CONQUEST Documentation

Release 1.4

CONQUEST Developers

Jan 14, 2025

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).

CHAPTER

ONE

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 DFT

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

Go to *top*

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 (v4.x or later).

The library locations are set in the `system.make` file in the `src/system` directory, along with other parameters needed for compilation. The default file name is `system.make` but you can select another file with `make SYSTEM=label` which would then use the file `system.label.make` in the `src/system` directory. `system.<systemname>.make` files are provided for some HPC systems used by the community, but if you want to run locally or on a different system, you will need to create an appropriate `system.make` file. Use `src/system/system.example.make` as a starting point.

- `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_v4` for LibXC v4.x, or `XC_LIBRARY=LibXC_v5` for LibXC v5.x and v6.x)
- 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.

Go to *top*

Multi-threading

CONQUEST can use OpenMP for multi-threading; some multi-threading is available throughout the code, while there are specific matrix multiplication routines which can use multi-threading for the linear scaling solver. The number of threads is set via the environment variable `OMP_NUM_THREADS`.

Compiler flags to enable OpenMP are dependent on the vendor, but should be specified via `OMPFLAGS` in the `system.make` file. If compiling with OpenMP then you should also change the variable `OMP_DUMMY` in the same file to be blank to enable the number of threads to be included in the output.

On some systems, the default stack size for OpenMP is set to be rather small, and this can cause a segmentation fault when running with multiple threads. We recommend testing the effect of the environment variable `OMP_STACKSIZE` (and suggest setting it to 50M or larger as a first test).

Most OpenMP multi-threading in CONQUEST uses the `runtime` schedule. This means the type of scheduling of work to threads can be set by the user by setting the `OMP_SCHEDULE` **`variable<https://www.openmp.org/spec-html/5.0/openmpse49.html>`_**. If the variable is unset, OpenMP will use a default implementation defined schedule.

Go to *top*

1.4.3 Installing with Spack

CONQUEST and all of its dependencies can be installed with Spack. The CONQUEST package requires Spack v0.21 or later. If Spack isn't available or up to date on your system, it is relatively straightforward to install it with user permissions following the install instructions. When setting up Spack on a new system, it is recommended to configure it to use available system compilers and system packages. Once spack is installed and set up, install CONQUEST with:

```
spack install conquest
```

and load the `Conquest` executable to `PATH` with

```
spack load conquest
```

The build can be customized by adding options to the Spack spec `conquest`. The CONQUEST package includes variants for OpenMP support and different matrix multiplication kernels; more details can be found in the Spack CONQUEST package.

1.4.4 Installing on Ubuntu

CONQUEST can be compiled on Ubuntu after installing the required packages. The below instructions are given for Ubuntu 22.04 LTS and Ubuntu 24.04 LTS. The source files will be downloaded into the `${USER}/local/src` directory. The ${USER} variable will be automatically replaced by the current username. If compilation is successful, the executable file can be found in `${USER}/local/src/conquest_master/bin`.

Install needed packages

```
sudo apt update
sudo apt upgrade

sudo apt install -y build-essential libtool                                  # GCC and other␣
↪tools for software development
sudo apt install -y openmpi-bin libopenmpi-dev                               # MPI
sudo apt install -y libfftw3-dev                                             # FFT
sudo apt install -y libblas-dev liblapack-dev libscalapack-openmpi-dev # Linear algebra
```

Install libxc

```
cd $HOME
mkdir -p ${HOME}/local/src
cd ${HOME}/local/src

wget https://gitlab.com/libxc/libxc/-/archive/6.2.2/libxc-6.2.2.tar.bz2 -O libxc.tar.bz2
tar -xf libxc.tar.bz2
cd libxc-6.2.2 && autoreconf -i && ./configure --prefix=$HOME/local
make
make check && make install
```

Download CONQUEST

```
cd ${HOME}/local/src
git clone https://github.com/OrderN/CONQUEST-release.git conquest_master
cd conquest_master/src
```

Prepare makefile

```
# Prepare system.make file for Ubuntu.
cat > system/system.make << EOF

# Set compilers
FC=mpif90

# OpenMP flags
# Set this to "OMPFLAGS= " if compiling without openmp
# Set this to "OMPFLAGS= -fopenmp" if compiling with openmp
OMPFLAGS=
# Set this to "OMP_DUMMY = DUMMY" if compiling without openmp
# Set this to "OMP_DUMMY = " if compiling with openmp
OMP_DUMMY = DUMMY

# Set BLAS and LAPACK libraries
# MacOS X
# BLAS= -lvecLibFort
# Intel MKL use the Intel tool
# Generic
BLAS= -llapack -lblas
# Full scalapack library call; remove -lscalapack if using dummy diag module.
# If using OpenMPI, use -lscalapack-openmpi instead.
# If using Cray-libsci, use -llibsci_cray_mpi instead.
SCALAPACK = -lscalapack-openmpi

# LibXC compatibility
# Choose LibXC version: v4 (deprecated) or v5/6 (v5 and v6 have the same interface)
# XC_LIBRARY = LibXC_v4
XC_LIBRARY = LibXC_v5
XC_LIB = -lxcf90 -lxc
XC_COMPFLAGS = -I\${HOME}/local/include -I/usr/local/include

# Set FFT library
```

(continues on next page)

(continued from previous page)

```
FFT_LIB=-lfftw3
FFT_OBJ=fft_fftw3.o

LIBS= \$(FFT_LIB) \$(XC_LIB)  \$(SCALAPACK) \$(BLAS)

# Compilation flags
# NB for gcc10 you need to add -fallow-argument-mismatch
COMPFLAGS= -O3 \$(OMPFLAGS) \$(XC_COMPFLAGS) -fallow-argument-mismatch

# Linking flags
LINKFLAGS= -L\${HOME}/local/lib -L/usr/local/lib \$(OMPFLAGS)

# Matrix multiplication kernel type
MULT_KERN = default
# Use dummy DiagModule or not
DIAG_DUMMY =

EOF
```

Compile CONQUEST

```
dos2unix ./makedeps       # For Windows Subsystem for Linux (WSL), there may be some␣
↪incompatibilities thus file conversion is recommended.
make                      # Or make -j`nproc` for parallel compilation using all␣
↪available cores
```

Go to *top*

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` util-

ity (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

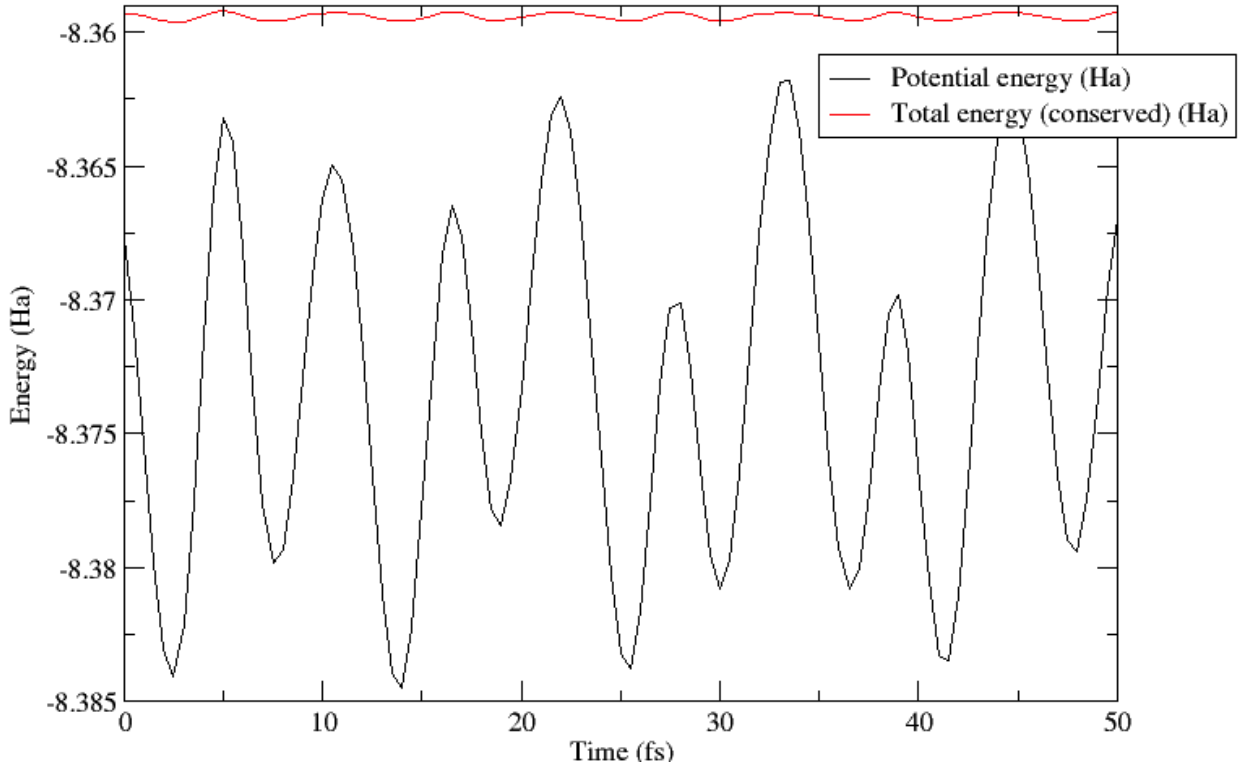
We will perform NVE molecular dynamics for methane, CH4, as a simple example of how to do this kind of calculation. You should use the same coordinate file and ion files as you did for the structural relaxation, but change the atomic movement flags in the coordinate file to allow all atoms to move (the 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*.

CHAPTER

TWO

USER GUIDE

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`, `sqnm`, `cg`
- `IO.Coordinates` File name
- `IO.Iprint` 0-3 (controls amount of output)* `DM.SolutionMethod` diagon
  - `Diag.MPMesh` T/F
  - `Diag.MPMeshX` (and `Y` and `Z`) N
  - `Diag.GammaCentred` T/F

- `Grid.GridCutoff` Energy in Ha (sets a grid spacing $\delta x = \pi/\sqrt{2E}$ for cutoff E in Ha; this spacing can also be set manually using `Grid.GridSpacing` in Bohr)
- `AtomMove.NumSteps` N
- `AtomMove.MaxForceTol` in Ha/bohr
- `AtomMove.OptCell` T/F (optimises simulation cell size)
- `General.NumberOfSpecies` N
- `%block ChemicalSpeciesLabel` Block specifying element number, mass and ion file name
- `IO.FractionalAtomicCoords` T/F
- `Spin.SpinPolarised` T/F
  - `Spin.FixSpin` T/F
  - `Spin.Magn` Difference between spin channel occupations
- `minE.SCTolerance` Fractional tolerance on magnitude of residual divided by number of electrons
- `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 the LDA, PBE and PBEsol 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 input file is `Conquest_ion_input`, and the key parameters to be set 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, and generally named `XX.pot` for species XX)
  - `Atom.BasisSize` to specify the size of the basis; at present this can take the values: `minimal`; `small`; `medium`; and `large`.

These are all included in the default input files. 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 T`). 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. For production runs, we expect `IO.Iprint 0` to give sufficient detail; `IO.Iprint 3` provides a level of detail that would normally only be needed for debugging.

Warnings from the calculation (including indications that the convergence should be improved, and technical issues) are written to the `Conquest_warnings` file, which should be checked after each run. The warnings are also written to the output file at certain `IO.Iprint` levels.

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 `eigenvalues.dat` 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:

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

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
```

For the Monkhorst-Pack approach, instead of specifying the number of points in x, y and z explicitly, they can be set automatically by giving a spacing in reciprocal space: `Diag.dk` where units are inverse Bohr radii. The number of points will be chosen so that $2\pi/(a \times dk)$ is less than the value specified.

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*.

Padding Hamiltonian matrix by setting block size

With the default setting, the size of Hamiltonian and overlap matrices is determined by the total number of support functions. It can be a prime number and timing of diagonalisation can be very slow in such cases, since the division of the matrix into small pieces is difficult.

By padding, we can change the size of Hamiltonian matrix to improve the efficiency of the diagonalisation. To set an appropriate value for the block size of the matrix, specify the following two variables.

```
Diag.BlockSizeR       20
Diag.BlockSizeC       20
```

Note that these two numbers should be the same when padding (and when using ELPA which will be introduced to CONQUEST soon). We suggest that an appropriate value is between 20 and 200, but this should be tested.

The option for padding was introduced after v1.2, and if you would like to remove it, set the following variable.

```
Diag.PaddingHmatrix             F
```

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*.

Restarting SCF

The SCF cycle can be restarted from a previous density matrix or charge density, which may significantly speed up convergence. The density matrix is automatically written out in the files `Kmatrix2.*` or `Lmatrix2.*` (depending on whether diagonalisation or linear scaling is being used). These files are read in, and the initial charge density made from them by setting the flags:

```
General.LoadDM T
SC.MakeInitialChargeFromK T
```

The charge density is not written out by default; this can be changed by setting `IO.DumpChargeDensity T` which results in the files `chden.nnn` being created. To read these in as the initial charge density, the flag `General.LoadRho T` should be set.

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:

$$\rho^{in}_{n+1} = \sum_i \alpha_i \left[\rho^{in}_i + AR_i\right]$$

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
```

(continues on next page)

(continued from previous page)

```
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 (in Bohr radii):

```
Grid.GridSpacing d
```

Or it can be set 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:

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

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, SZ)
- 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
```

(continues on next page)

(continued from previous page)

```
# 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 $\frac{2\pi}{\sqrt{2E_{\mathrm{cutoff}}}}$ 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 Electronic Structure

CONQUEST can be used to produce a wide variety of information on the electronic structure of different systems, including: density of states (DOS) and atom-projected DOS (or pDOS); band-resolved charge density; band structure; and electronic polarisation. Many of these are produced with the *post-processing* code using a *converged charge*

density. All of these (at present) require the exact diagonalisation approach to the ground state; linear scaling solutions are not possible.

2.5.1 Converged charge density

In most cases (except *polarisation*) the data required is produced by a *non-self-consistent* calculation which reads in a well-converged charge density. The convergence is mainly with respect to *Brillouin zone sampling*, but also self-consistency (a tight tolerance should be used). The basic procedure is:

1. Perform a well-converged calculation, writing out charge density (ensure that the Brillouin zone is well sampled, the SCF tolerance is tight (`minE.SCTolerance`) and that the flag `IO.DumpChargeDensity T` is set)
2. Perform a non-self-consistent calculation for the quantity desired (set `minE.SelfConsistent F` and `General.LoadRho T` to read and fix the charge density) using an appropriate Brillouin zone sampling
3. Run the appropriate *post-processing* to generate the data

However, note that the charge density often converges much faster with respect to Brillouin zone sampling than the detailed electronic structure, so the use of a non-self-consistent calculation is more efficient. Often it is most efficient and accurate to use a very high density k-mesh for the final, non-SCF calculation, but a lower density k-mesh to generate the charge density (which converges faster with respect to Brillouin zone sampling than DOS and other quantities).

Go to *top*.

2.5.2 Density of states

The total density of states (DOS) is generated from the file `eigenvalues.dat` which is written by all diagonalisation calculations. See *density of states* for details on parameters which can be set.

The atom-projected DOS resolves the total DOS into contributions from individual atoms using the pseudo-atomic orbitals, and can further decompose this into *l*-resolved or *lm*-resolved densities of states. It requires the wave-function coefficients, which will be generated by setting `IO.write_proj_DOS T`; further analysis is performed in *post-processing*.

Go to *top*.

2.5.3 Band structure

The band structure along a series of lines in reciprocal space can be generated. See *post-processing* for more details.

Go to *top*.

2.5.4 Band-resolved densities

A band-resolved density is the quantity $\mid \psi_n(\mathbf{r}) \mid^2$ for the n^{th} Kohn-Sham eigenstate (we plot density because the eigenstates are in general complex). It requires wavefunction coefficients which are generated by setting `IO.outputWF T`. Full details are found in the *band density* section of the *post-processing* part of the manual.

Go to *top*.

2.5.5 Electronic Polarisation

The electronic polarisation (the response of a material to an external electric field) can be calculated using the approach of Resta [ES1] by setting the tag `General.CalcPol T`. The direction in which polarisation is found is set using the tag `General.PolDir` (choosing 1-3 gives x, y or z, respectively, while choosing 0 gives all three directions, though this is normally not recommended).

The Resta approach is a version of the modern theory of polarisation (MTP) (perhaps better known in the method of King-Smith and Vanderbilt [ES2]) where the polarisation is found as:

$$\mathbf{P} = -\frac{e\mathrm{L}}{\pi V}\mathrm{Imln}\mathrm{det}\mathbf{S}$$
$$\mathrm{S}_{mn} = \langle\psi_m|\exp i2\pi\mathbf{r}/L|\psi_n\rangle$$

where L is a simulation cell length along an appropriate direction and V is the simulation cell volume. This approach is **only valid** in the large simulation cell limit, with Γ point sampling (e.g. for BaTiO3, a minimum of 3x3x3 formula units is needed, though this is perhaps a little too small).

As with all calculations in the MTP, the only valid physical quantity is a *change* of polarisation between two configurations. A very common quantity to calculate is the Born effective charge (BEC), which is defined as $Z^*_{k,\alpha\beta} = V\partial P_\alpha/\partial u_{k,\beta}$ for species k and Cartesian directions α and β. It is most easily calculated by finding the change in polarisation as one atom (or one set of atoms in a sublattice) is moved a small amount.

Go to *top*.

Go to *top*.

2.6 Structural relaxation

This section describes how to find the zero-Kelvin equilibrium atomic structure, given a starting structure with non-zero forces and/or stresses. CONQUEST can employ a variety of algorithms to minimise energy with respect to atomic positions, including: stabilised quasi-Newton method (SQNM); L-BFGS; conjugate gradients (CG); 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 and XCrysDen. Setting `AtomMove.AppendCoords T` will append the structure at each step to `UpdatedAtoms.dat` in the format of a CONQUEST structure input.

For the SQNM, 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*.

2.6.1 Ionic relaxation

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

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

The parameter `AtomMove.TypeOfRun` can take the values `sqnm`, `lbfgs` or `cg` for iterative optimisation. All three algorithms are robust and relatively efficient in most instances; SQNM [SR1] is recommended in most cases, though if the initial forces are large it may be worth performing quenched MD to reduce them (see below) before applying SQNM. 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. When using CG, the line minimiser can be chosen: `safe` uses a robust though sometimes slow line minimiser; `backtrack` uses a simple back-tracking line minimiser (starting with a step size of 1 and reducing if necessary to ensure the energy goes down); `adapt` uses an adaptive back-tracking line minimiser (which increases the starting step size if the energy goes down on the first step). In many cases the back-tracking line minimiser is more efficient, though the efficiency of the adaptive approach varies with problem.

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 large initial forces or problematic cases where the relaxation algorithms fail to find a downhill search direction, it may be worth trying quenched molecular dynamics, 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 [SR2] 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*.

2.6.2 Simulation cell optimisation

The simulation 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.ReuseDM T
AtomMove.EnthalpyTolerance 1E-5
AtomMove.StressTolerance 0.1
```

Note that stress is in GPa and enthalpy is in Ha by default.

Go to *top*.

2.6.3 Combined optimisation

For simple crystals, the fractional ionic coordinates vary trivially with changes in the simulation cell lengths; however for more complicated systems such as molecular crystals and amorphous materials, it is necessary simultaneously relax the ionic positions and simulation cell lengths (recalling that CONQUEST only allows *orthorhombic* unit cells). This can be done by setting `AtomMove.OptCellMethod 2` or `AtomMove.OptCellMethod 3`

```
AtomMove.TypeOfRun cg
AtomMove.OptCell T
AtomMove.OptCellMethod 2
AtomMove.ReuseDM T
AtomMove.MaxForceTol 5e-4
AtomMove.EnthalpyTolerance 1E-5
AtomMove.StressTolerance 0.1
```

Note that stress is in GPa and enthalpy is in Ha by default.

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 and stress tolerance, if it is even possible. For combined optimisation, we recommend using `AtomMove.OptCellMethod 2`, which uses a simple but robust double-loop minimisation: a full ionic relaxation (using either cg or sqnm) followed by a full simulation cell relaxation (using cg). While this may be less efficient than optimising all degrees of freedom simultaneously, it is much more robust. It is also possible to optimise cell vectors and atomic positions simultaneously, using `AtomMove.OptCellMethod 3`, but this should be monitored carefully, as it can be unstable.

Go to *top*.

Go to *top*.

2.7 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*.

2.7.1 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*.

2.7.2 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*.

2.7.3 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*.

2.7.4 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*.

2.7.5 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.

2.7.6 Postprocessing tools

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

Go to *top*.

Go to *top*.

2.8 Post-processing CONQUEST output

2.8.1 Introduction

The utility `PostProcessCQ` allows users to post-process the output of a CONQUEST calculation, to produce structure files, densities of states, and charge density, band densities and STM images as CUBE files (which can be read by the freely available VESTA code).

There are a number of different analyses which can be performed: coordinate conversion (to formats which can be plotted); conversion of total charge density to CUBE file format; production of band-resolved (optionally k-point resolved) densities in CUBE file format; simple Tersoff-Hamann STM simulation; and calculation of densities of states, including projected DOS. You should ensure that all the files produced during the CONQUEST run are available for the post-processing (including `eigenvalues.dat`, `chden.NNN`, `make_blk.dat` or `hilbert_make_blk.dat` and `ProcessNNNNNNNWF.dat` and `ProcessSijNNNNNNNWF.dat` as applicable) as well as the input files.

Note that the utility reads the `Conquest_input` file, taking some flags from the CONQUEST run that generated the output, and some utility-specific flags that are detailed below.

Note also that projected DOS, band density and STM simulation are not at present compatible with multi-site support functions (MSSF), though we hope to implement this soon.

Go to *top*.

2.8.2 Coordinate conversion

Set `Process.Job coo` to output a coordinate file for further processing or plotting. The utility will read the file specified by `Process.Coordinates` (which defaults to the file specified by `IO.Coordinates`). The output format is selected by specifying the `Process.CoordFormat` tag. The default output format is XYZ (which adds a `.xyz` suffix to the file name) using `xyz`. The CASTEP `.cell` output format can also be selected using `cell`. We plan to expand this conversion to other formats in the future.

Note that for a structural relaxation or molecular dynamics calculation, if you do not specify `Process.Coordinates` then the `IO.Coordinates` file, which will be converted, will be the *input* structure, not the output structure. Parameters that can be set are:

```
Process.Coordinates string (default: IO.Coordinates value)
Process.CoordFormat string (default: xyz; options: xyz, cell)
```

Go to *top*.

2.8.3 Charge density

Setting `Process.Job` to `cha`, `chg` or `den` will convert the files `chden.NNN` which are written by CONQUEST to a cube file. The processing will use the files `chden.NNN`, `Conquest_input` and `hilbert_make_blk.dat` or `raster_make_blk.dat`. Parameters that can be set include:

```
Process.ChargeStub string (default: chden)
```

The ChargeStub simply defines the filename which will be read, and used for output.

Note that to output the `chden.NNN` files from CONQUEST, you must set the flag `IO.DumpChargeDensity T` in the CONQUEST run.

Go to *top*.

2.8.4 Band density

Setting `Process.Job` to `ban` produces band densities from wave function coefficients output by CONQUEST. The CONQUEST run must have the following tags set:

```
IO.outputWF T
```

A set of bands whose coefficients are output are specified either with an energy range (the default is to produce *all* bands):

```
IO.WFRangeRelative T/F
IO.min_wf_E real (Ha)
IO.max_wf_E real (Ha)
```

or with a list of bands:

```
IO.maxnoWF n

%block WaveFunctionsOut
n entries, each a band number
%endblock
```

The wavefunction range can be relative to the Fermi level (`IO.WFRangeRelative T`) otherwise it is absolute. Either of these will produce a file containing all eigenvalues at all k-points (`eigenvalues.dat`) and a series of files containing the wavefunction expansion coefficients for the selected bands (`ProcessNNNNNNNWF.dat`). These files are output as binary (unformatted) by default (this can be changed by setting `IO.MatrixFile.BinaryFormat F` before the CONQUEST run) and will be read using the same format (it is important to check this!).

From these wavefunction coefficient files, band densities can be produced in post-processing, using similar tags; either a range:

```
Process.min_wf_E real (Ha)
Process.max_wf_E real (Ha)
Process.WFRangeRelative T/F
```

or an explicit list of bands:

```
Process.noWF n

%block WaveFunctionsProcess
n entries, each a band number
%endblock
```

Note that the bands to be processed must be a subset of the bands output by CONQUEST. The bands can be output summed over k-points, or at individual k-points, by setting `Process.outputWF_by_kpoint` to `F` or `T` respectively.

Go to *top*.

2.8.5 Tersoff-Hamann STM simulation

Setting `Process.Job ter` will use a very simple Tersoff-Hamann approach to STM simulation, summing over band densities between the Fermi level and the bias voltage (this is often surprisingly accurate). The following parameters can be set:

```
STM.BiasVoltage    real (eV)
STM.FermiOffset    real (eV)
Process.MinZ       real (Bohr)
Process.MaxZ       real (Bohr)
Process.RootFile   string (default: STM)
```

The `FermiOffset` tag allows the user to shift the Fermi level (to simulate charging or an external field). The height of the simulation cell in which the STM image is calculated is set by the `MinZ` and `MaxZ` tags, and the filename by the `RootFile` tag.

Go to *top*.

2.8.6 Density of states (DOS)

Setting `Process.Job dos` will produce a total density of states (DOS) for the system, using the eigenvalues output by CONQUEST. The following parameters can be set:

```
Process.min_DOS_E real    (Ha, default lowest eigenvalue)
Process.max_DOS_E real    (Ha, default highest eigenvalue)
Process.sigma_DOS real    (Ha, default 0.001)
Process.n_DOS     integer (default 1001)
```

The limits for the DOS are set by the first two parameters (note that CONQUEST will output all eigenvalues, so the limits on these are set by the eigenspectrum). The broadening applied to each state is set by `sigma_DOS`, while the number of bins is set by `n_DOS`. The integrated DOS is also calculated; the user can choose whether this is the total integrated DOS (i.e. from the lowest eigenvalue, regardless of the lower limit for DOS) or just the local integrated DOS (i.e. over the interval specified for the DOS) by setting `Process.TotalIntegratedDOS` to `T` or `F`, respectively.

We recommend that, for accurate DOS, CONQUEST should be run non-self-consistently with a very high k-point density, after reading in a well-converged input charge density: set `minE.SelfConsistent F` and `General.LoadRho T` (which will require that the converged charge density is written out by CONQUEST by setting `IO.DumpChargeDensity T`).

Go to *top*.

2.8.7 Atom-projected DOS

Setting `Process.Job pdos` will produce a total density of states as above, as well as the density of states projected onto the individual atoms. Given support functions $\phi_{i\alpha}(\mathbf{r})$ which are the basis functions of the Kohn-Sham eigenstates $\psi_n(\mathbf{r}) = \sum_{i\alpha} c^n_{i\alpha}\phi_{i\alpha}(\mathbf{r})$, then the projection of a given state, n, onto an atom i can be written as $\sum_{\alpha j\beta} c^n_{i\alpha} S_{i\alpha,j\beta} c^{n\mathbf{k}}_{j\beta}$. The projected DOS is constructed using these projections.

If using *pseudo-atomic orbitals (PAOs)* as the basis set, then the atom-projected DOS can be further resolved by angular momentum (either just l or both l and m). If using *pseudo-atomic orbitals (PAOs)* with *multi-site support functions* or *blip functions* then it is not possible to decompose the DOS any further (in future, it may be possible to resolve the MSSF coefficients into the individual PAOs, and hence decompose pDOS by angular momentum). To output the necessary coefficients to produce atom-projected DOS, a CONQUEST run must be performed with the following parameters set:

```
IO.writeDOS T
IO.write_proj_DOS T
```

As for the DOS, very high Brillouin zone sampling is required for accurate projected DOS, which is most efficiently generated using a converged charge density and a non-self-consistent calculation with much higher k-point density. CONQUEST will produce the wavefunction files (`ProcessNNNNNNNWF.dat` and `ProcessSijNNNNNNNWF.dat`) as binary (unformatted) by default (change using the flag `IO.MatrixFile.BinaryFormat F`).

Once the files have been generated by CONQUEST, the output can be processed by setting the output tag:

```
Process.Job pdos
```

This is all that is needed for the simplest output. The number of bins and smearing of the peaks can be set using:

```
Process.sigma_DOS 0.002
Process.n_DOS 10001
```

To resolve the DOS by angular momentum as well as by atom, then the following flags can be set:

```
Process.pDOS_l_resolved T
Process.pDOS_lm_resolved T
```

Note that only one of these is needed, depending on what level of resolution is required. At present, angular momentum resolution is only available for the PAO basis set (not MSSF or blips) though it is under development for the MSSF basis (by projection onto the underlying PAO basis).

The energy range for the projected DOS can also be specified:

```
Process.min_DOS_E -0.35
Process.max_DOS_E  0.35
Process.WFRangeRelative T
```

where the final tag sets the minimum and maximum values relative to the Fermi level.

If you only want to produce pDOS for a few atoms, then you can set the variable `Process.n_atoms_pDOS` and list the atoms you want in the block `pDOS_atoms`:

```
Process.n_atoms_pDOS 2
%block pDOS_atoms
1
12
%endblock
```

Go to *top*.

2.8.8 Band structure

The band structure of a material can be generated by CONQUEST by performing a non-self-consistent calculation, after reading a well-converged charge density: set `minE.SelfConsistent F` and `General.LoadRho T` (remember that to write a converged charge density from CONQUEST you set `IO.DumpChargeDensity T`). The k-points required can be specified as lines of points in k-space; setting `Diag.KspaceLines T` enables this (replacing the usual MP mesh), while the number of lines (e.g. Gamma to L; L to X; would be two lines) is set with `Diag.NumKptLines` and the number of points along a line with `Diag.NumKpts`. The k-point lines themselves are set with a block labelled `Diag.KpointLines` which should have two entries (starting and finishing k-points) for each k-point line. (In constructing the k-point list, CONQUEST will automatically remove any duplicate points, so that the output can be plotted smoothly.) So to create a bandstructure from X-Γ-L-X (3 lines: X-Γ; Γ-L; L-X) with 11 points in each line, you would use the following input:

```
Diag.KspaceLines T
Diag.NumKptLines 3
Diag.NumKpts 11
%block Diag.KpointLines
0.5 0.0 0.0
0.0 0.0 0.0
0.0 0.0 0.0
0.5 0.5 0.5
0.5 0.5 0.5
0.5 0.0 0.0
%endblock
```

After running CONQUEST, setting `Process.Job bst` and running the post-processing will read the resulting `eigenvalues.dat` file, and produce a file `BandStructure.dat`. The x-axis will be the k-point index by default, but specifying `Process.BandStrucAxis` (taking value `n` for index, `x`, `y` or `z` for a single direction in k-space, or `a` to give all k-point coordinates) will allow you to control this. Limits on the energies to select the bands produced can be set with `Process.min_DOS_E` and `Process.max_DOS_E`.

Go to *top*.

2.9 Managing Conquest with ASE

Below we give an introduction how to setup the ASE environment with respect to CONQUEST repository along with a few examples of ASE/Conquest capabilities. We assume that a python script or jupyter-notebook is used.

2.9.1 Setup

Environment variables

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. These variables are:

- `ASE_CONQUEST_COMMAND`: the Conquest executable command including MPI/openMPI prefix.
- `CQ_PP_PATH`: the PAO path directory to where are located the the `.ion` files.
- (optional) `CQ_GEN_BASIS_CMD` : the PAO generation executable `MakeIonFiles`.

Given the Conquest root directory `CQ_ROOT`, initialisation might look to something like

```
import os

CQ_ROOT = 'PATH_TO_CONQUEST_DIRECTORY'
```

(continues on next page)

(continued from previous page)

```
os.environ['ASE_CONQUEST_COMMAND'] = 'mpirun -np 4 '+CQ_ROOT+'/bin/Conquest'
os.environ["CQ_GEN_BASIS_CMD"] = CQ_ROOT+'/bin/MakeIonFiles"
os.environ['CQ_PP_PATH'] = CQ_ROOT+'/pseudo-and-pao/'
```

Go to *top*.

Pseudopotential/PAO files

Conquest atomic pseudotential and basis functions are store in the `.ion` files which will ne referred as to PAO files. Provided the pseudopotential files `.pot` available in `CQ_PP_PATH`, automatic generation of numerical PAOs is possible using the program *MakeIonFiles* available from the Conquest package.

Provided the PAO files, the basis set is specified through a python dictionary, for example:

```
basis = {'O' : {'file': 'O_SZP.ion'},
         'H' : {'file': 'H_SZP.ion'},
         'C' : {'file': 'C_SZP.ion'}}
```

In this case they are all assumed to be obtained from Hamann pseudopotentials, which are the default. Knowing the the exchange and correlation functional <XC> from the Conquest input (*vide infra*) and the chemical symbol <X>, the *Calculator* will search the `.ion` file in different places:

```
CQ_PP_PATH
CQ_PP_PATH/lib/
CQ_PP_PATH/<XC>/<X>/
```

including the current directory and the ASE working directory (*vide infra*). If your PAO file is located in a different place you can include the path in the basis dictionary:

```
basis = {'O' : {'file': 'O_SZP.ion',
                'directory': '<PATH_TO_FILE>'},
         'H' : {'file' : 'H_SZP.ion'},
         'C' : {'file' : 'C_SZP.ion'}}
```

For generating the PAO files, the keyword `gen_basis` should be set to `True` (default is `False`) and the size be provided (default is `medium`). For instance:

```
basis = {'O' : {'gen_basis' : True,
                'basis_size': 'small'},
         'C' : {'gen_basis' : True,
                'basis_size': 'medium'},
         'H' : {'gen_basis' : True,
                'basis_size': 'large'}}
```

will create the `O.ion`, `C.ion` and `H.ion` files where `small`, `medium` and `large` are *default size basis set*. You are allowed to choose the functional and to add options for basis set generation:

```
basis = {'H' : {'gen_basis' : True,
                'basis_size': 'small',
                'xc'        : 'LDA',
                'Atom.Perturbative_Polarised': False}}
```

Note

Only Hamann pseudopotentials for LDA, PBE and PBEsol are available within the CONQUEST distribution. For using other functionals see *Generating new pseudopotentials*.

Warning

Generating polarised PAOs for some atoms can be problematic (mainly group I and II). Please review carefuly the `MakeIonFiles` input files named `Conquest_ion_input` which are collected in `CQ_PP_PATH/<XC>/<X>/` if you are not sure about what you are doing, and check your PAOs.

Go to *top*.

2.9.2 CONQUEST Calculator

The CONQUEST *Calculator* class can be invoked from the ase Calculator set as described in the example below:

```
from ase.calculators.conquest import Conquest
```

A minimal example is given below for setting the CONQUEST *Calculator* (named `calc`) of the ASE Atoms object named *struct*:

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

struct = bulk('NaCl', crystalstructure='rocksalt', a=5.71, cubic=True)
basis  = {'Cl' : {'file' : 'Cl.ion'}, 'Na' : {'file' : 'Na.ion'}}

calc = Conquest(basis=basis,atoms=struct)
```

or, equivalently,

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

struct = bulk('NaCl', crystalstructure='rocksalt', a=5.71, cubic=True)
basis  = {'Cl' : {'file' : 'Cl.ion'}, 'Na' : {'file' : 'Na.ion'}}

struct.calc = Conquest(basis=basis)
```

In basic calculate mode (compute `energy`), the *Calculator* comes with 3 *methods*:

- **`write_input()`:**
  this function will setup the *input files*. For CONQUEST, the PAO basis will be generated/copied with respect to the dictionary key/value pairs, and `Conquest_input` file including the calculation parameters will be written, a long with the `coordinate` file, containing the lattice vectors (in Bohr Unit) and atomic positions (in fractional coordinates).
- **`execute()`:**
  this function execute the calculation. For CONQUEST, it will launch the `ASE_CONQUEST_COMMAND` setup in *the environment varaibles*.
- **`read_results()`:**
  this function post-process the the output file. For CONQUEST, the `energy`, `forces`, `stress` and

`eigenvalues` will be extracted from the `Conquest_out_ase` output file.

> **Note**
>
> The funtion `read_results()` operate on the `Conquest_out_ase` file. This output file is not created by default by CONQUEST. If you want to post-process a calculation with an input generated by hand you must add `IO.WriteOutToASEFile True` in `conquest_input`.

The **indirect** way for managing CONQUEST calculation with ASE is:

```
struct.calc = Conquest(basis=basis)

struct.calc.write_input(struct)
struct.calc.execute()
struct.calc.read_results(struct)
```

where `struct.calc.execute()` can be ignored when, for instance, the calculation is performed on a supercomputer and the output file is then copied back to the current directory for post-processing.

The **direct** way is simply:

```
struct.calc = Conquest(basis=basis)

struct.calc.calculate(struct)
```

or, equivalently,

```
struct.calc = Conquest(basis=basis)

struct.get_potential_energy()
```

Go to *top*.

2.9.3 Keywords for generating the Conquest_input file

In principle all the Conquest input parameters can be added to `Conquest_out_ase` using key/value pairs in a dictionary. There are 3 class of parameters:

- mandatory : they are parsed to the *Calculator* and have no defaults ; there are mandatory.
- important : they are parsed to the *Calculator* they can be freely modified. Some of them are pure ASE keywords.
- defaults : they are set as defaults ; some of them must not be modified. They are read by the *Calculator* through a dictionay `conquest_flags`.

Mandatory keywords

keyword	type	default value	description
`atoms`	`atoms`	None	an atoms object constructed either via ASE or read from an input
`basis`	`dict`	None	a dictionary specifying the pseudopotential/basis files

Important keywords

keyword	CONQUEST equivalence	type	default value	description
`directory`	None	`str`	None	directory used for storing input/output and calculation files
`label`	None	`str`	None	basename for working files (only used by ASE, eg. NEB)
`kpts`	None	`list` or `tuple`	None	k-points grid ; converted to CONQUEST Monkhorst-Pack grid
`grid_cutoff`	`Grid.GridCutoff`	`float`	100	integration grid in Ha
`xc`	`General.FunctionalType`	`str`	'PBE'	exchange and correlation functional
`self_consist`	`minE.SelfConsistent`	`bool`	True	choose either SCF or non-SCF
`scf_toleranc`	`minE.SCTolerance`	`float`	1e-6	Self-consistent-field convergence tolerance in Ha
`nspin`	`Spin.SpinPolarised`	`int`	1	spin polarisation: 1 for unpolarized or 2 for polarised
`conquest_fla`	None	`dict`	None	other CONQUET keyword arguments

Defaults keywords

keyword	type	default value	description
`IO.WriteOutToASEFile`	`bool`	True	write ASE output file ; **must always be True when using ASE for post-processing**
`IO.Iprint`	`int`	1	verbose for the output ; **must always be 1 when using ASE for post-processing**
`DM.SolutionMethod`	`str`	'diagon'	'diagon' stands for diagonalisation other is 'ordern' (base on density matrix)
`General.PseudopotentialType`	`str`	'Hamann'	kind of pseudopotential other type are 'siesta' and 'abinit'
`SC.MaxIters`	`int`	50	maximum number SCF cycles
`AtomMove.TypeOfRun`	`str`	'static'	'static' stands for single (non)SCF other are 'md' or optimisation algorithms.
`Diag.SmearingType`	`int`	1	1 for Methfessel-Paxton ; 0 for Fermi-Dirac
`Diag.kT`	`float`	0.001	smearing temperature in Ha

2.9.4 Some examples

An example of more advanced Calculator setup is given below for a SCF calculation on BCC-Na where for a PBE calculation using a k-point grid of $6 \times 6 \times 6$ using the Fermi-Dirac distribution for the occupation with a smearing of 0.005 Ha:

```
struct = bulk('Na', crystalstructure='bcc', a=4.17, cubic=True)
basis  = {'Na' : {'file' : 'NaCQ.ion'}}

conquest_flags = {'Diag.SmearingType': 0,
                  'Diag.kT'          : 0.005}
```

(continues on next page)

(continued from previous page)

```
struct.calc = Conquest(directory      = 'Na_bcc_example',
                       grid_cutoff    = 90.0,
                       self_consistent= True,
                       xc     = 'PBE',
                       basis = basis,
                       kpts  = [6,6,6],
                       nspin = 1,
                       **conquest_flags)

struct.get_potential_energy()
```

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

```
conquest_flags = {'DM.SolutionMethod' : 'ordern',
                  'DM.L_range'        : 12.0,
                  '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.

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

CQ_ROOT = 'PATH_TO_CONQUEST_DIRECTORY'

os.environ['ASE_CONQUEST_COMMAND'] = 'mpirun -np 4 '+CQ_ROOT+'/bin/Conquest'
os.environ["CQ_GEN_BASIS_CMD"] = CQ_ROOT+'/bin/MakeIonFiles"
os.environ['CQ_PP_PATH'] = CQ_ROOT+'/pseudo-and-pao/'

diamond = bulk('C', 'diamond', a=3.6, cubic=True)  # The atoms object
conquest_flags = {'DM.SolutionMethod' : 'ordern',  # Conquest keywords
                  'DM.L_range'        : 12.0,
                  'minE.LTolerance'   : 1.0e-6}

basis = {'C': {'basis_size' : 'minimal', # Generate a minimal basis
               'gen_basis'  : True}

calc = Conquest(grid_cutoff = 80,    # Set the calculator keywords
                xc = 'PBE',
                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:

```
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:

```
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.

```
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)
```

(continues on next page)

(continued from previous page)

```
basis = {'C': {"basis_size": 'minimal',
               "gen_basis": True}}

calc = Conquest(grid_cutoff = 50,
                xc = "PBE",
                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()

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

Go to *top*.

2.10 External tools

2.10.1 Post-processing for charge density, band density, DOS, STM

The utility `PostProcessCQ` allows users to post-process the output of a CONQUEST calculation, to produce the charge density, band densities, DOS and STM images in useful forms. It is described fully *here*.

2.10.2 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

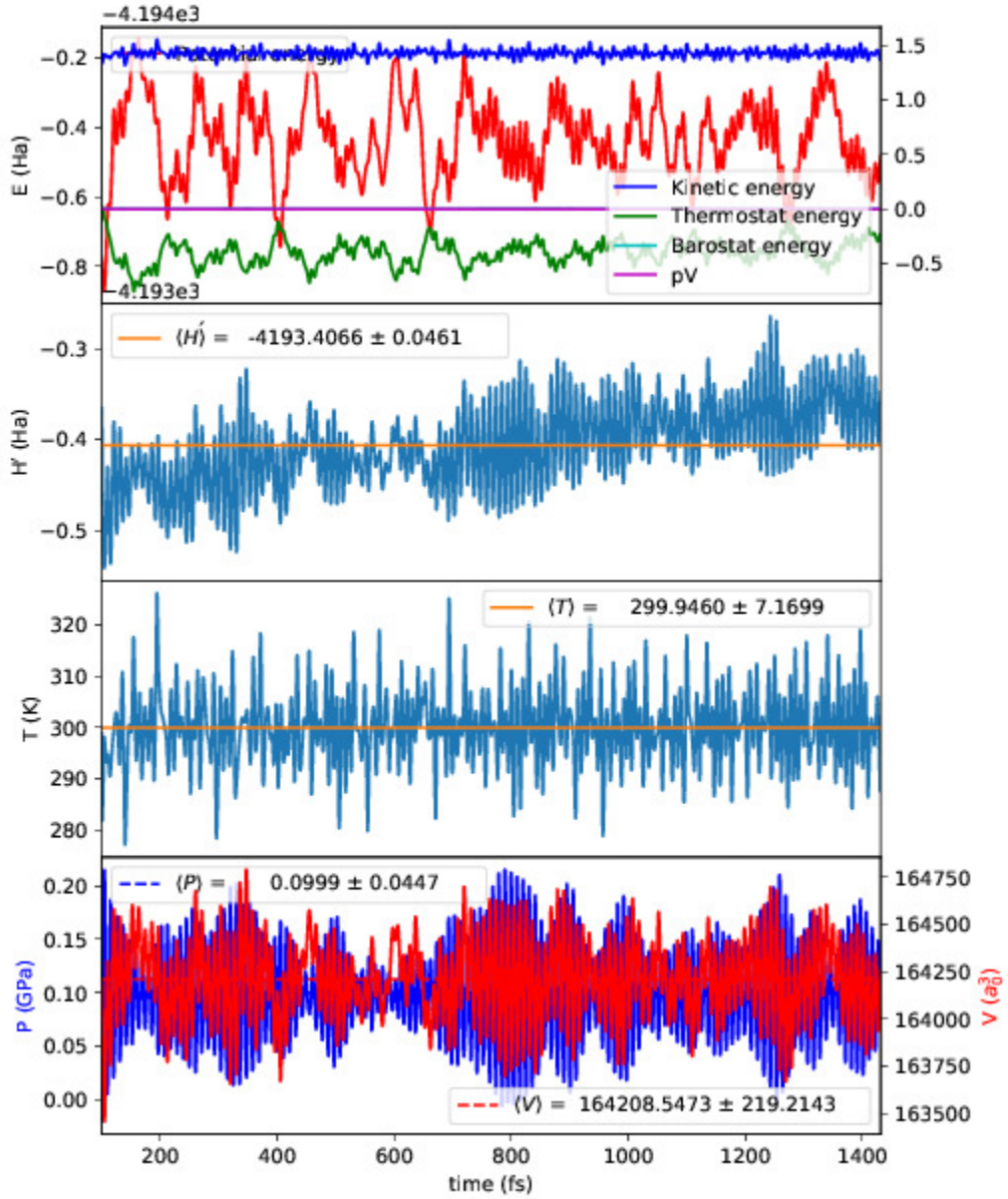
read the CONQUEST input flags to determine, for example, what ensemble is used, and process the results accordingly.

Go to *top*.

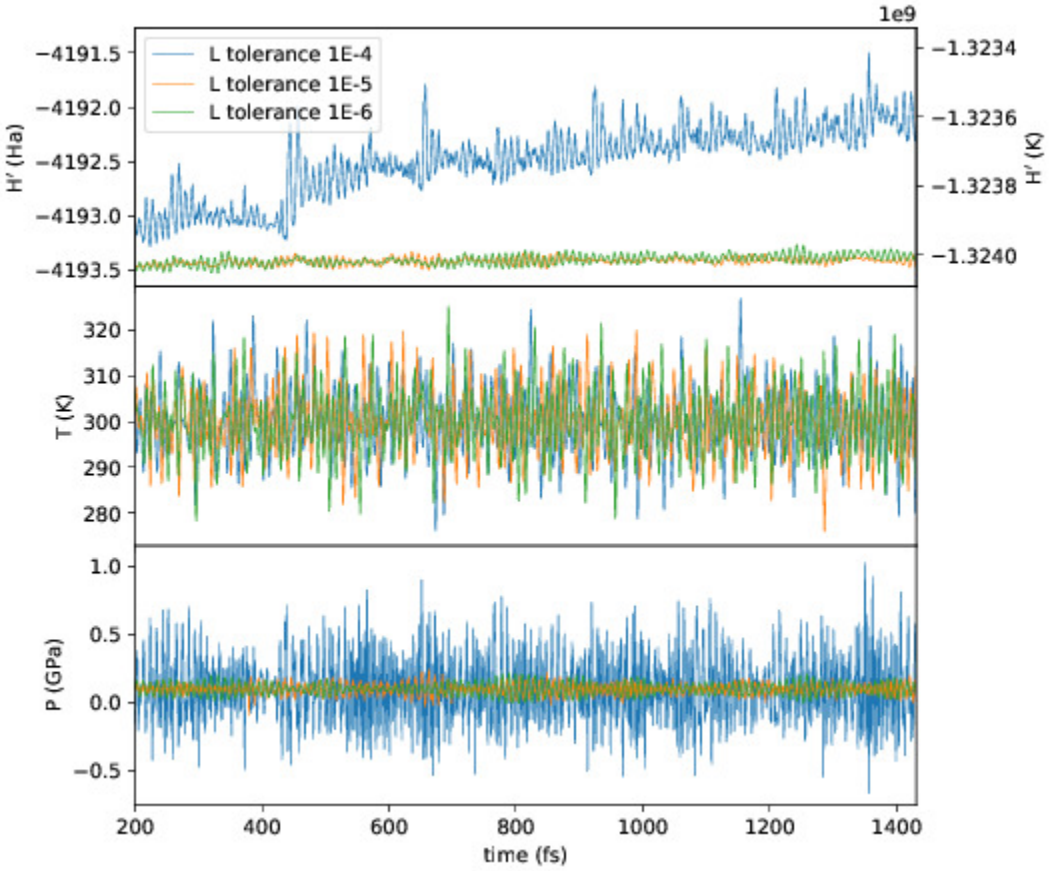
Plotting statistics

```
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 description" "dir2 description"`. The following example compares the effect of changing the L tolerance 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:
                        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)
```

(continues on next page)

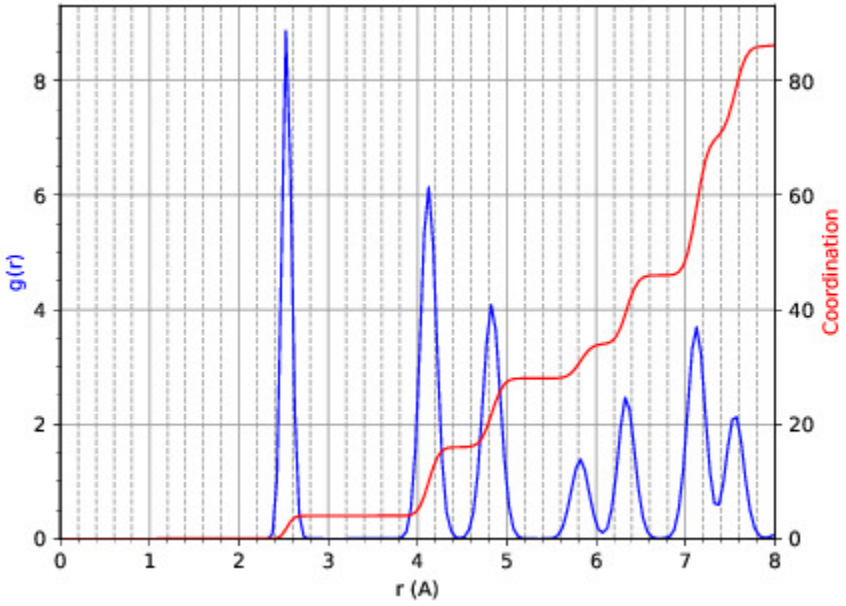
(continued from previous page)

```
  --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*.

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:
```

(continues on next page)

(continued from previous page)

```
                        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*.

2.10.3 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, also called *Calculator* so that it can be used to calculate `energies`, `forces` and `stresses` as inputs to other calculations such as Phonon or NEB that are not implemented in CONQUEST. ASE is a versatil tool to manage CONQUEST calculations without pain either:

- in a **direct** way where pre-processing, calculation and post-processing are managed on-the-fly by ASE,
- or in an **indirect** way where the calculation step is performed outside the workflow, ie. on a supercomputer.

The ASE repository containing the Conquest calculator can be found here. Detailed documentation on how to manage Conquest calculations with ASE is available *here*.

Go to *top*.

2.11 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.

2.12 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

2.12.1 General

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]
hybrid PBE0 (25% of exact exchange)	201	[E8]

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

At the moment, only hybrid functionals with GGA PBE are allowed. Fraction of exact exchange can be tuned. **Warning**: EXX contribution to forces are not implemented yet.

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.LoadDM (*boolean*)
Specifies whether to load a previous density matrix (K or L depending on whether diagonalisation or linear scaling are selected) 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*.

2.12.2 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*.

2.12.3 Input-Output General Tags

IO.Title (*string*)
Title for calculation

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.DumpChargeDensity (*boolean*)
Whether to write out the charge density. If T, then the charge density will be written out at self-consistency; additionally, if `IO.Iprint_SC` is larger than 2, the charge density will be written out at every step of the SCF cycle. The resulting `chden.nnn` files can be converted to cube format files using the *post-processing utility*.

default: F

IO.Dump[Har|XC|PS|ES|Tot]Pot (*boolean*)
Flags to allow dumping of different local potentials (Hartree, XC, pseudopotential, electrostatic, total). Only active when a static self-consistent run is chosen. (NB each flag must be set to true for output, such as `IO.DumpHarPot T` etc.) Files can be converted to cube format as for charge density by setting `Process.ChargeStub` appropriately (e.g. `locpsHar` with other files replacing Har with XC, PS, ES and Tot)

default: F

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*.

2.12.4 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*.

2.12.5 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 **IO.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.

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 Overview of the SCF cycle and atom movement
2 More detail on SCF cycle, atom movement
3 Extensive detail on SCF cycle, atom movement
4 Details of energy breakdown
5 Excessive output, only for developers debugging

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*.

2.12.6 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: 50 Ha.

Grid.GridSpacing (*real*)
As an alternative, the grid spacing in Bohr radii can be set (the code will determine a number of grid points that will be below this value)

Default: zero (value taken from Grid.GridCutoff above)

Go to *top*.

2.12.7 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*.

2.12.8 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

SC.MakeInitialChargeFromK (*boolean*)
Flag determining whether initial charge is made from the density matrix

default: T

Go to *top*.

2.12.9 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. [E9] and Palser and Manolopoulos [E10].)

default: diagon

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*.

2.12.10 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.dk (*real*)
Sets the number of k-points in the Monkhorst-Pack method so that the spacing in reciprocal space is less than the specified value.

default: 0.0

Diag.PaddingHmatrix (*boolean*)
Setting this flag allows the Hamiltonian and overlap matrices to be made larger than their physical size, so that ScaLAPACK block sizes can be set to any value (which can significantly improve efficiency). At present, the automatic setting of block sizes does not use this functionality; if desired, block sizes must be set manually (note that the optimum block size is likely to be different on different machines). (Available from v1.2)

default: T

Diag.BlockSizeR (*integer*)
Block size for rows (See next). From v1.4, the default value is 32 when Diag.PaddingHmatrix is true. It is recommended to check the efficiency (CPU time) on your platform by changing this value. Usually 20-40 is appropriate.

default: 32 or Determined automatically (if Diag.PaddingHmatrix= true)

Diag.BlockSizeC (*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. $\sum_{\mathrm{atoms}} \mathrm{NSF}(\mathrm{atom})$). A value of 64 is considered optimal by the ScaLAPACK user's guide.

If Diag.PaddingHmatrix is set to true then the block sizes can take any value, but BlockSizeR and BlockSizeC must be the same.

default: Determined automatically

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

Diag.ProcRows (*integer*)
Number of rows in the processor grid for SCALAPACK within each k-point processor group

default: Determined automatically

Diag.ProcCols (*integer*)
Number of columns in the processor grid for SCALAPACK within each k-point processor group. 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: Determined automatically

Go to *top*.

2.12.11 Moving Atoms

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

Options:

static — Single point calculation

cg — Structure optimisation by conjugate gradients

sqnm - Stabilised Quasi-Newton Minimisation (recommended approach)

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.MaxSQNMStep (*real*)
The maximum distance any atom can move during SQNM (in Bohr). Applies to the part of the search direction not in the SQNM subspace (scaled directly by a step size, which is limited to ensure this value is not exceeded).

default: 0.2 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.XsfFreq` steps

default: T

AtomMove.XsfFreq (*integer*)
Frequency of output of atomic coordinates to `trajectory.xsf`

default: same as `AtomMove.OutputFreq`

AtomMove.WriteXYZ (*boolean*)
Write atomic coordinates to `trajectory.xyz` for `AtomMove.TypeOfRun = md`, every `AtomMove.XyzFreq` steps

default: T

AtomMove.XyzFreq (*integer*)
Frequency of output of atomic coordinates to `trajectory.xyz`

default: same as `AtomMove.OutputFreq`

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 (except for method 2 below where sqnm will result in SQNM for atomic positions and CG for cell vectors).

default: F

AtomMove.OptCellMethod (*integer*)
Cell optimisation method.

default: 1

Options:

1. Fixed fractional coordinates (only cell vectors)
2. Alternating atomic position and cell vector optimisation (recommended for simultaneous optimisation)
3. Simultaneous cell and atomic conjugate gradients relaxation; caution recommended (can be unstable)

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

default: 1×10^{-5} Ha

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

default: 0.1 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*.

2.12.12 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*)
$< n1 >< n2 >< n3 > \ldots$ Masses of NHC heat baths

default: 1 1 1 1 1

MD.CellNHCMass (*block*)
$< n1 >< n2 >< n3 > \ldots$ 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

MD.VariableTemperature (*boolean*)
Simulation with a variable temperature if .True.

default: F

MD.VariableTemperatureMethod (*string*)
Type of temperature profile. Only `linear` temperature profile is implemented.

default: linear

MD.VariableTemperatureRate (*real*)
Change rate for the temperature. In units of K/fs. If positive, heating. If negative, cooling.

default: 0.0

MD.InitialTemperature(*real*)
Initial temperature.

default: same as AtomMove.IonTemperature

MD.FinalTemperature(*real*)
Final temperature.

default: same as AtomMove.IonTemperature

Go to *top*.

2.12.13 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*.

2.12.14 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*.

2.12.15 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*.

2.12.16 Exact exchange (EXX)

EXX.Alpha (*real*)
Fraction of exact exchange for the density functional XC functional. For example, a value of 1 yields to full EXX with no GGA exchange.

default: 0.25

EXX.Scheme (*integer*)
Select the algorithm to compute EXX matrix elements based on local numerical Poisson solver. Either the contraction reduction integral (CRI) method or full/screened computation of the electron repulsion integrals (ERIs) at each SCF step. For the latter, possibility of storing the integrals computed at the first SCF step is available.

- 1 Direct SCF using the CRI algorithm

- 2 Direct SCF using explicit calculation of ERIs
- 3 Indirect SCF using explicit calculation of ERIs and storage

We recommand either 1 or 3.

default: 1

EXX.Grid (*string*)
Grid accuracy for numerical solution of local the Poisson equation. Choose either `coarse`, `standard` or `fine`.

default: `standard`

Go to *top*.

2.12.17 vdW-DF

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

default:

Go to *top*.

2.12.18 DFT-D2

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

default:

Go to *top*.

2.12.19 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*.

2.12.20 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)

General.MixXCGGAInOut (*real*)
For non-SCF calculations only, chooses how to mix the proportions of GGA XC stress contribution (from the change of the electron density gradient) found using input (0.0 gives pure input) and output (1.0 gives pure output) densities. Note that this is an approximation but varying the value significantly away from 0.5 will give inconsistency between stress and energy.

default: 0.5

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

IO.AtomOutputThreshold (*integer*)
Threshold below which atomic positions are output on initialisation, and atomic forces are output at the end of a static run.

default: 200

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 [E11]
- blips (default) — B-splines [E12]

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*.

2.13 Generating PAOs

2.13.1 Introduction

CONQUEST includes a utility for generating the PAO basis files (`MakeIonFiles` with source code in the directory `tools/BasisGeneration`), and we also provide pseudopotential files (from the PseudoDojo database). The input files will generate a reasonable default basis set (though we can offer no guarantees: users must test the accuracy and convergence of their basis sets). Here we will discuss how to generate both default and custom basis sets. Full details of the basis sets can be found in a recent paper [GP1].

2.13.2 Default basis sets

To generate basis functions, radii for the PAOs must be specified; by default, the utility will set these radii automatically. The radii can be set so that the different shells either share the same radii, or share an energy shift associated with confinement. The default behaviour is to generate basis sets where shells share radii (to change this, add the line `Atom.Cutoffs energy` to the input file). There are four default basis set sizes:

- minimal (single zeta, SZ)
- small (single zeta and polarisation, SZP)
- medium (double zeta, single polarisation, DZP)
- large (triple zeta, triple polarisation, TZTP)

Generally, reasonable results will be obtained with a medium (DZP) basis, though this should always be tested. Minimal and small basis sets are much faster (and are the only basis sets compatible with *linear scaling*), but less reliable. The large basis set will be slower (often it is twice the size of the medium basis set, so diagonalisation will be up to eight times slower) but more reliable and accurate.

We note that Group I and II atoms are a little problematic: the standard approach for most other elements may produce a somewhat limited basis set, so we have created a more accurate, customised input file for these elements (with the exception of Na and Mg, where the pseudopotential does not include l=2 components, so the default approach is all that is possible). These should be tested carefully.

Go to *top*.

2.13.3 Specifying basis sets

The generation utility gives the user complete control over the basis sets that are produced. As an example, we reproduce below the input file for strontium (Sr) and discuss the layout.

```
%block Sr
Atom.PseudopotentialFile Sr.in
Atom.VKBFile Sr.pot
Atom.Perturbative_Polarised F
Atom.PAO_N_Shells 5
Atom.BasisBlock SrBlock
%endblock

%block SrBlock
# n, l, number of zetas
4 0 1
4 1 1
5 0 2
5 1 1
4 2 1
# Radii for PAOs (bohr)
4.0
5.0
10.1 5.7
10.1
10.1
%endblock
```

In this case, we specify five shells (combinations of n and l) via the `Atom.PAO_N_Shells` tag, with polarisation functions found simply by solving the Schrodinger equation in the usual way (the alternative, perturbative polarisation, is discussed below). We must then specify a block that defines these shells (`Atom.BasisBlock SrBlock`). Within that block, we give the number of zeta functions for each (n,l) pair (specified as a line `n l nzeta`) followed by the radii for the zeta functions.

Setting radii for the different shells is a complex process which requires considerable time and care, with an extensive literature; we cannot provide significant help, but only make suggestions. In the first instance, the default radii are a good starting point. Note that the default setting `Atom.Cutoffs radii` averages the radii between shells, while `Atom.Cutoffs energy` finds different radii for each shell (so that the energy change due to confinement is the same for all shells). We recommend starting from one of these sets of radii, and then testing and optimising the radii against some key properties of the system.

A common approach to the generation of polarisation functions (i.e. unoccupied states) is to perturb a valence state (typically the highest energy valence state) to generate a function with angular momentum increased by one; this is the default behaviour. In this case, the radii for the polarisation state should be the same as the shell being polarised (so

for Si, we would perturb the 3p (n=3, l=1) state to get the 3d (n=3, l=2) state), and at present the same number of zeta functions must be specified for the polarisation shell as for the unperturbed shell. For instance, for Si:

```
%block Si
Atom.PseudopotentialFile Si.in
Atom.VKBFile Si.pot
Atom.PAO_N_Shells 3
Atom.BasisBlock SiBlock
%endblock

%block SiBlock
# n, l, number of zetas
3 0 2
3 1 2
3 2 2
# Radii for PAOs (bohr)
8.0 4.0
8.0 4.0
8.0 4.0
%endblock
```

The perturbative option can be turned off by specifying *Atom.Perturbative_Polarised F* in the input file. (Note that in the strontium example above we have specified two polarisation shells, so cannot use the perturbative approach.)

By default, the utility calculates radii which are shared between shells; it is possible to specify instead shared energy shifts using `Atom.Cutoffs energy`, but this can only be done for valence shells, and so *must* use the perturbative polarisation approach for polarisation functions.

Go to *top*.

2.13.4 Compiling

To compile the code, the same `system.make` can be used as is specified for the main code. Once this is done, simply issue the ccommand `make` in the `tools/BasisGeneration` directory. The resulting executable will be placed in the `bin` directory.

Go to *top*.

2.13.5 Generating new pseudopotentials

CONQUEST is supplied with a complete set of pseudopotentials for the elements in the PseudoDojo database (covering LDA, PBE and PBEsol exchange-correlation functionals). In order to generate new pseudopotential files, users will need the Hamann pseudopotential code ONCVPSP v3.3.1 (the current release) and the patch file `Conquest_ONCVPSP_output.patch` which is in the `tools` directory. After patching and compiling the Hamann code (to patch the code, copy the patch to the ONCVPSP `src` directory, and issue the command `patch -p0 < Conquest_ONCVPSP_output.patch`; we cannot provide any support for this) the `oncvpsp.x` utility will generate a file `VPS.dat` which should be renamed (something like `element.pot` as in the CONQUEST pseudopotential files) and specified in the input file using the `Atom.VKBFile` tag.

Go to *top*.

Go to *top*.

CHAPTER

THREE

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 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`; no default)
- Ion files (suffix `.ion`), which provide the pseudopotentials and pseudo-atomic orbitals (PAOs)

The input file requires the user to provide a certain amount of information. The minimal file that is provided for this tutorial gives most of these:

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

# General Parameters
General.NumberOfSpecies 1

%block ChemicalSpeciesLabel
1  28.0850   Si_SZ
%endblock

# Moving Atoms
```

(continues on next page)

(continued from previous page)

```
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 key 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 Brillouin zone sampling must be investigated carefully, as for all periodic electronic structure calculations. The Monkhorst-Pack mesh (`Diag.MPMesh`) offers a convenient way to do this systematically. The job title is purely for reference. Further parameters are discussed in the next tutorial

- The coordinate file `IO.Coordinates`
- The number of species `General.NumberOfSpecies`
- The ion files for the species
- The basic input file
- The output
- Changing the output level and destination
- Controlling the SCF (tolerance and iterations, options)

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

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

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`

3.2.2 Atomic relaxation: Conjugate gradients

- Parameters that are needed
- The output
- Files in `Structural_relaxation_2`

3.2.3 Atomic relaxation: Quenched MD

- Parameters that are needed
- The output
- Files in `Structural_relaxation_3`

3.2.4 Cell relaxation: Conjugate gradients

- Parameters that are needed
- The output
- Files in `Structural_relaxation_4`

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`

3.3.2 Molecular dynamics: NVT with SVR

- Parameters that are needed
- The output
- Files in `MD_2`

3.3.3 Molecular dynamics: NVT with Nose-Hoover chains

- Parameters that are needed
- The output
- Files in `MD_3`

3.3.4 Molecular dynamics: NPT with SVR and Parrinello-Rahman

- Parameters that are needed
- The output
- Files in `MD_4`

3.3.5 Molecular dynamics: NPT with Nose-Hoover chains

- Parameters that are needed
- The output
- Files in `MD_5`

3.3.6 Molecular dynamics: Continuing a run

- Parameters that are needed
- The output
- Files in `MD_6`

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`

3.4.2 MSSF: Optimisation

- Parameters that are needed
- The output
- Files in `Basis_sets_2`

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`

3.5.2 An introduction to linear scaling

- Parameters that are needed
- The output
- Files in `Advanced_2`

CHAPTER

FOUR

THEORY

4.1 Background on energy, forces and stress

A number of different ways of formulating the energy exist in Conquest at the moment, involving both self-consistent and non-self-consistent densities and potentials, both with and without the neutral atom potential. With self-consistency, all formulations should give the same result, though numerical issues may give small differences; without self-consistency the Harris-Foulkes functional is more accurate.

4.1.1 Self-consistent calculations

We define the energy in Conquest in two ways that are equivalent at the self-consistent ground state. The Harris-Foulkes energy is given as:

$$E_{HF} = 2\mathrm{Tr}\left[KH\right] + \Delta E_{Ha} + \Delta E_{XC} + E_{II}$$

where the first term is the band structure energy, equivalent to the sum over the energies of the occupied states, the second two terms compensate for double counting and the final term gives the ion-ion interaction:

$$E_{II} = \frac{1}{2}\left(\sum_{ij}\frac{Z_i Z_j}{\mid \mathbf{R}_i - \mathbf{R}_j \mid}\right)$$

The Hamiltonian is defined as:

$$\hat{H} = \hat{T} + \hat{V}_L + \hat{V}_{NL} + V_{Ha} + V_{XC}$$

where the operators are the kinetic energy, the local and non-local pseudopotentials, the Hartree potential, defined as $V_{Ha} = \int d\mathbf{r}' n(\mathbf{r}')/ \mid \mathbf{r} - \mathbf{r}' \mid$, and the exchange-correlation potential. The alternative form, often known as the DFT energy, is:

$$E_{DFT} = 2\mathrm{Tr}\left[K(T + V_L + V_{NL})\right] + E_{Ha} + E_{XC} + E_{II}$$

with the Hartree energy defined as usual:

$$E_{Ha} = \frac{1}{2}\int\int d\mathbf{r}d\mathbf{r}'\frac{n(\mathbf{r})n(\mathbf{r}')}{\mid \mathbf{r} - \mathbf{r}' \mid}$$

along with the exchange-correlation energy:

$$E_{XC} = \int d\mathbf{r}\epsilon_{XC}\left[n\right]n(\mathbf{r})$$

For the Harris-Foulkes and DFT energies to be equal, it is easy to see that the double counting correction terms in the Harris-Foulkes formalism must be:

$$\Delta E_{Ha} = -E_{Ha} = -\frac{1}{2}\int\int d\mathbf{r}d\mathbf{r}'\frac{n(\mathbf{r})n(\mathbf{r}')}{\mid \mathbf{r}-\mathbf{r}' \mid}$$

and

$$\Delta E_{XC} = \int d\mathbf{r}\left(\epsilon_{XC}[n] - V_{XC}[n]\right)n(\mathbf{r})$$

When calculating forces and stress with self-consistency, we generally use the differentials of the DFT energy rather than the Harris-Foulkes energy; this enables us to separate contributions that are calculated in different ways (in particular on those that are calculated on the integration grid from those that are not).

Go to *top*.

Neutral atom potential

In a DFT code using local orbitals as basis functions, the total energy is most conveniently written in terms of the interaction of neutral atoms: this is simply a reformulation of the total energy which, in particular, reduces the ion-ion interaction to a sum over short-range pair-wise interactions. The charge density of interest is now the *difference* between the total charge density and a superposition of atomic densities, notated as $\delta n(\mathbf{r}) = n(\mathbf{r}) - \sum_i n_i(\mathbf{r})$ for atomic densities $n_i(\mathbf{r})$. We write:

$$E_{DFT,NA} = 2\mathrm{Tr}\left[K(T + V_{NA} + V_{NL})\right] + E_{\delta Ha} + E_{XC} + E_{SII}$$

where the second term is defined as:

$$E_{\delta Ha} = \frac{1}{2}\int\int d\mathbf{r}d\mathbf{r}'\frac{\delta n(\mathbf{r})\delta n(\mathbf{r}')}{\mid \mathbf{r}-\mathbf{r}' \mid}$$

The final term, the screened ion-ion interaction, is short-ranged, and written as:

$$E_{SII} = \frac{1}{2}\left(\sum_{ij}\frac{Z_i Z_j}{\mid \mathbf{R}_i - \mathbf{R}_j \mid} - \int d\mathbf{r} n_i(\mathbf{r})V_{Ha,j}(\mathbf{r})\right)$$

where $V_{Ha,i}(\mathbf{r})$ is the Hartree potential from the atomic density $n_i(\mathbf{r})$. We define the neutral atom potential for an atom as $V_{NA,i}(\mathbf{r}) = V_{L,i}(\mathbf{r}) + V_{Ha,i}(\mathbf{r})$, combining the local potential and the Hartree potential for the atomic density; the overall neutral atom potential is given as the sum over the atomic densities, $V_{NA}(\mathbf{r}) = \sum_i V_{NA,i}(\mathbf{r})$. If we write the pseudo-atomic density as $n_{PAD}(\mathbf{r}) = \sum_i n_i(\mathbf{r})$ then we can also write $V_{NA}(\mathbf{r}) = V_L(\mathbf{r}) + V_{Ha,PAD}(\mathbf{r})$.

In this case, we can write the Harris-Foulkes energy as:

$$E_{HF} = 2\mathrm{Tr}\left[KH\right] + \Delta E_{Ha} + \Delta E_{XC} + E_{SII}$$

with the Hamiltonian defined as:

$$\hat{H} = \hat{T} + \hat{V}_{NA} + \hat{V}_{NL} + V_{\delta Ha} + V_{XC}$$

where $V_{\delta Ha}(\mathbf{r}) = \int d\mathbf{r}'\delta n(\mathbf{r}')/\mid \mathbf{r}-\mathbf{r}' \mid$. Accordingly, the double counting Hartree correction term has to change:

$$\Delta E_{Ha} = -E_{\delta Ha} - \int d\mathbf{r}\delta n(\mathbf{r})\sum_i V_{Ha,i}(\mathbf{r}).$$

Go to *top*.

4.1.2 Non-self-consistent calculations

In non-self-consistent calculations, we use the Harris-Foulkes functional, along with a reasonable guess for the input density, which is normally taken as the superposition of atomic densities, $n_{in}(\mathbf{r})$ and write:

$$E_{NSC} = 2\mathrm{Tr}\left[KH\right] + \Delta E_{Ha}\left[n_{in}\right] + \Delta E_{XC}\left[n_{in}\right] + E_{II}$$

Notice that we effectively have two densities being used here: n_{in} (which is normally the superposition of atomic densities used in the neutral atom case) and and *effective* output density, $n_{out} = \sum_{ij}\phi_i K_{ij}\phi_j$ which comes from the band energy (first term); this complicates the calculation of forces and stress compared to the self-consistent case, as we have to consider contributions from both densities.

For the *neutral atom potential*, $\delta n(\mathbf{r}) = 0$ by definition, which also means that $E_{\delta Ha} = 0$ and $\Delta E_{Ha} = 0$.

Go to *top*.

4.1.3 Partial core corrections

Also known as non-linear core corrections, partial core corrections (PCC) [EFS1] add a model core charge to the pseudopotential to allow for the non-linear exchange-correlation interation between core and valence charge (which is linearised in standard pseudopotentials); this generally improves the accuracy of the pseudopotential. The exchange-correlation potential is evaluated in terms of the combined charge density, $n_v(\mathbf{r}) + n_c(\mathbf{r})$ where the valence charge is input or output charge density defined above: $V_{XC}\left[n_v + n_c\right]$. The exchange-correlation energy becomes:

$$E_{XC} = \int d\mathbf{r}\left(n_v(\mathbf{r}) + n_c(\mathbf{r})\right) V_{XC}\left[n_v + n_c\right].$$

Once this change to the charge density has been made, there is no change to the DFT energy. However, the double counting term for Harris-Foulkes needs redefining, since XC contribution to the band energy is $2Tr[KV_{XC}] = \int d\mathbf{r} n_v(\mathbf{r}) V_{XC}[n_v + n_c]$. We write:

$$\Delta E_{XC} =$$

$$\int d\mathbf{r}\left(n_v(\mathbf{r}) + n_c(\mathbf{r})\right)\epsilon_{XC}[n_v + n_c] - \int d\mathbf{r} n_v(\mathbf{r}) V_{XC}[n_v + n_c]$$

$$=$$

$$\int d\mathbf{r} n_c(\mathbf{r})\epsilon_{XC}[n_v + n_c] + \int d\mathbf{r}\left(\epsilon_{XC}[n_v + n_c] - V_{XC}[n_v + n_c]\right) n_v(\mathbf{r})$$

There is an extra factor of $\int d\mathbf{r} n_c(\mathbf{r})\epsilon_{XC}[n_v + n_c]$ over and above the usual term.

Go to *top*.

4.1.4 Forces and Stresses

It is important that the forces and stresses be the exact derivatives of the energy, for consistency. In particular, this means that as the energy is calculated in different ways for different contributions, the force or stress contribution must be calculated in the same way.

Go to *top*.

Forces

Forces are defined as the change in energy with respect to atomic positions; as the basis functions move with the atoms, these changes will also include Pulay terms. The forces found in Conquest are documented extensively elsewhere [EFS2, EFS3] though the changes needed to account for *PCC*, particularly in the *non-self-consistent* case, have not been published and are given here for completeness. As well as the Hellmann-Feynman forces (which come from the movement of the local and non-local pseudopotentials with the atoms) we define Pulay forces (divided into two parts, known as phi-Pulay which come from changes in the Hamiltonian matrix, and S-Pulay, which come from changes in

the overlap matrix; the phi-Pulay forces are calculated in three contributions, which depend on how the respective parts of the Hamiltonian matrix are calculated: the kinetic energy; the non-local pseudopotential; and the remaining terms which are all found on the integration grid). The ion-ion interactions also contribute forces.

The inclusion of *PCC* adds an extra term to the forces in all calculations, which comes from the change of the core density as the atoms move; the force on atom i is given as:

$$\mathbf{F}_i^{PCC} = -\int d\mathbf{r} \nabla_i n_i^c(\mathbf{r}) V_{XC}[n_v + n_c]$$

If the *non-self-consistent* formalism is used, then a further term is added (the non-self-consistent force changes) to include the gradient of the core charge. The non-self-consistent force is now written as:

$$\mathbf{F}_i^{NSC} = -\int d\mathbf{r} V_{\delta Ha}(\mathbf{r}) \nabla_i n_i^v(\mathbf{r}) - \int d\mathbf{r} \delta n(\mathbf{r}) V'_{XC} \left[n_v^{in} + n_c\right] \left(\nabla_i n_i^v(\mathbf{r}) + \nabla_i n_i^c(\mathbf{r})\right)$$

where V'_{XC} is the derivative of the exchange-correlation potential with respect to charge density.

Go to *top*.

Stress

The stress includes all contributions to the change of energy with the lattice constants; the calculation of stress in Conquest is documented in a paper being prepared for publication, but we give a brief overview here. As Conquest uses orthorhombic cells, only the diagonal stress components ($\sigma_{\alpha\alpha}$) are calculated.

In most cases, forces also contribute to the stress; it is easy to show that the stress contribution is given by:

$$\sigma_{\alpha\alpha} = \sum_i F_{i\alpha} R_{i\alpha}$$

where $R_{i\alpha}$ is the position of the atom. As well as these contributions, there are more subtle terms. Any energies calculated on the grid will contribute to the stress as the integration grid changes with cell size (the stress is simply the energy calculated), and the Hartree potential contributes a term related to the change in the reciprocal lattice vectors (as it is calculated by Fourier transforming the charge density). If the exchange-correlation functional is a GGA functional, then a further term coming from the change of the gradient of the density with the cell size arises. (For non-self-consistent calculations this leads to some complications, as this term technically requires both input and output densities; at present, we approximate this as a mixture of the term calculated with input density and the term calculated with output density; the proportion can be adjusted using the parameter `General.MixXCGGAInOut` documented in the *Advanced and obscure tags* section of the manual, though we do not recommend changing it.)

Go to *top*.

Go to *top*.

4.2 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.

4.2.1 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*.

4.2.2 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*.

4.2.3 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*.

4.3 Molecular Dynamics: Theory

4.3.1 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*.

4.3.2 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],

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

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,

$$\begin{aligned} m_i\ddot{\mathbf{r}}_\mathbf{i}&=-\frac{\partial U[\mathbf{R};\mathbf{LS}]}{\partial\mathbf{r}_i}=\mathbf{F}_\mathbf{i}\\ \ddot{\mathbf{X}}&=\omega^2(\mathbf{LS}-\mathbf{X}), \end{aligned}$$

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]:

$$\begin{aligned} \mathbf{X}(t+\delta t)=2\mathbf{X}(t)-\mathbf{X}(t-\delta t)+\delta t^2\omega^2\left[\mathbf{L}(t)\mathbf{S}(t)-\mathbf{X}(t)\right]\\ +a\sum_{m=0}^{M}c_m\mathbf{X}(t-m\delta t) \end{aligned}$$

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 $\mathbf{XS}^{-1}$ 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*.

4.3.3 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,

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

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}:

$$\begin{aligned}
\dot{\mathbf{r}}_\mathbf{i} &= \frac{\mathbf{p}_i}{m_i} \\
\dot{\mathbf{p}}_\mathbf{i} &= -\frac{\partial U(\mathbf{r})}{\partial \mathbf{r}_i} - \frac{p_{\eta_1}}{Q_1}\mathbf{p}_i \\
\dot{\eta}_k &= \frac{p_{\eta_k}}{Q_k} \\
\dot{p}_{\eta_1} &= \left(\sum_{i=1}^{N}\frac{\mathbf{p}_i}{m_i} - n_f k_B T\right) - \frac{p_{\eta_2}}{Q_{\eta_2}}p_{\eta_1} \\
\dot{p}_{\eta_k} &= \left(\frac{p_{\eta_{k-1}}^2}{Q_{k-1}} - k_B T\right) - \frac{p_{\eta_{k+1}}}{Q_{k+1}}p_{\eta_k} \\
\dot{p}_{\eta_M} &= \left(\frac{p_{\eta_{M-1}}^2}{Q_{M-1}} - k_B T\right)
\end{aligned}$$

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_{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,

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

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,

$$\begin{aligned}
\dot{\mathbf{r}}_i &= \frac{\mathbf{p}_i}{m_i} + \frac{\mathbf{p}_g}{W_g}\mathbf{r}_i \\
\dot{\mathbf{p}}_i &= \mathbf{F}_i - \frac{\mathbf{p}_g}{W_g}\mathbf{p}_i - \left(\frac{1}{N_f}\right)\frac{\mathrm{Tr}[\mathbf{p}_g]}{W_g}\mathbf{p}_i - \frac{p_\xi}{Q}\mathbf{p}_i \\
\dot{\mathbf{h}} &= \frac{\mathbf{p}_g \mathbf{h}}{W_g} \\
\dot{\mathbf{p}}_\mathbf{g} &= V(\mathbf{P}_{\mathrm{int}} - \mathbf{I}P_{\mathrm{ext}}) + \left[\frac{1}{N_f}\sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i}\right]\mathbf{I} - \frac{p_\xi}{Q}\mathbf{p}_g \\
\dot{\xi} &= \frac{p_\xi}{Q} \\
\dot{\mathbf{p}}_g &= \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i} + \frac{1}{W_g}\mathrm{Tr}[\mathbf{p}_g^T\mathbf{p}_g] - (N_f + d^2)kT
\end{aligned}$$

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,

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

Using Liouville's theorem, we have,

$$
\begin{aligned}
iL_r &= \sum_{i=1}^{N} [\mathbf{v}_i + \mathbf{v}_g \mathbf{r}_i] \cdot \nabla_{\mathbf{r}_i} \\
iL_h &= \sum_{\alpha,\beta} \mathbf{v}_{g,\alpha\beta} \mathbf{h}_{\alpha\beta} \frac{\partial}{\partial \mathbf{h}_{\alpha\beta}} \\
iL_v &= \sum_{i=1}^{N} \left(\frac{\mathbf{F}_i}{m_i} \right) \cdot \nabla_{\mathbf{v}_i} \\
iL_{\mathrm{bath}} &= iL_{\mathrm{vpart}} + iL_{\mathrm{vbox}} + iL_{\xi} + iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}} \\
&= \sum_{i=1}^{N} \left[- \left\{ \mathbf{v}_g + \frac{1}{N_f} \mathrm{Tr}(\mathbf{v}_g) + v_{\xi_1} \right\} \mathbf{v}_i \right] \nabla_{\mathbf{v}_i} \\
&+ \sum_{\alpha,\beta} \left[\frac{F_{\mathrm{box}}}{W} - v_{\xi_1} \mathbf{v}_{g,\alpha\beta} \right] \frac{\partial}{\partial \mathbf{v}_{g,\alpha\beta}} \\
&+ \sum_{k=1}^{M} v_{\xi_k} \frac{\partial}{\partial \xi_k} \\
&+ \left[\frac{F_{\mathrm{NHC}_1}}{Q_1} - v_{\xi_1} v_{\xi_2} \right] \frac{\partial}{\partial v_{\xi_1}} \\
&+ \sum_{k=2}^{M} \left[\frac{1}{Q_k} (Q_{k-1} v_{\xi_{k-1}}^2 - kT_{\mathrm{ext}}) - v_{\xi_k} v_{\xi_{k+1}} \right] \frac{\partial}{\partial v_{\xi_k}} \\
&+ \left[\frac{1}{Q_M} (Q_{M-1} v_{\xi_{M-1}}^2 - kT_{\mathrm{ext}}) \right] \frac{\partial}{\partial v_{\xi_M}}
\end{aligned}
$$

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

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

The Liouvillian for the heat baths can be further expanded:

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

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

$$
\begin{aligned}
e^{\left(iL_{v_{\xi_1}} + iL_{v_{\xi_k}} + iL_{v_{\xi_M}} \right) \frac{\Delta t}{4}} &= e^{-i \left(-v_{\xi_1} v_{\xi_2} \frac{\partial}{\partial \xi_1} - \sum_{k=2}^{M} v_{\xi_k} v_{\xi_{k+1}} \frac{\partial}{\partial \xi_k} - v_{\xi_{M-1}} v_{\xi_M} \frac{\partial}{\partial \xi_M} \right) \frac{\Delta t}{8}} \\
&\times e^{i \left(F_{\mathrm{NHC}_1} \frac{\partial}{\partial v_{\xi_1}} + F_{\mathrm{NHC}_k} \frac{\partial}{\partial v_{\xi_k}} + F_{\mathrm{NHC}_M} \frac{\partial}{\partial v_{\xi_M}} \right) \frac{\Delta t}{4}} \\
&\times e^{-i \left(-v_{\xi_1} v_{\xi_2} \frac{\partial}{\partial \xi_1} - \sum_{k=2}^{M} v_{\xi_k} v_{\xi_{k+1}} \frac{\partial}{\partial \xi_k} - v_{\xi_{M-1}} v_{\xi_M} \frac{\partial}{\partial \xi_M} \right) \frac{\Delta t}{8}}
\end{aligned}
$$

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 $\mathbf{v}_i \rightarrow \lambda \mathbf{v}_i$, 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*.

CHAPTER

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).

CHAPTER

SIX

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:

- A. Nakata, J. S. Baker, S. Y. Mujahed, J. T. L. Poulton, S. Arapan, J. Lin, Z. Raza, S. Yadav, L. Truflandier, T. Miyazaki, and D. R. Bowler, J. Chem. Phys. **152**, 164112 (2020) DOI:10.1063/5.0005074
- T. Miyazaki, D. R. Bowler, R. Choudhury and M. J. Gillan, J. Chem. Phys. **121**, 6186–6194 (2004) DOI:10.1063/1.1787832
- D. R. Bowler, T. Miyazaki and M. J. Gillan, J. Phys. Condens. Matter **14**, 2781–2798 (2002) DOI:10.1088/0953-8984/14/11/303

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.

[ES1] R. Resta. Macroscopic polarization from electronic wave functions. *Int. J. Quantum Chem.*, 75:599–606, 1999. doi:10.1002/(SICI)1097-461X(1999)75:4/5%3C599::AID-QUA25%3E3.0.CO;2-8.

[ES2] R. D. King-Smith and D. Vanderbilt. Theory of polarization of crystalline solids. *Phys. Rev. B*, 47:1651–1654, 1993. doi:10.1103/PhysRevB.47.1651.

[SR1] Bastian Schaefer, S. Alireza Ghasemi, Shantanu Roy, and Stefan Goedecker. Stabilized quasi-newton optimization of noisy potential energy surfaces. *J. Chem. Phys.*, 142(3):034112, 2015. doi:10.1063/1.4905665.

[SR2] 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] John P. Perdew, Matthias Ernzerhof, and Kieron Burke. Rationale for mixing exact exchange with density functional approximations. *J. Chem. Phys.*, 105(22):9982, 1996. doi:10.1063/1.472933.

[E9] 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.

[E10] 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.

[E11] 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.

[E12] 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.

[GP1] 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.

[EFS1] Steven G. Louie, Sverre Froyen, and Marvin L. Cohen. Nonlinear ionic pseudopotentials in spin-density-functional calculations. *Phys. Rev. B*, 26:1738–1742, Aug 1982. doi:10.1103/PhysRevB.26.1738.

[EFS2] T. Miyazaki, D. R. Bowler, R. Choudhury, and M. J. Gillan. Atomic force algorithms in density functional theory electronic-structure techniques based on local orbitals. *J. Chem. Phys.*, 121:6186, 2004. doi:10.1063/1.1787832.

[EFS3] Antonio S Torralba, David R Bowler, Tsuyoshi Miyazaki, and Michael J Gillan. Non-self-consistent Density-Functional Theory Exchange-Correlation Forces for GGA Functionals. *J. Chem. Theory Comput.*, 5:1499, 2009. doi:10.1021/ct8005425.

[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.