy. Used in Methfessel-Paxton smearing method (see *Electronic occupation smearing*)

> *default*: 10.0

Diag.KProcGroups (*integer*) Number of k-point processor groups for k-point parallelisation (see *K-point parallelization*)

> *default*: 1

Go to *top*.

2.9.11 Moving Atoms

AtomMove.TypeOfRun (*string*) values: static/cg/lbfgs/md

> Options:
>
> static — Single point calculation
>
> cg — Structure optimisation by conjugate gradients
>
> lbfgs — Structure optimisation by LBFGS (Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm)
>
> md — Velocity Verlet algorithm
>
> *default*: static

AtomMove.QuenchMD (*boolean*) Selects Quenched MD for structure relaxation (with `AtomMove.TypeOfRun md`)

> *default*: F

AtomMove.FIRE (*boolean*) Selects FIRE method for structure relaxation (with `AtomMove.TypeOfRun md`)

> *default*: F

AtomMove.NumSteps (*integer*) Maximum number of steps for a structure optimisation or molecular dynamics run

> *default*: 100

AtomMove.MaxForceTol (*real*) The structure optimisation will stop when the maximum force component is less than **MD.MaxForceTol**

> *default*: 0.0005 Ha/bohr

AtomMove.Timestep (*real*) Time step for molecular dynamics

> *default*: 0.5

AtomMove.IonTemperature (*real*) Initial temperature for molecular dynamics

> *default*: 300 K for MD, 0 for Quench MD or FIRE

AtomMove.ReadVelocity (*boolean*) Read velocity from file `md.checkpoint` (when `AtomMove.RestartRun` `T`)

> or `velocity.dat` (when `AtomMove.RestartRun` `F`, very rare)

> *default*: F (when `AtomMove.RestartRun` `F`)

> or T (when `AtomMove.RestartRun` `T`)

AtomMove.AppendCoords (*boolean*) Chooses whether to append coordinates to `UpdatedAtoms.dat` during atomic movement (T) or to overwrite (F)

> *default*: T

AtomMove.OutputFreq (*integer*) Frequency of output of information. *Not properly implemented*

> *default*: 50

AtomMove.WriteXSF (*boolean*) Write atomic coordinates to `trajectory.xsf` for `AtomMove.TypeOfRun` = `md` or `cg`, every `AtomMove.OutputFreq` steps

> *default*: T

AtomMove.TestForces (*boolean*) Flag for testing forces with comparison of analytic and numerical calculations. Can produce *large* amounts of output

> *default*: F

AtomMove.TestAllForces (*boolean*) Switch to test *all* force contributions or not

> *default*: F

AtomMove.CalcStress (*boolean*) Toggle calculation of the stress tensor. Switching off can improve performace.

> *default*: T

AtomMove.FullStress (*boolean*) Toggle calculation of the off-diagonal elements of the stress tensor, which can be expensive, but is required for calculating certain properties.

> *default*: F

AtomMove.AtomicStress (*boolean*) Toggle calculation of atomic contributions to the stress tensor. Used in heat flux/thermal conductivity calculations. Significantly increases memory demands.

> *default*: F

AtomMove.OptCell (*boolean*) Turns on conjugate gradient relaxation of the simulation box dimensions a, b and c. Note that AtomMove.TypeOfRun must also be set to cg.

> *default*: F

AtomMove.OptCellMethod (*integer*) Cell optimisation method. 1. fixed fractional coordinates 2. double loop - inner: full atomic cg optimisation, outer: single cell steepest descent step. Generally only useful for systems that are extremely problematic to relax 3. simultaneous cell and atomic conjugate gradients relaxation

> *default*: 1

AtomMove.EnthalpyTolerance (*real*) Enthalpy tolerance for cell optimisation

> *default*: 1×10^{-5} Ha/Bohr

AtomMove.StressTolerance (*real*) Stress tolerance for cell optimisation

>   *default*: 0.5×10^{-2} GPa

AtomMove.TargetPressure (*real*) External pressure for NPT molecular dynamics and cell optimisation

>   *default*: 0.0 GPa

AtomMove.OptCell.Constraint (*string*) Applies a constraint to the relaxation.

>   none: Unconstrained relaxation.

>   *Fixing a single cell dimension:*

>   a: Fix the x-dimension of the simulation box

>   b: Fix the y-dimension of the simulation box

>   c: Fix the z-dimension of the simulation box

>   *Fixing multiple cell dimensions:*

>   any combination of the above separated by a space character. e.g: "a b" fixes both the x and y dimensions of the simulation box

>   *Fixing Ratios:*

>   Any combination of a, b or c separated by a "/" character. e.g "c/a" fixes the initial ratio of the z-dimension to the x-direction.

>   *Global scaling factor:*

>   volume: minimize the total energy by scaling each simulation box dimension by the same global scaling factor. Search directions are set by the mean stress.

AtomMove.TestSpecificForce (*integer*) Label for which force contribution to test. Note that for PAOs non-local Pulay and Hellman-Feynman forces are found together as part of the HF calculation; ϕ Pulay refers to changes in $\phi(\mathbf{r})$ when atoms move, while S Pulay refers to changes in S when atoms move. Options:

>   1 Total 2 Total Hellman-Feynman 3 Total Pulay 4 Non-SC Correction 5 Non-local ϕ Pulay 6 KE ϕ Pulay 7 Local ϕ Pulay 8 S Pulay

>   *default*: 1

AtomMove.TestForceDirection (*integer*) Direction in which atom will be moved (1=x; 2=y; 3=z)

>   *default*: 1

AtomMove.TestForceAtom (*integer*) Atom to move

>   *default*: 1

AtomMove.TestForceDelta (*real*) Distance atom will be moved for numerical evaluation of force

>   *default*: 10^{-5} bohr

AtomMove.RestartRun (*boolean*) Restart a MD run. Note that this will set `General.LoadL T`, `AtomMove.MakeInitialChargeFromSC T` and `XL.LoadX T` if using the extended Lagrangian. The atomic coordinates will be read from `md.positions` and the velocities and extended system variables from `md.checkpoint`.

>   *default*: F

AtomMove.ReuseDM (*boolean*) Selects the use of last-step L-matrix (`ordern`) or K-matrix(`diagon`) during MD or structure relaxation

>   *default*: T

AtomMove.ReuseSFcoeff (*boolean*) Selects the use of last-step PAO coefficients of multi-site support functions during MD or structure relaxation

  default: T

AtomMove.ReuseInvS (*boolean*) Selects the use of T-matrix in MD run (rare)

  default: F

AtomMove.SkipEarlyDM (*boolean*) Selects the skip of earlyDM calculation in MD run

  default: F

AtomMove.McWeenyFreq (*integer*) McWeeny step is applied every N steps (with "AtomMove.ReuseDM T")

  default:

AtomMove.ExtendedLagrangian (*boolean*) Selects XL-BOMD (with "AtomMove.ReuseDM T")

  default: F

AtomMove.FixCentreOfMass (*boolean*) Remove the centre of mass velocity at every time step

  default: T

Go to *top*.

2.9.12 Molecular Dynamics

MD.Ensemble (*string*) values: nve/nvt/npt/nph

  The molecular dynamics ensemble

  default: nve

MD.Thermostat (*string*) values: none/nhc/berendsen/svr

  Thermostat type

  none No thermostat (used for calculating temperature only)

  berendsen Berendsen weak coupling thermostat

  svr Stochastic velocity rescaling

  default: none

MD.Barostat (*string*) values: none/berendsen/iso-mttk/ortho-mttk/mttk

  Barostat type. The following are the only valid thermostat/barostat combinations for the NPT ensemble: `berendsen`/ `berendsen`, `nhc`/ `pr`, `svr`/ `pr`

  none No barostat (used for calculating pressure only)

  berendsen Berendsen weak coupling barostat

  pr Parrinello-Rahman (extended system) barostat

  default: none

MD.tauT (*real*) Coupling time constant for thermostat. Required for Berendsen thermostat, or if `MD.CalculateXLMass = T`. Note that this number means different things for the Berendsen and NHC thermostats.

  default: 1.0

MD.TDrag (*real*) Add a drag coefficient to the thermostat. The thermostat velocities are reduced by a factor $1-\tau/D_T$ every step.

default: 0.0

MD.nNHC (*integer*) Number of Nosé-Hoover thermostats in chain

default: 5

MD.CellNHC (*boolean*) Use a separate Nosé-Hoover chain for thermostating the unit cell (NPT only)

default: T

MD.NHCMass (*blocks*) $< n1 >< n2 >< n3 > \dots$ Masses of NHC heat baths

default: 1 1 1 1 1

MD.CellNHCMass (*block*) $< n1 >< n2 >< n3 > \dots$ Masses of NHC heat baths for unit cell

default: 1 1 1 1 1

MD.BulkModulusEst (*real*) Bulk modulus estimate for system. Only necessary for Berendsen weak pressure coupling (`MD.Barostat = berendsen` or `MD.BerendsenEquil > 0`)

default: 100

MD.tauP (*real*) Coupling time constant for barostat. Required for Berendsen barostat, or if MD.CalculateXLMass = T. Note that this number means different things for the Berendsen and Parrinello-Rahman barostats.

default: 10.0 (Berendsen) or 100.0 (MTTK)

MD.PDrag (*real*) Add a drag coefficient to the barostat. The barostat velocities are reduced by a factor $1 - \tau/D_P$ every step. This is useful when the lattice parameters are varying rapidly.

default: 0.0

MD.BoxMass (*real*) Mass of box for extended system formalism (MTTK barostats)

default: 1

MD.CalculateXLMass (*boolean*) Calculate the mass of the extended system components (thermostats, barostat) using the MTTK formulae.

default: T

MD.nYoshida (*integer*) values: 1/3/5/7/15/25/125/625

Order of Yoshida-Suzuki integration

default: 1

MD.nMTS (*integer*) Number of time steps in inner loop of MTS scheme

default: 1

MD.BerendsenEquil (*integer*) Equilibrate the system for n steps using Berendsen weak coupling

default: 0

MD.TDEP (*boolean*) Dump data in a format readable by the Temperature Dependent Effective Potential (TDEP) code.

default: F

MD.ThermoDebug (*boolean*) Print detailed information about thermostat and extended variables in `thermostat.dat`

default: F

MD.BaroDebug (*boolean*) Print detailed information about barostat and extended variables in `barostat.dat`

> *default*: F

Go to *top*.

2.9.13 Spin Polarisation

Spin.SpinPolarised (*boolean*) Determines if the calculation is spin polarised (collinear) or non-spin polarised.

> *default*: F

Spin.FixSpin (*boolean*) Determines if spin populations are to be fixed. Only read if **Spin.FixPolarised** is set.

> *default*: F

Spin.NeUP (*real*) Total number of electrons in spin up channel at start of calculation.

> *default*: 0.0

Spin.NeDN (*real*) Total number of electrons in spin down channel at start of calculation.

> *default*: 0.0

Go to *top*.

2.9.14 DeltaSCF

flag_DeltaSCF (*boolean*) Selects delta SCF calculation

> *default*:

DeltaSCF.SourceLevel (*integer*) Eigenstate number to remove electron from (source)

> *default*:

DeltaSCF.TargetLevel (*integer*) Eigenstate number to promote electron to (target)

> *default*:

DeltaSCF.SourceChannel (*integer*) Spin channel for electron source

> *default*:

DeltaSCF.TargetChannel (*integer*) Spin channel for electron target

> *default*:

DeltaSCF.SourceNFold (*integer*) Allows selection of more than one level for excitation source (N-fold)

> *default*:

DeltaSCF.TargetNFold (*integer*) Multiplicity of target (N-fold)

> *default*:

DeltaSCF.LocalExcitation (*boolean*) Select an excitation localised on a group of atoms

> *default*:

DeltaSCF.HOMOLimit (*integer*) How many states down from HOMO to search for localised excitation

> *default*:

DeltaSCF.LUMOLimit (*integer*) How many states up from LUMO to search for localised excitation

> *default*:

DeltaSCF.HOMOThresh (*real*) (*please fill in*)

> *default*:

DeltaSCF.LUMOThresh (*real*) Threshold for identifying localised excitation (sum over square moduli of coefficients)

> *default*:

Go to *top*.

2.9.15 Constrained DFT (cDFT)

cDFT.Perform_cDFT (*boolean*) Selects cDFT operation

> *default*:

cDFT.Type (*integer*) values: 1 or 2

> Selects constraint to be for absolute charge on groups (1) or difference between two groups (2)

> *default*:

cDFT.MaxIterations (*integer*) Maximum iterations permitted

> *default*:

cDFT.Tolerance (*real*) Tolerance on charge

> *default*:

cDFT.NumberAtomGroups (*integer*) Number of groups of atoms

> *default*:

cDFT.AtomGroups (*block*) Block with each line specifying: Number of atoms, target charge, label for block. For each line, there should be a corresponding block with the appropriate label; the block consists of a list of atom numbers for the atoms in the group

Go to *top*.

2.9.16 vdW-DF

vdWDFT.LDAFunctionalType (*string*) Selects LDA functional to use with vdW-DF

> *default*:

Go to *top*.

2.9.17 DFT-D2

DFT-D2_range (*real*) DFT-D2 cutoff range (bohr)

> *default*:

Go to *top*.

2.9.18 XL-BOMD

XL.Kappa (*real*) Value of kappa

>*default*: 2.0

XL.PropagateX (*boolean*) Selects the propagation of LS in XL-BOMD

>*default*: T

XL.PropagateL (*boolean*) Selects the propagation of L matrix in XL-BOMD (inappropriate)

>*default*: F

XL.Dissipation (*boolean*) Selects the addition of dissipative force

>*default*:

XL.MaxDissipation (*integer*) Order of dissipative force term

>*default*: 5

XL.Integrator (*string*) Selects the Verlet method or velocity Verlet method

>*default*: velocityVerlet

XL.ResetFreq (*integer*) Frequency to reset the propagation of X matrix in XL-BOMD

>*default*: 0 (no reset)

Go to *top*.

2.9.19 Advanced and obscure tags

General

General.LoadInvS (*boolean*) Selects loading of inverse S matrix from previous step (not recommended)

>*default*: F

General.NeutralAtomProjector (*boolean*) Selects projector expansion of neutral atom potential; still in development. Only for expert use. (Allows specification of maximum l value for projectors and list of number of projectors for each l value.)

>*default*: F

General.PAOFromFiles (*boolean*) Allows you to give explicit file name for .ion files in atom block

>*default*: F

General.MaxTempMatrices (*integer*) Allows user to increase number of temporary matrices; sometimes required for wavefunction output.

>*default*: 100

General.EwaldAccuracy (*real*) Accuracy required for Ewald sum

>*default*:1×10^{-10}

General.CheckDFT (*boolean*) Calculates DFT energy using output density

>*default*: F

General.AverageAtomicDiameter (*real*) Related to space-filling

>*default*: 5.0

General.GapThreshold (*real*) Related to space-filling

> *default*: 2.0*(largest support radius)

General.only_Dispersion (*boolean*) Selects only DFT_D2 calculation (no electronic structure etc)

Go to *top*.

Atomic Specification

Atom.ValenceCharge (*real*) Valence charge of species (e.g. 4 for carbon, 6 for oxygen)

> *default*: read from ion file

Atom.NumberOfSupports (*integer*) Number of support functions per atom for a species. Don't confuse support functions and PAOs ! Support functions can be expanded in a basis set of PAOs or blips

> *default*: number of PAOs read from ion file

Atom.SupportFunctionRange (*real*) Confinement radius for the support functions for a given species

> *default*: maximal PAO radius read from ion file

Atom.SupportGridSpacing (*real*) The spacing of the blip grid (if using). Equivalent (under certain circumstances) to a maximum g-vector of π/**SupportGridSpacing** plane wave cutoff as region radius and L matrix radius go to infinity. *Not used for PAO calculations*. N.B. Grid.GridCutoff will be reset to *at least* half SupportGridSpacing if too small.

> *default*: none

Atom.NonLocalFactor (*real*) This is an adjustment factor: the Hamiltonian range is (strictly) $2 \times$ (support function radius + non-local projector radius). However, generally without affecting the results, the Hamiltonian range can be set to $2 \times$ (support function radius + non_local_factor$\times$ non-local projector radius). If you have non_local_factor = 1.0 then you get the full range, if 0.0 then the same range as the S matrix.

> *default*: 0.0

Atom.InvSRange (*real*) Range of inverse S matrix (though actual matrix range is twice this for consistency with S matrix range).

> *default*: support function range

Atom.SpinNeUp (*real*) Specify the population of spin-up electrons for setting initial spin state of atomic densities

> *default*: 0.0

Atom.SpinNeDn (*real*) Specify the population of spin-down electrons for setting initial spin state of atomic densities

> *default*: 0.0

Go to *top*.

I/O General

IO.Partitions (*string*) Name for file containing distribution of partitions over processors (generated by accompanying utilities)

> *default*: `make_prt.dat`

IO.TimingOn (*boolean*) Whether time information will be measured and written to output

> *default*: F

IO.TimeAllProcessors (*boolean*) Specifies whether time information will be written for all processors or just for the input/output process (the default)

> *default*: F

IO.WriteTimeFile (*boolean*) Whether time files are written or not.   This flag will be ignored if `IO.TimeAllProcessors` is true, in which case time files are always written.

> *default*: T

IO.TimeFileRoot (*string*) Root to be used in the time files, with an extension indicating the processor number, e.g. `.001`

> *default*: `time`

Go to *top*.

I/O Atomic Coordinates

IO.PdbAltLoc (*string*) In case of PDB files with multiple locations selects an alternate location. Values: A, B, etc., as listed in the pdb file. Note that if the keyword is present in the input file but no value is given, only the parts of the system without any alternate location specification will be taken into account

> *default*: none

IO.PdbOut (*boolean*) Format of the output coordinate file. Writes a PDB file if set to T. In that case, either the input must be in pdb format or a PDB "template" file needs to be specified (keyword General.PdbTemplate)

> *default*: F

IO.PdbTemplate (*string*) A file used as a template for writing out coordinate files in the PDB format, i.e., the output file will contain the same information as the template, only the atomic coordinates will be overwritten. If the input file is in PDB format, it will also be used as the template, although this can still be overwritten with this keyword

> *default*: coordinate file

Go to *top*.

Basis Set

Basis.BasisSet (*string*) values: blips/PAOs

> Selects the basis set in which to expand the support functions (localised orbitals).

> Options:

> - PAOs — Pseudo-atomic orbitals [E10]
> - blips (default) — B-splines [E11]

> *default*: PAOs

Basis.LoadBlip (*boolean*) Load blip or PAO coefficients from file. If set to T, for blips the code will look for a set of files containing blip coefficients, which is taken to be `blip_coeffs.nnn`, where `nnn` is processor number (padded with zeroes); for PAOs, the code will look for a *single* file which is `supp_pao.dat` by default, but can be set with `Basis.SupportPaoFile`

> *default*: F

Basis.SupportPaoFile (*string*) Specifies filename for PAO coefficients

> *default*: `supp_pao.dat`

Basis.UsePulayForPAOs (*boolean*) Determines whether to use Pulay DIIS for minimisation of PAO basis coefficients

default: F

Basis.PaoKspaceOlGridspace (*real*) Determines the reciprocal-space grid spacing for PAO integrals

default: 0.1

Basis.PaoKspaceOlCutoff (*real*) Determines the cutoff for reciprocal-space grid spacing for PAO integrals

default: 1000.0

Basis.PAOs_StoreAllAtomsInCell (*boolean*) Determines whether coefficients for all atoms in cell are stored on each processor (improves speed but potentially memory expensive, particularly with large systems) or only local atom coefficients (increases communication overhead)

default: T

Basis.SymmetryBreaking (*boolean*) Determines whether symmetry-breaking assignment of PAOs to support functions is allowed. In general, it is *highly* recommended that all atoms have sufficient support functions to span the space of angular momenta used in PAOs (i.e. $2l + 1$ support functions for each l channel used for PAOs); reducing the number potentially results in symmetry breaking and unphysical behaviour

default: F

Basis.PaoNormFlag (*integer*) Determines whether PAOs are normalised

default: 0

Basis.TestBasisGradients (*boolean*) Chooses whether gradients of energy with respect to basis function coefficients should be tested (using numerical vs. analytical gradients). **WARNING :** this produces large amounts of data

default: F

Basis.TestBasisGradTot (*boolean*) Test total gradient ?

default: F

Basis.TestBasisGradBoth (*boolean*) Test both S- and H-derived gradients (i.e. gradients arising from change of S or H when support functions vary) ?

default: F

Basis.TestBasisGrad_S (*boolean*) Test S-derived gradient ?

default: F

Basis.TestBasisGrad_H (*boolean*) Test H-derived gradient ?

default: F

Basis.PAOs_OneToOne (*boolean*) Assign PAOs to individual support functions (implies no support function optimisation)

default: F

Go to *top*.

Integration Grid

Grid.PointsAlong[X/Y/Z] (*integer*) Grid points along x (y,z). Overwrites the values set by **Grid.GridCutoff**. The default FFT code requires that the number of grid points have prime factors of 2, 3 or 5

>   *default*: 0

Grid.InBlock[X/Y/Z] (*integer*) This is the size of a grid point block (i.e., how many grid points are in one block in the x (y,z) direction), which must be a multiple of 2, 3, or 5 (larger values may impact on parallel efficiency).

>   *default*: 4

Grid.ReadBlocks (*boolean*) If specified, the code reads information about blocks from the file make_blk.dat

>   *default*: F

Go to *top*.

Go to *top*.

TUTORIALS

- *Introductory Tutorials*
- *Structural Relaxation Tutorials*
- *Molecular Dynamics Tutorials*
- *Basis Function Tutorials*
- *Advanced Tutorials*

3.1 Introductory Tutorials

These introductory tutorials will give you an overview of how to run CONQUEST, the files and parameter settings required, and what output to expect.

3.1.1 Bulk silicon: input, output and SCF

We start with a very basic introduction to the input required for CONQUEST, the output generated, and the self-consistency (SCF) procedure; it uses the same system (bulk silicon) as the first of the examples in the manual, but provides more detail. The files are found in `docs/tutorials/Introductory_1`.

CONQUEST requires the following files to run:

- The input file: `Conquest_input`
- A coordinates file (name set in `Conquest_input` with the flag `IO.Coordinates`; no default)
- Ion files (suffix `.ion`), which provide the pseudopotentials and pseudo-atomic orbitals (PAOs) (names set in `Conquest_input` in the `ChemicalSpeciesLabel` block)

We will discuss the files and their contents, and then how to run the code and interpret the output.

Conquest_input file

The `Conquest_input` file requires the user to provide a certain minimal amount of information, with most parameters having reasonable defaults set. The file that is provided for this tutorial gives the most basic parameters:

```
# Input/Output
IO.Title Bulk Si 8 atoms static
IO.Coordinates ionpos.dat

# General Parameters
```

(continues on next page)

```
General.NumberOfSpecies 1

%block ChemicalSpeciesLabel
1  28.0850   Si_SZ
%endblock

# Moving Atoms
AtomMove.TypeOfRun static

# Finding the density matrix
DM.SolutionMethod diagon

# k-points
Diag.GammaCentred T
Diag.MPMesh T
Diag.MPMeshX 2
Diag.MPMeshY 2
Diag.MPMeshZ 2
```

The most important entries are:

- the coordinate file (`IO.Coordinates`);

- the number of species (`General.NumberOfSpecies`);

- the specification for the species (the block `ChemicalSpeciesLabel` gives the atomic mass and the ion file name for all species);

- the type of run (`AtomMove.TypeOfRun` which defaults to `static`)

The job title is purely for the user's convenience and reference. The remaining parameters given above, which related to the Brillouin zone sampling, will be discussed in more detail in the next tutorial; these settings suffice for now.

The species entries in the `ChemicalSpeciesLabel` block have three entries: the species number (which must be ordered); the atomic mass; and the species label. The ion file which is read for each species is, by default, `species_label.ion`: for this tutorial, it is `Si_SZ.ion`. (A file name can be specified *after* the species label for each species by setting the flag `General.PAOFromFiles T`.)

Coordinates

As described briefly in *the manual*, the coordinate file requires an orthorhombic simulation cell, with coordinates specified either in fractional form (the default) or Cartesian (if the tag `IO.FractionalAtomicCoords F` is set). The units for distance are Bohr radii by default, though can be changed to Angstroms (set `General.DistanceUnits Ang`). The file in this case, `ionpos.dat`, is:

```
10.2600  0.0000   0.0000
 0.0000 10.2600   0.0000
 0.0000  0.0000  10.2600
 8
 0.0000  0.0000  0.0000   1 T T T
 0.5000  0.5000  0.0000   1 T T T
 0.5000  0.0000  0.5000   1 T T T
 0.0000  0.5000  0.5000   1 T T T
 0.2500  0.2500  0.2500   1 T T T
 0.2500  0.7500  0.7500   1 T T T
 0.7500  0.2500  0.7500   1 T T T
 0.7500  0.7500  0.2500   1 T T T
```

The first three lines specify the simulation cell axes (any non-zero off-diagonal entries will be discarded in the present version). The fourth line specifies the total number of atoms in the system. The subsequent lines give the coordinates, the species index, and then three flags which control whether the atom is free to move in the Cartesian directions (x/y/z). (CONQUEST checks for possible errors in the coordinates: atoms which are too close to each other (less than **find parameter**) or Cartesian coordinates in the input file when fractional coordinates have been specified.) CONQUEST can also read and write PDB files(**ref**).

Ion files

Ion files contain the pseudopotentials and pseudo-atomic orbitals (PAOs) for a given species. The ion file needed for this tutorial, `Si_SZ.ion`, is included along with the input files. In general, ion files are generated by the utility `MakeIonFiles`, which is found in `tools/BasisGeneration` and documented **somewhere**. The input files required for all elements with the LDA, PBE GGA and PBEsol GGA are found in `pseudo-and-pao/`. Default basis sets with different accuracies can be generated by adjusting the basis size in the `Conquest_ion_input` file (minimal/small/medium/large). Further details are found **somewhere**.

Output files and running CONQUEST

By default, the output is written to the file `Conquest_out`, though this can be changed: to a different file, using the input parameter `IO.OutputFile`; or to the console (`stdout`), using the input parameter `IO.WriteOutToFile`. The amount of output is controlled using the input parameter `IO.Iprint` (different areas of the code can be set to have different output levels, with all levels defaulting to the global `IO.Iprint` level).

Detailed discussion of the output file is a little difficult while we're reworking the output, so I'm postponing this for now

- Beginning: a summary of the job
- Bibliography
- Energies and residuals
- Forces and stresses

The SCF cycle

Go to *top*.

3.1.2 Bulk silicon: parameters to converge

- The files that are needed
    - Coordinates
    - Ion files
    - Input file: `Conquest_input`
- Integration grid
- Brillouin zone sampling
- Possibly basis set size

The Monkhorst-Pack mesh (`Diag.MPMesh`) offers a convenient way to do this systematically. Full documentation is found in the *diagonalisation* section of the manual.

Go to *top*.

3.1.3 Bulk silicon: analysis

- The files that are needed

    - Coordinates

    - Ion files

    - Input file: `Conquest_input`

- Total DOS

- Atom-projected DOS

- Band structure output

- Charge density and bands

- Atomic charges

Go to *top*.

3.2 Structural Relaxation Tutorials

We now turn to structural relaxation, covering both atomic relaxation and simulation cell relaxation.

3.2.1 Atomic relaxation: lbfgs

- Parameters that are needed

- The output

- Files in `Structural_relaxation_1`

Go to *top*.

3.2.2 Atomic relaxation: Conjugate gradients

- Parameters that are needed

- The output

- Files in `Structural_relaxation_2`

Go to *top*.

3.2.3 Atomic relaxation: Quenched MD

- Parameters that are needed

- The output

- Files in `Structural_relaxation_3`

Go to *top*.

3.2.4 Cell relaxation: Conjugate gradients

- Parameters that are needed

- The output

- Files in `Structural_relaxation_4`

Go to *top*.

3.3 Molecular Dynamics Tutorials

These tutorials introduce molecular dynamics (MD) in CONQUEST, though not the topic of MD itself, for which you should consult an appropriate textbook.

3.3.1 Molecular dynamics: NVE

- Parameters that are needed

- The output

- Files in `MD_1`

Go to *top*.

3.3.2 Molecular dynamics: NVT with SVR

- Parameters that are needed

- The output

- Files in `MD_2`

Go to *top*.

3.3.3 Molecular dynamics: NVT with Nose-Hoover chains

- Parameters that are needed

- The output

- Files in `MD_3`

Go to *top*.

3.3.4 Molecular dynamics: NPT with SVR and Parrinello-Rahman

- Parameters that are needed

- The output

- Files in `MD_4`

Go to *top*.

3.3.5 Molecular dynamics: NPT with Nose-Hoover chains

- Parameters that are needed

- The output

- Files in `MD_5`

Go to *top*.

3.3.6 Molecular dynamics: Continuing a run

- Parameters that are needed

- The output

- Files in `MD_6`

Go to *top*.

3.4 Basis Function Tutorials

We now introduce details of the basis sets used in CONQUEST, and how they are specified and optimised.

3.4.1 MSSF: local filter diagonalisation

- Parameters that are needed

- The output

- Files in `Basis_sets_1`

Go to *top*.

3.4.2 MSSF: Optimisation

- Parameters that are needed

- The output

- Files in `Basis_sets_2`

Go to *top*.

3.5 Advanced Tutorials

These tutorials cover more advanced topics in CONQUEST, and assume a reasonable familiarity and confidence with the general operation of the code.

3.5.1 Initialising spins

- Parameters that are needed

- The output

- Files in `Advanced_1`

Go to *top*.

3.5.2 An introduction to linear scaling

- Parameters that are needed

- The output

- Files in `Advanced_2`

Go to *top*.

THEORY

- *Structural relaxation: Theory*
- *Molecular Dynamics: Theory*

4.1 Structural relaxation: Theory

Structural relaxation involves optimisation of the ionic coordinates, optimisation of the simulation cell, or both, with respect to the DFT total energy or the enthalpy if the cell is not fixed.

4.1.1 Ionic relaxation

L-BFGS:

To be written. . .

Conjugate gradients

The most naive geometry optimisation algorithm is steepest descent: we calculate the gradient of the DFT total energy (i.e. the force) and propagate the system in the direction of the steepest gradient (the direction of the force vector) until the energy stops decreasing. We choose the direction (largest gradient in this case) and perform a line search. This will be sufficient if the potential energy surface is well-behaved, but in most cases convergence will require many iterations. Conjugate gradients is a well-established method the improves upon steepest descent in the choice of search direction. Without going into too much detail, we choose a new search direction that is orthogonal to all previous search directions using the *conjugacy ratio* β. At iteration n, it is given by,

$$\beta_n = \beta_{n-1} + \frac{\mathbf{f}_n^T \mathbf{f}_n}{\mathbf{f}_{n-1}^T \mathbf{f}_{n-1}}$$

This is the Fletcher-Reeves formulation; note that $\beta_0 = 0$. We can then construct the search direction at step n, D_n,

$$D_n = \beta_n D_{n-1} + \mathbf{f}_n,$$

and peform the line minimisation in this direction. This process is repeated until the maximum force component is below some threshold.

Go to *top*.

Quenched MD

The system is propagated in the direction of steepest descent as determined by the DFT forces, and the velocity is scaled down as the system approaches its zero-temperature equilibrium configuration.

Go to *top*.

FIRE Quenched MD

The system is propagated using the modified equation of motion [Tb1],

$$\dot{\mathbf{v}}(t) = \mathbf{F}(t)/m - \gamma(t)|\mathbf{v}(t)|[\hat{\mathbf{v}}(t) - \hat{\mathbf{F}}(t)]$$

which has the effect of introducing an acceleration in a direction that is steeper than the current direction of motion. If the power $P(t) = \mathbf{F}(t) \cdot \mathbf{v}(t)$ is positive then the system is moving "downhill" on the potential energy surface, and the stopping criterion is when it becomes negative (moving "uphill").

Go to *top*.

4.1.2 Cell optimisation

When optimising the cell with *fixed fractional ionic coordinates*, the same conjugate gradients method is used as above, but minimising the enthalpy with respect to the cell vectors.

Go to *top*.

4.1.3 Combined optimisation

The ionic and cell degrees of freedom can be relaxed simultaneously by combining all of their coordinates into a single vector and optimising them with respect to the enthalpy of the system. However, this atomic forces and total stresses having numerical values of the same order of magnitude, and changes in ionic coordinates and cell vectors being of the same order of magnitude. Using the method of Pfrommer *et al*. [Tb2], the latter can be enforced by using fractional coordinates for the ionic positions, and fractional lattice vectors of the form $h = (1 + \epsilon)h_0$ where h is the matrix of lattice vectors, h_0 is the matrix for some reference configuration and epsilon is the strain matrix. The "fractional" force on the i th atom is then $\mathbf{F}_i = g\mathbf{f}_i$ where $\mathbf{f}_i$ is the DFT-calculated force multiplied by the metric tensor $g = h^T h$. The "fractional" stress is,

$$f^{(\epsilon)} = -(\sigma + p\Omega)(1 + \epsilon^T)$$

where σ is the DFT-calculated stress, p is the target pressure and Ω is the volume. The resulting vector is optimised using the same conjugate gradients algorithm as before, minimising the enthalpy.

Go to *top*.

4.2 Molecular Dynamics: Theory

4.2.1 Microcanonical (NVE) ensemble

The Hamiltonian for the microcanonical ensemble is,

$$\mathcal{H} = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} + U(\mathbf{r}_i)$$

where $\mathbf{p}_i$ and $\mathbf{r}_i$ are the position and momentum of particle i and U is the DFT total (potential) energy. Hamilton's equations can be solved to give the following equations of motion:

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i}$$

$$\dot{\mathbf{p}}_i = \frac{\partial U(\mathbf{r}_i)}{\partial \mathbf{r}_i} = \mathbf{F_i}$$

In order to construct a time-reversible algorithm from these equations, the Liouvillian formulation is employed [Ta1] (trivially, in this case). The Liouville operator L can be defined in terms of position and momentum components:

$$iL = \dot{\mathbf{r}}\frac{\partial}{\partial \mathbf{r}} + \dot{\mathbf{p}}\frac{\partial}{\partial \mathbf{p}} = i(L_r + L_p).$$

The Liouvillian can be used to construct the classical propagator, which relates the state f of the system at time 0 to its state at time t:

$$f[\mathbf{p}^N(t), \mathbf{r}^N(t)] = e^{iLt} f[\mathbf{p}^N(0), \mathbf{r}^N(0)]$$

Taking the individual position and momentum parts of the Liouvillian L_r and L_p, it can be shown that applying it to the state f result in a simple linear shift in coordinates and a simple linear shift in momentum respectively:

$$iL_r f(t) = f[\mathbf{p}^N(0), \mathbf{r}^N(0) + \dot{\mathbf{r}}^N(0)t]$$
$$iL_p f(t) = f[\mathbf{p}^N(0) + \mathbf{F}^N(0)t, \mathbf{r}^N(0)]$$

However, we cannot simply replace e^{iLt} with $e^{iL_r t}$ because iL_r and iL_p are non-commuting operators, so we must employ the Trotter-Suzuki identity:

$$e^{A+B} = \lim_{P \to \infty} \left(e^{A/2P} e^{B/P} e^{A/2P} \right)^P$$

Thus for a small enough time step $\Delta t = t/P$ and to first order, a discrete time step corresponds to the application of the discrete time propagator G,

$$G(\Delta t) = U_1\left(\frac{\Delta t}{2}\right) U_2\left(\Delta t\right) U_1\left(\frac{\Delta t}{2}\right) = e^{iL_1 \frac{\Delta t}{2}} e^{iL_2 \Delta t} e^{iL_1 \frac{\Delta t}{2}},$$

which can be shown to be unitary and therefore time-reversible. Applying the operators U in the sequence determined by the Trotter decomposition generates the velocity Verlet algorithm, which is used to integrate microcanonical molecular dynamics in CONQUEST. For a detailed derivation of the algorithm, refer to Frenkel & Smit [Ta1].

Go to *top*.

4.2.2 Extended Lagrangian Born-Oppenheimer MD (XL-BOMD)

If the electronic density from the previous ionic step is used as an initila guess for the next SCF cycle, a problem arises because this process breaks the time-reversibility of the dynamics. This is manifested as a gradual drift in the total energy in the case of a NVE simulation, or the conserved quantity in the case of non-Hamiltonian dynamics. The solution proposed by Niklasson [Ta2][Ta3] is to introduce auxilliary electronic degrees of freedom into the Lagrangian, which can be propagated via time-reversible integrators.

The extended Lagrangian used in CONQUEST is [Ta4],

$$\mathcal{L}^{\text{XBO}}\left(\mathbf{X}, \dot{\mathbf{X}}, \mathbf{R}, \dot{\mathbf{R}}\right) = \mathcal{L}^{\text{BO}}\left(\mathbf{R}, \dot{\mathbf{R}}\right) + \frac{1}{2}\mu \text{Tr}\left[\dot{\mathbf{X}}^2\right] - \frac{1}{2}\mu\omega^2 \text{Tr}\left[(\mathbf{LS} - \mathbf{X})^2\right],$$

where $\mathbf{X}$ is a sparse matrix with the same range as $\mathbf{LS}$, μ is the fictitious electron mass and ω is the curvature of the auxiliary harmonic potential. The Euler-Lagrange equations of motion are then,

$$m_i\ddot{\mathbf{r}}_{\mathbf{i}} = -\frac{\partial U[\mathbf{R}; \mathbf{LS}]}{\partial \mathbf{r}_i} = \mathbf{F}_{\mathbf{i}}$$

$$\ddot{\mathbf{X}} = \omega^2(\mathbf{LS} - \mathbf{X}),$$

The first of these is simply Newton's second law, and the velocity update equation of motion in the microcanonical ensemble. The second can be integrated using a time-reversible algorithm, the velocity Verlet scheme in the case of CONQUEST [Ta4]:

$$\mathbf{X}(t + \delta t) = 2\mathbf{X}(t) - \mathbf{X}(t - \delta t) + \delta t^2 \omega^2 \left[\mathbf{L}(t)\mathbf{S}(t) - \mathbf{X}(t) \right]$$

$$+ a \sum_{m=0}^{M} c_m \mathbf{X}(t - m\delta t)$$

i.e. the trajectory of $\mathbf{X}(t)$ is time-reversible, and evolves in a harmonic potential centred on the ground state density $\mathbf{L}(t)\mathbf{S}(t)$. The matrix $\mathbf{X}\mathbf{S}^{-1}$ is a good guess for the $\mathbf{L}$ matrix in the Order(N) scheme.

Despite the time-reversitibility, the $\mathbf{X}$ matrix tends in practice to gradually drift from the harmonic centre over time, increasing the number of SCF iterations required to reach the minimum over the course of the simulation. To remove such numerical errors, the final dissipative term is included, and is found to have a minimal effect on the time-reversibility. We note that since the auxiliary variable X is used to generate an intial guess for the SCF process, it does not appear in the conserved (pseudo-Hamiltonian) quantity for the dynamics.

Go to *top*.

4.2.3 Non-Hamiltonian dynamics

Extended system method

Hamiltonian dynamics generally describes systems that are isolated from their surroundings, but in the canonical and isobaric-isothermal ensembles, we need to couple the system to an external heat bath and/or stress. It is possible to model such systems by positing a set of equations of *non-Hamiltonian* equations of motion, and proving that they generate the correct statistical ensemble [Ta5]. This is the extended system approach: we modify the Hamiltonian to include the thermostat and/or barostat degrees of freedom, derive the (pseudo-) Hamiltonian equations of motion, and demostrate that the correct phase space distribution for the ensemble is recovered.

Go to *top*.

Canonical (NVT) ensemble

In the Nose-Hoover formulation [Ta6][Ta7], the Hamiltonian for a system in the canonical ensemble can be written,

$$\mathcal{H} = \sum_i \frac{1}{2} m_i s^2 \dot{\mathbf{r}}_i^2 + U(\mathbf{r}_i) + \frac{1}{2} Q \dot{s}^2 - (n_f + 1) k_B T \ln s,$$

where $\mathbf{r}_i$ and $\dot{\mathbf{r}}_{\mathbf{i}}$ are respectively the position and velocity of particle i, U is the potential energy, in this case the DFT total energy, s is a dimensionless quantity that can be interpreted post-hoc as a time step scaling factor, Q is the fictitious mass of the heat bath and n_f is the number of ionic degrees of freedom. Hamilton's equations can be solved to generate the Nose-Hoover equations of motion. However Martyna *et al.* demonstrate that this method does not generate an ergodic trajectory, and proposed an alternative formulation [Ta8] in which the temperature is controlled

by a chain of M coupled thermostats of mass Q_k, notional position η_k and conjugate momentum p_{η_k}:

$$\dot{\mathbf{r}}_{\mathbf{i}} = \frac{\mathbf{p}_i}{m_i}$$

$$\dot{\mathbf{p}}_{\mathbf{i}} = -\frac{\partial U(\mathbf{r})}{\partial \mathbf{r}_i} - \frac{p_{\eta_1}}{Q_1}\mathbf{p}_i$$

$$\dot{\eta}_k = \frac{p_{\eta_k}}{Q_k}$$

$$\dot{p}_{\eta_1} = \left(\sum_{i=1}^{N}\frac{\mathbf{p}_i}{m_i} - n_f k_B T\right) - \frac{p_{\eta_2}}{Q_{\eta_2}}p_{\eta_1}$$

$$\dot{p}_{\eta_k} = \left(\frac{p_{\eta_{k-1}}^2}{Q_{k-1}} - k_B T\right) - \frac{p_{\eta_{k+1}}}{Q_{k+1}}p_{\eta_k}$$

$$\dot{p}_{\eta_M} = \left(\frac{p_{\eta_{M-1}}^2}{Q_{M-1}} - k_B T\right)$$

The Liouvillian for these equations of motion can be non-uniquely decomposed into components of ionic position (iL_r) and momentum (iL_p) as in the microcanonical case, the extended Lagrangian $(iL_{\mathrm{XL}}$, and a Nose-Hoover chain component (iL_{NHC})

$$iL = iL_{\mathrm{NHC}} + iL_p + iL_{\mathrm{XL}} + iL_r,$$

which is directly translated into an algorithm with the Trotter-Suzuki expansion,

$$\exp(iL\Delta t) = \exp\left(iL_{\mathrm{NHC}}\frac{\Delta t}{2}\right)\exp\left(iL_p\frac{\Delta t}{2}\right) \times$$
$$\exp\left(iL_{\mathrm{XL}}\frac{\Delta t}{2}\right)\exp\left(iL_r\Delta t\right)\exp\left(iL_{\mathrm{XL}}\frac{\Delta t}{2}\right) \times$$
$$\exp\left(iL_p\frac{\Delta t}{2}\right)\exp\left(iL_{\mathrm{NHC}}\frac{\Delta t}{2}\right)$$

This is recognisable as the velocity Verlet algorithm with extended Lagrangian integration which can be reduced to a single step, as described in *Extended Lagrangian Born-Oppenheimer MD (XL-BOMD)*, with a half time step integration of the Nose-Hoover chain equations of motion before and after. For full details of the integration scheme, see Hirakawa *et al.* [Ta9].

Go to *top*.

Isobaric-Isothermal (NPT) ensemble

The Parinello-Rahman equations of motion [Ta10] extend the fixed cell equations of motion to include the cell degrees of freedom in the extended system approach. We use the Martyna-Tobias-Tuckerman-Klein modification [Ta11], which couples the variable cell equations of motion to a Nose-Hoover chain the themrostat the system, recovering the isobaric-isothermal (NPT) ensemble. For an unconstrained cell (i.e. the lattice vectors can change freely), the equations of motion are,

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i} + \frac{\mathbf{p}_g}{W_g}\mathbf{r}_i$$

$$\dot{\mathbf{p}}_i = \mathbf{F}_i - \frac{\mathbf{p}_g}{W_g}\mathbf{p}_i - \left(\frac{1}{N_f}\right)\frac{\mathrm{Tr}[\mathbf{p}_g]}{W_g}\mathbf{p}_i - \frac{p_\xi}{Q}\mathbf{p}_i$$

$$\dot{\mathbf{h}} = \frac{\mathbf{p}_g\mathbf{h}}{W_g}$$

$$\dot{\mathbf{p}}_\mathbf{g} = V(\mathbf{P}_{\mathrm{int}} - \mathbf{I}P_{\mathrm{ext}}) + \left[\frac{1}{N_f}\sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i}\right]\mathbf{I} - \frac{p_\xi}{Q}\mathbf{p}_g$$

$$\dot{\xi} = \frac{p_\xi}{Q}$$

$$\dot{\mathbf{p}}_g = \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i} + \frac{1}{W_g}\mathrm{Tr}[\mathbf{p}_g^T\mathbf{p}_g] - (N_f + d^2)kT$$

Here, $\mathbf{r}_i$, $\mathbf{p}_i$ and m_i are the position, momentum and mass of particle i respectively, ξ, p_ξ and Q are the position, momentum and mass of the thermostat, and $\mathbf{h}$, $\mathbf{p}_g$ and W_g are the matrix of lattice vectors, matrix of cell velocities and cell mass respectively. Note that these equations only include one Nose-Hoover thermostat for simplicity. Conquest uses the Shinoda-Shiga-Mikami splitting of the Liouvillian [Ta12] to propagate the system. The Liouvillian is decomposed as,

$$iL = iL_r + iL_h + iL_v + iL_{\mathrm{bath}},$$

which can be further split,

$$iL_{\mathrm{bath}} = iL_{\mathrm{box}} + iL_{\mathrm{particles}}$$
$$iL_{\mathrm{box}} = iL_{\mathrm{vbox}} + iL_\xi + iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}}$$
$$iL_{\mathrm{particles}} = iL_{\mathrm{vpart}} + iL_\xi + iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}}$$

Using Liouville's theorem, we have,

$$iL_r = \sum_{i=1}^{N} [\mathbf{v}_i + \mathbf{v}_g \mathbf{r}_i] \cdot \nabla_{\mathbf{r}_i}$$

$$iL_h = \sum_{\alpha,\beta} \mathbf{v}_{g,\alpha\beta} \mathbf{h}_{\alpha\beta} \frac{\partial}{\partial \mathbf{h}_{\alpha\beta}}$$

$$iL_v = \sum_{i=1}^{N} \left(\frac{\mathbf{F}_i}{m_i} \right) \cdot \nabla_{\mathbf{v}_i}$$

$$iL_{\text{bath}} = iL_{\text{vpart}} + iL_{\text{vbox}} + iL_\xi + iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}}$$

$$= \sum_{i=1}^{N} \left[- \left\{ \mathbf{v}_g + \frac{1}{N_f} \text{Tr}(\mathbf{v}_g) + v_{\xi_1} \right\} \mathbf{v}_i \right] \nabla_{\mathbf{v}_i}$$

$$+ \sum_{\alpha,\beta} \left[\frac{F_{\text{box}}}{W} - v_{\xi_1} \mathbf{v}_{g,\alpha\beta} \right] \frac{\partial}{\partial \mathbf{v}_{g,\alpha\beta}}$$

$$+ \sum_{k=1}^{M} v_{\xi_k} \frac{\partial}{\partial \xi_k}$$

$$+ \left[\frac{F_{\text{NHC}_1}}{Q_1} - v_{\xi_1} v_{\xi_2} \right] \frac{\partial}{\partial v_{\xi_1}}$$

$$+ \sum_{k=2}^{M} \left[\frac{1}{Q_k} (Q_{k-1} v_{\xi_{k-1}}^2 - kT_{\text{ext}}) - v_{\xi_k} v_{\xi_{k+1}} \right] \frac{\partial}{\partial v_{\xi_k}}$$

$$+ \left[\frac{1}{Q_M} (Q_{M-1} v_{\xi_{M-1}}^2 - kT_{\text{ext}}) \right] \frac{\partial}{\partial v_{\xi_M}}$$

Here we use M heat baths in a Nose-Hoover chain. The Trotter-Suzuki expansion is,

$$e^{iL\Delta t} = e^{iL_{\text{bath}} \frac{\Delta t}{2}} e^{iL_v \frac{\Delta t}{2}} e^{iL_h \frac{\Delta t}{2}} e^{iL_r \Delta t} e^{iL_h \frac{\Delta t}{2}} e^{iL_v \frac{\Delta t}{2}} e^{iL_{\text{bath}} \frac{\Delta t}{2}}.$$

The Liouvillian for the heat baths can be further expanded:

$$e^{iL_{\text{particles}} \frac{\Delta t}{2}} = e^{\left(iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}} \right) \frac{\Delta t}{4}} e^{(iL_\xi + iL_{\text{vpart}}) \frac{\Delta t}{2}} e^{\left(iL_\xi + iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}} \right) \frac{\Delta t}{4}}$$

Finally, expanding the first propagator in the previous expression, we have,

$$e^{\left(iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}} \right) \frac{\Delta t}{4}} = e^{-i \left(-v_{\xi_1} v_{\xi_2} \frac{\partial}{\partial \xi_1} - \sum_{k=2}^{M} v_{\xi_k} v_{\xi_{k+1}} \frac{\partial}{\partial \xi_k} - v_{\xi_{M-1}} v_{\xi_M} \frac{\partial}{\partial \xi_M} \right) \frac{\Delta t}{8}}$$

$$\times e^{i \left(F_{\text{NHC}_1} \frac{\partial}{\partial v_{\xi_1}} + F_{\text{NHC}_k} \frac{\partial}{\partial v_{\xi_k}} + F_{\text{NHC}_M} \frac{\partial}{\partial v_{\xi_M}} \right) \frac{\Delta t}{4}}$$

$$\times e^{-i \left(-v_{\xi_1} v_{\xi_2} \frac{\partial}{\partial \xi_1} - \sum_{k=2}^{M} v_{\xi_k} v_{\xi_{k+1}} \frac{\partial}{\partial \xi_k} - v_{\xi_{M-1}} v_{\xi_M} \frac{\partial}{\partial \xi_M} \right) \frac{\Delta t}{8}}$$

These expressions are directly translated into the integration algorithm.

Go to *top*.

Weak coupling thermostat/barostat

Instead of modifying the Hamiltonian, the Berendsen-type weak coupling method [Ta13] involves coupling the ionic degrees of freedom to a an external temperature and/or pressure bath via "the principle of least local perturbation consistent with the required global coupling." Thermostatting is acheived via a Langevin-type equation of motion, in which the system is globally coupled to a heat bath and subjected to random noise:

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + m_i \gamma \left(\frac{T_0}{T} - 1 \right) \dot{\mathbf{r}}_i,$$

where γ is a global friction constant chosen to be the same for all particles. This can be acheived in practice by rescaling the velocities $\mathbf{v}_i \to \lambda \mathbf{v}_i$, where λ is,

$$\lambda = \left[1 + \frac{\Delta t}{\tau_T} \left(\frac{T_0}{T} - 1 \right) \right]^{\frac{1}{2}}$$

A similar argument can be applied for weak coupling to an external pressure bath. In the isobaric-isoenthalpic ensemble, the velocity of the particles can be expressed,

$$\dot{\mathbf{r}} = \mathbf{v} - \frac{\beta(P_0 - P)}{3\tau_P} \mathbf{r},$$

i.e. the fractional coordinates are scaled by a factor determined by the difference between the internal and external pressures, the isothermal compressibility β and a pressure coupling time constant tau_P. In the isotropic case, the cell scaling factor μ can be expressed,

$$\mu = \left[1 - \frac{\Delta t}{\tau_P} (P_0 - P) \right]^{\frac{1}{3}},$$

where the compressibility is absorbed into the time time constant τ_P. Allowing for fluctuations of all cell degrees of freedom, the scaling factor becomes,

$$\mu = \mathbf{I} - \frac{\beta \Delta t}{3\tau_P} (\mathbf{P}_0 - \mathbf{P})$$

While trivial to implement and in general stable, the weak-coupling method does not recover the correct phase space distribution for the canonical or isobaric-isothermal ensembles, for which the extended system method is required. It is no longer supported in CONQUEST, but the concepts are useful.

Go to *top*.

Stochastic velocity rescaling

Stochastic velocity rescaling (SVR) [Ta14] is a modification of the weak coupling method, in which a correctly constructed random force is added to enforce the correct NVT (or NPT) phase space distribution. The kinetic energy is rescaled such that the change in kinetic energy between thermostatting steps is,

$$dK = (\bar{K} - K) \frac{dt}{\tau} + 2 \sqrt{\frac{K\bar{K}}{N_f}} \frac{dW}{\sqrt{\tau}}$$

where $\bar{K}$ is the target kinetic energy (external temperature), dt is the time step, τ is the time scale of the thermostat, N_f is the number of degrees of freedom and dW is a Wiener process. Practically, the particle velocities are rescaled by a factor of α, defined via,

$$\alpha^2 = e^{-\Delta t/\tau} + \frac{\bar{K}}{N_f K} \left(1 - e^{-\Delta t/\tau} \right) \left(R_1^2 + \sum_{i=2}^{N_f} R_i^2 \right) + 2e^{-\Delta t/2\tau} \sqrt{\frac{\bar{K}}{N_f K} \left(1 - e^{-\Delta t/\tau} \right)} R_1$$

Where R_i is a set of N_f normally distributed random numbers with unitary variance. This method can be applied to thermostat the NPT ensemble by barostatting the system with the Parinello-Rahman method, and using the above expressions, but with additional R_i's for the cell degrees of freedom, and thermostatting the cell velocities as well as the particle velocities [Ta15].

Go to *top*.

FIVE

GET IN TOUCH

- If you have suggestions for developing the code, please use GitHub issues. The developers cannot guarantee to offer support, though we will try to help.

- Report bugs, or suggest features on GitHub issues. View the source code on the main GitHub page.

- You can ask for help and discuss any problems that you may have on the Conquest mailing list (to register for this list, please send an email to mlsystem@ml.nims.go.jp with the subject "sub conquest-user", though please note that, for a little while, some of the system emails may be in Japanese; you will receive a confirmation email to which you should simply reply without adding any text).

LICENCE

CONQUEST is available freely under the open source MIT Licence. We ask that you acknowledge use of the code by citing appropriate papers, which will be given in the output file (a BiBTeX file containing these references is also output). The key CONQUEST references are:

- D. R. Bowler, T. Miyazaki and M. J. Gillan, J. Phys. Condens. Matter **14**, 2781–2798 (2002)

- T. Miyazaki, D. R. Bowler, R. Choudhury and M. J. Gillan, J. Chem. Phys. **121**, 6186–6194 (2004)

- D. R. Bowler, R. Choudhury, M. J. Gillan and T. Miyazaki, phys. stat. sol. b **243**, 989–1000 (2006)

BIBLIOGRAPHY

[G1]   J. Harris. Simplified method for calculating the energy of weakly interacting fragments. *Phys. Rev. B*, 31:1770, 1985. doi:10.1103/PhysRevB.31.1770.

[G2]   W. Foulkes and R. Haydock. Tight-binding models and density-functional theory. *Phys. Rev. B*, 39:12520, 1989. doi:10.1103/PhysRevB.39.12520.

[G3]   H. J. Monkhorst and J. D. Pack. Special points for brillouin-zone integrations. *Phys. Rev. B*, 13:5188, 1976. doi:10.1103/PhysRevB.13.5188.

[G4]   M. Methfessel and A. T. Paxton. High-precision sampling for brillouin-zone integration in metals. *Phys. Rev. B*, 40:3616–3621, Aug 1989. URL: https://link.aps.org/doi/10.1103/PhysRevB.40.3616, doi:10.1103/PhysRevB.40.3616.

[B1]   David R. Bowler, Jack S. Baker, Jack T. L. Poulton, Shereif Y. Mujahed, Jianbo Lin, Sushma Yadav, Zamaan Raza, and Tsuyoshi Miyazaki. Highly accurate local basis sets for large-scale DFT calculations in conquest. *Jap. J. Appl. Phys.*, 58:100503, 2019. doi:10.7567/1347-4065/ab45af.

[B2]   Ayako Nakata, David R. Bowler, and Tsuyoshi Miyazaki. Efficient Calculations with Multisite Local Orbitals in a Large-Scale DFT Code CONQUEST. *J. Chem. Theory Comput.*, 10:4813, 2014. doi:10.1039/C5CP00934K.

[B3]   Ayako Nakata, David Bowler, and Tsuyoshi Miyazaki. Optimized multi-site local orbitals in the large-scale DFT program CONQUEST. *Phys. Chem. Chem. Phys.*, 17:31427, 2015. doi:10.1021/ct5004934.

[B4]   E. Hernández, M. J. Gillan, and C. M. Goringe. Basis functions for linear-scaling first-principles calculations. *Phys. Rev. B*, 55:13485–13493, 1997. doi:10.1103/PhysRevB.55.13485.

[B5]   P.D. Haynes, C.-K. Skylaris, A.A. Mostofi, and M.C. Payne. Elimination of basis set superposition error in linear-scaling density-functional calculations with local orbitals optimised in situ. *Chem. Phys. Lett.*, 422:345 – 349, 2006. doi:10.1016/j.cplett.2006.02.086.

[B6]   S.F. Boys and F. Bernardi. The calculation of small molecular interactions by the differences of separate total energies. some procedures with reduced errors. *Mol. Phys.*, 19:553, 1970. doi:10.1080/00268977000101561.

[SR1] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch. Structural Relaxation Made Simple. *Phys. Rev. Lett.*, 97:2897, 2006. doi:10.1103/PhysRevLett.97.170201.

[MD1] A. M. N. Niklasson. Extended Born-Oppenheimer Molecular Dynamics. *Phys. Rev. Lett.*, 100:123004, 2008. doi:10.1103/PhysRevLett.100.123004.

[MD2] G. Bussi, D. Donadio, and M. Parrinello. Canonical sampling through velocity rescaling. *J. Chem. Phys.*, 126:014101, 2007. doi:10.1063/1.2408420.

[MD3] S. Nosé. A unified formulation of the constant temperature molecular dynamics methods. *J. Chem. Phys.*, 81:511, 1984. doi:10.1063/1.447334.

[MD4]   W. G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695, 1985. doi:10.1103/PhysRevA.31.1695.

[MD5]   G. J. Martyna, M. L. Klein, and M. Tuckerman. Nosé–hoover chains: the canonical ensemble via continuous dynamics. *J. Chem. Phys.*, 97:2635, 1992. doi:10.1063/1.463940.

[MD6]   M. Parrinello and A. Rahman. Polymorphic transitions in single crystals: A new molecular dynamics method. *J. Appl. Phys.*, 52:7182–7190, December 1981. doi:10.1063/1.328693.

[E1]   J. P. Perdew and A. Zunger. Self-interaction correction to density-functional approximations for many-electron systems. *Phys. Rev. B*, 23:5048, 1981. doi:10.1103/PhysRevB.23.5048.

[E2]   S. Goedecker, M. Teter, and J. Hutter. Separable dual-space gaussian pseudopotentials. *Phys. Rev. B*, 54:1703, 1996. doi:10.1103/PhysRevB.54.1703.

[E3]   John P. Perdew and Yue Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, 45:13244, 1992. doi:10.1103/PhysRevB.45.13244.

[E4]   J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, 77(18):3865, 1996. doi:10.1103/PhysRevLett.77.3865.

[E5]   Y. Zhang and W. Yang. Comment on "generalized gradient approximation made simple". *Phys. Rev. Lett.*, 80(4):890–890, 01 1998. URL: https://link.aps.org/doi/10.1103/PhysRevLett.80.890, doi:10.1103/PhysRevLett.80.890.

[E6]   B. Hammer, L. B. Hansen, and J. K. Nørskov. Improved adsorption energetics within density-functional theory using revised perdew-burke-ernzerhof functionals. *Phys. Rev. B*, 59:7413, 1999. doi:10.1103/PhysRevB.59.7413.

[E7]   Z. Wu and R. E. Cohen. More accurate generalized gradient approximation for solids. *Phys. Rev. B*, 73(23):235116, 06 2006. doi:10.1103/PhysRevB.73.235116.

[E8]   X.-P. Li, R. W. Nunes, and D. Vanderbilt. Density-matrix electronic-structure method with linear system-size scaling. *Phys. Rev. B*, 47:10891, 1993. doi:10.1103/physrevb.47.10891.

[E9]   A. H. R. Palser and D. E. Manolopoulos. Canonical purification of the density matrix in electronic-structure theory. *Phys. Rev. B*, 58(19):12704, 1998. doi:10.1103/PhysRevB.58.12704.

[E10]   E. Artacho, D. Sanchez-Portal, P. Ordejon, A. Garcia, and J. M. Soler. Linear-scaling ab-initio calculations for large and complex systems. *Phys. Stat. Solidi B*, 215:809, 1999. doi:10.1002/(SICI)1521-3951(199909)215:1<809::AID-PSSB809>3.0.CO;2-0.

[E11]   E. Hernández, M. J. Gillan, and C. M. Goringe. Basis functions for linear-scaling first-principles calculations. *Phys. Rev. B*, 55:13485–13493, 1997. doi:10.1103/PhysRevB.55.13485.

[Tb1]   E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch. Structural Relaxation Made Simple. *Phys. Rev. Lett.*, 97:2897, 2006. doi:10.1103/PhysRevLett.97.170201.

[Tb2]   B. G. Pfrommer, M. Côté, S. Louie, and M. L. Cohen. Relaxation of Crystals with the Quasi-Newton Method. *J. Comput. Phys.*, 131:233, 1997. doi:10.1006/jcph.1996.5612.

[Ta1]   D. Frenkel and B. Smit. *Understanding molecular simulation: from algorithms to application*. Academic Press, 2002.

[Ta2]   A. M. N. Niklasson. Extended Born-Oppenheimer Molecular Dynamics. *Phys. Rev. Lett.*, 100:123004, 2008. doi:10.1103/PhysRevLett.100.123004.

[Ta3]   A. M. N. Niklasson and M. J. Cawkwell. Generalized extended Lagrangian Born-Oppenheimer molecular dynamics. *J. Chem. Phys.*, 141:164123, 2014. doi:10.1063/1.4898803.

[Ta4]   M. Arita, D. R. Bowler, and T. Miyazaki. Stable and Efficient Linear Scaling First-Principles Molecular Dynamics for 10000+ Atoms. *J. Chem. Theor. Comput.*, 10:5419, 2014. doi:10.1021/ct500847y.

[Ta5]   M. E. Tuckerman. *Statistical mechanics: theory and molecular simulations*. Oxford Graduate Texts, 2010.

[Ta6] S. Nosé. A unified formulation of the constant temperature molecular dynamics methods. *J. Chem. Phys.*, 81:511, 1984. doi:10.1063/1.447334.

[Ta7] W. G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695, 1985. doi:10.1103/PhysRevA.31.1695.

[Ta8] G. J. Martyna, M. L. Klein, and M. Tuckerman. Nosé–hoover chains: the canonical ensemble via continuous dynamics. *J. Chem. Phys.*, 97:2635, 1992. doi:10.1063/1.463940.

[Ta9] T. Hirakawa, T. Suzuki, D. R. Bowler, and T. Miyazaki. Canonical-ensemble extended lagrangian born-oppenheimer molecular dynamics for the linear scaling density functional theory. *J. Phys.: Condens. Matter*, 29:405901, 2017. doi:10.1088/1361-648X/aa810d.

[Ta10] M. Parrinello and A. Rahman. Polymorphic transitions in single crystals: A new molecular dynamics method. *J. Appl. Phys.*, 52:7182–7190, December 1981. doi:10.1063/1.328693.

[Ta11] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, and M. L. Klein. Explicit reversible integrators for extended systems dynamics. *Mol. Phys.*, 87:1117, 1996. doi:10.1080/002689799163235.

[Ta12] W. Shinoda, M. Shiga, and M. Mikami. Rapid estimation of elastic constants by molecular dynamics simulation under constant stress. *Phys. Rev. B*, 69:4396, 2004. doi:10.1103/PhysRevB.69.134103.

[Ta13] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. Dinola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.*, 81:3684, 1984. doi:10.1063/1.448118.

[Ta14] G. Bussi, D. Donadio, and M. Parrinello. Canonical sampling through velocity rescaling. *J. Chem. Phys.*, 126:014101, 2007. doi:10.1063/1.2408420.

[Ta15] G. Bussi, T. Zykova-Timan, and M. Parrinello. Isothermal-isobaric molecular dynamics using stochastic velocity rescaling. *J. Chem. Phys.*, 130:074101, 2009. doi:10.1063/1.3073889.