ONETEP Documentation

Release 7.1.8

Joseph Prentice

Jan 26, 2024

CONTENTS:

CHAPTER

ONE

STARTING WITH ONETEP

Author
Arihant Bhandari, University of Southampton

Author
Rebecca J. Clements, University of Southampton

Author
Jacek Dziedzic, University of Southampton

Date
April 2022 (revised July 2023)

1.1 Obtaining a copy of ONETEP

If you are an academic license user, you should have received a tarball of a personalised ONETEP copy once you have signed an academic licence agreement. Create a new directory, unpack the tarball there (`tar -xfz your_tarball_name.tar.gz`), and you're done. You can now go straight to *Compiling, testing and running ONETEP*.

If you plan to have continuous access to the current ONETEP source code, likely because you are a ONETEP developer or contributor, or collaborate with one, you will want to have access to the official ONETEP GitHub repository.

1. Take a look at *Repository location and structure* and *ONETEP GitHub set-up* to get acquainted with the overall set-up.
2. If you don't have a GitHub account yet, follow the steps at *Creating a GitHub user account* to create one.
3. Then, create a GitHub *personal access token* by following the steps at *Creating a GitHub personal access token (PAT)*. Without it you will not be able to access the repository due to GitHub policies (unless you already have a *PAT*).
4. Follow *Getting access to the ONETEP repository* to get access. Be sure to follow further instuctions there, depending on whether you plan to be a *User* or a *Contributor*. Congratulations, you have your own copy of ONETEP!

Continue along to *Compiling, testing and running ONETEP*.

1.2 Compiling, testing and running ONETEP

These are the instructions for setting the environment prior to compiling ONETEP, for how to compile ONETEP, and how to run quality-check ("QC") tests that will give you confidence in the robustness of your installation. There are also instructions on how to actually submit (start) your jobs.

These instructions differ depending on your environment (the machine on which you will run ONETEP and the installed software, such as compilers or external libraries). **They are provided separately**.

1. Go to the directory with your ONETEP installation.
2. Change to the `hpc_resources` directory. There you will see a number of well-supported systems, such as `Archer2` or `Young` (and more).
3. If your machine is well-supported by ONETEP, just follow the compilation, testing and running instructions there. For instance, instructions for Iridis5 would be found at hpc_resources/Iridis5/instructions_iridis5.pdf.

   If your machine is not in the list of well-supported systems, you will need to adapt one of the existing configs. Look into the subdirectories under `hpc_resources` and also in the `config` directory in your ONETEP installation.

Important point about compiling ONETEP following a git merge:

> **Warning:** If you just updated your ONETEP source via `git merge` or `git pull` (rather than cloning via `git clone` or unpacking ONETEP from a tarball), issue a `make cleanall` before compiling. The script for cascade avoidance used when making ONETEP can get confused as to what needs to be rebuilt after a `git merge` or `git pull`. If you forget about `make cleanall`, you may get an executable that builds correctly but malfunctions in strange ways (usually producing very unexpected error messages).

Important point about running QC tests:

> **Warning:** Never run the QC tests in the actual repository. Create a copy of your ONETEP installation directory (e.g. `cp -a onetep_jd onetep_jd_qc`) and run the tests in the directory of the copy. Much of the stuff in the install is not needed to run the tests (e.g. all of `src/`), but it's just less hassle to copy the entire thing than pick out the necessary subdirectories.
>
> Why not run the QC tests in the actual repository? Because they produce outputs, which can become messy to clean up back to pristine state, polluting your repository with spurious changes. This becomes more problematic because symbolic links are used in the QC test suite. Trying to update your repository via `git pull` or `git fetch` may be tricky when you have unfinished or improperly cleaned up QC tests. Just don't run them straight in the repository.

1.3 Creating input files

Go to the ONETEP website, onetep.org. There, under *Resources* you will find the *Tutorials* section, which introduces running various kinds of ONETEP calculations. Take a look at some of the input files at the bottom of the page. Input files in ONETEP have the `.dat` file extension. Should any files get downloaded having a `.txt` extension, you will need to rename them to end with `.dat`.

Input files contain keywords, instructing ONETEP on what calculations to run, and to set the parameters needed to run them. Check out the keywords on the webpage onetep.org/Main/Keywords to see what they mean. If not specified, most of them have default settings, as listed on the webpage.

The keywords come in different types: `logical`, `integer`, `real`, `text`, `physical` and `block`. Keywords of the type `logical` can have a value of `T` (true) or `F` (false). Keywords that are `integer` and `real` are numbers. Keywords of type `text` are a string of characters (for example a filename). Keywords of the type `physical` refer to physical variables, which come with units such as angstroem, bohr, joule, hartree, etc. A `block` indicates more than one line of input, these are often used for specifying coordinates.

Some of the important keywords to get started are:

- `task` – to choose what main calculation you would like ONETEP to perform, e.g. a single point energy calculation or geometry optimisation. You can run a properties calculation this way, using output files generated from a single point energy calculation or using `task singlepoint` and a separate keyword `do_properties` set to `T`.
- `xc_functional` – to choose how to approximate the exchange-correlation term in the Kohn Sham DFT energy expression.
- `%block lattice_cart` – to define the dimensions of the simulation cell.
- `%block positions_abs` – to define the atomic positions in Cartesian coordinates.

As can be seen from the example input files, all `block` keywords must end with a corresponding `endblock`. Be default all coordinates are in atomic units (bohr). To switch to angstroems, add `ang` in the first line of the block:

```
%block positions_abs
ang
C 16.521413 15.320039 23.535776
O 16.498729 15.308934 24.717249
...
%endblock positions_abs
```

The `species` and `species_pot` blocks detail the parameters of the atoms. Non-orthogonal Generalised Wannier Functions (NGWFs) are used to model the atomic orbitals. In the `species` block, the name we give to each type atom in the system is given first, followed by the element of the atom, its atomic number, the number of NGWFs to use (use -1 for an educated guess) and the radius of each NGWF typically around 8.0-10.0 (in bohr) for an accurate calculation. For instance for carbon you might use:

```
C C 6 4 8.0
```

The `species_pot` block specifies the location of the pseudopotential used for each element of the system. The standard ONETEP norm-conserving pseudopotentials (`.recpot` files) exclude core electrons. Core electrons are included in `.paw` files. Some of these can be found in your repository's `pseudo` directory. A complete database of all pseudopotentials for all elements in the `.paw` format can be downloaded from https://www.physics.rutgers.edu/gbrv/all_pbe_paw_v1.5.tar.gz. There are also PAWs and pseudopotential (`.recpot`) files in the `utils-devel` repository at https://github.com/onetep-devel/utils-devel.

To continue a calculation if it has run out of computation time, use the keywords below. The original input must have the `write` keywords, but no `read` keywords because the files aren't available to read at this stage. Any continuing input files must include the `read` keywords. If the input file name isn't changed upon continuation, the output file will be overwrite with the results of the continuation, so make sure to back up files before continuing.

```
write_denskern T
write_tightbox_ngwfs T
read_denskern T
read_tightbox_ngwfs T
```

If you are running an ensemble DFT (EDFT) calculation you will also need to add

`write_hamiltonian T`
`read_hamiltonian T`
to the above list.

1.4 Running ONETEP in parallel environments

ONETEP is typically run on more than one CPU core – whether on a desktop computer, or at a high-performance computing (HPC) facility. This is termed *parallel operation*. There are two main modes of parallel operation – *distributed-memory* computing (sometimes termed simply *parallelism*), and *shared-memory* computing (sometimes termed *concurrency*). ONETEP combines both of them, so it will be crucial to understand how they work.

1.4.1 Distributed-memory parallelism (MPI)

In this scenario a collection of individual *processes* (that is, running instances of a program) work together on the same calculation. The processes can all reside on the same physical machine (often termed *node*) – e.g. when you run them on your many-core desktop machine – or on separate machines (nodes) – e.g. when you run them at an HPC facility.

In both cases processes reside in separate memory spaces, which is a fancy way of saying they *do not share* memory – each of them gets a chunk of memory and they don't know what the other processes have in their chunks. Yes, even when they are on the same machine.

The problem they work on has to be somehow subdivided between them – this is known as *parallel decomposition*. One common way of doing that – and one that ONETEP employs – is *data decomposition*, where it's the data in the problem that is subdivided across processes. In ONETEP the grids on which quantities like electronic density or external potential are calculated are divided across processes, with each process "owning" a slab of the grid. Similarly, the atoms in the system are divided across processes, with each process "owning" a subset of atoms. Both of these concepts are illustrated in Fig. 1.1.

From the point of view of the operating system, the processes running on a machine are separate entities (see Fig. 1.2), and collaboration between them almost always necessitates some form of communication (because, remember, they do not share memory) – e.g. process #1 may need to ask process #2 "what are the positions of *your* atoms?" This is accomplished by a dedicated software library known as Message Passing Interface (MPI). This is why we often call the processes *MPI processes*, or, more technically, *MPI ranks*.

MPI facilitates starting multiple processes as part of a single calculation, which can become slightly tricky when there are multiple machines (nodes) involved. Your MPI installation will provide a dedicated command for running multiple processes. The command is often called `mpirun`, `aprun`, `gerun`, `srun` or something similar (it will certainly be stated in the documentation for your system). On a desktop machine its invocation typically looks like this:

```
mpirun -np 4 ./onetep_launcher input.dat >input.out 2>input.err
```

Here, `mpirun` is the name of the command for launching multiple processes, `-np 4` asks for four processes, `onetep_launcher` is the name of the script for launching ONETEP – it's the script that will actually be run on four CPU cores, and each instance will start one ONETEP process for you – here we assume it's in the current directory (`./`), `input.dat` is your ONETEP input file. Output will be sent ("redirected") to `onetep.out`, and error messages (if any), will be redirected to `input.err`. All four processes will be started on the same machine.

In HPC environments the syntax will be slightly different, because the number of processes will be automatically inferred by the batch (queueing) system, the batch system will also take care of instructing `mpirun` (or equivalent) what machines to put the processes on.

MPI lets you run your calculation on as many processes as you like – even tens of thousands. However, there are practical limitations to how far you can go with ONETEP. Looking at Fig. 1.1 it becomes clear that you cannot have more MPI processes than atoms – or some processes would be left without work to do. In fact this limitation is even slightly stricter – to divide work more evenly ONETEP tries to give each processes a similar number of NGWFs, not

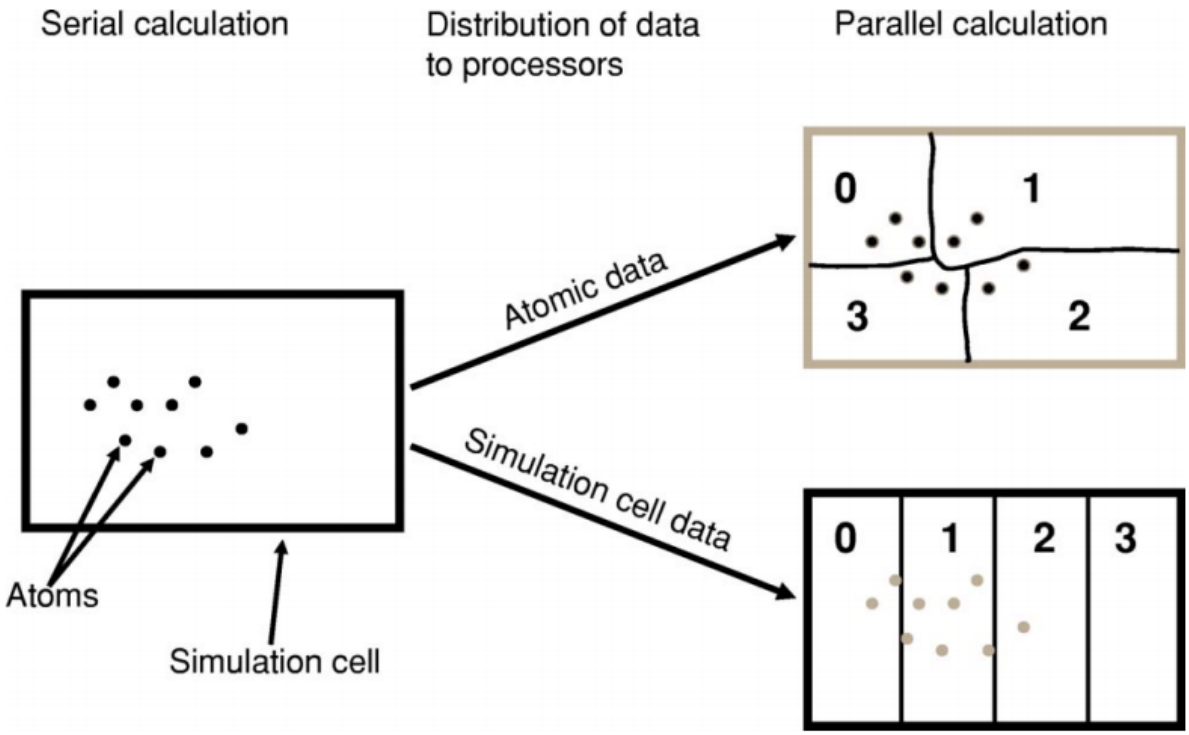

Fig. 1.1: Illustration of parallel data decomposition in ONETEP. Figure borrowed from J. Chem. Phys. **122**, 084119 (2005), https://doi.org/10.1063/1.1839852, which you are well-advised to read.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15742 jd12g09   20   0  616772 292396  24152 R 100.0  0.4   0:04.78 onetep.RH79
15802 jd12g09   20   0  620652 285552  20508 R 100.0  0.4   0:04.77 onetep.RH79
15803 jd12g09   20   0  627420 291792  20452 R 100.0  0.4   0:04.76 onetep.RH79
15804 jd12g09   20   0  620728 286260  20324 R 100.0  0.4   0:04.77 onetep.RH79
    1 root      20   0  191980   4896   2616 S   0.0  0.0   0:10.28 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.20 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
```

Fig. 1.2: Four ONETEP processes running on one machine, each utilising 100% of a CPU core and 0.4% of available memory.

atoms. For instance, for a water molecule run on two processes, it makes sense to assign the O atom and its 4 NGWFs to one process, and both H atoms (1 NGWF each) to the second process. If you try to run a calculation on H:sub:`2`O on *three* processes, it's very likely that ONETEP will do the same thing – assign O to one processes, both H's to another process and the third process will wind up with no atoms. This will cause the calculation to abort. So, one limitation is **you will not be able to use more MPI processes that you have atoms in your system, and even slightly smaller numbers of MPI processes might not work**. Even if they do, you don't really want that, because load balancing will be rather poor – the processor that gets the O atom has roughly twice as much work to do as the one that gets the two H atoms. The bottom line is – *you should have at least several atoms per MPI rank* – in the interest of efficiency.

1.4.2 Shared-memory parallelism (OMP)

This approach, sometimes known as *concurrency*, *concurrent processing* or colloquially as *threads*, uses *shared memory*. The way it works is a process *spawns* (starts) a number of *threads of execution*, with each thread delegated to a separate CPU core. Typically each thread works with a subset of data, and, in contrast to processes, threads within the same process can access each other's memory. For example, if a process was given 50 atoms to work with, it can spawn 4 threads and tell each thread to work on 12-13 atoms. Because threads share memory, they do not need special mechanisms to communicate – they can just use memory for this. What they need instead are special mechanisms for synchronisation – e.g. so that thread 1 knows thread 2 finished writing something to memory and it's safe to try to read it. These mechanisms are described by a standard known as `OpenMP`, or `OMP` for short.

In ONETEP threads are most conveniently handled using the launcher's `-t` option, which instructs it how many threads each process should spawn. For instance the command

```
./onetep_launcher -t 8 input.dat >input.out 2>input.err
```

runs one process (note the absence of `mpirun`), which spawns eight threads. This is what it looks like to the operating system:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1553 jd12g09   20   0 2101636 829840  24080 R 799.3  1.3   4:00.93 onetep.RH79
    9 root      20   0       0      0      0 S   0.3  0.0   0:15.44 rcu_sched
```

Fig. 1.3: One ONETEP process that spawned eight threads, running on one machine, utilising almost 800% of a CPU core and 1.3% of available memory – this is for the entire process encompassing eight threads.

Thread-based processing has a number of limitations. As threads reside within a process, you cannot feasibly run more threads than you have CPU cores on a node – in other words, threading is limited to a single node. Moreover, large numbers of threads quickly become inefficient. If a processes owns 10 atoms, using more than 10 threads will not give you any advantage, because the additional threads will not have anything to work with (fortunately, this does not lead to the calculation aborting, only to some threads idling). Even with four threads you will lose some efficiency, because some threads will get 3 atoms and some only 2. ONETEP works best with about 4-6 threads, unless you are using Hartree-Fock exchange (HFx), which is the most efficient on large thread counts.

Threads are easiest to control via `onetep_launcher`, which you are advised to use, but ONETEP also provides keywords for controlling them manually – these are `threads_max`, `threads_per_fftbox`, `threads_num_fftboxes`, `threads_per_cellfft` and `threads_num_mkl`. Each of these sets the number of threads spawned from a single process for some part of ONETEP's functionality. This is advanced stuff and will not be covered in this beginners' document.

Another point to note is that each thread requires its own *stack* (a region of memory for intermediate data) in addition to the global (per-process) stack. This per-thread stack needs to be large enough – almost always 64 MB suffices. So, if you spawn 16 threads from a process, that's an extra 1024 MB of memory that you need, per process. If you use `onetep_launcher`, it takes care of setting this stack for you. If you don't – you'll need to take care of this on your own (by exporting a suitable `OMP_STACKSIZE`) or you risk ugly crashes when the stack runs out. Not recommended.

1.4.3 Hybrid (combined MPI+OMP) parallelism

For anything but the smallest of systems, combining MPI processes with OMP threads is the most efficient approach. This is known as *hybrid parallelism*. In ONETEP this is realised simply by combining `mpirun` (or equivalent) with `onetep_launcher`'s `-t` option, like this:

```
mpirun -np 4 ./onetep_launcher -t 8 input.dat >input.out 2>input.err
```

Here we are starting 4 processes, each of which spawns 8 threads. This set-up would fully saturate a large, 32-core desktop machine.

Setting up processes and threads looks slightly different in HPC systems, where you need to start them on separate nodes. Your submission script (you will find ones for common architectures in the `hpc_resources` directory of your ONETEP installation) defines all the parameters at the top of the script and then accordingly invokes `mpirun` (or equivalent) and `onetep_launcher`. Look at the beginning of the script to see what I mean.

1.4.4 How many nodes, processes and threads should I use?

There are a few points worth considering here. First of all, efficiency almost universally decreases with the number of CPU cores assigned to a problem. That is to say, throwing 100 cores at a problem is likely to reduce the time to solution by less than 100-fold. This is because of communication overheads, load imbalance and small sections of the algorithm that remain sequential, and is formally known as Amdahl's law. It's worth keeping this in mind if you have a large number of calculations to perform (known as *task farming*) – if you have 1000 calculations to perform, and have 500 CPU cores at your disposal, time to solution will be the shortest if you run 500 1-core jobs first, followed by 500 1-core jobs next. If you opt for running a job on 500 CPU cores simultaneously, and do this for 1000 jobs in sequence, your time to solution will be much, much worse, because of efficiency diminishing with the number of cores.

Having said that, task farming is not the only scenario in the world. Sometimes you have few jobs, or only one, that you want to run quickly. Here, you're not overly worried about efficiency – if running the job on 1 CPU core takes a month, and using 100 CPU cores reduces it to a day, you'd still take 100 CPU cores, or even more. You just have to remember that the returns will be diminishing with each CPU core you add[1].

The remaining points can be summarised as follows:

1. Avoid using 1-2 atoms per MPI process, unless there's no other way. Try to have at least several atoms per MPI process – for good load-balancing.
2. For OMP threads the sweet spot is typically 4-5 threads. If you have a giant system, so that you have a hundred atoms or more per MPI process, you might be better off using 2 threads or even 1 (using purely distributed-memory parallelism). This is because load balancing will be very good with high numbers of atoms per MPI process. If you have a small system, or if already using large numbers of MPI processes, so that you wind up with very few atoms per MPI process (say, below 5), you might find that using higher numbers of threads (say, 8) to reduce the number of MPI processes is beneficial.
3. Know how many CPU cores you have on a node. Make sure the number of MPI processes *per node* and OMP threads per process saturate all the node's cores. For instance, if you have 40 CPU cores on a node, you should aim for 10 processes per node, 4 threads each; or 20 processes per node, 2 threads each; or 40 processes per node, 1 thread each; or 4 processes per node, 10 threads each. 2 processes per node with 20 threads each would also work, but would likely be suboptimal. Just don't do things like 3 processes with 10 threads, because then you leave 10 CPU cores idle, and don't do things like 6 processes with 10 threads, because you then oversubscribe the node (meaning you have more threads than CPU cores) – this degrades performance.
4. Nodes are often divided internally into *NUMA regions* – most often there are two NUMA regions per node. The only thing you need to know about NUMA regions is that you don't want a process to span across them. This is why in the example above you did not see 5 processes with 8 OMP threads each or 8 processes with 5 OMP

[1] Hartree-Fock exchange calculations being an important exception

threads each – I assumed there are two NUMA regions with 20 CPU cores each. In both examples here you would have a process spanning across two NUMA regions. It works, but is much slower.

5. Points 1-3 above do not apply to Hartree-Fock exchange calculations. Point 4 applies. When doing HFx calculations (this includes calculations with hybrid exchange-correlation functionals, like B3LYP) follow the more detailed instructions in the HFx manual.

6. If you find that your calculation is running out of memory, your first step should be to increase the number of nodes (because it splits the problem across a set-up with more total RAM). Another idea is to shift the MPI/OMP balance towards more threads and fewer MPI processes (because this reduces the number of buffers for communication between MPI processes). So if your job runs out of memory on 2 nodes with 10 processes on each and with 4 threads per process, give it 4 or more nodes with 10 processes on each with 4 threads per each, or switch to 4 processes with 10 OMP threads.

CHAPTER

TWO

WORKING WITH GITHUB

Author
Jacek Dziedzic, University of Southampton

Author
Nicholas Hine, University of Warwick

Author
James C. Womack, University of Southampton

Date
July 2023

This chapter describes a number of procedures for working with GitHub. These are separate procedures, each performing a simple GitHub-related operation. They are not meant to be invoked in numerical sequence.

2.1 Repository location and structure

The ONETEP GitHub repository is located at https://github.com/onetep-devel/.

`onetep-devel` is the *organisation*. It contains four repositories (which you can see by clicking `Repositories` at the top):

1. `documentation` (public) – containing the source of the documentation, for instance this document.
2. `tutorials` (public) – containing the source of the ONETEP tutorials.
3. `utils-devel` (public) – containing the utilities and resources that are not directly distributed with ONETEP, such as scripts, GFN parameters, vdW parameters, PAW datasets, and developer documentation.
4. `onetep` (private) – containing the ONETEP distribution itself, which is mostly Fortran source code (`src/`), quality-check tests (`tests/`) and utilities (`utils/`).

The `onetep` repository is private, so if you haven't been granted access to it, you will only see *three* repositories at https://github.com/onetep-devel, and trying to go directly to https://github.com/onetep-devel/onetep will give you a *404 Page not found*.

2.2 ONETEP GitHub set-up

Fig. 2.1 shows how the ONETEP workflow is set up on GitHub and locally, on your drive.

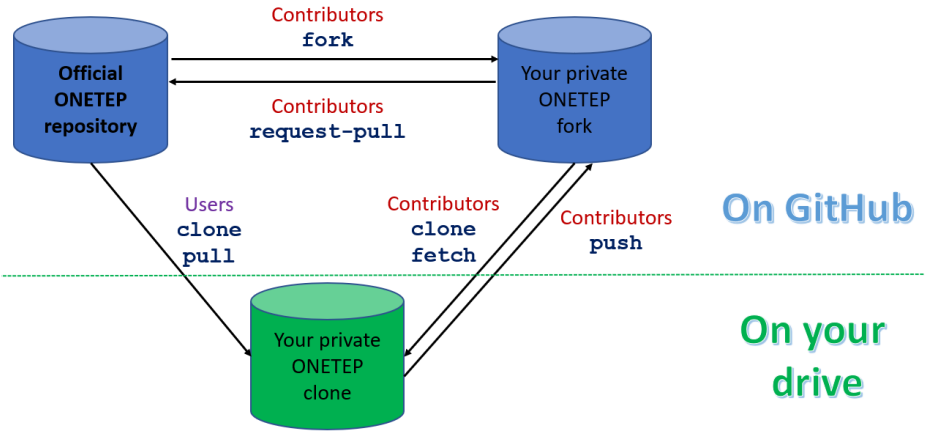

Fig. 2.1: Set-up of ONETEP on GitHub. There is one official repository. *Users* `clone` the repository (once), and later `pull` if they want to update it with the latest changes. They cannot contribute. *Contributors* `fork` the repository first (once), then `clone` the fork (once), not the official repository. They `fetch` (and merge) from the fork when they want to update. When they want to contribute changes, *Contributors* `push` to their fork first, then submit a `pull request` from the fork to the official repository. No pushing to the official repository happens at any stage. All contributions happen via `pull requests`.

2.3 Creating a GitHub user account

Go to https://github.com/signup and follow the instructions there. Use your university or organisation email.

2.4 Creating a GitHub *personal access token* (PAT)

To be able to access the ONETEP repository via `git`, you will need to create a *personal access token* or PAT on GitHub. This is a distinct password from the one you use to access the GitHub web interface, it will only be used for accessing the repository through `git` – the two passwords are not interchangeable. You must have a PAT set up, you will not be able to access the ONETEP repository with your GitHub password.

For technical details, read https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls

To create a PAT, on your GitHub page click your profile picture (top right) and choose `Settings` from the drop-down list. In the panel on the left, scroll to the very bottom and click <> `Developer settings`. On the page that opens, in the panel on the left choose `Personal access tokens` and then `Tokens (classic)`. Click on `Generate new token`, then `Generate new token (classic)`. On the page that opens up, put something descriptive in the `Note` field, like *My personal access token for accessing ONETEP*. Choose an expiration date for your token if you want it to expire after a certain time, or `No expiration` if you don't want it to expire. Under `Select scopes` check `repo`, and nothing else. Click on the green `Generate token` button at the bottom. Your PAT has been created. **Make sure to copy it to a safe place at this point** – once you leave this page, you will not be able to retrieve it (just create a new PAT if this happens).

2.5 Getting access to the ONETEP repository

The `documentation`, `tutorials` and `utils-devel` repositories are public – you don't need any special access, just go to https://github.com/orgs/onetep-devel/repositories.

To get access to the `onetep` repository (with the ONETEP source code), you will need to be granted access. Contact Chris-Kriton Skylaris (`C.Skylaris[-at-]soton.ac.uk`) and ask to be added as a member to the `onetep` repository. If you do not have a GitHub account yet, make sure to create one *first* – follow the instructions at *Creating a GitHub user account*.

To be able to access the repository via `git`, you must create a *personal access token* on GitHub. Follow the instructions at *Creating a GitHub personal access token (PAT)* if you haven't done that already.

Once your request to join the `onetep` repository is approved, you will get a GitHub invitation in your email. In the email message, accept the invitation by clicking the green button. On the webpage that opens click `Join onetep-devel`. You are now a member of the `onetep` repository.

If you plan to be a ONETEP *User* (you'd like to use ONETEP and maybe view the source code), follow the steps described in *Cloning the official ONETEP repository*, then go to *Compiling, testing and running ONETEP*.

If you plan to be a ONETEP *Contributor* (you'd like not only to use ONETEP, but also contribute to it), follow the steps described in *Forking the official ONETEP repository*, then in *Cloning your private ONETEP fork*, then go to *Compiling, testing and running ONETEP*.

2.6 Cloning the official ONETEP repository

This procedure is only meant for *Users*. *Contributors* should go to *Forking the official ONETEP repository* instead, then to *Cloning your private ONETEP fork*.

Cloning the ONETEP repository lets you get a copy of the ONETEP distribution, including the source code, the quality-check (QC) tests, and essential scripts. Think of this clone as of your personal copy of ONETEP. This copy will reside locally, on your disk. See also Fig. 2.1, the left-hand side.

To clone the ONETEP repository, go to your terminal and issue:

```
git clone https://github.com/onetep-devel/onetep.git
```

When prompted for username, enter your GitHub username. When prompted for password, type in (or preferably paste) your *GitHub personal access token*, **not** your GitHub password.

See *Storing GitHub personal access token (PAT) in git* for instructions on how to have `git` store your credentials so that you don't have to type or paste them each time you want to perform an action on your repository.

As a *User* you don't have permissions to write to the repository. Attempts to do so will finish with:

```
remote: Write access to repository not granted.
```

2.7 Cloning the `documentation`, `tutorials` or `utils-devel` repositories

This procedure is only meant for *Users*. *Contributors* should go to *Forking the documentation, tutorials or utils-devel repositories* instead, then to *Cloning your private documentation, tutorials or utils-devel fork*.

These repositories are public.

Cloning the `documentation` repository lets you get a copy of the *source* of the ONETEP documentation (so, `.rst` files). If you are just interested in the compiled documentation (`.pdf` or `.html`), you might be better off just visiting https://onetep.org/resources/documentation.

Cloning the `tutorials` repository lets you get a copy of the *source* of the ONETEP tutorials (so, `.rst` files). If you are just interested in the compiled tutorials (`.pdf` or `.html`), you might be better off just visiting https://onetep.org/resources/tutorials.

Cloning the `utils-devel` repository lets you get a copy of the additional utilities useful mostly for developers, but ONETEP users may benefit from having a copy too. ONETEP will offer to clone this repository for you after you compile it, so perhaps it's not worth it to clone it manually.

Think of the above clones as of your personal copies of, respectively, the documentation source, the source for the tutorials and the utilities. This copy/copies will reside locally, on your disk. See also Fig. 2.1, the left-hand side.

To clone the documentation repository, go to your terminal and issue:

```
git clone https://github.com/onetep-devel/documentation.git
```

To clone the tutorials repository, go to your terminal and issue:

```
git clone https://github.com/onetep-devel/tutorials.git
```

To clone the utils-devel repository, go to your terminal and issue:

```
git clone https://github.com/onetep-devel/utils-devel.git
```

When prompted for username, enter your GitHub username. When prompted for password, type in (or preferably paste) your *GitHub personal access token*, **not** your GitHub password.

See *Storing GitHub personal access token (PAT) in git* for instructions on how to have `git` store your credentials so that you don't have to type or paste them each time you want to perform an action on your repository.

As a *User* you don't have permissions to write to the repository. Attempts to do so will finish with:

```
remote: Write access to repository not granted.
```

2.8 Forking the official ONETEP repository

This procedure is only meant for *Contributors*. *Users* should go to *Cloning the official ONETEP repository* instead.

Forking the ONETEP repository lets you get a copy of the ONETEP distribution, including the source code, the quality-check (QC) tests, and essential scripts. Think of this fork as of your personal copy of ONETEP. This copy will reside on GitHub. You will be able to *clone* this copy, to get a personal copy of ONETEP locally on your disk. See Fig. 2.1, the parts marked with *Contributors*.

To fork the official repository, assuming you have been given access (see *Getting access to the ONETEP repository*), follow these steps:

1. Go to `https://github.com/onetep-devel/onetep`.

2. Click on `Fork`, top right.
3. Choose a repository name for your fork. A good name is `onetep_` followed by your initials, e.g. for Jane Doe, you could choose `onetep_jd`. The fork will be created in your private workspace, so there's no need to worry about any conflicts with someone else who has the same initials.
4. Add a description, e.g. *Jane's fork of ONETEP on GitHub*.
5. **Uncheck** the check-box `copy the master branch only`.
6. Click `Create fork`.

You now have your private fork, accessible via something like https://github.com/JaneDoe/onetep_jd.

2.9 Forking the `documentation`, `tutorials` or `utils-devel` repositories

The procedure is the same as *Forking the official ONETEP repository*, except `onetep` should be replaced with `documentation`, `tutorials` or `utils-devel`.

2.10 Cloning your private ONETEP fork

This procedure is only meant for *Contributors*. *Users* should go to *Cloning the official ONETEP repository* instead.

Cloning your ONETEP fork lets you get a local copy of the ONETEP distribution, including the source code, the quality-check (QC) tests, and essential scripts. See also Fig. 2.1, the left-hand side.

To clone your ONETEP fork, follow these steps:

1. Go to your GitHub profile – something like https://github.com/JaneDoe, where `JaneDoe` would be replaced by your GitHub user name.
2. Click your profile picture (top right) and choose `Your repositories` from the drop-down list.
3. Click the name of your fork that you created earlier, something like `onetep_jd`. You are now on the GitHub page of your private fork.
4. Click the green `<> Code` button.
5. Click the `Copy` icon (looks like two overlapping squares) to the right of the address.
6. Go to your terminal, change to a directory where you want your clone to reside, and type `git clone`, followed by a space. Paste the address copied earlier. You should get something like:

   ```
   git clone https://github.com/JaneDoe/onetep_jd.git
   ```

   for our Jane Doe example.
7. When prompted for username, enter your GitHub username. When prompted for password, type in (or preferably paste) your *GitHub personal access token*, **not** your GitHub password.

You should now have your personal clone.

See *Storing GitHub personal access token (PAT) in git* for instructions on how to have `git` store your credentials so that you don't have to type or paste them each time you want to perform an action on your repository.

You are now ready to develop the code inside your private ONETEP fork – see *Development within a fork* for more details.

As a *Contributor* you have permissions to write (push) to the repository from your clone. The commits will, of course, wind up in your personal fork. To get them to the official ONETEP repository, follow the steps in *Creating a pull request*.

2.11 Cloning your private `documentation`, `tutorials` or `utils-devel` fork

The procedure is the same as *Cloning your private ONETEP fork*, except `onetep` should be replaced with `documentation`, `tutorials` or `utils-devel`.

2.12 Development within a fork

This procedure is only meant for *Contributors*.

Develop within your local clone on your disk! Any changes to the code should be made in your local clone. Once you are satisfied with them, you can commit them, and push them to your private fork. If you want them to become a part of official ONETEP, you should then create a pull request from your private fork to the official repository. Here's how to do that: *Creating a pull request*.

Whether you are a *Contributor* or a *User*, you might want to update your repository with any latest changes that might have occurred in the official repository. *Users* might be interested in recent bug fixes or new functionality, *Contributors* will want to update their copy before committing any changes of their own. Keep your fork up-to-date with changes in the official repository by regularly merging with the official repository, i.e.

- Add the official repository as a remote to your local repository (only once):

```
git remote add github_official https://github.com/onetep-devel/onetep.git
```

  If you are developing documentation, tutorials or the utilities, replace `onetep` with `documentation`, `tutorials` or `utils-devel` in the above.

- Fetch changes from the official repository (each time):

```
git fetch github_official
```

- Merge changes from the official repository into your currently checked-out branch (each time):

```
git merge github_official/master
```

It is often the case that `make cleanall` must be issued after merging, the script for cascade avoidance used when making ONETEP can get confused as to what needs to be rebuilt after a `git merge`.

Remember to commit your changes regularly and push these to your fork on GitHub so that they are backed up, e.g. `git push origin <branch_name>` where `<branch_name>` is the name of the git branch you are working on.

You can choose how you develop in your private fork as the branches, tags etc. that you create in the fork do not affect the official repository.

You can apply bugfixes to release branches in your private fork, too, i.e.

- Fetch changes from the official repository:

```
git fetch github_official
```

- Check out a release branch:

```
git checkout academic_release_v5.0
```

- Commit your bugfix to the branch and push to your changes back to GitHub.

The above text describes a basic `git` workflow within a private fork. If you are not familiar with `git`, or source code version control in general, it may be worthwhile to spend some time working through a tutorial or guide before proceeding with any serious ONETEP development. There are numerous resources available online. Here is a small selection:

- [Official Git (git-scm.com) documentation](https://git-scm.com/doc)
- [Software carpentry: "Version control with Git"](https://swcarpentry.github.io/git-novice/)

More experienced `git` users may prefer to use a different process. You may use whatever workflow you like in your own private fork, since your changes will be selectively applied to the official repository via the pull request process.

2.13 Creating a pull request

This procedure is only meant for *Contributors*.

When your changes are ready for contributing to the official ONETEP repository, you need to create a pull request via the GitHub web interface.

Before you make your pull request:

- Make sure that your changes satisfy the code quality requirements (see *Advice for Contributors*), or your pull request will be rejected.
- Make sure that your code is up-to-date with official ONETEP repository – use the instructions in *Development within a fork* to merge changes into the branch you are working on in your fork.

Once you are sure your code satisfies the requirements and is synchronized with the official repository, follow these steps.

1. Open the GitHub project page for your private fork in a web browser. This will be something like https://github.com/JaneDoe/onetep_jd.
2. Click `Contribute` below and to the left of the green <> `Code` button.
3. Click `Open pull request`.
4. Edit the title and description of what you want to commit.
5. Choose Reviewer(s) on the right. Read the two **notes** below.
6. Click `Create pull request`.

Your pull request has now been created. You should wait for the Reviewer(s).

- If they have issues with your proposed changes, they will let you know and maybe suggest how to fix them. There is no need to create a new pull request in that case. You can apply the requested changes directly to the branch in your private fork for which the pull request was issued and they will be automatically added your pull request.
- If they have no objections to your pull request, you should get a notification that it has been merged. The merging is done by the Reviewer.

Note: It looks like at our settings it is not possible to have more than one reviewer, unless we upgrade to *Pro*, *Team* or *Enterprise* plan, see: https://github.com/orgs/community/discussions/23978

Note: Also, unless you are one of the repository Owners, you will simply not see the option to select a Reviewer. This is a GitHub feature meant to prevent any spamming from people who forked public repositories. Intead of adding a Reviewer, *tag* the person or people you'd like to review your change in the text that you enter in the *Leave a comment* box. For instance, type `@JacekDziedzic` to add Jacek as a reviewer – he will then get a notification once you submit the pull request.

If the pull request is for the documentation or tutorials, once it is merged, the changes will be deployed automatically to the ONETEP website (a `documentation-deploy` or `tutorials-deploy` **GitHub Action**). When the pull request is submitted, this step will be skipped (because you have not been authorised to make the changes at this point yet). It will only be run following the merge.

2.14 Reviewing and merging pull requests

This procedure is only meant for *Reviewers* on pull requests.

To review and merge a pull request on which you are a *Reviewer*, follow these steps.

1. Go to the ONETEP official repository, https://github.com/onetep-devel/onetep.
2. Click `Pull requests` at the top.
3. Locate the pull request in question and click on it.
4. If you have comments, you can type them in the `Leave a comment` text field and click `Comment`.
5. To finish the review, click the green button `Add your review` at the top right.
6. If you would like to reject this pull request, click `Close pull request` at the bottom. This will close the pull request. You can later reopen it.
7. If you would like to merge this commit, click *the arrow* near the green button (the button caption may vary, e.g. `Merge pull request` or `Squash and merge`), and choose `Squash and merge`.
8. Click `Squash and merge`.
9. Click `Confirm squash and merge`.

2.15 Simple changes to `documentation` or `tutorials`

If you have a small change to the documentation or tutorials, for instance you want to add or change a single file or several files, *and* you're an Owner, you can just upload the new/updated files straight from the GUI. Make sure you are logged in, then choose `Add File` (to the left of the green button). Follow the instructions on screen. Once you're done, this will create a commit.

2.16 Storing GitHub personal access token (PAT) in git

It's likely that you will soon find typing or pasting the GitHub *personal access token* any time you want to perform an operation on your repository cumbersome. To avoid having to do this, go to your local repository clone and, from the command line, issue the following command:

```
git config credential.helper store
```

The next time you are prompted for a password will be the last time – `git` will store it for you.

2.17 Adding a new user to the repository

This section is meant for *Owners*. It explains how to add users to the ONETEP GitHub repository.

Note that only the `onetep` repository is private. Anyone can access the public repositories of the organisation `onetep-devel` (which are `documentation`, `tutorials` and `utils-defvel`). To add a user to the `onetep` repository, you yourself must be a user in the `Owner` role.

To add a new user, follow these steps:

1. Go to the organisation level: https://github.com/onetep-devel.
2. Choose `People` at the top, then `Invite member`.
3. Type in the invitee's GitHub username or email address, click `Invite`.
4. Choose a suitable `Role`. To add a typical user, just leave `Member` selected. To add a privileged user, switch to `Owner`. Click `Send Invitation`.
5. The invitee should now appear in the `Invitations` pane on the left.
6. That's it. Once the invitee accepts the invitation, they will be visible in the `People` menu.

CHAPTER

THREE

GROUND STATE CALCULATION SETUP

3.1 Using the Pseudoatomic Solver to Generate NGWFs

Author
Nicholas D.M. Hine, University of Warwick

Date
September 2011

Date
Updated February 2016

3.1.1 What is being calculated?

When the atomic solver is used [Ruiz-Serrano2012], a Kohn-Sham DFT calculation is performed for a "pseudoatom". This means that the pseudopotential of a single isolated ion is used as the external potential, and the single-electron Kohn-Sham states are solved self-consistently, for a given XC functional. The resulting states can be expected to form an 'ideal' atomic orbital basis for a given calculation [Soler2002], [Artacho1999], [Blum2009], [Tarralba2008], [Chen2010]. In practice they are a good starting point for initial NGWFs, or a passable fixed basis for calculations without NGWF optimisation, in which context they should be at least comparable to the basis sets generated in SIESTA, for example (though note that a high cutoff energy is required for accurate representation on the grid in such cases).

The pseudoatomic orbitals we are looking for solve the Kohn-Sham equation:

$$\hat{H}_{\mathrm{KS}}\psi_n(\mathbf{r}) = \epsilon_n\psi_n(\mathbf{r})$$

where

$$\hat{H}_{\mathrm{KS}} = -\frac{1}{2}\nabla^2 + V_{\mathrm{loc}}(r) + \sum_i |p_i\rangle D_{ij}\langle p_j|$$

is the Hamiltonian in terms of kinetic, local potential and nonlocal potential contributions. For a norm conserving pseudopotential the matrix D_{ij} is diagonal and there is one term per angular momentum channel, so there is generally only one contributing nonlocal projector for each wavefunction. In PAW and USPs, there may be more, and nonzero cross-terms D_{ij} for $i \neq j$. Because it is a spherical potential, solutions of this form of the Kohn-Sham equation are easily separable into an angular part, solved by the spherical harmonics ($Y_{lm}(\hat{\mathbf{r}})$, or equivalently the real spherical harmonics $S_{lm}(\hat{\mathbf{r}})$) multiplied by radial part, which we will be solving explicitly in terms of a basis.

The atomic orbitals are solved in a sphere of radius R_c. The most appropriate basis to use therefore consists of normalised spherical Bessel functions of given angular momentum l. The radial parts of the basis functions, $B_{l,\nu}(r)$ are

$$B_{l,\nu}(r) = j_l(q_{l,\nu}r) \,/\, \left[\int_0^{R_c} |j_l(q_{l,\nu}r)|^2 r^2 dr\right]^{\frac{1}{2}},$$

with $q_{l,\nu}$ such that $q_{l,\nu}R_c$ are the zeros of the spherical Bessel functions. Thence $B_{l,\nu}(R_c) = 0$ and $\int_0^{R_c} |B_{l,\nu}(r)|^2 r^2 dr = 1$ for all ν. The basis would be complete if ν were infinite: in practice it must be truncated, and the number of functions included is determined by a kinetic energy cutoff $E_{\rm cut}$. The criterion $\frac{1}{2}q_{l,\nu}^2 < E_{\rm cut}$ determines the largest ν for each l.

In the pseudoatom calculation we are therefore calculating Kohn-Sham states of the form

$$\psi_n(\mathbf{r}) = \sum_\nu c_{n,\nu}\, B_{l_n,\nu}(r)\, S_{l_n m_n}(\hat{\mathbf{r}}) \ ,$$

which have eigenvalues ϵ_n and occupancies f_n which include spin-degeneracy. The occupancies are fixed, and determined before the main calculation, such that they sum to the number of valence electrons. Spherical symmetry of the density is assumed, so the occupancies of all members of a given set of m-degenerate orbitals are always equal — and in fact in practice the m states for a given l, n pair are amalgamated into one state with f_n summed over all the degenerate m's. For example, for a nitrogen ion with valence configuration $2s^2\,2p^3$, we would have $f_{2s} = 2$, $f_{2p} = 3$. For this, we therefore need to find the lowest-energy self-consistent eigenstate of each of $l = 0$ and $l = 1$. Henceforth we will only consider the radial dependence $\psi_n(r)$. All radial quantities will be considered to have been integrated over solid angle already, so factors of 4π are omitted and $\int |\psi_n(r)|^2 r^2 dr = 1$ for a normalised orbital.

We define the local potential through

$$V_{\rm loc}(r) = V_{\rm psloc}(r) + V_H[n](r) + V_{XC}[n](r) + V_{\rm conf}(r)$$

where for a spherical charge distribution $n(r) = \sum_n f_n |\psi_n(r)|^2$, the Hartree potential is given by

$$V_H(r) = \frac{1}{r}\int_0^r n(r')r'^2 dr' + \int_r^\infty n(r')r'\, dr' \ .$$

and the XC potential is $V_{XC}[n](r) = \frac{\partial E_{XC}[n]}{\partial n(r)}$. $V_{\rm conf}(r)$ is an optional confining potential whose specific form will be discussed later [Blum2009].

For each l we can define the Hamiltonian matrix

$$H^l_{\nu,\nu'} = \int_0^{R_c} B_{l,\nu}(r)\left[\hat{H}B_{l,\nu'}(r)\right] r^2 dr$$

and the overlap matrix

$$S^l_{\nu,\nu'} = \int_0^{R_c} B_{l,\nu}(r) B_{l,\nu'}(r) r^2 dr$$

We then solve the secular equation

$$\mathbf{H}^l.\mathbf{c}_n = \epsilon_n \mathbf{S}^l.\mathbf{c}_n$$

to give the coefficients $c_{n,\nu}$ which describe the orbitals. The orbitals are generated on the real-space grid and density mixing with a variable mixing parameter α is then used until self-consistency is obtained. The result is deemed to be converged once a) the Harris-Foulkes estimate of the total energy (the bandstructure energy) matches the total energy as determined from the density to within a given tolerance (10^{-5} Ha) and the energy has stopped changing each iteration, to within a given tolerance (10^{-7} Ha).

3.1.2 Performing a Calculation with the Pseudoatomic Solver

The atomic solver is the default approach to NGWF initialisation, so if you do not need to change any settings for any species, simply omit the `%block species_atomic_set` block.

If there are any tweaks to be made to the default, this block is required, and for each element symbol the string "SOLVE" should appear in the entry. If you want to use automatic initialisation of the number of NGWFs, then specify that to

be -1 in the species block. The code will attempt to determine how many orbitals to use, which orbitals constitute the valence, and what their default occupancies should be. To illustrate what will happen, we present some simple examples.

Let us imagine setting up a calculation with only nitrogen, for which $Z_{\mathrm{atom}} = 7$ and $Z_{\mathrm{ion}} = 5$. The valence manifold consists of 4 NGWFs of radius $R_c = 8.0$ per atom, so we would have the following blocks in our input file:

```
%block species
N N 7 4 8.0
%endblock species
%block species_atomic_set
N "SOLVE"
%endblock species_atomic_set
```

Note that because we may well want to add extra options to this string later, it's best to always use the "" quotes around SOLVE. These settings will activate the pseudoatomic solver and it will attempt to guess a default configuration for the atom. Since $Z_{\mathrm{ion}} = 5$, the code will count back five electrons from the end of the default neutral atom occupancy, which is $1s^2\,2s^2\,2p^3$, and will discover that the valence states are $2s^2\,2p^3$. Since we have asked for $N = 4$ NGWFs, the solver will then count forward from the start of the valence states and determine that by including the whole first set of s and p states it has enough to span the valence space and create four orbitals (and thus four NGWFs). The solver will therefore solve for one state with $l = 0$, $f = 2$ and one state with $l = 1$, $f = 3$, all with radius $R_c = 8.0$, and from these states will produce one s-like NGWF and the three degenerate p_x, p_y and p_z NGWFs.

A slightly more complex example would be if we were generating orbitals for iron ($Z_{\mathrm{atom}} = 26$, $Z_{\mathrm{ion}} = 8$):

```
%block species
Fe Fe 26 9 10.0
%endblock species
%block species_atomic_set
Fe "SOLVE"
%endblock species_atomic_set
```

This time, to find the default configuration, the solver initialisation routines will count back 8 electrons from the neutral atom configuration of $1s^2\,2s^2\,2p^6\,3s^2\,3p^6\,3d^6\,4s^2$ and thus will determine that the valence states are $3d^6\,4s^2$. However, this time we have asked for 9 NGWFs, so it will then count forward from $3d$, include the fivefold-degenerate lowermost d-like state and the lowest s-like state. This only makes six, so it then will also have to include the threefold-degenerate $4p$-like state. The solver will have to solve for one unoccupied p-like orbital, which will have $f = 0$ throughout the calculation.

Controlling the configuration

The default neutral-atom configurations for all the elements up to $Z = 92$ are included in the code, and will be used by default to generate the configuration. However, it is also possible to override these default configurations. For example, to generate NGWFs for iron in the 3+ state, we might want to set the occupancies to $3d^5\,4s^0$. To do this we use the "conf=" directive after the SOLVE string:

```
%block species_atomic_set
```

```
Fe "SOLVE conf=3d5 4s0"
%endblock species_atomic_set
```

Any terms in the configuration which are not overridden are left at their default values. Another example might be if we wanted to force the partial occupation of more higher-lying states than would otherwise be occupied for the neutral atom:

```
%block species_atomic_set
C "SOLVE conf=2s1.5 2p2.5"
%endblock species_atomic_set
```

Note that the solver counts through the configuration terms strictly in the order n, l, i.e. n is looped over outermost, then $l = 0$ to $l = n-1$ for each n innermost. This means that sometimes a little thought is required to get the terms one actually wants, and not spurious extra ones. For example, if we wanted to run a calculation of oxygen with 9 NGWFs per atom, what we probably wanted would be to run with 1 s-like NGWF, 3 p-like NGWFs and 5 d-like NGWFs. However, this is not by default what one will get if one asks for

---

```
%block species
O O 8 9 9.0
%endblock species
%block species_atomic_set
O "SOLVE"
%endblock species_atomic_set
```

This will identify $2s^2\,2p^4$ as the valence orbitals, and counting forward will identify $2s$, $2p$, $3s$, $3p$ and just 1 of the 5 degenerate $3d$ states as the NGWFs required. Therefore, we must instruct the atomsolver to ignore the unwanted excited $3s$ and $3p$ terms. We do this with an "X", which instructs the solver to knock out this term:

```
%block species_atomic_set
O "SOLVE conf=2s2 2p4 3sX 3pX 3d0"
%endblock species_atomic_set
```

Strictly speaking, the $2s$, $2p$ and $3d$ strings are not needed, as they are the default values anyway, but they are left in for clarity. I find it advisable, so that I can keep track of the terms which will generate the NGWFs, to add explicitly the terms with zero occupancy to the conf string.

Generating larger, non-minimal bases

ONETEP is generally used to create an *in-situ-optimised*, minimal basis (eg 4 NGWFs/atom for C, N, O etc). However, it is also possible to fix the NGWFs and run with a much larger, unoptimised basis, in a manner akin to other DFT codes designed for large-scale simulations (eg SIESTA). One would then normally want to use multiple NGWFs for each angular momentum channel corresponding to the valence orbitals. This is known as using a "multiple-zeta" basis set, where zeta refers to the radial part of the valence atomic orbitals. For example, a "triple-zeta" basis for carbon would have 3 s-like functions and 3 of each of p_x, p_y, and p_z-like functions. There are two approaches to generating these extra radial functions. This simplest is just generate the higher-lying orbitals of a given angular momentum. For carbon, for example, a double-zeta basis in this scheme would include $3s$ and $3p$-like states. This approach, however, is not very quick to converge with basis size.

It is often better to apply the commonly-used "split-valence" approach. This allows the orbitals that have been generated to be "split" into multiple functions, so as to generate so-called "split-valence multiple-zeta" basis sets. In this formalism, one function $f(r)$ can be split into two functions $g_1(r)$ and $g_2(r)$ according to the following:

1. A matching radius r_m is chosen: for $r > r_m$, we set $g_2(r) = f(r)$. For $r \leq r_m$, we set $g_2(r) = r^l(a_l - b_l r^2)$, where a_l and b_l are chosen such that $g_2(r_m) = f(r_m)$ and $g_2'(r_m) = f'(r_m)$.
2. The other function, $g_1(r)$, is set to $f(r) - g_2(r)$, so $g_1(r) = 0$ for $r \geq r_m$.
3. Both functions are renormalised, so $\int_0^{R_c} |g_1(r)|^2 r^2 dr = 1$ and $\int_0^{R_c} |g_2(r)|^2 r^2 dr = 1$.

Splitting of a term is activated by adding a colon after the term and specifying the "split norm" value. This is the fraction p of the total norm of the orbital which is beyond the matching radius r_m, such that $\int_{r_m}^{R_c} |f(r)|^2 r^2 dr = p$. If this colon is present, the solver will take into account the total number of orbitals which will result from this term *after splitting,* when counting forward in the configuration terms to determine which orbitals to solve. For example, if we wished to generate a Double-Zeta Polarisation (DZP) basis for oxygen ($2 \times 1 \times s$, $2 \times 3 \times p$,$1 \times 5 \times d$), where the last 15% of the norm was matched for the s and p-orbitals, we would use the following:

```
%block species
O O 8 13 9.0
%endblock species
%block species_atomic_set
O "SOLVE conf=2s2:0.15 2p4:0.15 3sX 3pX 3d0"
%endblock species_atomic_set
```

So that you can tell that it is happening, the code will output a message along the following lines when splitting a given orbital.

```
Splitting orbital 1, splitnorm= 0.150000000
Splitting orbital 1, splitnorm= 0.060000000
```

The result of the splitting can be viewed in the "initial_rad_ngwf_xx" files.

Obtaining Polarisation Orbitals through Perturbation

As well as including more flexibility for the valence orbitals, in the form of multiple-zeta basis sets, one frequently also wants to expand the basis by extending it to higher angular momentum channels. This can be done by simply increasing the number of NGWFs requested and ensuring through the 'conf=' string that the extra functions obtained are of the right angular momentum. However, the resulting high-l states tend to be unbound in the free atom, and therefore do not necessarily add anything particularly useful to the basis.

There is an alternative means to generate higher-l states, using perturbation theory. In this, one effectively applies an electric field to the valence states of angular momentum l and polarises them, resulting in a set of states of angular momentum $l + 1$. Imagine we have an orbital $\psi_0(\mathbf{r})$ of angular momentum l, m which is an eigenstates of the original Hamiltonian $\hat{H}_0$ with eigenvalue ϵ_0:

$$\psi_0(\mathbf{r}) = \sum_\nu c_{0,\nu}\, B_{l,\nu}(r)\, S_{lm}(\hat{\mathbf{r}}) \ .$$

We wish to polarise this orbital by applying an electric field $\mathcal{E}$ in the z-direction:

$$\hat{H}_1 = \mathcal{E} r\, S_{10}(\hat{\mathbf{r}}) \ ,$$

(since $S_{10}(\hat{\mathbf{r}}) = z/r$). Perturbing ψ_0 with $\hat{H}_1$ gives us no shift in energy to first-order, since the perturbation is an odd multiplicative function of z, meaning $\epsilon_1 = 0$. What about the change in the wavefunction? This obeys

$$(\hat{H}_0 - \epsilon_0)\psi_1(\mathbf{r}) = -(\hat{H}_1 - \epsilon_1)\psi_0(\mathbf{r})$$

In principle, $\psi_1(\mathbf{r})$ could have any angular momentum components, but we can see that in practice it only contains $L = l \pm 1$, since the dipole selection rule excludes all other channels. We already have $l - 1$ states in our basis, so we conclude that $\psi_1(\mathbf{r})$ need only include $l + 1$, and we can expand ψ_1 in terms of the $l + 1$ basis functions:

$$\psi_1(\mathbf{r}) = \sum_{\nu} c_{1,\nu}\, B_{l+1,\nu}\, S_{l+1,m}(\hat{\mathbf{r}})$$

Therefore we can generate a shifted Hamiltonian

$$H^{l+1}_{\nu,\nu'} = \int_0^{R_c} B_{l+1,\nu}(r) \left[(\hat{H}^{l+1} - \epsilon_0) B_{l+1,\nu'}(r)\right] r^2 dr \,, \tag{3.1}$$

and the components of the RHS of Eq. (3.1)

$$D_\nu = -\int_0^{R_c} B_{l+1,\nu}(r) r \psi_0(r) r^2 dr \,.$$

To solve for $c_{1,\nu}$ we just need to invert $H^{l+1}_{\nu,\nu'}$and apply it to D_ν, and then renormalise the result to have a norm of 1.

In practice, polarisation of a given configuration term of angular momentum l, to form a perturbative polarisation orbital for $l + 1$, is achieved by adding "|P" to the term, for example:

```
%block species

O O 8 13 9.0

%endblock species

%block species_atomic_set

O "SOLVE conf=2s2:0.15 2p4:0.15|P"

%endblock species_atomic_set
```

So that you can tell that it is happening, the code will output a message along the following lines when polarising a given orbital:

```
Polarising orbital 1 to generate l= 2 function (orbital 3)
```

Again, the result can be viewed by plotting the relevant "initial_rad_ngwf_xx" file.

Overriding radii

By default, the cutoff radius used for all the orbitals of an atom is the same R_c as defined in the `%block species` entry for that element. However, we can override this, either for all orbitals, or for certain angular momentum channels.

To override the radius for all channels, for example to $7.0a_0$, would we add the flag "R=7.0" to the SOLVE string:

```
%block species_atomic_set

O "SOLVE conf=2s2:0.15 2p4:0.15 3sX 3pX 3d0 R=7.0"
```

```
%endblock species_atomic_set
```

Or leave the default values for all other channels, but override the d-channel only to $5.0a_0$, we would use

---

```
%block species_atomic_set
O "SOLVE conf=2s2:0.15 2p4:0.15 3sX 3pX 3d0 Rd=5.0"
%endblock species_atomic_set
```

Adjusting confining potentials

By default, a confining potential is applied, of the form:

$$V_{\mathrm{conf}}(r) = S\ \exp[-w_l/(r - R_c + w_l)]/(r - R_c)^2$$

where S is the maximum height of the confining potential (at $r = R_c$), and w_l is the width of the region over which it is applied. By default, $S = 100$ Ha, and$w_l = 3.0a_0$ for all l-channels used. These can also be overridden, either all at once or for specific l-values in the case of w.

For example, to set no confining potential on the confined d-orbitals in Zinc, but to keep the default one on all the other orbitals, we could set $w_d = 0$:

```
%block species_atomic_set
Zn "SOLVE conf=3d10 4s2 Rd=5.0 wd=0.0"
%endblock species_atomic_set
```

Or to turn off confinement potentials entirely, and generate $R_c = 15.0a_0$ orbitals to match those generated by CASTEP's atomsolver (this should allow direct comparison of energies, given suitable tweaking of the energy cutoffs so that they exactly match:

```
%block species_atomic_set
O "SOLVE R=15.0 S=0.0"
%endblock species_atomic_set
```

Note that there can be problems with convergence for certain choices of confining potential. In particular, if you apply different confining potentials to different *occupied* orbitals, there will be problems obtaining agreement between the Harris-Foulkes estimator and the actual total energy - because the band energy will incorporate the different confining potentials, but the total energy cannot. The confining potential on angular momentum channels with no occupied orbitals can therefore be whatever you like, but those of occupied orbitals must all match. The exception to this is if the cutoff radius ment of one channel is less than the onset radius for the others. In this case, there is no need to apply a confinement to the lower-cutoff channel at all (eg in the example above for Zinc).

Initial Guess Density: Setting initial charges and spins

The atomsolver solutions are by default also used to provide an initial guess for the valence electron density. This is used to generate the initial Hamiltonian, which determines the occupation of the orbitals via Palser-Manolopoulos or other kernel optimisation schemes. Therefore it is important that this initial density is a reasonably good guess to the real density.

A superposition of atomic densities is usually fine for neutral systems without large magnetic moments. Sometimes, however, one needs to adjust the charges and spins of this initial density. It appears to be a rather bad idea to actually adjust the orbital occupations of the pseudoatoms self-consistently: it becomes impossible to disentangle the effect of changing the orbital from that of changing the density.

A better approach, therefore, is to retain the same pseudoatomic solutions for the neutral atom, but adjust the orbital occupations only at the point where they are used to generate the density.

This can be done by specifying "INIT CHARGE=X SPIN=Y" in the SOLVE string. Either CHARGE or SPIN can be omitted if they are not required. For example, for a manganese ion with charge +3 and spin 4, we might set

---

```
%block species_atomic_set

Mn "SOLVE conf=3d5 4s2 wd=7.0 INIT SPIN=4 CHARGE=+3"

%endblock species_atomic_set
```

The default occupation for the neutral atom is $3d^5\ 4s^2$. However, we ask it to apply a charge of +3, and this will remove occupation number from the orbitals with the highest energy until the right charge is obtained. In this case the resulting occupation will be $3d^4\ 4s^0$. Next, the spin is applied, resulting in $3d^4_\uparrow\ 3d^0_\downarrow$. Note that the charge is applied first, followed by the spin.

[Ruiz-Serrano2012] A. Ruiz-Serrano, N.D.M. Hine and C.-K. Skylaris, *in press*, (2012).

[Soler2002] J.M. Soler, E. Artacho, J.D. Gale, A. Garcia, J. Junquera, P. Ordejon, and D. Sanchez-Portal,*The SIESTA method for ab initio order-N materials simulation*, J. Phys. Condens. Matter 14, (2002).

[Artacho1999] E. Artacho, D. Sanchez-Portal, P. Ordejon, A. Garca, and J. M. Soler, *Linear-scaling ab-initio calculations for large and complex systems*, Phys. Status Solidi B 215, 809 (1999).

[Blum2009] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler: *Ab initio molecular simulations with numeric atom-centered orbitals*, Comput. Phys. Commun. 180, 2175 (2009).

[Tarralba2008] A. S. Torralba, M. Todorovic, V. Brazdova, R. Choudhury, T. Miyazaki, M. J. Gillan, and D. R. Bowler. *Pseudo-atomic orbitals as basis sets for the O(N) DFT code CONQUEST*, J. Phys. Condens. Matt. 20(29), (2008).

[Chen2010] M. Chen, G.-C. Guo, and L. He, *Systematically improvable optimized atomic basis sets for ab initio calculations*, J. Phys. Condens. Matt. 22, 445501, (2010).

3.2 Conduction NGWF optimisation and optical absorption spectra

Author
Laura E. Ratcliff, Imperial College London

Date
July 2011

3.2.1 Conduction calculations

As a consequence of the NGWF optimisation process in ONETEP the occupied (valence) Kohn-Sham states are well represented by the NGWFs, but the unoccupied (conduction) NGWFs are not, so that upon diagonalisation of the Hamiltonian at the end of a calculation, if one were to compare the resulting eigenvalues with a conventional cubic-scaling DFT code such as CASTEP [Clark2005], the ONETEP conduction states would be higher in energy than the CASTEP states, and some conduction states might be missing [Skylaris2005]. In order to correct this problem, a method has been implemented whereby a second set of NGWFs (referred to as the conduction NGWFs) are optimised to accurately represent the Kohn-Sham conduction states. This is done with asymptotic linear-scaling computational effort by constructing an idempotent density matrix representing the manifold of conduction states (rather than by solving for them explicitly). Optimisation of the NGWFs describing the conduction states proceeds as with the valence states, using a dual-loop system by which the conduction density kernel and the conduction NGWF coefficients are simultaneously optimised.

It should be noted that the Kohn-Sham eigenvalues will of course not be expected to exactly correspond to the true quasi-particle energies, however in practice reasonable agreement with experiment has been seen to occur in a number of systems, particularly when using the scissor operator [Godby1986], [Gygi1989].

The conduction NGWF optimisation takes the form of a non-self-consistent calculation following a ground-state calculation, where the density and potential calculated in the ground-state calculation are re-used. A projected Hamiltonian is then constructed in the conduction NGWF basis, using the density operator as a projection operator. This projected Hamiltonian is modified to avoid problems which might occur if the Hamiltonian and density operators do not commute perfectly. Additionally, the valence states are shifted up in energy by some amount w, such that they become higher in energy than the conduction states. The projected conduction Hamiltonian is thus written:

$$\begin{aligned}\left(H_{\chi}^{\text{proj}}\right)_{\alpha\beta} = & \quad \langle\chi_{\alpha}|\hat{H} - \hat{\rho}\left(\hat{H} - w\right)\hat{\rho}|\chi_{\beta}\rangle \\ = & \quad \left(H_{\chi}\right)_{\alpha\beta} - \left(T^{\dagger}KH_{\phi}KT\right)_{\alpha\beta} \\ & \quad + w\left(T^{\dagger}KS_{\phi}KT\right)_{\alpha\beta},\end{aligned}$$

where $\{|\phi_{\alpha}\rangle\}$ is the set of valence NGWFs and $\{|\chi_{\alpha}\rangle\}$ the set of conduction NGWFs. $\boldsymbol{\rho}$ is the valence density matrix, $\mathbf{K}$ is the valence density kernel, $\mathbf{S}_{\phi}$ is the valence overlap matrix and $\mathbf{H}_{\phi}$ is the valence Hamiltonian. $\mathbf{S}_{\chi}$ is the conduction overlap matrix, $\mathbf{T}$ is the valence-conduction cross overlap matrix defined as $T_{\alpha\beta} = \langle\phi_{\alpha}|\chi_{\beta}\rangle$, $\mathbf{H}_{\chi}$ is the (unprojected) conduction Hamiltonian, $\mathbf{H}_{\chi}^{\text{proj}}$ is the projected conduction Hamiltonian, $\mathbf{Q}$ is the conduction density matrix and $\mathbf{M}$ is the conduction density kernel. The conduction NGWFs and kernel are then minimised with respect to the energy expression $E = \text{tr}\left[\mathbf{Q}\mathbf{H}_{\chi}^{\text{proj}}\right]$, following the same procedure as in a standard ONETEP calculation. The shift can either be set to a constant value, or updated during a calculation, by setting it to be higher than the highest eigenvalue as calculated in the conduction NGWF basis.

At the end of the conduction NGWF optimisation process, the valence and conduction NGWF basis sets are combined into a new 'joint' basis, which will be capable of accurately representing both the occupied and unoccupied Kohn-Sham states. Other properties such as optical absorption spectra can then be calculated in this joint basis.

For further information see Ratcliff *et al.* [Ratcliff2011].

3.2.2 Performing conduction calculations in ONETEP

In order to optimise a set of NGWFs capable of accurately representing the Kohn-Sham conduction states in ONETEP, it is first necessary to have performed a standard ONETEP ground-state calculation and have retained the density kernel and NGWF output files. No special parameter values are required for this stage, although it may be worth setting ODD_PSINC_GRID to true, as conduction NGWF radii generally need to be larger than valence NGWF radii in order to achieve large convergence, and so it is more likely that the FFT box will be required to be equal to the psinc grid, and as both stages of the calculation must have the same cut-off energy and therefore grid size, it is desirable to have an odd grid for both the cell size and FFT box.

Once a ground-state calculation has been performed, a conduction calculation can be performed by setting TASK=COND. The number of conduction NGWFs per species and their radii must then be specified in the SPECIES_COND block, which follows the same pattern as the species block. The initial NGWFs, if not specified, will be equal to the initial NGWFs used for the valence density matrix. This choice can be overridden by specifying different choices in a SPECIES_ATOMIC_SET_COND block, which can be set to use the pseudoatomic solver by setting "SOLVE" for each species. One useful option is to specify that certain valence states, particularly those known to be fully filled and thus not expected to contribute to the manifold of conduction states, should be included in the calculation of the ground state of the pseudoatom but left out of the conduction NGWF set. For example, when generating conduction NGWFs for Cadmium, one might want to include the 10 filled 4d states in the atom calculation, but since they are not expected to contribute to the unoccupied states, they can be omitted from the conduction NGWFs by setting a splitnorm of "-1" for them, through the following solver string: "SOLVE conf=4d10:-1".

The conduction density kernel must contain a specific number of occupied states, and only these states will contribute to the NGWF gradient: COND_NUM_STATES is used to specify the number of conduction states to be optimised. In principle this could be any number, but in practice the higher energy conduction states converge rather slowly with respect to conduction NGWF radii, and in particular completely delocalised conduction states are very hard to represent using localised basis functions. Therefore results should be treated with caution when optimising high energy conduction states. A good rule-of-thumb is, if at all possible, to try to choose a manifold of conduction states above which there is a reasonable sized gap in the density-of-states (as calculated with the valence NGWFs).

For truly asymptotically linear-scaling calculations, one must truncate the conduction density kernel. The cutoff for this truncation is specified using COND_KERNEL_CUTOFF, although it is expected that high levels of kernel truncation will significantly limit the accuracy of the calculated conduction states. If unsure, do not use kernel truncation unless you are confident you have verified that the properties you are interested in are unaffected by the truncation.

At the end of a conduction calculation, diagonalisations are automatically performed of the valence Hamiltonian, both the projected and unprojected conduction Hamiltonians and the joint valence-conduction Hamiltonian. The eigenvalues are written to the corresponding .bands files. However, no joint basis density kernel is generated and so the occupancies are not calculated within this basis. The unprojected conduction eigenvalues are of limited use to most users, as it is difficult to determine which are conduction states and which are poorly represented valence states. For the projected conduction eigenvalues, the gap referred to in the output is not the usual gap, rather it is the gap between the highest optimised conduction state and the lowest unoptimised conduction state. If required, it is also possible to plot the orbitals in either the valence and conduction NGWF basis sets, and/or in the joint basis set, using the keywords COND_PLOT_VC_ORBITALS and COND_PLOT_JOINT_ORBITALS.

A standalone properties calculation can also be performed on the basis of sets of valence and conduction NGWFs and kernels which have already been calculated. To enable this, set TASK=PROPERTIES_COND: the options COND_READ_TIGHTBOX_NGWFS and COND_READ_DENSKERN will automatically be enabled, the NGWF optimisation will skipped and the calculation will proceed straight to the stage of diagonalisation of the Hamiltonian and plotting of the orbitals.

Automatic setup of COND_NUM_STATES

Since it is not always straightforward to guess a sensible number of conduction states to converge, the code will by default attempt to choose an appropriate number of states for the user. By default, at the start of a conduction optimisation, the code will count the number of unoccupied states of the valence Hamiltonian with negative eigenvalues to arrive at a guess of the number of bound states in a finite system. The code will also check for any degeneracy of the highest unoccupied state included in the calculation and automatically include more states until an energy gap of at least 0.001 Ha between states that get optimised and states that are unoptimised is achieved.

Since counting the number of eigenstates with negative eigenvalues in order to obtain the number of bound states is only strictly valid in finite systems, it is possible for the user to define an energy range, as measured from the HOMO energy level, and the code will attempt to optimise all states within that energy range. The two keywords controlling the automatic conduction state setup are given by COND_ENERGY_RANGE and COND_ENERGY_GAP. The first keyword defines the desired energy range in Hartree while COND_ENERGY_GAP defines the minimum required energy gap between the highest conduction state that is optimised and the lowest conduction state that stays unoptimised.

Setting the shift

There are a number of parameters relating to the shift, w, used in the projected conduction Hamiltonian. It is possible to keep the shift at some fixed value (defined using COND_INIT_SHIFT) during the calculation, by setting COND_FIXED_SHIFT to true. Alternatively, it can be automatically updated during the calculation, which is usually the safest way to proceed. This is achieved by calculating the highest eigenvalue within the conduction NGWF basis at the start of each NGWF iteration (providing COND_CALC_MAX_EIGEN is set to true), and comparing the current shift to this eigenvalue. Providing the shift is higher than the highest eigenvalue, it remains unchanged, but if the maximum eigenvalue has become greater than the current shift, it is updated to equal the maximum eigenvalue plus some extra buffer value (defined by COND_SHIFT_BUFFER).

Local minima

In practice, it is sometimes possible to become trapped in local minima, where the ordering of states within the initial unoptimised basis doesn't correspond to the correct order, and so sometimes states are missed. The problem can be identified by decreasing NGWF_THRESHOLD_ORIG and seeing if the gradient stagnates while the energy continues to decrease, or by plotting convergence graphs with conduction NGWF radii where sharp changes in energy are sometimes observed with small changes in conduction NGWF radii. In practice it is therefore very important to systematically converge calculations with respect to the conduction NGWF radii, which might require larger values than ground-state ONETEP calculations. This problem can typically be avoided by optimising some extra states (COND_NUM_EXTRA_STATES) above the required number of conduction states for a few iterations (COND_NUM_EXTRA_ITS) (typically 5-10 iterations). Selecting the required number of extra states to include is mostly a trial and error process whereby the number of extra states should be increased until no changes are seen in the calculated conduction energy.

Additional notes on input parameters

As the ground-state NGWFs and density kernel are required for the conduction calculation, READ_TIGHTBOX_NGWFS and READ_DENSKERN are automatically set to true. There are separate variables for the corresponding conduction quantities (COND_READ_TIGHTBOX_NGWFS and COND_READ_DENSKERN) which can be set to true for restarting conduction calculations. The parameters WRITE_TIGHTBOX_NGWFS and WRITE_DENSKERN are not independently specified for the conduction and valence NGWF basis sets.

Conduction calculations in implicit solvent

Some care has to be taken when performing a conduction optimisation for a system embedded in an implicit solvent (see the separate Implicit Solvation documentation on how to perform a ground state calculation in implicit solvent). The reason for this is that the ground state in the implicit solvation model is often computed in a two step process. In the first step the solvation cavity is computed as an isosurface of the ground state density in vacuum, while in the second step the ground state of the system is evaluated for that fixed cavity. In order for the conduction calculation to be consistent with the ground state calculation, the same solvation cavity has to be used in both cases. To ensure that the code uses the correct restart files when setting up the ground state Hamiltonian at the beginning of a conduction optimisation, the following sets of keywords should be used:

```
Task :  Singlepoint Cond
is_implicit_solvent :  T
is_auto_solvation :  T
is_smeared_ion_rep :  T
```

The code will then automatically perform a SinglePoint and a conduction calculation in the implicit solvent, using the same solvation cavity for both the ground state and the conduction state calculation. This is achieved by writing .vacuum_dkn and .vacuum_tightbox_ngwfs files that are used to set up the solvation cavity.

If further conduction calculations are required using the same ground state, for example in order to change the number of conduction states converged, it is possible to change the Task to COND and include the keyword is_separate_restart_files: T. This triggers the use of the .vacuum files to set up the correct solvation cavity at the beginning of the COND calculation.

3.2.3 Optical absorption spectra

The calculation of matrix elements for the generation of optical absorption spectra using Fermi's golden rule has been implemented in ONETEP following the method used in CASTEP, as outlined by Pickard [Pickard1997]. Using the dipole approximation, the imaginary component of the dielectric function is defined as

$$\varepsilon_2(\omega) = \frac{2e^2\pi}{\Omega\epsilon_0} \sum_{\mathbf{k},v,c} |\langle\psi^c_{\mathbf{k}}|\hat{\mathbf{q}}\cdot\mathbf{r}|\psi^v_{\mathbf{k}}\rangle|^2 \,\delta\left(E^c_{\mathbf{k}} - E^v_{\mathbf{k}} - \hbar\omega\right), \tag{3.2}$$

where v and c denote valence and conduction bands respectively, $|\psi^n_{\mathbf{k}}\rangle$ is the nth eigenstate at a given $\mathbf{k}$-point with a corresponding energy $E^n_{\mathbf{k}}$, Ω is the cell volume, $\hat{\mathbf{q}}$ is the direction of polarization of the photon and $\hbar\omega$ its energy. Currently only the Γ point is included in the sum over $\mathbf{k}$-points.

As the position operator is ill-defined in periodic boundary conditions, this should instead be calculated using a momentum operator formalism, where the two are related via [Read1991]:

$$\langle\phi_f|\mathbf{r}|\phi_i\rangle = \frac{1}{i\omega m}\langle\phi_f|\mathbf{p}|\phi_i\rangle + \frac{1}{\hbar\omega}\langle\phi_f|\left[\hat{V}_{nl},\mathbf{r}\right]|\phi_i\rangle.$$

The commutator term can then be found using the identity [Motta2010]:

$$\begin{aligned}&(\nabla_{\mathbf{k}} + \nabla_{\mathbf{k}'})\left[\int e^{-i\mathbf{k}\cdot\mathbf{r}} V_{nl}(\mathbf{r},\mathbf{r}')\, e^{i\mathbf{k}'\cdot\mathbf{r}'}\, d\mathbf{r}\, d\mathbf{r}'\right]\\ =\;& i\int e^{-i\mathbf{k}\cdot\mathbf{r}}\left[V_{nl}(\mathbf{r},\mathbf{r}')\,\mathbf{r}' - \mathbf{r}V_{nl}(\mathbf{r},\mathbf{r}')\right] e^{i\mathbf{k}'\cdot\mathbf{r}'}\, d\mathbf{r}\, d\mathbf{r}',\end{aligned}$$

where the derivative can either be calculated directly or using finite differences in reciprocal space. Once the matrix elements have been calculated in this manner, they can then be used to form a weighted density of states according to equation (3.2).

3.2.4 Calculating optical absorption spectra in ONETEP

The calculation of matrix elements for optical absorption spectra is activated by setting COND_CALC_OPTICAL_SPECTRA to true. The matrix elements are then calculated at the end of a conduction calculation in both the valence and joint valence-conduction NGWF basis sets. Various options can be modified, including the choice of calculating the matrix elements in either the position or momentum representation, using the parameter COND_SPEC_CALC_MOM_MAT_ELS. For accurate results, the position operator should only be used for molecules, where the conduction NGWFs are sufficiently small compared to the size of the unit cell that they do not overlap with any periodic copies. If using the momentum formulation, the default behaviour is to also calculate the commutator between the nonlocal potential and the position operator, although setting COND_SPEC_CALC_NONLOC_COMM will switch this off. Additionally, the method of calculation of the commutator can be specified using COND_SPEC_CONT_DERIV, so that either a continuous derivative or finite difference method is employed. If using the finite difference option, the finite difference shifting parameter can also be specified using COND_SPEC_NONLOC_COMM_SHIFT.

Outputs

If the input filename is seed.dat then the matrix elements will be written to seed_val_OPT_MAT_ELS.txt and seed_joint_OPT_MAT_ELS.txt. These contain the matrix elements between all states in the x, y and z directions, and the energies of each state, as well as the transition energy, are also printed. For calculations in the momentum representation, the real and imaginary components of the matrix element are printed in the additional two columns at the end.

[Clark2005] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson and M. C. Payne, Z. Kristallogr. **220**, 567 (2005).

[Skylaris2005] C.-K. Skylaris, P. D. Haynes, A. A. Mostofi and M. C. Payne, J. Phys. Condens. Matter **17**, 5757 (2005).

[Godby1986] R. W. Godby, M. Schlüter and L. J. Sham, Phys. Rev. Lett **56**, 2415 (1986).

[Gygi1989] F. Gygi and A. Baldereschi, Phys. Rev. Lett **62**, 2160 (1989).

[Ratcliff2011] L. E. Ratcliff, N. D. M. Hine and P. D. Haynes *In Preparation* (2011).

[Pickard1997] C. J. Pickard, Ph.D. thesis, University of Cambridge (1997).

[Read1991] A. J. Read and R. J. Needs, Phys. Rev. B **44**, 13071 (1991).

[Motta2010] C. Motta, M. Giantomassi, M. Cazzaniga, K. Gal-Nagy and X. Gonze, Comput. Mater. Sci. **50**, 698 (2010).

3.3 Finite-temperature DFT calculations using the Ensemble-DFT method

Author
Álvaro Ruiz Serrano, University of Southampton

Author
Extended to spin relaxation by Kevin Duff, University of Cambridge

Author
Extended to include Grand Canonical Ensemble by Arihant Bhandari, University of Southampton

Author
Extended to include Pulay mixing and input trial step by Gabriel Bramley, University of Southampton

Date
August 2013

Date
Extended by Kevin Duff April 2018

Date
Extended by Arihant Bhandari September 2020

Date
Extended by Gabriel Bramley November 2020

3.3.1 Basic principles

This manual describes how to run finite-temperature calculations using ONETEP. A recent implementation uses a direct minimisation technique based on the Ensemble-DFT method [Marzari1997]. The Helmholtz free energy functional is minimised in two nested loops. The inner loop performs a line-search in the space of Hamiltonian matrices (in a similar fashion as described in Ref. [Freysoldt2009]), for a fixed set of NGWFs. Then, the outer loop optimises the NGWFs psinc expansion coefficients for a fixed density kernel. For a more detailed description and discussion of this method in ONETEP, see Ref. [Ruiz-Serrano2013].

Using the NGWF representation, the Helmholtz free energy functional becomes:

$$A_{\mathcal{T}}\left[\{H_{\alpha\beta}\},\{|\phi_{\alpha}\rangle\}\right] = E\left[\{H_{\alpha\beta}\},\{|\phi_{\alpha}\rangle\}\right] - \mathcal{T}S\left[\{f_i\}\right]. \tag{3.3}$$

where $\{H_{\alpha\beta}\}$ is the NGWF representation of the Hamiltonian matrix, $\{|\phi_{\alpha}\rangle\}$ is the current set of NGWFs, $\mathcal{T}$ is the electronic temperature, $E\left[\{H_{\alpha\beta}\},\{|\phi_{\alpha}\rangle\}\right]$ is the energy functional and $S\left[\{f_i\}\right]$ is an entropy term. The occupancies of the Kohn-Sham states, $\{f_i\}$ are calculated from the energy levels, $\{\epsilon_i\}$, using the Fermi-Dirac distribution:

$$f_i\left(\epsilon_i\right) = \left(1+\exp\left[\frac{\epsilon_i-\mu}{k_{\mathrm{B}}\mathcal{T}}\right]\right)^{-1}.$$

where μ is the Fermi level. To obtain the energy eigenvalues, $\{\epsilon_i\}$, the Hamiltonian matrix is diagonalised as:

$$H_{\alpha\beta}M_i^{\beta} = S_{\alpha\beta}M_i^{\beta}\epsilon_i,$$

where $\{S_{\alpha\beta}\}$ are the elements of the NGWF overlap matrix, and $\left\{M_i^{\beta}\right\}$ are the expansion coefficients of the Kohn-Sham eigenstates in the NGWF basis set. At the moment, this is a cubic-scaling operation that requires dealing dense matrices, which makes it memory-demanding.

3.3.2 Free- and fixed-spin EDFT

By default in spin polarized runs, the total occupancy of each spin channel is held fixed; each spin channel has its own Fermi level determined by this constraint. Alternatively the whole system can be held at one Fermi level dictated by the conservation of the total number of electrons in the system, allowing the net spin to freely relax.

Free-spin EDFT should be appropriate for most applications unless there's a reason to hold the system fixed at a given net spin. As with any minimization with potentially many minima, the final state may depend on initial conditions. As a special case, free-spin EDFT may not be able to symmetry-break a system that wants to have any kind of spin polarization but that is initialized to have 0 net spin. The general advice for simple systems like basic ferromagnets (though this should not replace good system-specific judgment) is to slightly over-specify the expected net spin on each atom and hold the spin fixed for a few iterations before being allowed to relax. For example a cobalt cluster is expected to have a net spin per atom lower than that of an isolated atom, that decreases to bulk-like as a function of cluster size. A good initialization may be to give each atom atomic-like net spin and hold the net spin fixed for 3-5 NGWF CG iterations, then allow it to relax.

3.3.3 Compilation

By default, ONETEP is linked against the Lapack library [lapack_web] for linear algebra. The Lapack eigensolver DSYGVX [DSYGVX], can only be executed in one CPU at a time. Therefore, EDFT calculations with Lapack are limited to small systems (a few tens of atoms). Calculations on large systems are possible if, instead, ONETEP is linked against ScaLapack library [scalapack_web] during compilation time. The ScaLapack eigensolver, PDSYGVX, can be run in parallel using many CPUs simultaneously. Moreover, ScaLapack can distribute the storage of dense matrices across many CPUs, thus allowing to increase the total memory allocated to a given calculation in a systematic manner, simply by requesting more processors. For the compilation against ScaLapack to take effect, the flag `-DSCALAPACK` must be specified during the compilation of ONETEP.

3.3.4 Pulay Mixing EDFT

In default EDFT, the Hamiltonian is updated using a damped fixed point update routine:

$$H_{\alpha\beta}^{(m+1)} = H_{\alpha\beta}^{(m)} + \lambda R[H_{\alpha\beta}^{(m)}]$$

Where the λ defines the mixing parameter and residual is defined as:

$$R[H_{\alpha\beta}^{(m)}] = \tilde{H}_{\alpha\beta}^{(m)} - H_{\alpha\beta}^{(m)}$$

Where $\tilde{H}_{\alpha\beta}^{(m)}$ is the diagonlised Hamiltonian obtained at step m. At a sufficiently low value of λ, most systems will achieve convergence, but at an increasingly slow rate as the system increases in size. Convergence can be accelerated using quasi-Newton update methods such as Broyden or Pulay methods, the latter of which is implemented in EDFT as an alternative to the damped fixed point method.

The implementation in ONETEP uses a similar logic to other DFT implementations of Pulay's method, except the Hamiltonian is optimised instead of the density:

$$H_{\alpha\beta}^{(m+1)} = \sum_{j=m-n+1}^{m} c_j H_{\alpha\beta}^{(j)} + \lambda \sum_{j=m-n+1}^{m} c_j R[H_{\alpha\beta}^{(j)}]$$

Where the history length is defined n and the co-efficients c_j are obtained through the procedure outlined by Ref. [Kresse1996]. For the systems tested, this method leads to improved convergence, especially for larger metallic systems. Further information can be found in Ref. [Woods2019].

3.3.5 Increased Calculation Speed Using Fixed Step Sizes

As described in the Section on Pulay mixing, λ defines the step length taken at each inner loop iteration. In the default algorithm, an optimal λ value which gives the greatest decrease in the Lagrangian is determined by a line search routine. Although this improves the robustness of the algorithm, the line search requires two or more energy evaluations per inner loop step to obtain the optimum λ value. If λ varies very little over the course of the calculation, this can double the computational expense of each inner loop iteration for a negligible increase in the accuracy for each step.

Alternatively, one can fix the λ to a reasonable value for a significant speed-up by ensuring only one energy evaluation is performed per inner loop iteration. However, this option is less robust than the default line search algorithm, as the fixed λ value may produce either sub-optimal energy decreases or energy increases for certain steps. Furthermore, if λ is chosen to be too high, your answer may diverge from the ground state by taking several consecutive positive Lagrangian steps (A warning will be provided if this occurs too often). Conversely, convergence will be very slow if λ is chosen to be too low. λ is set with the `edft_trial_step` keyword, which switches from the line search algorithm if greater than 0, and uses the fixed λ value specified.

User input values of λ can be determined by running a standard EDFT calculation for a single NGWF iteration with line search and plotting the 'step' value printed at each iteration (in VERBOSE output mode). The safest option is to choose a value close to the minimum step value, but a slightly higher value can be selected, especially if larger step values are common. The first two steps of your calculation choose λ with line search regardless of your input, as optimal step sizes for these iterations are significantly larger than subsequent inner loop iterations. As such, these two iterations should be disregarded from your λ value selection analysis. As step sizes which yield stable convergence are system dependent, it is recommended to manually determine different λ values for systems with large differences in species or size.

3.3.6 Commands for the inner loop

Basic setup

- `edft: T/F` [Boolean, default `edft: F`]. If true, it enables Ensemble-DFT calculations.
- `edft_maxit: n` [Integer, default `edft_maxit: 10`]. Number of EDFT iterations in the ONETEP inner loop.
- `edft_smearing_width: x units` [Real physical, default `edft_smearing_width: 0.1 eV`]. Sets the value of the smearing width, $k_B\mathcal{T}$, of the Fermi-Dirac distribution. It takes units of energy (eV, Hartree) or temperature. For example, `edft_smearing_width: 1500 K` will set $\mathcal{T} = 1500$ degree Kelvin.
- `edft_update_scheme: damp_fixpoint/pulay_mix` [Character, default `dft_update_scheme: damp_fixpoint`]. Defines the mixing scheme for EDFT in the ONETEP inner loop.
- `edft_ham_diis_size: x` [Integer, default `edft_ham_diis_size: 10`]. Specifies the maximum number of Hamiltonians used from previous iterations to generate the new guess through Pulay mixing.
- `spin: x` [Real, default `spin: 0.0`]. For EDFT runs this value does not need to be an integer. Because we are considering an ensemble of states it can have any real value between $-\frac{n_{\text{elec}}}{2}$ to $\frac{n_{\text{elec}}}{2}$. Make sure you have enough bands to cover the more populated spin channel.
- `edft_spin_fix` [Integer, default `edft_spin_fix: -1`]. Control for whether the net spin of the system should remain fixed at `spin`, or relax during the run. Any negative number will fix the net spin. Nonnegative numbers n will hold the net spin fixed for n iterations then let it relax for the rest of the calculation.
- `edft_trial_step` [Integer, default `edft_trial_step: 0`]. Sets the value of λ, which fixes the step size in the EDFT inner loop, and switches off the line search for optimum λ values. If set to 0, the normal line search routine is used.

Tolerance thresholds

- `edft_free_energy_thres: x units` [Real physical, default `edft_free_energy_thres: 1.0e-6 Ha/Atom`]. Maximum difference in the Helmholtz free energy functional per atom between two consecutive iterations.
- `edft_energy_thres: x units` [Real physical, default `edft_energy_thres: 1.0e-6 Ha/Atom`]. Maximum difference in the energy functional per atom between two consecutive iterations.
- `edft_entropy_thres: x units` [Real physical, default `edft_entropy_thres: 1.0e-6 Ha/Atom`]. Maximum difference in the entropy per atom functional between two consecutive iterations.
- `edft_rms_gradient_thres: x` [Real, default `edft_rms_gradient_thres: 1.0e-4`]. Maximum RMS gradient $\frac{dA_{\mathcal{T}}}{df_i}$.
- `edft_commutator_thres: x units` [Real physical, default `edft_commutator_thres: 1.0e-5 Hartree`]. Maximum value of the Hamiltonian-Kernel commutator.
- `edft_fermi_thres: x units` [Real physical, default `edft_fermi_thres: 1.0e-3 Hartree`]. Maximum change in the Fermi energy between two consecutive iterations.

Advanced setup

- `edft_extra_bands: n` [Integer, default `edft_extra_bands: -1`]. Number of extra energy bands. The total number of bands is equal to the number of NGWFs plus `edft_extra_bands`. When set to a negative number, no extra bands are added.
- `edft_round_evals: n` [Integer, default `edft_round_evals: -1`]. When set to a positive integer value, the occupancies that result from the Fermi-Dirac distribution are rounded to `n` significant figures. This feature can reduce some numerical errors arising from the grid-based representation of the NGWFs.
- `edft_write_occ: T/F` [Boolean, default `edft_write_occ: F`]. Save fractional occupancies in a file.
- `edft_max_step: x` [Real, default `edft_max_step: 1.0`]. Maximum step during the EDFT line search.

3.3.7 Commands for the outer loop

The standard ONETEP commands for NGWF optimisation apply to the EDFT calculations as well. The only flag that is different is:

- `ngwf_cg_rotate: T/F` [Integer, default `ngwf_cg_rotate: T`]. This flag is always true in EDFT calculations. It ensures that the eigenvectors M_i^β are rotated to the new NGWF representation once these are updated.

3.3.8 Restarting an EDFT calculation

- `write_hamiltonian: T/F` [Boolean, default `write_hamiltonian: F`]. Save the last Hamiltonian matrix on a file.
- `read_hamiltonian: T/F` [Boolean, default `read_hamiltonian: F`]. Read the Hamiltonian matrix from a file, and continue the calculation from this point.
- `write_tightbox_ngwfs: T/F` [Boolean, default `write_tightbox_ngwfs: T`]. Save the last NGWFs on a file.
- `read_tightbox_ngwfs: T/F` [Boolean, default `read_tightbox_ngwfs: F`]. Read the NGWFs from a file and continue the calculation from this point.

If a calculation is intended to be restarted at some point in the future, then run the calculation with
`write_tightbox_ngwfs: T`
`write_hamiltonian: T`
to save the Hamiltonian and the NGWFs on disk. Two new files will be created, with extensions `.ham` and `.tightbox_ngwfs`, respectively. Then, to restart the calculation, set
`read_tightbox_ngwfs: T`
`read_hamiltonian: T`
to tell ONETEP to read the files that were previously saved on disk. Remember to keep a backup of the output of the first run before restarting the calculation.

the density kernel is not necessary to restart an EDFT calculation. However, it is necessary to calculate the electronic properties of the system, once the energy minimisation has completed. To save the density kernel on a file, set: `write_denskern: T`
to generate a `.dkn` file containing the density kernel. To read in the density kernel, set
`read_denskern: T`

3.3.9 Controlling the parallel eigensolver

Currently, only the ScaLapack PDSYGVX parallel eigensolver is available. A complete manual to this routine can be found by following the link in Ref. [PDSYGVX]. If ONETEP is interfaced to ScaLapack, the following directives can be used:

- `eigensolver_orfac: x` [Real, default `eigensolver_orfac: 1.0e-4`]. Precision to which the eigensolver will orthogonalise degenerate Hamiltonian eigenvectors. Set to a negative number to avoid reorthogonalisation with the ScaLapack eigensolver.
- `eigensolver_abstol: x` [Real, default `eigensolver_abstol: 1.0e-9`]. Precision to which the parallel eigensolver will calculate the eigenvalues. Set to a negative number to use ScaLapack defaults.

The abovementioned directives are useful in calculations where the ScaLapack eigensolver fails to orthonormalise the eigenvectors. In such cases, the following error will be printed in the input file:

`(P)DSYGVX in subroutine dense_eigensolve returned info= 2`.

Many times (although not always) this error might cause the calculation to fail. If this situation occurs, set
`eigensolver_orfac: -1`
`eigensolver_abstol: -1`
in the input file and restart the calculation. ScaLapack will not reorthonormalise the eigenvectors. Instead, an external Löwdin orthonormalisation process [Lowdin1950] will be triggered. This is usually more efficient for larger systems.

3.3.10 Grand Canonical Ensemble DFT

In simulations of electrochemical electrodes, the electrons can freely exchange between the electrode and the electrical circuit. So, there is no constraint on the number of electrons N. Rather, the electrode potential U is fixed, with respect to a reference electrochemical potential μ_{ref} which fixes the chemical potential of electrons μ:

$$\mu = \mu_{ref} - eU$$

Typical experiments use a standard hydrogen electrode as the reference electrode with $\mu_{ref}^{SHE} = -4.44$ eV. Once the chemical potential of electrons is fixed, the number of electrons changes as a dependent variable according to the Fermi-Dirac distribution in eq. .

$$N = \sum_i f_i$$

Thermodynamically, this corresponds to switching the electrons from the finite-temperature, fixed-number canonical ensemble to the finite-temperature, fixed-potential grand-canonical ensemble. Correspondingly, the relevant free energy minimized at equilibrium is the grand potential [Sundararaman2017]:

$$\Omega = A - \mu N$$

The following keywords are used for the grand-canonical ensemble DFT:

- `edft_grand_canonical: T/F` [Boolean, default `edft_grand_canonical: F`]. Switch to fixed-potential grand-canonical ensemble.
- `edft_reference_potential: x units` [Real physical, default `edft_reference_potential: -4.44 eV`]. Set the reference potential μ_{ref}. If no units are given, atomic units are considered: Ha (hartrees).
- `edft_electrode_potential: x units` [Real physical, default `edft_electrode_potential: 0.0 V`]. Set the electrode potential U. If no units are given, atomic units are considered: Ha/e, hartrees per elementary charge.

- `edft_nelec_thres: x` [Real, default `edft_nelec_thres: 1.0e-06 per atom`]. Convergence threshold on the change in number of electrons per spin channel per atom.

[Sundararaman2017] R. Sundararaman, W. Goddard, and T. Arias. J. Chem. Phys., 146(11):114104, 2017.

[Marzari1997] N. Marzari, D. Vanderbilt, and M. C. Payne. Phys. Rev. Lett., 79(7):1337–1340, 1997.

[Freysoldt2009] C. Freysoldt, S. Boeck, and J. Neugebauer. Phys. Rev. B, 79(24):241103, 2009.

[Ruiz-Serrano2013] A. Ruiz-Serrano and C.-K. Skylaris. A variational method for density functional theory calculations on metallic systems with thousands of atoms. J. Chem. Phys., 139(5):054107, 2013.

[Lapack_web] Lapack. http://www.netlib.org/lapack/.

[DSYGVX] Lapack DSYGVX eigensolver. http://netlib.org/lapack/double/dsygvx.f.

[Scalapack_web] ScaLapack. http://www.netlib.org/scalapack/.

[PDSYGVX] ScaLapack PDSYGVX eigensolver. http://www.netlib.org/scalapack/double/pdsygvx.f.

[Lowdin1950] Per-Olov Lowdin. On the non-orthogonality problem connected with the use of atomic wave functions in the theory of molecules and crystals. J. Chem. Phys., 18(3):365–375, 1950.

[Kresse1996] G. Kresse and J. Furthmüller. Efficient iterative schemes for *ab initio* total-energy calculations using a plane-wave basis set. Phys. Rev. B, 54:11169, 1996.

[Woods2019] N. Woods, M. Payne and P. Hasnip. Computing the self-consistent field in Kohn–Sham density functional theory J. Phys. Condens. Matter, 31:453001, 2019.

3.4 Running linear-scaling DFT calculations for metallic systems with the AQUA-FOE method

Author
Jolyon Aarons, University of Southampton

Author
Chris-Kriton Skylaris, University of Southampton

This is a guide to performing FOE calculations on metals in ONETEP using the EDFT functionality. The methodology will be explained briefly, but for detailed information about how these methods work and are implemented, appropriate references are presented. We provide a example calculation, walk through and input files at the end of this document. Finally, to perform these calculations, a recent (as of 2018) version of ONETEP is required, for instance version 4.5.4.4, or the upcoming version 5.0.

3.4.1 Metals in ONETEP

Calculations on conductors have been possible in ONETEP with EDFT since 2013. In its default mode, EDFT will perform a diagonalisation of the Hamiltonian matrix and form the density kernel matrix from the eigenvectors and a Fermi-Dirac occupancy function of the eigenvalues. If your calculations have fewer than 1000 atoms, this mode is probably your best option. The method scales cubically with system size, however, and when you have a few thousand atoms calculations are impractically slow on current supercomputers.

On larger systems, it is possible that you will have sufficient sparsity in the Hamiltonian matrix that a Fermi-Operator-Expansion based approach to constructing the density kernel will be faster. The idea behind this approach is that a power series expansion of the Fermi-Dirac distribution function can be applied directly to the Hamiltonian matrix, as matrix multiplications are linear operations. Hence, the density kernel matrix can be constructed from matrix powers of the Hamiltonian matrix, without diagonalisation. The main caveat with this is that sparsity must be exploited, or such a method is likely to be slower than the diagonalisation based alternative, due to the many matrix multiplications

that are required. Since sufficient sparsity is only obtained for metals with more than about 1000 atoms, we recommend the use of this method only for larger systems.

The way that we apply the matrix analogue of the Fermi-Dirac distribution function,

$$K = [I + \exp((H - \mu I)\,\beta)\,]^{-1},$$

without diagonalisation or the poor conditioning of applying the above operations directly is to firstly compute an expansion of a system of electrons at an integer multiple (2^n) of the electronic temperature, so that:

$$K^{\mathrm{HOT}} = [I + \exp((H - \mu I)\beta/2^n)\,]^{-1}. \tag{3.4}$$

The advantage of applying an expansion to hotter electrons is that the argument of the exponential function is reduced in magnitude, so fewer terms in expansion of the Fermi-Dirac distribution function are needed for a good approximation. In practice, we use a low order Chebyshev approximation to equation (3.4). To recover K, the density kernel matrix at the target temperature, we need to anneal K^{HOT} by halving the temperature, $n-1$ times. This can be achieved by applying the matrix analogue of the hyperbolic tangent double angle formula after noting that

$$K^{\mathrm{HOT}} = \frac{1}{2}\left(I + \tanh\left((H - \mu I)\frac{\beta}{2^{n+1}}\right)\right).$$

The annealing formula is given as:

$$K^{\mathrm{HOT}/2} = \frac{1}{2}\left(\frac{4K^{\mathrm{HOT}} - 2I}{I + (2K^{\mathrm{HOT}} - I)^2} + I\right), \tag{3.5}$$

which we do not apply directly in practice (this would cost 1 matrix product and 1 matrix solve / inversion), but expand equation (3.5) as another Chebyshev polynomial expansion, which is cheaper due to equation (3.5) being very well conditioned (we know that the eigenvalues of the denominator are necessarily between 1.0 and 2.0).

The application of equation (3.4) assumes that we know the chemical potential, μ. In practice, we do not. We instead start with a trial chemical potential and improve K in the sense that we drive it towards the electron conserving chemical potential. We achieve this by knowing how wrong we were at a given μ_i,

$$\Delta N_e = N_e - \mathrm{trace}\left(K(\mu_i)\right),$$

and by using hyperbolic trigonometry to correct by this by a change in chemical potential, $\Delta\mu$ without recalculating the expansion:

$$[I + \exp\{(H - \mu I)\beta \pm \beta\Delta\mu I\}]^{-1} = \frac{1}{2}\left(I + \frac{2K \pm \tanh\left(\frac{\beta\Delta\mu}{2}\right) I - I}{I \pm \tanh\left(\frac{\beta\Delta\mu}{2}\right) \times (2K - I)}\right).$$

Likewise with this equation, we do not apply it directly but expand it as a Chebyshev polynomial to avoid an inversion. Once again, this works well because we know that this is a well conditioned problem. To know how much to adjust the chemical potential we can use the derivative:

$$\frac{\partial N_e}{\partial \mu} = -\frac{\beta}{4}\left(1 - \mathrm{trace}(K^2)\right),$$

and the electron residual ΔN_e.

The final piece in this puzzle is to construct the electronic entropy, which we find as a matrix, taking its trace to find the scalar electronic entropy. We avoid calculating the Entropy as

$$\mathcal{S} = \mathrm{tr}[\mathbf{K}\,\ln(\mathbf{K}) + [\mathbf{I} - \mathbf{K}]\,\ln(\mathbf{I} - \mathbf{K})]\ , \tag{3.6}$$

because this involves calculating two expensive matrix logarithms. Instead we use a further expansion for this quantity, this time taking that of the Fermi-Dirac entropy expression. When expanding this expression as a Chebyshev expansion,

many terms are required to reach a good approximation at zero and one occupancy. To avoid this problem, we expand a form which maintains the property that the single particle entropy is zero at zero and one occupancy with fewer terms. This is:

$$s(x) \simeq ax^2 + \frac{b}{c + dx - dx^2} - e - ax = y(x), \tag{3.7}$$

where $a = 1.96056$, $b = 0.0286723$, $c = 0.114753$, $d = 1.98880$ and $e = 0.249860$. We then refine this with further expansions.

In practice, because sparsity is imposed on the density kernel and entropy matrices, we do not have perfectly accurate implicit eigenvalues. Some of these may be above one, so when taking large powers of the matrix, as we do in the power expansion of equation (3.7), the (implicit) eigenvalues of the entropy may explode, causing a problem. To detect and avoid this problem, we use a very simple quadratic approximation to the entropy.

The quadratic

$$f(x) = 3x^2 - 3x,$$

the coefficients of 3 minimise the integral of $x*Ln(x)+(1-x)*Ln(1-x)-cx^2-cx$ over [0:1]. The maximum error of this approximation is at $x = 0.5$, where the error is 0.05685 or 8.2% of the correct value at 0.5. This indicates that in the very worst case where every eigenvalue of $\mathbf{KS}$ is 0.5, then the error on the entropy from this approximation is 8.2%. The value of this is that $\mathbf{KSKS}$ is easy to calculate reliably, compared with a high order series expansion… i.e. if there is an eigenvalue of K which is a little above 1 because of noise due to truncation, the corresponding eigenvalues in the entropy matrix can be very large. This can accumulate in the entropy value, if there are many of these greater than 1 eigenvalues, to give a completely wrong value. We can, therefore, check the value of entropy we have calculated by comparing it with this quadratic approximation. If the absolute difference between the two values is less than 8.2 the high order value, then we assume that things have gone well and do nothing. If this is not the case, then there was a high amount of error accumulated and so we'll print a warning and a refined quadratic approximation (a quartic).

In practice, when we take all of these power expansions, we need to consider the tensorial transformation properties of the quantities we are considering, for instance: the Hamiltonian matrix, $H_{\alpha\beta}$ is covariant, so taking a power requires that we raise the appropriate indices using the metric : $(H_{\alpha\beta})^n = (H_{\alpha\gamma}S^{\gamma\delta})^n S_{\delta\beta}$. In practice we can do this in one of two ways, either invert the metric / overlap matrix $S^{\alpha\beta} = (S_{\alpha\beta})^{-1}$ and multiply by the Hamiltonian to give a Hamiltonian matrix in the natural representation (contra-covariant) or solve an approximate problem in the style of Haydock *et al* [Gibson1993]. We then need to store the mixed contra-covariant Hamiltonian matrix, which we choose to represent in the sparsity pattern suggested by Haydock *et al*. This is the covariant Hamiltonian matrix sparsity pattern by default, but the user may specify an optional argument to effectively increase the interaction regions of this sparsity pattern with respect to the covariant case.

More information on our method may be found in [Aarons2018].

3.4.2 Setting up a Calculation

The most important thing to understand firstly is that this method is probably only sensible for large metallic systems. If your system has a band-gap, it would be wise to explore standard insulating ONETEP (LNV) first. This method should still work, but LNV ought to be much faster.

The calculation should perform EDFT as the electronic minimiser. Please set

```
EDFT : T
```

in your input file. As you will be performing a finite electronic temperature calculation with the Fermi-Dirac occupancy distribution function, it would also be wise to choose an electronic smearing (temperature). The default value is 0.1 eV, which may well be sufficient in your case. Please bear in mind that with FOE enabled, unlike with EDFT with diagonalisation the calculation is likely to be faster with higher electronic temperature. Depending on the property you wish to probe, you may get away with a higher temperature, but it is very unlikely that the smearing should be above 1.0 eV. Set the smearing as

```
EDFT_SMEARING_WIDTH : 0.25 eV .
```

You may also at this point wish to set some EDFT stopping criteria, as the defaults are likely to be too tight for EDFT with FOE. It is possible to tweak the FOE options to get under tight thresholds, but at the expense of reduced performance.

```
EDFT_FREE_ENERGY_THRES : 1e-5 Ha
EDFT_COMMUTATOR_THRES : 1e-4
```

should be quite realistic in most cases and will certainly serve as an adequate starting point.

The number of extra bands parameter in EDFT should always be set to the maximum, which is done by setting:

```
EDFT_EXTRA_BANDS : -1
```

This should be sufficient for the EDFT part of the calculation, next we will need to configure the FOE. Firstly, please enable it in your calculation with

```
FOE : T
```

This alone will perform a FOE calculation as we have described in this document, but unless you have already set a kernel truncation, then your density kernel matrix will be dense and you might as well just run a normal EDFT calculation with diagonalisation. One possible exception to this is if you do not have access to Scalapack or a ONETEP binary compiled against Scalapack, or your Scalapack distribution is poorly optimised or broken. In this case it may still be preferable to avoid the diagonalisation.

To achieve some sparsity in the density kernel matrix, we have various options and strategies.

1. Just to use a geometric kernel truncation:

   Here we enforce a simple sparsity pattern on the density kernel matrix by ensuring that matrix elements resulting from NGWFs on atoms separated by a large enough distance are zero.

   ```
   kernel_cutoff : 20 Ang
   ```

   a larger value will result in a more accurate answer, but also give a less sparse matrix which will have performance implications. This parameter is material-dependent and you are encouraged to play with it to find a happy medium.

2. Use a sparsity pattern inspired by the power series expansions which we use extensively in the FOE methods. The FOE method is based on the idea that we only need a finite number of terms in the power expansion to get sufficient accuracy. In the same spirit, we can truncate the expansion to give a sparsity pattern of the density kernel matrix as a low order power of the Hamiltonian matrix. In ONETEP, this can currently be set to be the squared power by using:

   ```
   H2DENSKERN_SPARSITY : T
   ```

If you opt for the second option, then the quadratic order term in the expansion may not be a sufficiently accurate sparsity pattern to achieve the accuracy you were hoping for, but higher order powers are likely to be too dense to be useful and are not exposed to the user. Instead to reach your desired tolerances, we recommend you consider using the radius multiplier option.

This option extends the effective range of the Hamiltonian matrix (when represented in contra-covarient, mixed form (as described above) and the density kernel matrix is formed from the square of this increased range matrix. The option is given in terms of a multiplier, which multiplies the NGWF radius to produce the resulting sparsity patterns. This option increases accuracy at the expense of performance and it is up to you as a user to find an appropriate value. A starting point may be around 1.2 to 1.5 times:

```
CONTRACOHAM_RADMULT : 1.2
```

It is worth pointing out here that you must also ensure your calculation is converged with respect to plane wave kinetic energy cutoff and NGWF radius, as well as number of NGWFs per atom. If you do opt for Hamiltonian matrix squared sparsity in the density kernel matrix, however, you will find that NGWF radius plays a particularly big role in the accuracy of your calculation. This is because it is effectively the parameter which controls the sparsity of your density kernel in this mode. Increasing the radius multiplier will be cheaper (to an extent) than increasing the NGWF radii, but you should make sure that your NGWF radii are sufficient first!

The final parameter which is important to achieve your tolerances is the μ-tolerance in the chemical potential search. In practice this should be *at least* an order of magnitude smaller than your EDFT energy tolerances. You may find that you can improve the performance of your calculations by adjusting this, but you will find that the gains will be minor, because the number of extra matrix multiplications near to convergence is minimal compared with the number far from the correct chemical potential. This is set as:

```
FOE_MU_TOL : 1e-6 eV
```

Beyond this, there is little else to know, from a user perspective about performing FOE calculations in ONETEP. A general point is that these calculations tend to be expensive. If you are running a metallic system which is big enough to exhibit exploitable sparsity in its density kernel matrix, then you are probably looking at a system with thousands of atoms. In this case you will need to be running on a supercomputer and using a parallel version of ONETEP with as many cores as you can use.

If you are looking for ways to speed up your calculations, look at OpenMP / MPI parallelism rather than just MPI, also try to optimise your convergence criteria to have the lowest values you can get away with while maintaining your desired level of accuracy. You may also want to look at the option of using fixed NGWFs and only optimising the density kernel. If you go down this route, please ensure that you use a multiple ζ set of NGWFs and probably polarisation functions. All of these extra points are really general ONETEP comments and information on them can be found in the other tutorials on the ONETEP website.

3.4.3 An Example

As an example we will set up a single-point energy calculation on bulk Magnesium, using PAW. We'll opt for a decent cut-off energy of 700 eV and the PBE exchange-correlation functional.

```
task singlepoint
cutoff_energy 700 eV
charge 0
xc_functional PBE
paw : T
```

We need to enable EDFT and set it up as we have described:

```
!EDFT
edft : T
edft_smearing_width: 0.5 eV
edft_maxit: 10
```

10 EDFT iterations may be too few, and 0.5 eV smearing may be too hot for production calculations, but for the sake of a demonstration, they speed things up a bit.

We also should opt for as much threading as we can get away with because this will be made use of in every matrix product:

```
!THEADING
threadsmax 8
threadsperfftbox 1
threadsnumfftboxes 8
threadspercellfft 8
threadsnummkl 8
```

For instance, if your supercomputer has two 8 core processors per node, you could opt for 8-way OpenMP maximally, although in practice, you might want to reduce this to 4-way, if you need the MPI ranks.

Now lets save some matrices and information:

```
! properties
 do_properties F

! I/O
 output_detail VERBOSE
 timings_level 1
 write_xyz T
 write_denskern T
 read_denskern F
 write_tightbox_ngwfs T
 read_tightbox_ngwfs F
 write_hamiltonian T
 read_hamiltonian F
```

In practice it is a good idea to write everything, in case you need to continue this calculation, but in the extreme case where your NGWFs and matrices are too big to write to disk (not enough space or it's taking too long) then you can disable this.

Now we can set up NGWF optimisation:

```
! NGWF optimization
 k_zero 2.5
 maxit_ngwf_cg 25
```

and FOE:

```
foe : T
dense_foe : F
H2DENSKERN_SPARSITY : T
```

Before setting up the Mg:

```
%block species
 Mg Mg 12 4 8.0
%endblock species

%block species_atomic_set
Mg "SOLVE"
%endblock species_atomic_set
```

for instance. Or, if you wanted to set maxit_ngwf_cg to be zero and run a fixed NGWF calculation, you could run a multiple ζ + polarisation calculation with:

```
%block species
 Mg Mg 12 13 8.0
%endblock species

 %block species_atomic_set
Mg "SOLVE conf=2s2 2p6 3s2 3p0|P"
%endblock species_atomic_set
```

For more info on this, please see the pseudoatomic solver tutorial. Add the PAW data:

```
%block species_pot
Mg "mg_pbe_v1_abinit.paw"
%endblock species_pot
```

Finally lets add a simulation cell and some atoms:

```
%BLOCK lattice_cart
ang
 13.2119999999999997   0.0000000000000213   0.0000000000000213
  0.0000000000000000  10.8379999999999992   0.0000000000000175
  0.0000000000000000   0.0000000000000000  10.4299999999999997
%ENDBLOCK lattice_cart

%BLOCK positions_abs
ang
Mg          3.30300         1.99300         0.00000
Mg          1.65100         4.70200         0.00000
Mg          3.30300         0.11500         2.60800
Mg          1.65100         2.82500         2.60800
Mg          0.00000         1.99300         0.00000
Mg          4.95400         4.70200         0.00000
Mg          0.00000         0.11500         2.60800
Mg          4.95400         2.82500         2.60800
Mg          3.30300         1.99300         5.21500
Mg          1.65100         4.70200         5.21500
Mg          3.30300         0.11500         7.82300
Mg          1.65100         2.82500         7.82300
Mg          0.00000         1.99300         5.21500
Mg          4.95400         4.70200         5.21500
Mg          0.00000         0.11500         7.82300
Mg          4.95400         2.82500         7.82300
Mg          3.30300         7.41200         0.00000
Mg          1.65100        10.12100         0.00000
Mg          3.30300         5.53400         2.60800
Mg          1.65100         8.24400         2.60800
Mg          0.00000         7.41200         0.00000
Mg          4.95400        10.12100         0.00000
Mg          0.00000         5.53400         2.60800
Mg          4.95400         8.24400         2.60800
Mg          3.30300         7.41200         5.21500
Mg          1.65100        10.12100         5.21500
Mg          3.30300         5.53400         7.82300
Mg          1.65100         8.24400         7.82300
```

(continues on next page)

(continued from previous page)

```
Mg          0.00000         7.41200         5.21500
Mg          4.95400        10.12100         5.21500
Mg          0.00000         5.53400         7.82300
Mg          4.95400         8.24400         7.82300
Mg          9.90900         1.99300         0.00000
Mg          8.25700         4.70200         0.00000
Mg          9.90900         0.11500         2.60800
Mg          8.25700         2.82500         2.60800
Mg          6.60600         1.99300         0.00000
Mg         11.56000         4.70200         0.00000
Mg          6.60600         0.11500         2.60800
Mg         11.56000         2.82500         2.60800
Mg          9.90900         1.99300         5.21500
Mg          8.25700         4.70200         5.21500
Mg          9.90900         0.11500         7.82300
Mg          8.25700         2.82500         7.82300
Mg          6.60600         1.99300         5.21500
Mg         11.56000         4.70200         5.21500
Mg          6.60600         0.11500         7.82300
Mg         11.56000         2.82500         7.82300
Mg          9.90900         7.41200         0.00000
Mg          8.25700        10.12100         0.00000
Mg          9.90900         5.53400         2.60800
Mg          8.25700         8.24400         2.60800
Mg          6.60600         7.41200         0.00000
Mg         11.56000        10.12100         0.00000
Mg          6.60600         5.53400         2.60800
Mg         11.56000         8.24400         2.60800
Mg          9.90900         7.41200         5.21500
Mg          8.25700        10.12100         5.21500
Mg          9.90900         5.53400         7.82300
Mg          8.25700         8.24400         7.82300
Mg          6.60600         7.41200         5.21500
Mg         11.56000        10.12100         5.21500
Mg          6.60600         5.53400         7.82300
Mg         11.56000         8.24400         7.82300
%ENDBLOCK positions_abs
```

[Gibson1993] A. Gibson, R. Haydock and J. P. LaFemina. *Ab initio electronic-structure computations with the recursive method*, Phys. Rev. B. **47**, 9229 (1993)

[Aarons2018] J. Aarons and C.-K. Skylaris. *Electronic annealing Fermi operator expansion for DFT calculations on metallic systems*, J. Chem. Phys. **148**, 074107 (2018)

3.5 Density mixing (Kernel-DIIS)

Author
Álvaro Ruiz Serrano, University of Southampton

3.5.1 Basic principles

This manual describes how to run density mixing (kernel DIIS) calculations in ONETEP. Currently, kernel DIIS is an alternative to the LNV density kernel optimisation in the ONETEP inner loop (where the NGWFs are fixed). Please note that:

- The current code is experimental and might be unstable.
- Kernel DIIS currently can only perform calculations on insulators, where the constraint of idempotency of the density matrix holds (i.e., the Kohn-Sham occupancies are integers, 1 for the valence states and 0 for the conductions states.)
- Kernel DIIS is cubic-scaling, always, even if the density matrix is truncated and localised.
- Use only in the rare occasions where LNV optimisation might not be optimal.

Density mixing is a type of self-consistent cycle used to minimise the total energy of the system. Currently, there are two types of mixing implemented in ONETEP: density kernel mixing or Hamiltonian mixing. In both cases, Hamiltonian diagonalisation is required, which makes the method cubic-scaling. The current implementation includes the linear mixing, ODA mixing [Cances2000], [Cances2001], Pulay mixing [Pulay1980], LiSTi [Wang2011] and LiSTb [Chen2011] mixing schemes. Pulay, LiSTi and LiSTb offer the best performance of all, but they must be used in conjunction with linear mixing in order to obtain numerical stability.

3.5.2 Compilation

By default, ONETEP is linked against the Lapack library [Lapack_web] for linear algebra. The Lapack eigensolver DSYGVX [DSYGVX], can only be executed in one CPU at a time. Therefore, kernel DIIS calculations with Lapack are limited to small systems (a few tens of atoms). Calculations on large systems are possible if, instead, ONETEP is linked against ScaLapack library [Scalapack_web] during compilation time. The ScaLapack eigensolver, PDSYGVX [PDSYGVX], can be run in parallel using many CPUs simultaneously. Moreover, ScaLapack can distribute the storage of dense matrices across many CPUs, thus allowing to increase the total memory allocated to a given calculation in a systematic manner, simply by requesting more processors. For the compilation against ScaLapack to take effect, the flag `-DSCALAPACK` must be specified during the compilation of ONETEP.

3.5.3 Commands for the inner loop

Mixing schemes

The keyword
`kernel_diis_scheme: string` [String, default `kernel_diis_scheme: NONE`]

is used to select the mixing method during the kernel-DIIS loop. By default, this keyword takes the value "`NONE`", which disables kernel DIIS and tells the program to proceed with the LNV optimisation. The following options are available:

- `kernel_diis_scheme: DKN_LINEAR`

linear mixing of density kernels. The new input density kernel is built from the *in* and *out* density kernels of the current iteration as $K_in^{n+1} = (1-\lambda)K_{in}^n + \lambda K_{out}^n$.

- `kernel_diis_scheme: HAM_LINEAR`
  linear mixing of Hamiltonians. The new input Hamiltonian is built from the and *out* Hamiltonians of the current iteration as $H_in^{n+1} = (1-\lambda)H_{in}^n + \lambda H_{out}^n$.
- `kernel_diis_scheme: DKN_PULAY`
  Pulay mixing of density kernels (see Ref. [Pulay1980]). The new input density kernel is built as a linear combination of the *output* density kernels of the N_{mix} previous iterations as $K_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m K_{out}^m$. Pulay mixing requires the storage of N_{mix} matrices.
- `kernel_diis_scheme: HAM_PULAY`
  Pulay mixing of Hamiltonians (see Ref. [Pulay1980]). The new input Hamiltonian is built as a linear combination of the *output* Hamiltonians of the N_{mix} previous iterations as $H_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m H_{out}^m$. Pulay mixing requires the storage of N_{mix} matrices.
- `kernel_diis_scheme: DKN_LISTI`
  LiSTi mixing of density kernels (see Ref. [Wang2011]). The new input density kernel is built as a linear combination of the *output* density kernels of the N_{mix} previous iterations as $K_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m K_{out}^m$. LiSTi mixing requires the storage of $4 \times N_{mix}$ matrices.
- `kernel_diis_scheme: HAM_LISTI`
  LiSTi mixing of Hamiltonians (see Ref. [Wang2011]). The new input Hamiltonian is built as a linear combination of the *output* Hamiltonians of the N_{mix} previous iterations as $H_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m H_{out}^m$. LiSTi requires the storage of $4 \times N_{mix}$ matrices.
- `kernel_diis_scheme: DKN_LISTB`
  LiSTb mixing of density kernels (see Ref. [Chen2011]). The new input density kernel is built as a linear combination of the *output* density kernels of the N_{mix} previous iterations as $K_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m K_{out}^m$. LiSTb mixing requires the storage of $4 \times N_{mix}$ matrices.
- `kernel_diis_scheme: HAM_LISTB`
  LiSTb mixing of Hamiltonians (see Ref. [Chen2011]). The new input Hamiltonian is built as a linear combination of the *output* Hamiltonians of the N_{mix} previous iterations as $H_in^{n+1} = \sum_{m=n-N_mix}^{n} \lambda_m H_{out}^m$. LiSTb mixing requires the storage of $4 \times N_{mix}$ matrices.
- `kernel_diis_scheme: DIAG`
  Hamiltonian diagonalisation only - no mixing takes place. Strongly NOT recommended.

NOTE: linear mixing can be used simultaneously with Pulay, LiSTi or LiSTb mixing to create a history of density kernels/Hamiltonians with optimal numerical properties. See the keywords `kernel_diis_coeff` and `kernel_diis_linear_iter` in the section on basic principles for more information.

Basic setup: controlling the mix

- `kernel_diis_coeff: x` [Real, default `kernel_diis_coeff: 0.1`]. Linear-mixing λ coefficient. Must be in the range $[0, 1]$. If set to a negative value, the ODA method (see Refs. [Cances2000], [Cances2001]) will automatically calculate the optimal value of λ. The value of `kernel_diis_coeff` can be made arbitrarily small in order to attain numerical stability. However, this can make convergence slow. A small value can be used in order to create a history of matrices before Pulay, LiSTi or LiSTb mixing. A value close to 1 will make the calculation potentially unstable.
- `kernel_diis_linear_iter: n` [Integer, default `kernel_diis_linear_iter: 5`]. Number of linear mixing iterations before Pulay, LiSTi or LiSTb mixing. This keyword will create and store a history of `n` previous matrices generated by linear mixing before Pulay, LiSTi or LiSTb mixing begin. Required for large systems in order to achieve stability.

- `kernel_diis_size: n` [Integer, default `kernel_diis_size: 10`]. Number of matrices (N_{mix}) to be mixed with the Pulay, LiSTi or LiSTb schemes.
- `kernel_diis_maxit: n` [Integer, default `kernel_diis_maxit: 25`]. Maximum number of iterations during the density mixing inner loop.

Tolerance thresholds

- `kernel_diis_threshold: x` [Real, default `kernel_diis_threshold: 1.0e-9`]. Numerical convergence threshold for the DIIS inner loop. Use in conjunction with `kernel_diis_conv_criteria`.
- `kernel_diis_conv_criteria: string` [String, default `kernel_diis_conv_criteria: 1000`]. Select the convergence criteria for the DIIS inner loop.`kernel_diis_conv_criteria` takes a string value 4 characters long. Each position acts as a logical switch, and can only take the values "`1`" (on) or "`0`" (off). The order is the following:
  - Position 1: residual $|K_{out} - K_{in}|$, if density kernel mixing, or $|H_{out} - H_{in}|$, if Hamiltonian mixing.
  - Position 2: Hamiltonian-density kernel commutator.
  - Position 3: band-gap variation between two consecutive iterations (in Hartree).
  - Position 4: total energy variation between two consecutive iterations (in Hartree).

  For example, `kernel_diis_conv_thres: 1101` will enable criteria 1,2 and 4 and disable criterion 3.

Advanced setup: level shifter

Extra stability can sometimes be achieved if the conduction energy values are artificially increased. This technique is known as level-shifting. See Ref. [Saunders1973] for further details.

- `kernel_diis_lshift: x units` [Real physical, default `kernel_diis_lshift: 1.0 Hartree`]. Energy shift of the conduction bands.
- `kernel_diis_ls_iter: n` [Integer, default `kernel_diis_ls_iter: 0`]. Total number of DIIS iterations with level-shifting enabled.

3.5.4 Commands for the outer loop

The standard ONETEP commands for NGWF optimisation apply.

3.5.5 Restarting a kernel DIIS calculation

- `write_denskern: T/F` [Boolean, default `write_denskern: F`]. Save the last density matrix on a file.
- `read_denskern: T/F` [Boolean, default `read_denskern: F`]. Read the density kernel matrix from a file, and continue the calculation from this point.
- `write_tightbox_ngwfs: T/F` [Boolean, default `write_tightbox_ngwfs: T`]. Save the last NGWFs on a file.
- `read_tightbox_ngwfs: T/F` [Boolean, default `read_tightbox_ngwfs: F`]. Read the NGWFs from a file and continue the calculation from this point.

  If a calculation is intended to be restarted at some point in the future, then run the calculation with `write_tightbox_ngwfs: T`

`write_denskern: T`

to save the density kernel and the NGWFs on disk. Two new files will be created, with extensions `.dkn` and `.tightbox_ngwfs`, respectively. Then, to restart the calculation, set

`read_tightbox_ngwfs: T`

`read_denskern: T`

to tell ONETEP to read the files that were previously saved on disk. Remember to keep a backup of the output of the first run before restarting the calculation.

3.5.6 Controlling the parallel eigensolver

Currently, only the ScaLapack PDSYGVX parallel eigensolver is available. A complete manual to this routine can be found by following the link in Ref. [PDSYGVX]. If ONETEP is interfaced to ScaLapack, the following directives can be used:

- `eigensolver_orfac: x` [Real, default `eigensolver_orfac: 1.0e-4`]. Precision to which the eigensolver will orthogonalise degenerate Hamiltonian eigenvectors. Set to a negative number to avoid reorthogonalisation with the ScaLapack eigensolver.
- `eigensolver_abstol: x` [Real, default `eigensolver_abstol: 1.0e-9`]. Precision to which the parallel eigensolver will calculate the eigenvalues. Set to a negative number to use ScaLapack defaults.

The abovementioned directives are useful in calculations where the ScaLapack eigensolver fails to orthonormalise the eigenvectors. In such cases, the following error will be printed in the input file:

`(P)DSYGVX in subroutine dense_eigensolve returned info= 2`.

Many times (although not always) this error might cause the calculation to fail.

[Cances2000] E. Cancès and C. Le Bris. Can we outperform the diis approach for electronic structure calculations? Int. J. Quantum Chem., 79(2):82, 2000.

[Cances2001] E. Cances. Self-consistent field algorithms for Kohn–Sham models with fractional occupation numbers. J. Chem. Phys., 114(24):10616, 2001.

[Pulay1980] P. Pulay. Convergence acceleration of iterative sequences - the case of SCF iteration. Chem. Phys. Lett., 73(2):393, 1980.

[Wang2011] Y. A. Wang, C. Y. Yam, Y. K. Chen, and G. Chen. Linear-expansion shooting techniques for accelerating self-consistent field convergence. J. Chem. Phys., 134(24):241103, 2011.

[Chen2011] Y. K. Chen and Y. A. Wang. LISTb: a better direct approach to LIST. J. Chem. Theory Comput., 7(10):3045, 2011.

[Lapack_web] Lapack. http://www.netlib.org/lapack/.

[DSYGVX] Lapack DSYGVX eigensolver. http://netlib.org/lapack/double/dsygvx.f.

[Scalapack_web] ScaLapack. http://www.netlib.org/scalapack/.

[PDSYGVX] ScaLapack PDSYGVX eigensolver. http://www.netlib.org/scalapack/double/pdsygvx.f.

[Saunders1973] V. R. Saunders and I. H. Hillier. Level-shifting method for converging closed shell Hartree-Fock wavefunctions. Int. J. Quantum Chem., 7(4):699, 1973.

3.6 Empirical Dispersion Correction

Author
Max Phipps, University of Southampton

Author
Arihant Bhandari, University of Southampton

Date
November 2020

3.6.1 Theory

Energy

The modification to the total DFT energy when correcting for dispersion is given by,

$$E_{DFT-D} = E_{KS-DFT} + E_{disp} \tag{3.8}$$

where the dispersion energy correction is given by raw-latex:[Vasp], [Hill2009]:

$$E_{disp} = -\frac{1}{2} \cdot s_6 \cdot \sum_{i=1}^{N_{at}} \sum_{j=1}^{N_{at}} \sum_{L}^{*} \frac{C_{6,ij}}{R_{ij}^6} f_{damp}(R_{ij}) g_{smooth}(R_{ij}) \tag{3.9}$$

where:

- s_6 is a DF-dependent global scaling factor,
- $C_{6,ij}$ is a dispersion coefficient for the atom pair ij,
- L denotes all translations of the unit cell within the van der Waals radial cutoff R_{cut},
- $*$ denotes $i \neq j$ in $L = 0$
- R_{ij} is the distance $|\overrightarrow{R_i^0} - \overrightarrow{R_j^L}|$ between atom i in the parent cell $L = 0$ and the atom j in all possible translations L,
- $f_{damp}(R_{ij})$ is a damping function that is unity at large distances and zero at small distances [Hill2009], [Grimme2006],
- $g_{smooth}(R_{ij})$ is smoothening function for truncating the interactions beyond van der Waals radial cutoff.
- The dispersion coefficients and the form of the damping function are dependent upon the empirical vdW correction model adopted. The damping functions for the different models are given below:

Index	Description	Damping function $f_{damp}(R_{ij})$	vdW radii ($R_{0,ij}$)
1	Damping function of Elstner [Elstner2001].	$(1 - \exp(-c_{damp}(R_{ij}/R_{0,ij})^7))^4$	$\frac{{R_{0,i}}^3+{R_{0,j}}^3}{{R_{0,i}}^2+{R_{0,j}}^2}$
2	First damping function of Wu and Yang [Wu2002].	$(1 - \exp(-c_{damp}(R_{ij}/R_{0,ij})^3))^2$	$\frac{{R_{0,i}}^3+{R_{0,j}}^3}{{R_{0,i}}^2+{R_{0,j}}^2}$
3	Second damping function of Wu and Yang [Wu2002].	$\frac{1}{1+\exp(-c_{damp}(R_{ij}/R_{0,ij}-1))}$	$\frac{{R_{0,i}}^3+{R_{0,j}}^3}{{R_{0,i}}^2+{R_{0,j}}^2}$
4	Damping function of D2 correction of Grimme [Grimme2006].	$\frac{1}{1+\exp(-c_{damp}(R_{ij}/R_{0,ij}-1))}$	$R_{0,i} + R_{0,j}$

where c_{damp} is a damping constant (referred to as d within the literature of Grimme [Grimme2006] and c_{damp} by Hill [Hill2009]) and $R_{0,ij}$ is determined by the vdW radii of the atomic pair i and j.

- The smoothening function has the following form for all dispersion models:

$$g_{smooth}(R_{ij}) = 1 - e^{-(R_{ij}-R_{cut})^2}$$

- where, R_{cut} is the radial cutoff for van der Waals interactions, and can be set by the following keyword in the input file:
  `vdw_radial_cutoff: x units` [Real physical, default `vdw_radial_cutoff: 100 bohr`].

Forces

During geometry optimization, the van der Waals forces are calculated from the derivative of the dispersion energy with respect to the ionic coordinates $\{s_i\}$:

$$\frac{\partial E_{disp}}{\partial s_i} = -\frac{1}{2} \cdot s_6 \cdot \sum_{j=1}^{N_{at}} \sum_{L}^{*} \frac{C_{6,ij}}{R_{ij}^6} \left[f(R_{ij}) g'(R_{ij}) + g(R_{ij}) \left(f'(R_{ij}) - \frac{6 f(R_{ij})}{R_{ij}} \right) \right] \frac{\partial R_{ij}}{\partial s_i}$$

3.6.2 Activating the dispersion corrections

Four vdW correction options have been implemented within ONETEP. Activation of the vdW corrections within ONETEP is achieved using the

`DISPERSION`

keyword followed by the dispersion index. eg. For the D2 correction of Grimme [Grimme2006],

```
DISPERSION 4
```

The exchange-correlation functionals available with optimized parameters for the dispersion models are given below:

Index	Keyword	Available XC functionals
1	ELSTNER	BLYP, PBE, PW91, REVPBE, RPBE, XLYP.
2	WUYANG1	BLYP, PBE, PW91, REVPBE, RPBE, XLYP.
3	WUYANG2	BLYP, PBE, PW91, REVPBE, RPBE, XLYP.
4	GRIMMED2	BLYP, PBE, B3LYP.

The $C_{6,ij}$ coefficients and the coefficients of the damping function of dispersion corrections 1–3 are optimized for each xc functional to minimize the root mean square difference of the interaction energy from the literature values for a selection of dispersion-dominant biological complexes from the JSCH-2005 and S22 sets [Jurecka2006]_` and the database of Morgado [Morgado2007]. For the D2 correction of Grimme the parameters are optimized as described in the literature [Grimme2006].

The s_6 parameters are unity for dispersion corrections 1–3 and are fitted by least squares optimization of interaction energy error for 40 noncovalently bound complexes for the D2 correction of Grimme.

In the case of using an unoptimized functional not given within the list, default unoptimized c_{damp}, $C_{6,ij}$) and $R_{0,i}$ values are adopted for the damping functions of Elstner or Wu and Yang as described by Hill [Hill2009] and a default s_6 value of 1.00 is adopted for Grimme's D2 correction model.

3.6.3 Overriding dispersion correction parameters

It is possible to override the default parameters of the dispersion damping functions. This option allows the user to specify parameters for elements and functionals for which values are not given. The atom-dependent variables $C_{6,i}$ (used to calculate $C_{6,ij}$), $R_{0,i}$ (related to the atomic vdW radius of an atom i), and n_{eff} (used in the calculation of $C_{6,ij}$ for all damping functions excluding the D2 correction of Grimme) are modified using the

`vdw_params`

block. This override block applies the parameter changes to atoms by their atomic number (nzatom). eg. To override the dispersion parameters associated with nitrogen,

```
%block vdw_params
! nzatom, c6coeff, radzero, neff
  7       21.1200  2.6200   2.5100
%endblock vdw_params
```

To override the damping constant c_{damp} associated with a damping function, the keyword

`vdw_dcoeff`

followed by the modified damping constant parameter is used. eg.

`vdw_dcoeff 11.0`

3.6.4 Boundary Conditions

The boundary conditions are set by the following keyword:

`vdw_bc P/O P/O P/O`

which accepts a string which should contain three characters (which may be separated by spaces), specifying the BCs along the x, y and z directions of the simulation cell. 'P' for periodic and 'O' for open. In case, the keyword is not specified, BCs set same as

`ion_ion_bc`.

[Vasp] https://www.vasp.at/wiki/index.php/DFT-D2

[Hill2009] Q. Hill and C-K Skylaris, *Proc. R. Soc. A* 465(2103):669-683, 2009

[Grimme2006] S. Grimme, *J. Comput. Chem.* 27(15):1787-1799, 2006

[Elstner2001] M. Elstner, P. Hobza, T. Frauenheim, S. Suhai and E. Kaxiras, *J. Chem. Phys.* 114(12):5149-5155, 2001

[Wu2002] Q. Wu and W. Yang, *J. Chem. Phys.*, 116(2):515-524, 2002

[Jurecka2006] P. Jurečka, J. Šponer, J. Černýa and P. Hobza, *Phys. Chem. Chem. Phys.* 8:1985-1993, 2006

[Morgado2007] C. A. Morgado, J. P. McNamara, I. H. Hillier, N. A. Burton and M. A. Vincent *J. Chem. Theory Comput.* 3(5):1656-1664, 2007

3.7 Using van der Waals Density Functionals

Author
Lampros Andrinopoulos, Imperial College London

Date
November 2012

3.7.1 Activating vdW-DF

The van der Waals energy is calculated in ONETEP using the van der Waals Density Functional method, developed by Dion *et al* [Dion2004].

The only input variable needed to activate the vdW-DF functional is to set `xc_functional VDWDF`. If a `vdW_df_kernel` file is not present in the working directory, it will be automatically generated.

3.7.2 Theory

The form for the exchange-correlation functional proposed by Dion *et al* is:

$$E_{xc} = E_x^{\mathrm{revPBE}} + E_c^{\mathrm{PW92}} + E_c^{\mathrm{nl}} \tag{3.10}$$

where the non-local exchange-correlation energy is given by:

$$E_c^{\mathrm{nl}} = \frac{1}{2}\int\int d\mathbf{r}d\mathbf{r}'\rho(\mathbf{r})\phi(\mathbf{r},\mathbf{r}')\rho(\mathbf{r}') \tag{3.11}$$

where $\rho(\mathbf{r})$ is the electron density at $\mathbf{r}$ and $\phi(\mathbf{r},\mathbf{r}')$ is the nonlocal exchange correlation kernel whose form is explained in [Dion2004].

Non-local correlation energy

The direct calculation of the integral in the form of Eq. (3.11) is very computationally expensive, as it involves a six-dimensional spatial integral.

The algorithm proposed later by Roman-Perez and Soler [Roman-Perez2009] improves the efficiency of the calculation. They observed that with the form used by Dion *et al* for ϕ, the above expression can be re-written as:

$$E_c^{\mathrm{nl}} = \frac{1}{2}\int\int d\mathbf{r}d\mathbf{r}'\rho(\mathbf{r})\phi(q,q',r)\rho(\mathbf{r}')$$

where $r = |\mathbf{r}-\mathbf{r}'|$, and q and q' are the values of a universal function $q_0[\rho(\mathbf{r}),|\nabla\rho(\mathbf{r})|]$ at $\mathbf{r}$ and $\mathbf{r}'$. They thus proposed a way to expand the kernel ϕ using interpolating polynomials $p_\alpha(q)$ for chosen values q_α of q, and tabulated functions $\phi_{\alpha\beta}(r)$ for the kernel corresponding to each pair of interpolating polynomials. The interpolating polynomials p_α are cubic splines that evaluate to a Kronecker delta on each respective interpolating point. A mesh of 20 interpolation points is used in Soler's implementation. The Soler form of the nonlocal energy can be written as:

$$\phi(q_1,q_2,r) = \sum_{\alpha\beta}\phi_{\alpha\beta}(r)p_\alpha(q_1)p_\beta(q_2) \tag{3.12}$$

The universal function $q_0(\mathbf{r})$ is in practice given by:

$$q_0(\mathbf{r}) = \left(1 + \frac{\epsilon_c^{\mathrm{PW92}}}{\epsilon_x^{\mathrm{LDA}}} + \frac{0.8491}{9}\left(\frac{\nabla\rho}{2\rho k_F}\right)^2\right)k_F \tag{3.13}$$

with $k_F = (3\pi^2\rho)^{1/3}$. The quantity q_0 is first "saturated" to limit its maximum value, according to:

$$q_0^{\text{sat}}(\rho, |\nabla\rho|) = q_c\left(1 - \exp\Big(-\sum_{m=1}^{m_c}\frac{(q/q_c)^m}{m}\Big)\right)$$

where q_c is the maximum value of the mesh of q_α.

To evaluate this, we first define a quantity $\theta_\alpha(\mathbf{r}) = \rho(\mathbf{r})p_\alpha(q(\rho(\mathbf{r}), \nabla\rho(\mathbf{r}))$ in real space. In terms of this, Eq. (3.11) can be written as:

$$E_c^{\text{nl}} = \frac{1}{2}\sum_{\alpha\beta}\int\int d\mathbf{r}d\mathbf{r}'\theta_\alpha(\mathbf{r})\theta_\beta(\mathbf{r}')\phi_{\alpha\beta}(r) \tag{3.14}$$

It can be shown that this can be written as a reciprocal space integral:

$$E_c^{\text{nl}} = \frac{1}{2}\sum_{\alpha\beta}\int d\mathbf{k}\theta_\alpha^*(\mathbf{k})\theta_\beta(\mathbf{k})\phi_{\alpha\beta}(k) \tag{3.15}$$

Since the kernel is radially dependent in real space, it is only dependent on the magnitude of the G-vectors, hence the kernel need only be evaluated as a 1-dimensional function $\phi_{\alpha\beta}(k)$ for each α, β.

The kernel ϕ and its second derivatives are tabulated for a specific set of radial points and transformed to reciprocal space. These values are then used to interpolate the kernel at every point $\mathbf{k}$ in reciprocal space required to calculate Eq. (3.15).

Kernel

This section details the evaluation of the NLXC kernel. The kernel $\phi(\mathbf{r}, \mathbf{r}')$ as specified by Dion *et al* [Dion2004] is given by (in atomic units):

$$\phi(\mathbf{r}, \mathbf{r}') = \frac{1}{\pi^2}\int_0^\infty a^2 da\int_0^\infty b^2 dbW(a,b)T(\nu(a),\nu(b),\nu'(a),\nu'(b))$$

where

$$T(w,x,y,z) = \frac{1}{2}\Big[\frac{1}{w+x}+\frac{1}{y+z}\Big]\Big[\frac{1}{(w+y)(x+z)}+\frac{1}{(w+z)(y+x)}\Big],$$

and

$$\begin{aligned}W(a,b) = 2\Big[&(3-a^2)b\cos b\sin a\\ &+(3-b^2)a\cos a\sin b\\ &+(a^2+b^2-3)\sin a\sin b\\ &-3ab\cos a\cos b\Big]/(a^3b^3),\end{aligned}$$

and

$$\nu(y) = 1 - e^{-\gamma y^2/d^2}; \quad \nu'(y) = 1 - e^{-\gamma y^2/d'^2};$$

where $d = |\mathbf{r}-\mathbf{r}'|q_0(\mathbf{r})$, $d' = |\mathbf{r}-\mathbf{r}'|q_0(\mathbf{r}')$

Following this chain of logic, it is clear that this the kernel can in fact be considered as a function only of $|\mathbf{r}-\mathbf{r}'|$, $q_0(\mathbf{r})$ and $q_0(\mathbf{r}')$, since all other variables are dummy variables which are integrated over. The kernel can therefore be written as

$$\phi(r, q_0(\mathbf{r}), q_0(\mathbf{r}')) \tag{3.16}$$

This makes it possible to evaluate the integrals above so as to tabulate the kernel values numerically for a pre-chosen set of radial points and q_0 values.

Non-local potential

Starting from (3.15), one can evaluate the potential $v^{\mathrm{nl}}(\mathbf{r})$ corresponding to this energy, by evaluating all terms in $\partial E_{\mathrm{nl}}/\partial n(\mathbf{r})$. The non-local potential v_i^{nl} at point $\mathbf{r}_i$ on the grid is thus written explicitly in terms of the derivatives of the θ_α quantities with respect to the values ρ_j at all other points on the grid:

$$v_i^{\mathrm{nl}} = \sum_\alpha (u_{\alpha i} \frac{\partial \theta_{\alpha i}}{\partial \rho_i} + \sum_j u_{\alpha j} \frac{\partial \theta_{\alpha j}}{\partial \nabla \rho_j} \frac{\partial \nabla \rho_j}{\partial \rho_i}) \tag{3.17}$$

This makes use of the quantities $u_\alpha(\mathbf{r}) = \sum_\beta \mathcal{F}(\theta_\beta(\mathbf{k})\phi_{\alpha\beta}(k))$: which are already calculated in the evaluation of the energy.

Using the White and Bird [White1994] approach, Eq. (3.17) can be written as:

$$v_{\mathrm{nl}}(\mathbf{r}) = \sum_\alpha \left(u_\alpha(\mathbf{r}) \frac{\partial \theta_\alpha(\mathbf{r})}{\partial \rho(\mathbf{r})} - \int\int i\mathbf{G} \cdot \frac{\nabla\rho(\mathbf{r}')}{|\nabla\rho(\mathbf{r}')|} \frac{\partial \theta_\alpha(\mathbf{r}')}{\partial |\nabla\rho(\mathbf{r}')|} e^{i\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}')} d\mathbf{r} d\mathbf{G} \right) \tag{3.18}$$

For this we need to calculate $\frac{\partial\theta}{\partial\rho}$ and $\frac{\partial\theta}{\partial|\nabla\rho|}$:

$$\begin{aligned}
\frac{\partial \theta_\alpha}{\partial \rho} &= p_\alpha + \rho \frac{\partial p_\alpha}{\partial \rho} \\
&= p_\alpha + \rho \frac{\partial p_\alpha}{\partial q} \frac{\partial q}{\partial \rho} \\
&= p_\alpha + \rho \frac{\partial p_\alpha}{\partial q} \frac{q}{k_F} \frac{\partial k_F}{\partial \rho} + \rho \frac{\partial p_\alpha}{\partial q} k_F \left(\frac{\partial \varepsilon_c}{\partial \rho} \varepsilon_x^{-1} - \varepsilon_c \varepsilon_x^{-2} \frac{\partial \varepsilon_x}{\partial \rho} - \frac{8}{3(3\pi^2)^{2/3}} \frac{Z}{4} (\nabla\rho)^2 \rho^{-11/3} \right) \\
&= p_\alpha + q/3 \frac{\partial p_\alpha}{\partial q} + k_F \rho \frac{\partial p_\alpha}{\partial q} \left(\frac{\partial \varepsilon_c}{\partial \rho} \varepsilon_x^{-1} - \varepsilon_c \varepsilon_x^{-2} \frac{\partial \varepsilon_x}{\partial \rho} - \frac{2Z}{3(3\pi^2)^{2/3}} |\nabla\rho|^2 \rho^{-11/3} \right)
\end{aligned}$$

$$\frac{\partial \theta_\alpha}{\partial |\nabla\rho|} = \rho \frac{\partial p_\alpha}{\partial q} \frac{\partial q}{\partial |\nabla\rho|} = \frac{Z}{2\rho k_F} \rho \frac{\partial p_\alpha}{\partial q} |\nabla\rho| \tag{3.19}$$

Combining Eqs. (3.18), (3.19) and (3.19) gives us the final expression for the nonlocal potential.

3.7.3 Overview of computational algorithm

Module `nlxc_mod`

The main module for the calculation of the non-local energy and potential is `nlxc_mod`. The tabulation of the kernel ϕ is performed only if a kernel file is not found, by `vdwdf_kernel`.

The input required to calculate the non-local energy and potential is essentially just the density and its gradient on the fine grid. The calculation of q and the Fourier transformed θ_α from Eq. (3.15) is performed first, in the routine `nlxc_q0_theta`. The derivatives of the θ_αs with respect to the density and the module of its gradient are calculated on-the-fly in the real-space loop for the calculation of the non-local potential v_{nl} in Eq. (3.17). This is to avoid storing unnecessary arrays. From Eq. (3.18) two transforms are required per α value, a forward FFT, followed by a backward FFT for calculating the non-local potential.

Subroutines to interpolate the polynomials as well as the kernel using cubic splines are used (`spline_interp` and `interpolate`). The interpolating polynomials p_α used are Kronecker deltas, so they take the value 1 on the interpolating point and are zero at the other points.

Module vdwdf_kernel

The kernel $\phi_{\alpha\beta}(k)$ is tabulated for 1024 radial reciprocal space points and 20 q_0 points. Gaussian quadrature is used to calculate Eq. (3.16) and then the result is Fourier transformed. The second derivatives of the kernel are calculated by interpolation, and also tabulated. The default name of the file is `vdw_df_kernel`. The program will first check if this file exists: if it does, it will be loaded in and need not be calculated. If it does not, it will be generated from scratch (which only takes a few minutes) and then it is written out for future re-use in the current working directory.

The format of the `vdw_df_kernel` file is:

```
N_alpha N_radial
max_radius
q_points(:)
kernel(0:N_radial,alpha,beta)
kernel2(0:N_radial,alpha,beta)
```

where `kernel2` is the array of second derivatives of the kernel.

[Dion2004] M. Dion, H. Rydberg, E. Schröder, D. C. Langreth, and B. I. Lundqvist, Phys. Rev. Lett. **92**, 246401 (2004).

[Roman-Perez2009] G. Román-Pérez and J. M. Soler, Phys. Rev. Lett. **103**, 096102 (2009).

[White1994] J. A. White and D. M. Bird, Phys. Rev. B **50**, 4954 (1994).

3.8 Realspace local pseudopotential in ONETEP

Author
Jacek Dziedzic, University of Southampton

Author
Chris-Kriton Skylaris, University of Southampton

3.8.1 Motivation

In standard ONETEP the local pseudopotential is obtained in reciprocal space by a discrete Fourier transform, by assuming the cell is periodically repeated in space. However, there are certain use-cases, where one is interested in the properties of an isolated (not periodically repeated) system. This is especially true if other energy terms, such the Hartree energy or the ion-ion energy are already calculated with open boundary conditions, which is the case, e.g., for implicit solvent calculations in ONETEP.

3.8.2 Theory

Assume that $v_{loc}\left(\vec{r}\right)$ is located on an atom A at a position $\vec{R}_A$ and we want to determine the contribution to the local pseudopotential coming from this atom. Owing to the spherical symmetry of the potential, we have

$$v_{loc,A}\left(\vec{r}\right) = v_{loc}\left(\vec{r} - \vec{R}_A\right) = v_{loc}\left(\left|\vec{r} - \vec{R}_A\right|\right).$$

The local pseudopotential is given to us in terms of its continuous Fourier coefficients, $\tilde{v}_{loc}\left(|\vec{g}|\right)$, read from a recpot file. To generate the pseudopotential at a point $\vec{r}$ in real space, we use the continuous Fourier transform:

$$v_{loc}\left(\vec{r} - \vec{R}_A\right) = \frac{1}{\left(2\pi\right)^3}\int \tilde{v}_{loc}\left(\vec{g}\right)e^{i\vec{g}\cdot\left(\vec{r}-\vec{R}_A\right)}d\vec{g} = \int \tilde{v}_{loc}\left(\vec{g}\right)e^{i\vec{g}\cdot\vec{x}}d\vec{g},$$

where we have set $\vec{x} = \vec{r} - \vec{R}_A$. Expanding the plane wave $e^{i\vec{g}\cdot\vec{x}}$ in terms of localised functions, we get

$$v_{loc,A}\left(\vec{r}\right) = \frac{1}{\left(2\pi\right)^3}\int \tilde{v}_{loc}\left(\vec{g}\right)\cdot\left[4\pi\sum_{l=0}^{\infty}\sum_{m=-l}^{l} i^l j_l\left(gx\right)Z_{lm}\left(\Omega_{\vec{g}}\right)Z_{lm}\left(\Omega_{\vec{x}}\right)d\vec{g}\right],$$

$$v_{loc,A}\left(\vec{r}\right) = \frac{1}{\left(2\pi\right)^3}4\pi\sum_{l=0}^{\infty}\sum_{m=-l}^{l} i^l Z_{lm}\left(\Omega_{\vec{x}}\right)\underbrace{\int \tilde{v}_{loc}\left(\vec{g}\right)j_l\left(gx\right)Z_{lm}\left(\Omega_{\vec{g}}\right)d\vec{g}}_{I_1}. \tag{3.20}$$

The orthogonality of harmonics means that all of the terms, except for that of $l = m = 0$, disappear and, after a change of coordinates ($g^2\sin\theta$ being the Jacobian), we obtain a new expression for the integral in ((3.20)):

$$I_1 = \int_0^{2\pi}\int_0^{\pi} Z_{lm}\left(\Omega_{\vec{g}}\right)Z_{00}\sin\theta\, d\theta\, d\varphi \cdot \int_0^{\infty}\tilde{v}_{loc}\left(g\right)j_l\left(gx\right)g^2 dg.$$

With $Z_{00} = 1/\sqrt{4\pi}$, the double integral simplifies to 1 and we obtain, after realizing that all terms except for $l = 0$ disappear,

$$v_{loc,A}\left(\vec{r}\right) = \frac{1}{\left(2\pi\right)^3}4\pi\int \tilde{v}_{loc}\left(g\right)j_0(gx)\,g^2 dg = \frac{1}{\left(2\pi\right)^3}4\pi\int \tilde{v}_{loc}\left(g\right)\frac{\sin\left(gx\right)}{gx}g^2\, dg.$$

ONETEP uses a convention where an additional factor of 4π is needed when transforming between real and reciprocal space. Thus the final formula for the local pseudopotential at a distance of x from an atom of species s becomes

$$v_{loc}^{s}\left(x\right) = \frac{2}{\pi}\int_0^{\infty}\tilde{v}_{loc}^{s}\left(g\right)\frac{\sin\left(gx\right)}{x}g\, dg. \tag{3.21}$$

3.8.3 Implementation

In practice, however, it is not possible to evaluate the integral (3.21) with ∞ as the upper limit, because $\tilde{v}_{loc}^{s}\left(g\right)$ is defined in the recpot file only up to a g_{max} of 100 Å^{-1}. Furthermore, to ensure the results are consistent with standard ONETEP, we must lower this limit even more, to prevent aliasing, as high g's will not be representable on our reciprocal space grid. Thus, in practice we evaluate

$$v_{loc}^{s}\left(x\right) = \frac{2}{\pi}\int_0^{g_{cut}}\tilde{v}_{loc}^{s}\left(g\right)\frac{\sin\left(gx\right)}{x}g\, dg, \tag{3.22}$$

where $g_{cut} = 2\pi\max\left(d_1, d_2, d_3\right)$ (d_i being the grid spacings of `pub_cell`) and will usually be in the order of 20-30 a_0^{-1}.

The integral is evaluated for x's on a fine radial grid running from 0 to the maximum possible distance, which is the magnitude of the cell diagonal. The calculation is distributed across nodes (each node deals with a portion of the fine radial grid). The total pseudopotential for any point on the real space fine grid is evaluated by interpolation from the fine radial grid and by summing over all atoms. This calculation is distributed across nodes as well (each node deals with its own slabs of the real space fine grid). The default number of points in the radial grid is 100000 and can be changed with the directive `openbc_pspot_finetune_nptsx`.

The integral (3.22) is difficult to evaluate numerically. One source of difficulties is the oscillatory nature of $\sin(gx)$. For larger cells, where the maximum interesting x is in the order of $100\,a_0$, this oscillates so rapidly that the resolution of the recpot file (0.05 Å^{-1}) is not enough and it becomes necessary to interpolate $\tilde{v}^s_{loc}(g)$, and the whole integrand, between the g-points specified in the recpot file. The result of the interpolation is stored on a fine radial g-grid, which is f times as fine as the original radial g-grid of the recpot file. f is determined automatically so that every full period of $\sin(gx)$ is sampled by at least 50 points. For typical cells, this yields f in the order of 5-50, depending on the cell size. Alternatively, f may be specified manually by the `openbc_pspot_finetune_f` directive.

Another difficulty is caused by the singularity in $\tilde{v}^s_{loc}(g)$ as $g \to 0$, where the behaviour of $\tilde{v}^s_{loc}(g)$ approaches that of $-Z_s/g^2$. Although the integral is convergent, this singularity cannot be numerically integrated in an accurate fashion. The singularity also presents problems when interpolating between the g points – the usual cubic interpolation of `services_1d_interpolation` becomes inaccurate at low g's. The second problem is solved by subtracting the Coulombic potential, $-Z_s/g^2$, before interpolation to the fine radial g-grid and then adding it back. The first problem is difficult to treat. An approach where at low g's $\tilde{v}^s_{loc}(g)$ is assumed to be exactly equal to $-Z_s/g^2$ (which allows the low-g part of integral ((3.22)) to be evaluated analytically) gives better results than attempting to numerically integrate the singularity, but is not accurate enough, leading to errors in the order of $50-100\,\mu$Ha in the energy for a hydrogen atom test-case (with a total energy of ca. 0.477Ha. Attempting to fit $A/g^2+B/g+C$ (which also allows analytical integration at low g's) gives similar results. The numerical inaccuracy presents itself as a near-constant shift of the obtained pseudopotential and clearly affects total energy.

To solve this problem, we observe that the local pseudopotential can be split into a long-range part and a short-range part:

$$v^s_{loc}(x) = v^{s(long)}_{loc}(x) + v^{s(short)}_{loc}(x),$$

$$\tilde{v}^s_{loc}(g) = \tilde{v}^{s(long)}_{loc}(g) + \tilde{v}^{s(short)}_{loc}(g).$$

Following [Martyna1999], we observe that $\tilde{v}^{s(long)}_{loc}(g) = \frac{4\pi}{g^2}\exp\left(\frac{-g^2}{4\alpha^2}\right)$ (where α is an adjustable parameter, controllable with `openbc_pspot_finetune_alpha`) which easily transforms to real space to give $v^{s(long)}_{loc}(x) = -\frac{\text{erf}(\alpha x)}{x}$ and is conveniently calculated in real space. The short-range part (corresponding to high g's) is $\tilde{v}^{s(long)}_{loc}(g) = \tilde{v}^s_{loc}(g)\cdot\left[1-\exp\left(\frac{-g^2}{4\alpha^2}\right)\right]$. In this way, the integral ((3.22)) can be rewritten as

$$v^s_{loc}(x) = -\frac{\text{erf}(\alpha x)}{x} + \underbrace{\frac{2}{\pi}\int_0^{g_{cut}} \tilde{v}^s_{loc}(g)\cdot\left[1-exp\left(\frac{-g^2}{4\alpha^2}\right)\right]\cdot\frac{\sin(gx)}{x}g\,dg}_{I_s(x)}. \tag{3.23}$$

Owing to the $\left[1-\exp\left(\frac{-g^2}{4\alpha^2}\right)\right]$ factor, the integral $I_s(x)$ is no longer singular at $g=0$ and can be accurately evaluated numerically, if α is large enough. Small values of α make the numerical integration more difficult (requiring larger values for f), because the oscillations at low g's are large in magnitude. Larger values of α allow for easy integration, but they cause the long-range behaviour to "kick in" earlier. As this long-range behaviour is calculated in real space, it lacks the oscillations that are present in standard ONETEP because of a finite value for g_{cut}. Even though these oscillations are an artifact, obtaining a long-range behaviour that is physically more correct, but without the oscillations, leads to aliasing in reciprocal space and to a departure from the results of standard ONETEP. For this reason we want α to be as small as possible, without negatively impacting the numerical integration. The accuracy of the obtained method can be judged by comparing the real space tail of the obtained pseudopotential with the Coulombic potential. Since we expect the obtained pseudopotential to oscillate slightly around $-Z_s/x$, a good measure of accuracy, which we will

call b, is the average value of $\frac{v_{loc}^{s}(x)-(-Z_s/x)}{-Z_s/x}$ over the tail of the pseudopotential, from, say, $5a_0$ to the maximum x for which $v_{loc}^{s}(x)$ is evaluated. Ideally, b should be zero. Numerical inaccuracies will cause a shift in $v_{loc}^{s}(x)$ which will present itself as a finite, non-zero value of b. Naïve numerical integration by a direct calculation of ((3.22)) led, for our test-case, to b in the order of 0.01, which can be reduced by an order of magnitude by using a very fine radial g-grid (high value of f). Subtracting out the Coulombic potential and integrating only the difference between $\tilde{v}_{loc}^{s}(g)$ and the Coulombic potential numerically, while integrating the remaining part analytically reduced b to about 0.0005. Application of the proposed formula ((3.23)) yielded $b = 5 \cdot 10^{-8}$ for $\alpha = 0.5/l$ and $b = 3 \cdot 10^{-9}$ for $\alpha = 0.1/l$ with a suitably large f to ease the numerical integration at low g (l is the box length). With the default value for f, the total energy is not sensitive (to more than 0.0001%) to the choice of α, provided it is in a resonable range of $0.1/l - 2/l$. The value of $0.3/l$ was chosen as a default.

The calculation of the realspace local pseudo is implemented in `norm_conserv_pseudo.F90` in the subroutine `pseudo_local_on_grid_openbc` and its internal subroutine `internal_Is_of_x`, which evaluates $I_s(x)$. A typical calculation would use default values for all the parameters. The realspace local pseudo is off by default and is turned on automatically when smeared ions or implicit solvent is in use. It can also be forced to be on (for development tests) by using `openbc_pspot T`.

Table 3.1: Directives controlling the calculation of the realspace local pseudo

Directive	Action	Rationale for use
`openbc_pspot T`	Forces the realspace pseudo to be used	Normally not needed, the realspace pseudo will be turned on when necessary. This directive allows turning it on even though the Hartree potential calculation and Ewald calculation proceed in reciprocal space, which might be useful for certain test calculations. A related directive, `openbc_ion_ion T` may be used in conjuction, to replace Ewald with a direct Coulombic sum.
`openbc_pspot_finetune_f` $value$ [$value$ is an integer.]	Sets the fineness parameter, f, to $value$.	Default value of -1 causes f to be determined automatically. Positive values can be used to increase f to obtain extra accuracy. Decreasing f will reduce accuracy and is not recommended.
`openbc_pspot_finetune_nptsx` $value$ [$value$ is an integer.]	Sets the number of radial grid points (distinct values of x) to $value$.	The default of 100000 should be enough, unless huge boxes are used, where it might make sense to increase it. Decreasing this value is not recommended, as it will impact accuracy.
`openbc_pspot_finetune_alpha` $value$ [$value$ is a real.]	Sets the short-range-long-range crossover parameter α to $value/l$, where l is the maximum dimension of the cell.	A default value of 0.3 should be OK for most applications. Increasing α will reduce the numerical inaccuracy in $I_s(x)$, but will cause the long-range behaviour to lack the oscillations of usual ONETEP and thus increase aliasing. Decreasing α will make $I_s(x)$ inaccurate, which can be helped, to a certain extent, by increasing f.

[Martyna1999] G. J. Martyna and M. E. Tuckerman *J. Chem. Phys.* **110** (1999).

3.9 Solvent and Electrolyte Model

Author
Jacek Dziedzic, University of Southampton

Author
James C. Womack, University of Southampton

Author
Arihant Bhandari, University of Southampton

Author
Gabriel Bramley, University of Southampton

Date
September 2022

This manual pertains to ONETEP versions v6.0.0 and later.
For older versions, see separate documentation on the ONETEP website.
Major changes relative to v6.0.0:

- **Soft-sphere model added in v6.1.1.8**
- **Surface Accessible Volume added in v6.1.3.0**
- **Conjugate gradient solver added in v6.1.3.6**
- **Self-consistent Continuum Solvation (SCCS) model added in v6.1.11.0**
- **Solvation forces in PBC added in v6.1.15.0**
- **Electrolyte forces added in v6.1.15.5**
- **Forces for soft-spheres solvation model added in v6.1.15.9**

WARNING to users of v6.1.3.0 and later.
The method used to calculate the surface area of the dielectric cavity was changed in version 6.1.3.0. The surface area is used to calculate the ΔG_{npol} component of the solvation. The new method is more mathematically consistent, but gives approximately 20% smaller values for the surface area. By default, we use the new method, which means the value of ΔG_{solv} and may not agree with earlier versions. If you need full compatibility with versions before 6.1.3, set `is_apolar_sasa_definition` to `isodensity`.

3.9.1 Overview of capabilities

First and foremost, ONETEP implements the Minimal Parameter Solvent Model (MPSM), first presented in Ref. [Dziedzic2011]. A more detailed description, including guidelines on the choice of parameters, is given in Ref. [Dziedzic2013]. MPSM offers a very accurate treatment of the polar (electrostatic) solvation contribution, and a rather simple, yet still accurate, treatment of the apolar terms: cavitation, dispersion and repulsion. MPSM is based on the Fattebert and Gygi model (later extended by Scherlis *et al.*) [Scherlis2006].

Two other models are available – Fisicaro's soft-sphere model (SSM) [Fisicaro2017], and Andreussi's Self-Consistent Continuum Solvation (SCCS) model [Andreussi2012].

Implicit solvation calculations in ONETEP can be performed, regardless of the choice of model, either in open boundary conditions (OBC) or in periodic boundary conditions (PBC), with OBC assumed by default. The solute can be

immersed in pure solvent (necessitating the solution of the Poisson equation (PE)) or in solvent with electrolyte (leading to the Poisson-Boltzmann equation (PBE)). Solvation energies can be calculated by running a calculation in vacuum first, followed by a calculation in solvent. This can be done either automatically ("auto solvation") or manually. Apart from energy terms due to solvation, ONETEP calculates solvation contributions to forces. It is thus possible to perform in-solvent geometry optimisation and molecular dynamics (with difficulty). Forces are supported both in OBC and PBC calculations since v6.1.15.0. Additional solvent exclusion regions can be specified to keep the solvent from predefined regions of the simulation cell. These currently work only in OBC.

3.9.2 The models

ONETEP includes solvation effects by defining a smooth dielectric cavity around the solute ("solute cavity"). In contrast to PCM-based approaches, the transition from a dielectric permittivity of 1 (in the immediate vicinity of the solute) to the bulk value of the solvent is smooth, rather than discontinuous. Thus, there is no "cavity surface", strictly speaking, but rather a thin region of space where the transition takes place. For MPSM and SCCS the cavity and the transition are defined by a simple function relating the dielectric permittivity at a point, $\epsilon(\vec{r})$, to the electronic density there, $n(\vec{r})$, yielding an isodensity model. For SSM the transition is also smooth, but the cavity is defined by atom-centered overlapping spheres.

ONETEP offers two modes of operation – one, where $n(\vec{r})$ is the *current* electronic density ("the self-consistently updated cavity mode"), and another one, where $n(\vec{r})$ is *fixed*, usually to the converged in-vacuum density ("the fixed cavity mode").

For MPSM the dielectric function $\epsilon(\cdot)$, defined in Ref. [Scherlis2006], Eq. 7, and in Ref. [Dziedzic2011], Eq. 1 depends on two parameters: ρ_0, the electronic density threshold, where the transition takes place; and β, which controls the steepness of the change in ϵ. A physical constant, ϵ_∞, the bulk permittivity of the solvent completes the description

For SCCS the dielectric function is defined in Ref. [Andreussi2012] in Eqs. 41-42. It has two parameters: ρ_{min} and ρ_{max}. A physical constant, ϵ_0, the bulk permittivity of the solvent completes the description. This is the same as ϵ_∞ in the MPSM model, only Ref. [Andreussi2012] chose a different notation.

For SSM the dielectric function is described in the section on the soft sphere cavity model below.

Once a solute cavity is constructed, the Poisson equation is then solved to obtain the potential due to the molecular density in the nonhomogeneous dielectric. A more general case, where electrolyte ions can be added to the solvent (specified via concentrations), requires solving the Poisson-Boltzmann equation.

Solute cavity

In the MPSM and SCCS models the dielectric cavity is determined wholly by the electronic density, freeing the model of any per-species parameters (such as van der Waals radii). Thus, the cavity will change shape every time the electronic density changes. From the physical point of view this is good, since it means the density can respond self-consistently to the polarisation of the dielectric and vice versa. From the computational point of view this is rather inconvenient, because it requires extra terms in the energy gradients (see e.g. Ref. [Scherlis2006], Eqs. 5 and 14). Because these terms tend to vary rapidly over very localised regions of space, their accurate calculation usually requires unreasonably fine grids and becomes prohibitively difficult for larger molecules.

Fixed cavity

One workaround for the above problem, which is straightforward, but introduces an approximation, consists in *fixing* the cavity and not allowing it to change shape. This is realised by performing an in-vacuum calculation first, then restarting the solvated calculation from the converged in-vacuum density, and using this density to generate the cavity that then remains fixed for the duration of the solvated calculation. Both calculations are self-consistent (in the DFT sense), only the cavity is *not* self-consistently updated in the in-solvent calculation.

How good is this approximation? From experience, it yields solvation energies within several percent of the accurate, self-consistent calculation, cf. Ref. [Dziedzic2011]. More specifically, the error in solvation energy is expected to be less than 3-4% percent for charged species and less than 1% for neutral species.

If you can spare the computational resources, it would be good to test it on a representative molecule, by comparing the solvation energy against a calculation with a self-consistently updated cavity.

As the cavity remains fixed, the difficult extra terms no longer need to be calculated, and the memory and CPU requirements are significantly reduced (because the grid does not need to be made finer). It is thus the recommended solution. The fixed cavity mode is activated by `is_dielectric_model FIX_INITIAL`, which is the default setting.

If embedded mean field theory (EMFT) is in use alongside the implicit solvent model, a choice can be made about whether the in-vacuum density is calculated using the standard density kernel (calculated at the lower level of theory), or the EMFT density kernel. In most situations, this should not strongly affect results, especially if the active region in EMFT is far from the edge of the cavity. This can be controlled using the keyword `is_emft_cavity`, which is false by default. This has only been tested for the fixed cavity approach, and not the self-consistent cavity approach. For more information, please see the EMFT documentation.

Self-consistently updated cavity

If one insists on performing calculations with the solute cavity self-consistently responding to changes in density (as in Ref. [Scherlis2006]), this can be achieved by `is_dielectric_model SELF_CONSISTENT`. As mentioned earlier, this is costly, because it almost always requires grids that are finer than the default. The relevant grid ("fine grid") can be made finer by `fine_grid_scale n`, with $n > 2$ (which is the default). Typically one would use 3, you might be able to get away with 2.5, you might need 3.5 or even more. The memory and CPU cost increase with the *cube* of this value, so, for instance, when using `fine_grid_scale 3.5` one would expect the computational cost to increase by a factor of $(3.5/2)^3 \approx 5.36$.

Even when using much finer grids, the additional gradient term due to the self-consistently updated cavity poses numerical difficulties. This is especially true if the changes in the density are rapid. For this reason, even if it is technically possible to run a calculation in solvent *without* a preceding calculation in vacuum, it is not recommended to do so – the initial, dramatic changes in the density will likely prove problematic. It will be much easier to run an in-vacuum calculation to convergence, and to restart a calculation in solvent from there. The auto solvation functionality (see section on this below) makes this easy.

Soft Sphere Cavity Model

In addition to MPSM and SCCS, the soft sphere cavity model of Fisicaro *et al.* (Ref. [Fisicaro2017]) has been implemented to provide a dielectric cavity function closer to the standard per-species parametrisation models. This feature is especially useful when the system under study requires significantly different solvation radii for its constituent species. This contrasts with MPSM, which applies the parameter controlling the dielectric cavity shape (the isodensity contour) globally, which leads to the dielectric cavity being too large/small for particular species for a single input isodensity value.

The dielectric cavity within the soft sphere model is composed of a set of interlocking, atom-centered spheres with radii assigned to each atom. Much like MPSM, the dielectric function for each atom varies smoothly from vacuum to bulk permitivity. The dielectric functions themselves are defined by: i) the soft sphere radius set by default by

Alvarez's database of van der Waals' radii (Ref. [Alvarez2013]) or manually set in the `is_soft_sphere_radii` block. The default radii can be uniformly scaled using `is_soft_sphere_scale`. ii) The steepness of the transition from vacuum to bulk permitivity is controlled by `is_soft_sphere_delta`. To activate the soft sphere cavity model, set `is_dielectric_function` to 'soft_sphere'. By default, `is_soft_sphere_scale` is set to 1.33 and `is_soft_sphere_delta` to 0.5, as determined by minimizing the error of the solvation free energy against empirical data for a set of small neutral, organic molecules. These cavity radii may not give accurate solvation energies for heavier elements/system types, and it is encouraged to perform further parametrization to minimize error with respect to selected experimental data. Forces for the soft sphere cavity model have been implemented.

Apolar terms: cavitation energy

All three models include the apolar cavitation term in the solvent-accessible surface-area (SASA) approximation, thus assuming the cavitation energy to be proportional to the surface area of the cavity, the constant of proportionality being the (actual physical) surface tension of the solvent, γ, and the constant term being zero. The cavitation energy term is calculated and added automatically, unless `is_include_apolar F` is explicitly stated. Surface tension of the solvent has to be specified (otherwise the default for water near room temperature (about 0.074 N/m) will be used). This can be done using `is_solvent_surf_tension`. Keep in mind that the apolar term is *scaled by default* to account for dispersion and repulsion (see section on this below). The scaling is controlled by `is_apolar_scaling_factor`, and the default is *not* unity.

Apolar terms: dispersion-repulsion energy

ONETEP includes a simple, approximate way for modeling solute-solvent dispersion-repulsion apolar energy term. This greatly improves the quality of obtained solvation energies for uncharged molecules, particularly so if they are large. This term is reasonably approximated with the same SASA approach that is used for cavitation, albeit with a smaller, and negative, prefactor. In practice this is most easily achieved by simply scaling the cavitation term down by a constant multiplicative factor. A good scaling factor for MPSM, and presumably for SSM, is 0.281705, which is what ONETEP uses by default (see Ref. [Dziedzic2011] for justification). The keyword controlling this parameter is `is_apolar_scaling_factor` (with the above default), and its argument is a unitless value. For SCCS there are two different parameterisations described in Ref. [Andreussi2012], termed "fit g09" and "fit g03", with the latter usually being more accurate. The corresponding values of the apolar scaling factor are 0.034722222 and 0.159722222.

Apolar terms: solvent sccessible volume (SAV)

The accuracy of the implicit solvent model can be further improved by adding the surface-accessible volume (SAV) to the apolar energy term:

$$\Delta G_{apol} = \tau\gamma S + pV$$

where γ is the physical surface tension (Section on cavitation energy), τ is the apolar scaling factor tuned by `is_apolar_scaling_factor` in the SASA model (Section on dispersion-repulsion energy), and p is the solvent pressure. This method is activated by setting `is_apolar_method` to 'SAV'. We note that the scaling factors τ, p, and (in the case of soft sphere) f, must be tuned to give accurate free energies of solvation compared to the original SASA model. By minimising the mean absolute error (MAE) of ΔG_{solv} with respect to experiment for a small set of neutral molecules, we found the optimum scaling factors for water ($\gamma = 0.07415\ Nm^{-1}$) are:

- Soft Sphere: $f = 1.20$, $\tau = 0.813$ and $p = -0.35\ GPa$
- MPSM: ρ_0=0.00035, $\tau = 0.684$ and $p = -0.35\ GPa$

Currently, these values are only fully optimised for the soft sphere implicit solvent model, but the values provided for the MPSM provide a starting estimate. We note that in this simple model, p does not correspond to the physical pressure of the solvent and acts as a fitting parameter to give optimum values of ΔG_{solv}, meaning it can assume negative values.

Furthermore, if fixed PAOs are used in place of optimised NGWFs, the optimum parameters change significantly. For the QZP basis, best parameters for the soft sphere model and MPSM are:

- Soft Sphere: $f = 1.21$, $\tau = 0.861$ and $p = -0.35\ GPa$
- MPSM: $\rho_0 = 0.00035$, $\tau = 0.785$ and $p = -0.35\ GPa$

Larger or smaller fixed PAO basis sets may require slightly different optimal parameters given the above values were calculated with the QZP basis only.

In summary:

- **polar, cavitation, dispersion and repulstion terms (SASA)**:
  `is_include_apolar T` (default)
  `is_apolar_method SASA` (default)
  `is_apolar_scaling_factor 0.281705` (default)
- **polar, cavitation, dispersion and repulstion terms (SAV)**:
  `is_include_apolar T` (default)
  `is_apolar_method SAV`
  `is_soft_sphere_scale 1.20` (optimised for soft sphere)
  `is_apolar_scaling_factor 0.813` (optimised for soft sphere)
  `is_solvent_pressure -0.35 GPa` (optimised for soft sphere)
- **polar and cavitation terms only**:
  `is_include_apolar T` (default)
  `is_apolar_scaling_factor 1.0`
- **polar term only**:
  `is_include_apolar F`

3.9.3 Practicalities

DL_MG solver

ONETEP uses a multigrid solver to solve the Poisson or Poisson-Boltzmann equation. Currently this is done by interfacing to a solver called DL_MG [Anton2020], [Womack_2018]. DL_MG is distributed with ONETEP and is compiled in by default. If your version does not include DL_MG your calculation will stop with a descriptive error message.

Solving the P(B)E is a memory- and time-consuming process, and you should expect solvation calculations to take about 2-3 times longer compared to standard ONETEP (also remembering that you will likely have to run two calculations per result – one in vacuum, and one in solvent). The memory requirement of the solver grows linearly with the volume of the system, meaning that padding with vacuum or with bulk solvent is not free, in contrast to calculations not employing the multigrid solver.

The solver uses a multigrid approach to solve the P(B)E to second order. To ensure the high-order accuracy necessary for solvation calculations, the solver then applies a high-order defect correction technique, which iteratively corrects the initial solution to a higher order. Consult Ref. [Dziedzic2013] for more information on the defect correction approach used in DL_MG.

Grid sizes

Under OBC

One limitation of DL_MG is that the grid sizes it uses are not created equal. Good grid sizes are divisible many times into grids twice as small. For example a grid with 161 points (and so 160 grid-edges in between them) is an excellent choice, since it divides into two grids with 81 points (160 splits into two 80's), these divide into two grids with 41 points, which in turn divide into two grids with 21 points, which divide into two grids with 11 points and so on. This lets the solver use many multigrid levels, increasing efficiency. For contrast, consider a grid with 174 points (and so 173 grid-edges). 173 is prime, and this grid cannot be subdivided at all, making it a poor choice.

Knowing about these limitations, ONETEP will sometimes slightly reduce (truncate) your fine grid dimensions when passing data to and from the multigrid solver. This truncation always affects the right-hand side of the grid, and by default between 1 and 7 grid lengths will be truncated, to give DL_MG enough flexibility. This is done automatically, and you will be informed about the details like this:

```
ONETEP fine grid is 126 x 126 x 126 gridpoints, 29.0000 x 29.0000 x 29.0000 bohr.
FD multigrid is     121 x 121 x 121 gridpoints, 27.8492 x 27.8492 x 27.8492 bohr.
```

Here, ONETEP discarded three slabs, each just over 1 a_0 thick, from your system, at the highest values of x, y, and z.

Even though this is done automatically, it is your responsibility to ensure that nothing of significance (read: any charge density) is in the margin that is thrown away. If any of your NGWFs extend into the margin, your calculation will be meaningless (and will likely stop with an error). Due to *Fourier ringing*, tails of very small, but nonzero charge density extend in all Cartesian directions from your system, even outside the localisation spheres of the NGWFs. It is thus good practice to pad your system with a little vacuum in all directions, say 10 a_0. This is in addition to the margin lost due to truncation.

Under PBC

Under PBC, the grid used by the multigrid solver must have the same dimensions as the simulation cell. This is necessary to ensure that the solution from the solver has the correct periodicity. The approach of truncating the grid (Section on grid sizes under OBC) to obtain a grid which satisfies the grid size constraints of the multigrid solver cannot therefore be used in periodic BCs. Instead, an appropriately sized grid for use by the multigrid solver is obtained by *scaling* ONETEP's fine grid, changing the number and spacing of grid points, while maintaining the same physical dimensions. This corresponds to slightly increasing the scale factor for the fine grid (used, among other things, for multigrid operations) with respect to the standard grid (determined by the kinetic energy cutoff) along each coordinate direction to ensure that the dimensions of the fine grid satisfy the requirements of the solver (see Ref. [Anton2020] for details about these requirements).

This is done automatically, and you will be informed about the details like this:

```
Grid scale modified to satisfy multigrid solver grid constraints
         Grid scale values after modification:         2.09  2.09  2.00
*******************************************************************************
[...]
ONETEP fine grid is 136 x 136 x 240 gridpoints, 32.4219 x 32.7579 x 60.0000 bohr.
FD multigrid is     136 x 136 x 240 gridpoints, 32.4219 x 32.7579 x 60.0000 bohr.
```

Here, the grid was scaled by a factor of 2.09 along the x and y coordinates, and no scaling was necessary for the z coordinate. The two grids (ONETEP fine grid and the grid seen by DL_MG) are identical.

Changing the fine grid scale factor causes ONETEP to use the modified fine grid throughout the calculation (not only when invoking the multigrid solver). This has the unfortunate consequence that ONETEP must perform additional work to interpolate and filter between the fine grid and a slightly smaller "double grid", which is used during other

parts of a ONETEP calculation. Normally this is avoided by making the fine and double grids the same size, but is no longer possible when the fine grid is modified for multigrid operations in PBCs.

Auto solvation

An in-solvent calculation is almost universally preceded by a calculation in vacuum. In the fixed cavity mode this is necessary to generate the cavity from a converged in-vacuum calculation. In the self-consistently updated cavity mode this helps mitigate stability issues associated with the cavity updates (cf. Section on self-consistently updated cavity). To make the procedure easier for users, ONETEP provides what is known as "auto solvation" – a mode of operation, where the two calculations (in vacuum and in solvent) are automatically run in sequence.

To enable auto solvation (which is *off* by default), use `is_auto_solvation T`. This will automatically run an in-vacuum calculation, followed by a calculation in solvent. Some input parameters might have to be adjusted along the way, but this will happen automatically and you will always be informed when this happens. Once the calculation in solvent completes, a detailed breakdown of the energies will be printed. It will look something like this:

```
Individual components of total energy in solvent:     hartree           kcal/mol
- Usual non-electrostatic DFT terms:       -26.28930636174560      -16496.788451
- Electrostatic fixed charge energy:         3.05443104938460        1916.684380
- Apolar cavitation energy:                  0.02080496905999          13.055315
- Apolar dispersion-repulsion energy:       -0.01495721238146          -9.385792
 ------------------------------------------------------------------------------
- Total energy in solvent:                 -23.22902755568246      -14576.434548
```

The above shows a breakdown of the total energy in solvent into the usual DFT terms (except for electrostatic energy), the electrostatic energy, the apolar cavitation energy and the apolar dispersion-repulsion energy.

```
Components of total energy in solvent:                hartree           kcal/mol
- Usual non-electrostatic DFT terms:       -26.28930636174560      -16496.788451
- Electrostatic energy:                      3.05443104938460        1916.684380
- Apolar energy terms:                       0.00584775667854           3.669523
 ------------------------------------------------------------------------------
- Total energy in solvent:                 -23.22902755568246      -14576.434548
```

In the above all the apolar terms have been summed together for convenience.

```
Calculation of free energy of solvation:              hartree           kcal/mol
- Total energy in solvent:          (+)     -23.22902755568246      -14576.434548
- Total energy in vacuum:           (-)     -23.20990671966879      -14564.436043
 ------------------------------------------------------------------------------
- Total free energy of solvation:            -0.01912083601367         -11.998505
```

The above is a direct calculation of the free energy of solvation as a difference of the in-solvent and in-vacuum energies.

```
Components of polar term in f.e. of solvation:        hartree           kcal/mol
- Electrostatic:                              -0.06759943752720         -42.419287
- Change in nonelectrostatic DFT terms:        0.04263084483500          26.751258
 ------------------------------------------------------------------------------
- Polar term in f.e. of solvation:            -0.02496859269221         -15.668028
```

The above is the calculation of the polar term to solvation, as a sum of the change in electrostatic energy between in-solvent and in-vacuum and the change in the remaining DFT terms.

```
Components of free energy of solvation:               hartree           kcal/mol
- Polar term in f.e. of solvation: (+)        -0.02496859269221         -15.668028
- Apolar (cavitation, dis., rep.): (+)         0.00584775667854           3.669523
 ------------------------------------------------------------------------------
- Total free energy of solvation:             -0.01912083601367         -11.998505
```

Finally, the total free energy of solvation is calculated as the sum of the polar and apolar terms calculated earlier. This is usually what you are after.

Auto solvation relies on restart files to achieve a seamless transition from the calculation in vacuum to the calculation in solvent. A `.vacuum_dkn` and a `.vacuum_tightbox_ngwfs` file will be written to disk once the calculation in vacuum is completed (and also earlier, if you used `write_denskern T` and/or `write_tightbox_ngwfs T`). These files are then read at the beginning of the calculation in solvent. This makes restarting in-solvent geometry optimisation and molecular dynamics runs very tricky – this is not recommended in practice. Please ensure such calculations run to completion without manual restarts.

Manual solvation and restarts

Occasionally you might want to run a calculation in solvent without automatically running a calculation in vacuum first. Perhaps you already have the calculation in vacuum and you prefer to manually restart it in solvent. This is known as "manual solvation". To activate it, use `is_implicit_solvent T` (the default is `F`), and make sure to have `is_auto_solvation F` (which is the default).

Make sure you know how the solute cavity is generated in this case. If this is a fresh calculation (not a restart), the cavity will be generated from the initial guess density. This is probably not what you want. In the fixed cavity mode, this will mean that you will be stuck with a cavity that is not very realistic (coming from a guess). In the self-consistently updated cavity mode, the cavity will adapt to the subsequent changes to the density, but the initial, dramatic changes might make this numerically unstable. Therefore, restarting from a converged in-vacuum run is recommended instead.

If you ran an in-vacuum calculation to convergence earlier and you have the requisite restart files, you can add `read_denskern T` and `read_tightbox_ngwfs T` to your input to effect a restart. ONETEP will look for a `.dkn` and a `.tightbox_ngwfs` file. The cavity will be constructed from the density generated from these files, and the

calculation will also proceed from this DKN and NGWFs. If the in-vacuum calculation you ran earlier was a part of an auto-solvation calculation, you will need to rename or link the `.vacuum_dkn` and `.vacuum_tightbox_ngwfs` files to their `.dkn` and `.tightbox_ngwfs` counterparts.

If you need to restart an auto-solvation calculation which stopped in the middle of the in-vacuum calculation, you can set the `is_restart_vac_from_vac` to `T`. This allows you to restart the in-vacuum calculation from the `.vacuum_dkn` and `.vacuum_tightbox_ngwfs` files.

Finally, if you want to restart an in-solvent calculation from an unfinished in-solvent calculation, you have to be careful. This is because you want the calculation to continue from the partially-converged in-solvent density, while still constructing the cavity from the converged in-vacuum density. To do this, use the `is_separate_restart_files` keyword. Setting it to `T` (the default is `F`) will instruct ONETEP to construct the solute cavity from the `.vacuum_dkn` and `.vacuum_tightbox_ngwfs` files, while the density for continuing the calculation will be generated from the `.dkn` and `.tightbox_ngwfs` files.

Solvation in PBC

Implicit solvation operates under OBC by default. However, ONETEP allows solvation calculations in PBC, with some caveats. Only fully periodic BCs are supported, i.e. where the system is periodic along all simulation cell directions. Support for mixed BCs (where OBC are applied along some directions and PBC along others) is planned for the future, but is not currently supported. If you intend to solvate slabs, surfaces or wires, you would probably be best off using PBC and suitable padding.

Boundary conditions can be specified individually for the multigrid solver, local pseudopotential, ion-ion interaction and the smeared ion representation using the following keywords:

- `multigrid_bc`,
- `pspot_bc`,
- `ion_ion_bc`,
- `smeared_ion_bc`.

Each of these keywords accepts a string which should contain three characters (which may be separated by spaces), specifying the BCs along the x, y and z directions of the simulation cell. For `multigrid_bc` the characters may be `0`, `P` or `Z`, corresponding to open (Coulombic), periodic and zero BCs, respectively. "Zero" BCs are open BCs, but with the potential set to zero at the boundary, rather than approximately computed. For `pspot_bc`, `ion_ion_bc` and `smeared_ion_bc`, the values can be `0` or `P`, defined as for `multigrid_bc`.

These keywords allow for flexible selection of mixtures of open and periodic BCs, but currently only fully open and fully periodic BCs are supported, corresponding to values of `0 0 0` and `P P P` (and `Z Z Z` to use zero BCs in the multigrid solver).

Key points on using the BC keywords

- If `multigrid_bc` is set in an input file, but the implicit solvent model is not activated (e.g. if other solvent model keywords are not used) then the multigrid solver is used to compute the Hartree potential in vacuum, without the smeared ion representation.
- Setting `smeared_ion_bc` is insufficient to activate the smeared ion representation—you must also set `is_smeared_ion_rep` (or use the full solvent model, e.g. via `is_implicit_solvent`).
- If BCs are not explicitly set using `multigrid_bc` and the multigrid solver is activated (for example, by setting `is_implicit_solvent: T`), then the BCs for the multigrid solver default to fully open BCs.
- If BCs are not explicitly set using `pspot_bc` then the BCs for the local pseudopotential are determined by the type of calculation being performed and should respect previous defaults. Setting `openbc_pspot: T` will set

fully open BCs, as will setting `is_implicit_solvent: T` or `is_smeared_ion_rep: T`. In vacuum (without smeared ions) the local pseudopotential defaults to fully periodic BCs, unless the cutoff Coulomb approach is used, in which case the BCs are determined by the value of the `coulomb_cutoff_type` keyword.

- If BCs are not explicitly set using `ion_ion_bc` then the BCs for the ion-ion interaction are determined by the type of calculation being performed and should respect previous defaults. Setting `openbc_ion_ion: T` will set fully open BCs, as will setting `is_implicit_solvent: T` or `is_smeared_ion_rep: T`. In vacuum (without smeared ions) the ion-ion interaction defaults to fully periodic BCs, but this can be changed (as normal) by using the cutoff Coulomb or Martyna-Tuckerman approaches.
- If `smeared_ion_bc` is not explicitly set, then the BCs used for smeared ions are the same as those used for the multigrid solver (with the exception that zero BCs for the multigrid solver are converted to open BCs for smeared ions).
- It is possible to specify inconsistent BCs for different interaction terms. A warning should be output if this is detected, but care is necessary to avoid unphysical results.

In short: the boundary conditions selected by default in previous versions of ONETEP should be respected if the new keywords are not explicitly set. If the keywords are set, then care must be taken to ensure that they are set consistently in order to obtain physically realistic results. An effort has been made to prevent inconsistencies between the setting of the new keywords for controlling BCs and earlier keywords (such as `openbc_hartree`, `openbc_pspot` and `openbc_ion_ion`), but this has not been extensively tested.

Smeared ions

The P(B)E is almost always solved for the molecular (total) density, because we are interested in how the solvent polarises in response to the total (valence electronic + core) charge density. The solution is the molecular potential, and not the electronic potential. To reconcile this with the usual DFT way of thinking in terms of valence-electronic and core densities and potentials separately (which is needed e.g. in the calculation of the NGWF gradient), a numerical trick known as the smeared-ion formalism is used. In this formalism ionic cores are modelled by narrow positive Gaussian distributions and the usual energy terms are re-cast (cf. Ref. [Dziedzic2013], Appendix):

- the usual Hartree energy is now replaced by the "molecular Hartree energy" (also called electrostatic energy), that is, the electrostatic energy of the molecule's total charge distribution in the potential this charge distribution generates, in the presence of dielectric;
- the local pseudopotential energy is corrected by an extra term that takes the smeared-ion nature of the cores into account;
- a self-interaction correction term is added to the total energy to account for the added Gaussian distributions (each of them self-interacts). This term does not depend on the electronic degrees of freedom, but depends on the ionic positions;
- a non-self-interaction correction term is added to the total energy to account for the added Gaussian distributions (they interact with each other). This term does not depend on the electronic degrees of freedom, but depends on the ionic positions.

In principle, the total energy of the system is unchanged by the application of the smeared-ion formalism, however, due to minor numerical inaccuracies some discrepancies may be observed. These cancel out when calculating energy differences between solvated and *in vacuo* systems, **provided the smeared-ion formalism is used for the vacuum calculation as well**. There is one parameter to the smeared-ion formalism, σ, which controls the width of the Gaussians placed on the ions. See Ref. [Dziedzic2013] for more details on the choice of this parameter. The default value is almost always OK.

The key takeaway message here is that you need to use smeared ions in **both** the in-vacuum calculation and the in-solvent calculation to ensure the energy expressions are comparable. To do that, add `is_smeared_ion_rep T` to your input file(s). If you forget about this in a solvation calculation (`is_implicit_solvent T`) or if you do auto solvation (`is_auto_solvation T`) it will be added automatically for you, but a warning will be produced. However, if you run

manual solvation, you need to remember to include `is_smeared_ion_rep T` in the in-vacuum calculation – ONETEP has no way of knowing you will follow this with an in-solvent calculation.

Forces

If you ask ONETEP to calculate forces, the force terms due to implicit solvent will be automatically calculated and included. The formulas employed are exact (to numerical accuracy) when a self-consistently updated cavity is used. In practice there will be some noise due to incomplete SCF convergence, but this is at a level comparable with standard ONETEP calculations in vacuum (say, ~0.1%). This is because the Hellmann-Feynman theorem holds only *approximately* due to the calculation being very close to the energy minimum, but not exactly there. Non-SCF forces help, but they too are approximate.

For the case of a fixed cavity, the solvation forces are approximate. The approximation is very good, but initial tests suggest that you might not be able to converge geometries to typical thresholds – although the noise in the forces will be small, it might be enough close to equilibrium to throw off the geometry optimiser. Keep this in mind. You should expect about 1-2% error in the force.

You should be able to do geometry optimisation and molecular dynamics without any problems with implicit solvent, provided that you use `is_auto_solvation T` (not needed, of course, for the soft-sphere model). Note that restarting these might be tricky if they are interrupted during the in-solvent stage – you will need to ensure the correct restart files (the vacuum restart files) are used to generate the solvent cavity upon restart, cf. Section on manual solvation and restarts.

Smeared-ion forces in vacuum are also implemented. These are numerically exact and practically negligible.

Solvation forces work both in OBC and in PBC.

Exclusion regions

This functionality enables excluding regions of space from the solvent. Any excluded region has its dielectric permittivity set to exactly 1, similarly to what happens in core regions (cf. `is_core_width`). This is useful for removing pockets of solvent that could otherwise appear in buried cavities, which are inaccessible to the solvent, yet the electronic density there is low enough to generate a dielectric with a permittivity notably larger than 1.

The regions are specified in a `%block is_dielectric_exclusions`, which looks like this:

```
%block is_dielectric_exclusions
sphere 20.0 22.0 18.0 4.0             ! x, y, z of centre; r (all in a0)
box 13.0 16.0  20.5 29.0  13.0 15.0   ! xmin xmax  ymin ymax  zmin zmax (all in a0)
xcyl 18.4 20.7 7.0                    ! y, z, r (all in a0)
%endblock is_dielectric_exclusions
```

The above excludes the solvent from a sphere centred at $(20, 22, 18)\,a_0$ with a radius of $4\,a_0$, from a box spanning from $(13, 20.5, 13)\,a_0$ to $(16, 29, 15)\,a_0$, and from a cylinder oriented along the X axis, passing through $y = 18.4\,a_0$, $z = 20.7\,a_0$ and a radius of $7\,a_0$. `sphere`, `box`, `xcyl`, `ycyl` and `zcyl` are the only region shapes supported now. All exclusion regions currently assume open boundary conditions **and do not work in PBC**. You can have as many as 10000 regions specified in the exclusion block.

It is crucial to ensure that discontinuities in the permittivity are avoided, because they prevent the solver from converging. Usually, exclusion regions can be chosen such that they merge quite smoothly with regions where the dielectric is naturally 1 (or reasonably close). If this is not possible, then the boundaries of the exclusion regions can be smoothed. This is achieved using a Fermi-Dirac function,

$$\varepsilon(d) = \varepsilon_\infty - \frac{\varepsilon_\infty - 1}{e^{d/d_0} + 1},$$

where d is the distance to the exclusion region boundary (and is negative if inside the exclusion region), and d_0 is the smearing length set by `is_dielectric_exclusions_smear`. By default, this is set to 0 a_0, giving hard-walled exclusion regions ($\varepsilon = 1$ inside and $\varepsilon = \varepsilon_\infty$ outside). But if exclusion regions interface directly with solvent regions, it should be chosen to be at least a couple of times larger than the multigrid spacing, so that the permittivity becomes sufficiently continuous for the solver to converge.

Solvent Polarization

The non-homogeneous Poisson equation:

$$\nabla \cdot (\varepsilon \nabla v) = -4\pi n_{\mathrm{tot}}$$

can be recast in the form of a solvent polarization density:

$$\nabla \cdot \nabla v = -4\pi \left(n_{\mathrm{tot}} + n_{\mathrm{pol}}\right)$$

Subtracting the two, the polarization density is calculated as [Andreussi2012]:

$$n_{\mathrm{pol}} = \frac{1}{4\pi} \nabla \cdot \left[(\varepsilon - 1)\, \nabla v\right]$$
$$n_{\mathrm{pol}} = \frac{1}{4\pi} \left[(\varepsilon - 1)\, \nabla^2 v + \nabla\,(\varepsilon - 1) \cdot \nabla v\right]$$

The polarization potential is calculated by solving the following Poisson eq:

$$\nabla \cdot \nabla v_{\mathrm{pol}} = -4\pi n_{\mathrm{pol}}$$

This is done in properties calculation with `is_solvation_properties T` and 3-D grid data for n_{pol} and v_{pol} is output.

3.9.4 Keywords used in solvation calculations

Basic

- `is_implicit_solvent T/F` turns on/off the implicit solvent. Default is off. Will be set automatically if auto solvation is used.
- `is_include_apolar T/F` turns on/off the apolar energy terms. Default is on.
- `is_apolar_sasa_definition density/isodensity` defines the method used in the difference method which calculates the solvent accessible surface area (SASA) of the dielectric cavity. `density` calculates the SASA by varying the electron density, and `isodensity` uses varying ρ_0 values. `density` is the recommended setting unless backwards compatibility with old versions is desired. Warning can be suppressed by defining this keyword. Default is `density`, but you will get a warning if it is not specified. Specify `density` or `isodensity` explicitly to suppress warnings. Only affects MPSM and SCCS. For SCCS `density` is the only available option.
- `is_apolar_method SASA/SAV` sets the definition of the cavitation term in terms of surface area or surface area with volume. Default is `SASA`.
- `is_apolar_scaling_factor x` controls the scaling of the apolar term with the aim of taking solute-solvent dispersion-repulsion into account. The default is 0.281075, which is good for MPSM, but not necessarily SCCS or SSM.
- `is_smeared_ion_rep T/F` turns on/off the smeared-ion representation. Default is off, but if ONETEP detects you're running a solvation calculation, it will turn it on for you and let you off with a warning. When comparing results of two calculations (e.g. results in vacuum and in solvent), always ensure this is set identically in both calculations.

- `is_density_threshold x` sets the MPSM model parameter ρ_0 to x (atomic units). The default is 0.00035, as per Ref. [Dziedzic2011].
- `is_solvation_beta x` sets the MPSM model parameter β to x (no unit). The default is 1.3, as per Ref. [Dziedzic2011].
- `is_bulk_permittivity x` sets the physical constant – solvent bulk permittivity ϵ_∞ to x (no unit). The default is 78.54 (suitable for water near room temperature and pressure and at low frequencies) if implicit solvent is on, and 1.0 is implicit solvent is off.
- `is_solvent_surf_tension x` sets the physical constant – solvent surface tension γ to x (unit must be supplied). The default is 0.07415 N/m (which is suitable for water near room temperature).
- `is_solvent_pressure x` sets the pressure used to calculate the SAV contribution to the apolar term. Does not correspond to physical water pressure and is optimised to obtain minimal errors with respect to experimental free energies of solvation. Default is -0.35 GPa (which is suitable for water near room temperature).
- `is_dielectric_model FIX_INITIAL/SELF_CONSISTENT` picks either the fixed cavity or the self-consistently updated cavity, as described in the section on the solute cavity.
- `is_auto_solvation x` automatically runs an in-vacuum calculation before any solvation calculation, thus relieving the user from the burden of manually restarting calculations. This attempts to automatically control the directives for restarting, running two calculations (vacuum and solvated) in succession. Using this directive is a must when doing implicit-solvent geometry optimisation, implicit-solvent molecular dynamics, implicit-solvent transition state search or implicit-solvent forcetest. This directive is compatible with conduction calculations.
- `is_dielectric_function FGF/SOFT_SPHERE/ANDREUSSI` Defines the function used to create dielectric cavity. Switches between the charge density based MPSM (`FGF`), the atomic radius-based soft sphere model (`SOFT_SPHERE`) and SCCS (`ANDREUSSI`).
- `is_density_min_threshold x` Only applies to SCCS. Sets the parameter ρ_{min}. The default is 0.0001, which corresponds to the "g03" fit in Ref. [Andreussi2012].
- `is_density_max_threshold x` Only applies to SCCS. Sets the parameter ρ_{max}. The default is 0.0050, which corresponds to the "g03" fit in Ref. [Andreussi2012].
- `is_soft_sphere_scale x` Only applies to SSM. Scales the default Alvarez vdW radii provided in ONETEP. The default is 1.33. This does not apply to radii defined in the `is_soft_sphere_radii` block.
- `is_soft_sphere_delta x` Only applies to SSM. Controls the steepness of the transition from vacuum to the bulk permitivity value. This applies to both default radii and those specified in the `is_soft_sphere_radii` block. The default is 0.5.
- `is_soft_sphere_radii` Only applies to SSM. Block sets the soft sphere radii for species defined. These values are unaffected by the scaling factor defined in `is_soft_sphere_scale`. Undefined species will use the default values defined by Alvarez vdW (Ref. [Alvarez2013]). Units bohr. e.g.

```
%block is_soft_sphere_radii
Li 2.5
Pt 4.6
%endblock is_soft_sphere_radii
```

Advanced

The default settings usually work fine and the advanced settings should only be changed if you know what you're doing.

- `is_bc_coarseness x` changes the size of the blocks into which charge is coarsened when boundary conditions are calculated. The default is 5. Smaller values may subtly increase accuracy, but will incur a computational cost that grows as x^{-3}. This can be perfectly acceptable for smaller molecules. For larger molecules (1000 atoms and more) use 7 or more to reduce computational cost. For the effect of this parameter on accuracy, cf. Ref. [Dziedzic2013].
- `is_bc_surface_coarseness x` changes the size of the surface blocks onto which charge is interpolated when boundary conditions are calculated. The default is 1 and is recommended. Larger values will improve computational cost (that grows as x^{-2}), but may decrease accuracy, especially for charged molecules. If the calculation of BCs becomes a bottleneck, prefer tweaking `is_bc_coarseness x` instead.
- `is_bc_allow_frac_charge T/F` (new in v6.1.1.28) when set to `T`, the calculation of boundary conditions for the multigrid solver will not check if the coarse-grained charge is close to an integer. This can be used in rare cases where you know this is not going to be a problem. The default is `F`.
- `is_separate_restart_files T/F` allows the set of restart files used to construct the solute cavity in solvent to be distinct from the set of restart files used to construct the initial density. This is useful if you need to restart a solvated calculation, but still want to construct the cavity from the converged vacuum density, and not the partially-converged solvated density. See section on manual solvation and restarts.
- `is_restart_vac_from_vac T/F` allows the in-vacuum calculation as part of an auto-solvation calculation to be restarted from `.vacuum_dkn` and `.vacuum_tightbox_ngwfs` files, rather than the usual `.dkn` and `.tightbox_ngwfs` files. See section on manual solvation and restarts.
- `is_solvation_properties T/F` when set to `T` it will produce scalarfields of quantities relevant in solvation during a properties calculation. This is useful for visualising potentials, densities, Boltzmann ion concentrations, electrolyte accessibilities, etc. Ensure you supplied `dx_format T` and/or `cube_format T` and/or `grd_format T`.
- `is_smeared_ion_width x` sets the width of the smeared-ion Gaussians, σ, to x (in units you supply). The default is 0.8 a_0 and should be OK for most calculations. Results should not depend on this parameter, but only if it's within rather narrow limits of sensibility. Too high values (anything larger than 1.0, roughly) are seriously unphysical, as they will lead to cores whose Gaussian tails stick out of the electronic density, especially for hydrogen atoms. This is very bad, since it does not change the energy *in vacuo* (the effect of the smearing, regardless of σ, is cancelled by the correction terms to energy), but changes the energy in solution (by polarising the solvent differently in reality the cores are screened by the electrons). Too low values (anything smaller than 0.6, roughly), on the other hand, will lead to Gaussians so thin and tall that they will become very difficult for the multigrid solver to treat, requiring high orders and unreasonably fine grids to obtain multigrid convergence. See Ref. [Dziedzic2013] for more details.
- `fine_grid_scale x` makes the ONETEP fine grid x (no unit) times as fine as the coarse grid, x does not have to be an integer. The solution of the P(B)E and associated finite-difference operations are performed on the fine grid (or its subset, for OBC). Increasing `fine_grid_scale` allows making this grid finer without unnecessarily increasing the kinetic energy cutoff of the calculation. The default is 2. Memory and computational effort increase with the cube of x.
- `is_dielectric_exclusions_smear x` sets the smearing for dielectric exclusion regions to x (in the units you supply). See section on exclusion regions.
- `is_emft_cavity T/F` if EMFT is enabled at the same time as implicit solvent, this controls whether the solvent cavity is determined using the standard density kernel (at the lower level of theory), or the EMFT kernel. The default is `F`. See EMFT documentation for more details.

Fine control over DL_MG

These keywords enable fine control over the behaviour of the DL_MG solver. See Ref. [Anton2020] for more details, particularly regarding convergence control.

- `mg_use_cg T/F` (new in v6.3.1.6) Turns on the conjugate gradient solver. This generally increases the stability of the solver, but is likely to reduce performance. It might be useful to turn this on if you have problems converging difficult cases – particularly in Poisson-Boltzmann solvation.
- `mg_use_error_damping T/F` can be used to turn on/off error damping in the defect correction procedure. This is often necessary when solving the full (non-linearised) Poisson-Boltzmann equation, but will likely not do much for the linearised Poisson-Boltzmann equation or for the Poisson equation. Accordingly, the default depends on `is_pbe` and is `F` for `is_pbe NONE` and `is_pbe LINEARISED`, and `T` for `is_pbe FULL`.
- `mg_continue_on_error T/F` if `T`, instructs the multigrid solver not to abort if a solution to the P(B)E cannot be converged to desired tolerances, and instead to return an underconverged solution. This can be useful for particularly stubborn cases, especially in Boltzmann solvation. Default is `F` when solving the Poisson equation and `T` if solving the Poisson-Boltzmann equation. If you want to turn it on for Boltzmann solvation, you will very likely need to increase `is_pbe_energy_tolerance` by a very large amount.
- `mg_defco_fd_order x` sets the discretization order used when solving the P(B)E to x (no unit). Available values are 2, 4, 6, 8, 10 and 12, the default is 8. With 2 no defect correction is performed. Values of 4 and above employ defect correction. The lowest values (2 and 4) are not recommended, because they offer poor accuracy. Generally the largest value (12) will offer best accuracy, but this has to be weighed against a likely drop in performance (higher orders often take longer) and possibility of Gibbs-like phenomena that may occur when high orders are used with steeply-changing dielectric permittivity, as is the case for larger values of β. 8 or 10 is a good starting value. Results should not depend on the choice of this parameter, but performance and multigrid convergence will. See the troubleshooting section below for details. See Ref. [Dziedzic2013] for more details.
- `mg_max_iters_vcycle x` sets the maximum number of multigrid V-cycle iterations to x (no unit). The default is 200. See Ref. [Womack2018] for a description of the solver, including the V-cycle scheme employed.
- `mg_max_iters_defco x` sets the maximum number of high-order defect correction iterations to x (no unit). The default is 30. See Ref. [Womack2018] for a description of the solver, including the defect correction procedure.
- `mg_max_iters_newton x` sets the maximum number of Newton method iterations to x (no unit). The default is 30. This is only relevant when solving the non-linear PBE. See Ref. [Womack2018] for a description of the inexact-Newton method employed by the solver in this scenario.
- `mg_max_iters_cg x` (new in v6.3.1.6) sets the maximum number of iterations for conjugate gradients to x (no unit). The default is 50. This is only relevant when `mg_use_cg` is `T`.
- `mg_max_res_ratio x` sets the threshold for the consecutive residual ratio which determines when the multigrid solver gives up (positive real value, no unit, the default is 0.999). This should not require tuning.
- `mg_vcyc_smoother_iter_pre x` sets the number of V-cycle smoother iterations pre-smoothing (integer, no unit, the default is 2). Difficult systems, particularly in PBCs, might benefit from an increase of this value to 4 or 8.
- `mg_vcyc_smoother_iter_post x` sets the number of V-cycle smoother iterations post-smoothing (integer, no unit, the default is 1). Difficult systems, particularly in PBCs, might benefit from an increase of this value to 4 or 8.
- `mg_tol_res_rel` x Set the relative tolerance in the norm of the residual for the defect correction procedure to x (no units, the default is 1.0e-2).
- `mg_tol_res_abs` x Set the absolute tolerance in the norm of the residual for the defect correction procedure to x (atomic units, the default is 5.0e-2).

- `mg_tol_pot_rel` x Set the relative tolerance in the norm of the potential for the defect correction procedure to x (no units, the default is 1.0e-6).
- `mg_tol_pot_abs` x Set the absolute tolerance in the norm of the potential for the defect correction procedure to x (atomic units, the default is 1.0e-6).
- `mg_tol_vcyc_rel` x Set the relative tolerance for the norm of the residual in multigrid V-cycle iterations to x (no units, the default is 1.0e-8).
- `mg_tol_vcyc_abs` x Set the absolute tolerance for the norm of the residual in multigrid V-cycle iterations to x (atomic units, the default is 1.0e-5).
- `mg_tol_newton_rel` x Set the relative tolerance for the norm of the residual in Newton method iterations to x (only applies when solving the nonlinear PBE, no units, the default is 1.0e-8).
- `mg_tol_newton_abs` x Set the absolute tolerance for the norm of the residual in Newton method iterations to x (only applies when solving the nonlinear PBE, atomic units, the default is 1.0e-5).
- `mg_tol_cg_res_rel` x (new in v6.3.1.6) Set the relative tolerance in the norm of the residual for the conjugate gradients to x (no units, the default is 1.0e-2). This is only relevant when `mg_use_cg` is `T`.
- `mg_tol_cg_res_abs` x (new in v6.3.1.6) Set the absolute tolerance in the norm of the residual for the conjugate gradients to x (atomic units, the default is 5.0e-2). This is only relevant when `mg_use_cg` is `T`.

Expert

These will only be listed here and not discussed. The last three keywords are discussed in a separate document devoted to the real space local pseudopotential (see ONETEP website).

- `mg_granularity_power`,
- `is_surface_thickness`,
- `is_bc_threshold`,
- `is_core_width`,
- `is_check_solv_energy_grad`,
- `openbc_pspot_finetune_nptsx`,
- `openbc_pspot_finetune_f`,
- `openbc_pspot_finetune_alpha`.

3.9.5 Boltzmann solvation (solute with electrolyte)

ONETEP has the ability to perform Poisson-Boltzmann implicit solvent calculations, that is, to include electrolyte in the implicit solvent. The electrolyte is represented by point particles ("Boltzmann ions"), which interact with one another only in the mean-field sense, and affect the reaction field, providing a rudimentary model of screening. The model is described in Ref. [Dziedzic2020] and Ref. [Bhandari2020]. Users would be well-advised to read these first.

Boltzmann solvation calculations in ONETEP can be performed in OBC and in PBC alike. In PBC care must be taken to suitably neutralise the simulation cell so that the electrostatic energy does not diverge. ONETEP offers a number of schemes to achieve this, including a novel NECS scheme – see `is_pbe_neutralisation_scheme` in the section on keywords controlling Boltzmann solvation and carefully read Ref. [Bhandari2020].

The inclusion of the electrolyte leads to the well-known nonlinear Poisson-Boltzmann equation. ONETEP (or rather DL_MG) can solve this equation as is, or the linearised approximation can be used – see `is_pbe` in the section on Boltzmann solvation keywords.

Because Boltzmann ions are point particles, they tend to concentrate in the immediate vicinity of the solute, often reaching unphysical concentrations. A number of ways have been proposed to address this problem. ONETEP implements a steric potential approach to keep the Boltzmann ions sufficiently far from the solute – see Ref. [Dziedzic2020] and `is_steric_pot_type` in the section on Boltzmann solvation keywords for a description.

In the presence of the electrolyte a number of additional terms appear in the grand potential. These are clearly listed in the output if auto solvation is used:

```
Individual components of total energy in solvent:     hartree           kcal/mol
- Usual non-electrostatic DFT terms:       -26.31256445391999      -16511.383124
- Electrostatic fixed charge energy:         3.11407548141888        1954.111825
- Electrostatic mobile charge energy:       -0.00000610048106          -0.003828
- Accessibility (steric) correction:         0.00000440501188           0.002764
- Osmotic pressure contribution:            -0.00045050433991          -0.282696
- Ionic atmosph. rearrangement entropy:     -0.00082199171218          -0.515808
- Chemical potential contribution:           0.00082978766243           0.520700
- Apolar cavitation energy:                  0.02130850899275          13.371291
- Apolar dispersion-repulsion energy:       -0.01531921982762          -9.612955
 ---------------------------------------------------------------------------
- Total energy in solvent:                 -23.19294408719482      -14553.791830
```

Similarly, the calculation of the free energy of solvation will include additional terms due to the electrolyte:

```
Components of free energy of solvation:              hartree           kcal/mol
- Polar term in f.e. of solvation: (+)      -0.04757925126662         -29.856430
- Apolar (cavitation, dis., rep.): (+)       0.00598928916514           3.758336
- Non-es. electrolyte terms:       (+)      -0.00044440385885          -0.278868
- Energy of pure electrolyte:      (-)      -0.00048258128208          -0.302824
 ---------------------------------------------------------------------------
- Total free energy of solvation:           -0.04154568419718         -26.070310
```

If you chose not to use auto solvation, you will have to rely on the ENERGY COMPONENTS table to find the individual terms, while the energy of pure electrolyte will be printed out for you at the end of the in-solvent calculation.

Forces

Forces are implemented for smooth hard core steric potential, as this is the only steric potential which is well differentiable. The force term due to implicit electrolyte are calculated and included by default. The formulas employed are exact (to numerical accuracy) at self-consistency. In practice there will be some noise due to incomplete SCF convergence, but this is at a level comparable with standard ONETEP calculations in vacuum (say, ~0.1%). This is because the Hellmann-Feynman theorem holds only *approximately* due to the calculation being very close to the energy minimum, but not exactly there. Non-SCF forces help, but they too are approximate. You should be able to do geometry optimisation and molecular dynamics without any problems with implicit electrolyte. Electrolyte forces work both in OBC and in PBC. Tests have shown that the magnitude of these forces is quite small as compared to other force contributions.

Keywords controlling Boltzmann solvation

The following keywords control the Poisson-Boltzmann implicit solvation functionality, which allows performing calculations in implicit solvent containing electrolyte represented by Boltzmann ions.

- `is_pbe NONE/LINEARISED/FULL` chooses the equation to be solved in implicit solvation. `NONE` chooses the (generalised) Poisson equation, which corresponds to solvation in the absence of an electrolyte. `LINEARISED` chooses the linearised Poisson-Boltzmann equation (LPBE), which is a simplified treatment of electrolyte. `FULL` chooses the full, non-linear Poisson-Boltzmann equation (NLPBE), which deals with the electrolyte without the simplifications offered by linearisation. The default is `NONE`.

Except where noted **(*)**, all the below keywords only have an effect if `is_pbe` is *not* `NONE`. Similarly, except where noted **(*)**, all the defaults given below only apply to calculations where `is_pbe` is *not* `NONE`.

- `is_pbe_temperature T` sets the temperature of the Boltzmann ions to `T` (in K). The default is 300 K.
- `is_pbe_bc_debye_screening T/F` includes (`T`) or does not include (`F`) the effect of Debye screening in the calculation of Dirichlet boundary conditions for calculations in solvent. This only has an effect in OBC. With Debye screening an additional multiplicative factor of $\exp\left(-r/\lambda_{\mathrm{D}}\right)$, where λ_{D} is the Debye length, is included in the boundary conditions. This is exact for LPBE and an approximation in NLPBE. Turning off Debye screening will cause ONETEP to use BCs that are appropriate for the case of no electrolyte, which will be unphysical. The default is `T`.
- `is_pbe_exp_cap c` – sets the exponential cap to c (no unit). This is only relevant to `is_pbe FULL`. In solving the NLPBE it is a well-known issue that the exponential factors that appear in certain expressions (e.g. for the Boltzmann ion concentration) are prone to exploding (in the usual floating-point representation) when the value of the argument to the $\exp$ function is large. To retain numerical stability, the arguments to the $\exp$ function are typically capped, i.e. they are not allowed to exceed a predefined constant. The default in ONETEP is 0.0, which means c is set to the default cap in DL_MG, which is currently 50.0. Specifying any value other than 0.0 will cause ONETEP to discard the default provided by DL_MG and to use the user-specified value.
- `is_pbe_neutralisation_scheme scheme` chooses a specified neutralisation scheme.

  (*) This keyword and its defaults can also apply to `is_pbe NONE`. This is only relevant for PBC calculations with non-zero total solute charge. In this scenario the total system charge (solute + electrolyte) must be zero for the electrostatic energy not to diverge. There are many ways of ensuring charge neutrality. ONETEP implements the following:

  - `NONE` ignores charge neutralisation. This is only meaningful for OBC or when the system is charge-neutral. This is the default in OBC.
  - `JELLIUM` applies the common jellium neutralisation, shifting the charge density by its negative average, so that the average density is zero. This is the default for PBC with no electrolyte (`is_pbe NONE`).
  - `ACCESSIBLE_JELLIUM` applies a modified jellium neutralisation (cf. Ref. [Bhandari2020], Sec. 3.3). This is only applicable when `is_pbe` is *not* `NONE`.
  - `COUNTERIONS_AUTO` applies neutralisation by electrolyte concentration shift (NECS) (cf. Ref. [Bhandari2020], Sec. 3.1) with optimal shift parameters (cf. Ref. [Bhandari2020], Eqs. 14 and 15). This is the default in PBC with electrolyte (`is_pbe LINEARISED` or `is_pbe FULL`).
  - `COUNTERIONS_AUTO_LINEAR` applies neutralisation by electrolyte concentration shift (NECS) (cf. Ref. [Bhandari2020], Sec. 3.1) with shift parameters derived from a linear approximation (cf. Ref. [Bhandari2020], Eq. 18).
  - `COUNTERIONS_FIXED` applies neutralisation by electrolyte concentration shift (NECS) (cf. Ref. [Bhandari2020], Sec. 3.1) with shift parameters specified by the user via `%block_sol_ions`.
- `is_pbe_energy_tolerance E` sets the tolerance for the discrepancy between two expressions for the mean-field contribution to the grand potential to E (in units you supply). The two expressions are (A): Ref. [Bhandari2020], Eq. 5, where individual terms are calculated according to Eqs. 10, 12, 13, 15, and 16 *except* for the

fixed electrostatic term (first term in brackets in Eq. 10), which is excluded here; and (B) Ref. [Bhandari2020], Eq. 31 (for PBC) or Eq. 35 (for OBC) *except* for the fixed electrostatic term (first term in brackets in Eq. 10), which is also excluded here. For `is_pbe FULL` we expect the two expressions to be identical (modulo numerical noise). For `is_pbe LINEARISED` we expect the two expressions to be identical to first order (modulo numerical noise). The check is useful for detecting poorly converged solutions of the PBE. The default is 0.01 kcal/mol for `is_pbe FULL`, and 0.05 kcal/mol for `is_pbe LINEARISED`. Normally you should not need to adjust this parameter. However, it might need to be increased, perhaps dramatically, if you set `mg_continue_on_error T`.

- `is_steric_pot_type X/H/M/S` specifies the type of steric potential that will affect Boltzmann ions. This is to prevent them from unphysically concentrating in the immediate vicinity of the solute. The available options are:
  - `X` no steric potential. This is not recommended, except in contrived test cases. This is currently the default, but this might change later.
  - `H` hard-core potential (see below). The hard-core potential is infinite within the radial cutoff r_c, and zero elsewhere. Numerically this is realised by setting the accessibility to $\gamma = 0$ within the radial cutoff, and to $\gamma = 1$ elsewhere. This choice can pose numerical difficulties because of the infinite steepness, and is not recommended.
  - `M` smoothed hard-core potential (see below). **This is the recommended choice**. Here the accessibility is defined as $\gamma = \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\,(\frac{r-r_\mathrm{c}}{\sigma})$, with values below 10^{-7} then set to 10^{-90}. The potential, as always, is $-kT\ln\gamma$. Here, r_c is the radial cutoff, σ is the smearing parameter.
  - `S` soft-core potential, that is, a potential of the form $Ar^{-12}\,\mathrm{erf}\,(\alpha r)^{12}$ (see the `is_sc_` keywords below). This potential does not seem to work well (too soft) and is not recommended.

  For both hard-core steric potentials the radial cutoff r_c is determined as a sum of two components: the solvent radial cutoff $r_\mathrm{c}^\mathrm{solvent}$, and the solute radial cutoff $r_\mathrm{c}^\mathrm{solute}$. The solvent radial cutoff is set by the `species_solvent_radius` block, with one value per *solute* (sic!) species. The solute radial cutoff is determined through `is_hc_steric_dens_isovalue` and also depends on the solute species (see below). The solute cutoff is determined for each species separately, by examining the radial valence-electronic density profile coming from the pseudoatomic solver. The radial density is scanned from infinity to zero for a value equal to or larger than n_0 (specified via `is_hc_steric_dens_isovalue`). The radial coordinate of this value is taken as $r_\mathrm{c}^\mathrm{solute}$. If `is_hc_steric_dens_isovalue` is negative, $r_\mathrm{c}^\mathrm{solute} = 0$. Thus, the solute radial cutoff is constant, and does depend on the current electronic density, only on the output of the atomic solver.
- `is_hc_steric_dens_isovalue n_0` sets the density isovalue used to determine $r_\mathrm{c}^\mathrm{solute}$ to n_0 atomic units. The default is 0.003. This only applies to `is_steric_pot_type H/M`.
- `is_hc_steric_smearing \sigma` sets the smearing width for the smoothed hard-core cutoff (`is_steric_pot_type M`) to σ (in units you supply). The default is 0.4 a_0.
- `is_sc_steric_magnitude A` sets the magnitude of the soft-core steric potential to A (in units you supply, dimension: energy $\times$ distance^{12}). Default is negative, to force users not to forget this parameter. This only applies to `is_steric_pot_type S`, which you should not be using anyway.
- `is_sc_steric_smoothing_alpha \alpha` sets the smoothing parameter of the soft-core steric potential to α (in units you supply, dimension: inverse distance). Default is 1.5 ${a_0}^{-1}$. This only applies to `is_steric_pot_type S`, which you should not be using anyway.
- `is_sc_steric_cutoff r_{\textrm{c}}` sets the radial cutoff for the soft-core steric potential to r_c (in units you supply). Since the range of this potential is technically infinite, we truncate it to zero beyond a specified distance, r_c. This only applies to `is_steric_pot_type S`, which you should not be using anyway.
- `is_steric_write T/F` if set to `T`, the steric potential and associated accessibility will be written out as scalarfields when initialised. Ensure you supplied `dx_format T` and/or `cube_format T` and/or `grd_format T`.
- `sol_ions` is a block describing the Boltzmann ions in the system. The format for n Boltzmann ions is as follows:

```
%block sol_ions
ion_species_1 charge_1 conc_1 x_1
ion_species_2 charge_2 conc_2 x_2
...
ion_species_n charge_n conc_n x_n
%endblock sol_ions
```

Here, `ion_species_i` is the name of the species of Boltzmann ion i (which is irrelevant from the physical point of view), `charge_i` is the charge on species i, `conc_i` is the concentration of that species (in mol/L), and `x_i`, which is optional, is a NECS shift parameter for species i, relevant only for `is_pbe_neutralisation_scheme COUNTERIONS_FIXED`. For example, to define a 1M NaCl electrolyte, you would use:

```
%block sol_ions
Na +1 1.0
Cl -1 1.0
%endblock sol_ions
```

- `species_solvent_radius` defines the solvent radial cutoff $r_{\mathrm{c}}^{\mathrm{solvent}}$ for every *solute* (sic!) species, as follows:

```
%block species_solvent_radius
species_1 r_1
species_2 r_2
...
species_n r_n
%endblock species_solvent_radius
```

For example, to keep all the Boltzmann ions an extra $3.5\ a_0$ away from your methane solute, you would use:

```
%block species_solvent_radius
C 3.5
H 3.5
%endblock species_solvent_radius
```

3.9.6 Various hints for a successful start

- Use one of the examples provided on the ONETEP website as a starting point.
- Make sure both your vacuum and solvated calculations use smeared ions.
- Make sure the parameters of both your vacuum and solvated calculations are identical (box sizes, KE cutoffs, `k_zero`, `mg_defco_fd_order`, `is_smeared_ion_width`, `is_bc_coarseness`, `is_bc_surface_coarseness`). Or just use `is_auto_solvation T`.
- Choose `FIX_INITIAL` over `SELF_CONSISTENT` for `is_dielectric_model`.
- Use an `mg_defco_fd_order` of 8 and `is_smeared_ion_width` of 0.8. Specify them explicitly, as the defaults may change in the future.
- Do not mess with expert directives.

- In OBC, have at least about 10 bohr of vacuum/solvent around the edges of your molecule's NGWFs (not atomic positions) on each side of the simulation cell, *after taking the truncation into account* – cf. section on grid sizes under OBC.
- Always start your calculation in solution as a restart from a fully converged *in vacuo* calculation. Or just use `is_auto_solvation T`.

3.9.7 Troubleshooting: Problems, causes and solutions

- **Problem A**: ONETEP crashes (e.g. catching `SIGKILL` or `SIGSEGV`) when evaluating the boundary conditions or solving the P(B)E.

  Cause (A1): You've run out of memory and the OOM killer killed the calculation. Solving the P(B)E often represents the peak memory usage of the calculation.

  Solution (A1): Increase available memory (perhaps by shifting the MPI/OMP balance towards more threads and fewer MPI processes) or decrease box size or decrease grid fineness.

  Cause (A2): You've run out of global stack space. Solving the P(B)E often represents the peak stack usage of the calculation.

  Solution (A2): Increase stack size using `ulimit -s`. Make sure you do that on compute nodes, not the login node. Or, preferably, use `onetep_launcher` and its `-s` parameter. **Cause (A3)**: You've run out of per-thread stack space. Solving the P(B)E often represents the peak per-thread stack usage of the calculation.

  Solution (A3): Increase per-thread stack size using `ulimit -s`. Make sure you do that on compute nodes, not the login node. Or, preferably, use `onetep_launcher` and its `-o` parameter.

- **Problem B**: Multigrid calculation does not converge (error message from DL_MG) or converges very slowly (as evidenced by the contents of a log file with a filename ending in `_dl_mg_log.txt`).

  Cause (B1): (Only applies to OBC calculations) Charge is not correctly localized (cell is too small or molecule otherwise too close to cell edge).

  Solution (B1): Check and fix the cell size, paying attention to the margin between the DL_MG grid and fine grid.

  Cause (B2): Dielectric permittivity too steeply changing on the cavity boundary for the current grid size, finite differences struggling to approximate the changes. This is often the culprit if the calculation ran fine *in vacuo* but struggles in solvent.

  Solution (B2): Preferable, but painful, solution is to make the grid finer (`fine_grid_scale`). Otherwise an increase or decrease of discretisation order may help (make sure it stays consistent across your calculations, though). A parameterisation with lower `is_solvation_beta` and `is_density_threshold` will usually help (make sure it stays consistent across your calculations, though).

  Cause (B3): The smearing width is too small, making the smeared cores too thin and tall, which is difficult for the finite differences. This is often the culprit if the calculation also struggles *in vacuo*.

  Solution (B3): Increasing `is_smeared_ion_width` will help (but mind the consequences), if it was too small in the first place. Increasing the discretisation order will help (especially if you've been using less than 10), but might lead to a similar problem (Cause (B2)) in solution.

  Cause (B4): Too lax thresholds for convergence of the defect correction in DL_MG.

  Solution (B4): To tighten the convergence threshold of the defect correction in DL_MG, adjust the values of `mg_tol_res_rel`, `mg_tol_res_abs`, `mg_tol_pot_rel` and `mg_tol_pot_abs`.

  Cause (B5): Too few defect correction iterations in DL_MG.

  Solution (B5): To increase the number of defect correction iterations in DL_MG, use `mg_max_iters_defco`, try 200 for good measure.

  Cause (B6): Too few smoother iterations in DL_MG, particularly if this is a Boltzmann calculation.

  Solution (B6): Increase the number of smoother iterations in DL_MG to 2 or 4 using `` mg_vcyc_smoother_iter_pre`` and `mg_vcyc_smoother_iter_post`.

  Cause (B7): Too few V-cycle iterations in DL_MG.

Solution (B7): Increase the number of V-cycle iterations in DL_MG using `mg_max_iters_vcyc`, try 200 for good measure.

Cause (B8): Too few Newton iterations in DL_MG. This only applies if you are solving the NLPBE in Boltzmann solvation.

Solution (B8): Increase the number of Newton iterations using `mg_max_iters_newton`, try 100 for good measure.

Cause (B9): Problem is too difficult for the solver – e.g. grids are not fine enough, the dielectric cavity has a steep boundary (usually happens when underconverged densities are used to generate it), Boltzmann-ionic concentrations changing too steeply, etc.

Solution (B9): Try using the conjugate gradient approach – add `mg_use_cg T` to your input file. This is only available in versions v6.1.3.6 and newer.

- **Problem C**: Calculation struggles to converge LNV or NGWFs or does not converge at all. RMS gradient stalls.

  Cause (C1): If you're using `is_dielectric_model SELF_CONSISTENT`, then this is normal, unless your grid is ridiculously fine (you will need `psinc_spacing 0.5` and `fine_grid_scale 3` or better, as a rule of thumb).

  Solution (C1): Use `is_dielectric_model FIX_INITIAL` if possible. If you are sure you need `is_dielectric_model SELF_CONSISTENT`, make the grid finer and have a lot of memory.

  Cause (C2): Density kernel is not converged enough.

  Solution (C2): `Try minit_lnv 6` and `maxit_lnv 6` (for smaller molecules) or `minit_lnv 10` and `maxit_lnv 10` (for large molecules).

3.9.8 Frequently asked questions

What are the values for the model parameters?

Two sets of values for MPSM will be proposed here. The first one will be called high-beta parameterisation. It offers the best quality (in terms of r.m.s. error from experiment) for both charged and neutral species. The drawback is that the high value of β means the multigrid convergence is poor and it often takes a while to converge. Or it may not converge. This should be your first choice **only** if accuracy trumps anything else. The parameters are:

```
is_solvation_beta 1.6
is_density_threshold 0.00055
```

The second parameterisation, called low-beta should pose no problems to the multigrid solver under any circumstances. Quality should be only marginally worse for anions and neutrals and comparable or better for cations. These are the default parameters, and they are:

```
is_solvation_beta 1.3
is_density_threshold 0.00035
```

Both parameterisations assume `is_bulk_permittivity 78.54`, which is suitable for water. It should be noted that the model is deficient in its treatment of anions, consistently underestimating the magnitude of the solvation effect by 10-25%. Work is ongoing to fix this, until then a different parameterisation may be used if one is only interested in anionic species.

Can you do solvents other than water?

Yes, provided you know the dielectric permittivity of the solvent and its surface tension. Accuracy has not been extensively tested, but it should work.

Can you do mixed boundary conditions?

Not yet, but we might in the future.

Is implicit solvation compatible with conduction calculations?

Yes, to the best of our knowledge.

Is implicit solvation compatible with PAW?

Yes, to the best of our knowledge.

3.9.9 Known issues and untested functionality

- PBC are not yet currently compatible with the `is_dielectric_exclusions` block.

3.9.10 Contact

General questions about implicit solvation in ONETEP should be directed to Jacek Dziedzic (`J.Dziedzic[-at-]soton.ac.uk`). Questions regarding Boltzmann (electrolyte) solvation should be directed to Arihant Bhandari (`A.Bhandari[-at-]soton.ac.uk`).

[Dziedzic2011] J. Dziedzic, H. H. Helal, C.-K. Skylaris, A. A. Mostofi, and M. C. Payne, *Minimal parameter implicit solvent model for ab initio electronic-structure calculations*, EPL **95** (2011).

[Dziedzic2013] J. Dziedzic, S. J. Fox, T. Fox, C. S. Tautermann, and C.-K. Skylaris, *Large-Scale DFT Calculations in Implicit Solvent – A Case Study on the T4 Lysozyme L99A/M102Q Protein*, International Journal of Quantum Chemistry **113** issue 6 (2013).

[Scherlis2006] D. A. Scherlis, J.-L. Fattebert, F. Gygi, M. Cococcioni, and N. Marzari, *A unified electrostatic and cavitation model for first-principles molecular dynamics in solution*, J. Chem. Phys. **124** (2006).

[Fisicaro2017] G. Fisicaro, L. Genovese, O. Andreussi, S. Mandal, N. Nair, N. Marzari and S. Goedecker, *Soft-Sphere Continuum Solvation in Electronic-Structure Calculations*, J. Chem. Theory Comput. **13** (2017).

[Alvarez2013] S. Alvarez *A cartography of the van der Waals territories*, Dalton Trans. **42** (2013).

[Anton2020] L. Anton, J. Womack, and J. Dziedzic, *DL_MG multigrid solver* (2020) http://www.dlmg.org

[Womack2018] J. C. Womack, L. Anton, J. Dziedzic, P. J. Hasnip, M. I. J. Probert, and C.-K. Skylaris, J. Chem. Theory Comput. **14**, 1412 (2018).

[Andreussi2012] O. Andreussi, I. Dabo and N. Marzari, *Revised self-consistent continuum solvation in electronic-structure calculations*, J. Chem. Phys. **136** (2012).

[Dziedzic2020] J. Dziedzic, A. Bhandari, L. Anton, C. Peng, J. C. Womack, M. Famili, D. Kramer, and C.-K. Skylaris, *Practical Approach to Large-Scale Electronic Structure Calculations in Electrolyte Solutions via Continuum-Embedded Linear-Scaling Density Functional Theory*, J. Phys. Chem. C **124** (2020).

[Bhandari2020] A. Bhandari, L. Anton, J. Dziedzic, C. Peng, D. Kramer, and C.-K. Skylaris, *Electronic Structure Calculations in Electrolyte Solutions: Methods for Neutralization of Extended Charged Interfaces*, J. Chem. Phys. **153** (2020).

3.10 Spherical wave resolution of identity (SWRI), Hartree-Fock exchange (HFx), hybrid functionals and distributed multipole analysis (DMA)

Author
Jacek Dziedzic, University of Southampton

Author
James C. Womack, University of Southampton

Date
June 2020

This manual pertains to ONETEP versions v5.3.4.0 and later.

3.10.1 The basics

ONETEP offers linear-scaling calculation of Hartree-Fock exchange (HFx) and linear-scaling distributed multipole analysis (DMA) through a variant of density fitting. In this approach products of two NGWFs are approximated (fitted) using an auxiliary basis of spherical waves centred on the atoms on which the two NGWFs reside. Before the expansion can be performed, an overlap matrix between spherical waves on all centres whose NGWFs overlap needs to be calculated. The resultant matrix will be termed the *metric matrix*, and the process through which it is obtained will be termed *spherical wave resolution of identity (SWRI)*. Understanding the keywords controlling SWRI is thus essential for any calculation involving HFx or DMA.

3.10.2 Limitations

All calculations using SWRI have these limitations:

- Open boundary conditions (OBCs) are assumed in all uses of SWRI. If your calculation uses OBCs (via cut-off Coulomb, Martyna-Tuckerman or multigrid [Hine2011]), this is a non-issue. If your calculation technically uses PBCs, but you have sufficient vacuum padding in every direction (effectively making it approximately OBC), this is a non-issue. If your system is truly extended in any direction (at least one NGWF sphere sticks out of the cell and wraps around), this is an issue, as neither HFx nor DMA will work (producing nonsense). This is not checked!
- All species must have identical NGWF radii. Without this simplification the numerical methods used in SWRI would get ugly. Typically this is a non-issue, just increase the NGWF radii to the largest value. This might admittedly be an issue if you have a single species that requires a large NGWF radius. This is checked against and ONETEP will not let you do SWRI unless all NGWF radii are identical.
- Although well-controllable, density fitting is an approximation. Using settings that are too crude can lead to inaccurate results and poor NGWF convergence.
- HFx is incompatible with PAW and ONETEP will refuse to use both simultaneously.
- HFx and DMA are incompatible with complex NGWFs and ONETEP will refuse to use both.
- HFx does not work with atoms that have 0 NGWFs, although this is hardly a limitation.

3.10.3 Spherical wave resolution of identity

Generating the metric matrix

Calculation of the metric matrix [Dziedzic2013] (Sec. II.D.1) is challenging. This is because products of spherical waves (SWs) and products of a spherical wave with a potential of a spherical wave (SWpot) oscillate rapidly and cannot be reliably integrated on a Cartesian or radial grid. ONETEP calculates the elements of the metric matrix by a piecewise approximation of SWs and SWpots with high-order Chebyshev polynomials. Once this is done, their overlaps can be calculated by overlapping a large number of polynomials, which is easy, but time consuming.

For a fixed set of atomic positions and chosen NGWF radii the metric matrix needs only be calculated once, this is done during initialisation. The metric matrix does not depend on the NGWFs themselves or even the number of NGWFs per atom.

In ONETEP versions prior to 5.1.2.3, metric matrix elements are evaluated using the method described in Ref. [Dziedzic2013] (section II.D.1), i.e. by 3-D integration over piecewise expansions of SWs/SWpots in Chebyshev polynomials. This is the "3-D Chebyshev" (3Dc) scheme.

For versions $\geq$ 5.1.2.3, an alternative metric matrix scheme is available in which metric matrix elements are decomposed into products of 2-D numerical and 1-D analytic integrals. This is the "2-D numerical, 1-D analytic" (2Dn-1Da) scheme.

The separation of the 3-D integral in the 2Dn-1Da scheme is made possible by expressing metric matrix elements for each atom pair in a *molecular* coordinate frame with the z-axis pointing along the vector between atomic centres and with a set of SWs aligned in this coordinate system. When expressed in spherical polar coordinates, the 1-D integration (over the azimuthal angle, ϕ) is simple to evaluate analytically. The 2-D integration over (r, θ) can be performed by the same approach used in the 3Dc method, i.e. evaluating integrals over piecewise expansions of the (r, θ)-dependent parts of the SWs and SWpots in Chebyshev polynomials.

To obtain the metric matrix in the original set of SWs, which are aligned parallel to the z-axis of a *global* (simulation cell) coordinate frame, the SWs in the molecular coordinate frame must be represented in terms of the SWs in the global coordinate frame. This is achieved by applying a rotation matrix to the real-spherical harmonic (RSH) components of the SWs/SWpots associated with each atom pair.

For versions $\geq$ 5.1.5.0, the 2Dn-1Da scheme is the default for evaluating the electrostatic metric matrix. The 3Dc scheme remains the default for the overlap metric matrix (0), since overlap metric matrix evaluation is not currently supported in the 2Dn-1Da scheme.

Use of the 2Dn-1Da scheme to evaluate the electrostatic metric matrix is strongly recommended, as it reduces the computational cost (in terms of memory and execution time) of evaluating the matrix by orders of magnitude compared to the 3Dc scheme (with no apparent loss of accuracy).

To set up SWRI (for both 2Dn-1Da and 3Dc schemes), define the following block:

```
%block swri
  *myname* :math:`l_{\textrm{max}}` :math:`q_{\textrm{max}}` *metric* :math:`N_{\textrm
↪{i}}` :math:`N_{\textrm{o}}` *flags*
%endblock swri
```

Replace ``*myname*`` with a user-readable name for the SWRI, you will use it later to tell HFx or DMA which SWRI to use (as you can have more than one SWRI).

[lmax] $l_{\max}$ is the maximum angular momentum in the SW basis. Supported values are 0 (s-like SWs only), 1 (s- and p-like SWs), 2 (s-, p- and d-like SWs), 3 (s-, p-, d- and f-like SWs) and 4 (you get the idea). For HFx choose 2 for a crude calculation, 3 for good quality, and 4 for extreme quality. 0 and 1 will likely be too crude to fully NGWF-converge. Expect 2 to recover more than 99% of HFx energy, and 3 to recover about 99.7%. See [Dziedzic2013] (Fig. 8) for an illustrative guide. For DMA choose 0 if you're only interested in atomic charges, 1 if you want charges and dipoles, and 2 if you want charges, dipoles and quadrupoles. There is no point in using 3 or 4 for DMA. Be aware that the

computational cost grows as $\mathcal{O}(l_{\mathrm{max}}^4)$ (for the entire calculation), and the memory requirement grows as $\mathcal{O}(l_{\mathrm{max}}^4)$ (for the entire calculation).

[qmax] q_{max} is the number of Bessel (radial) functions in the SW basis. Adding more Bessel functions increases the size of the basis (linearly) and improves quality. However, each subsequent Bessel function oscillates more rapidly and beyond a certain point (at about 15 Bessel functions) they become difficult to represent on Cartesian grids of typical fineness. As a guide, use 7 for a crude calculation, 10 for reasonable accuracy and 14 for extreme accuracy. Again, see [Dziedzic2013] (Fig. 8) to see a graphical representation of the impact of this setting on accuracy. There is no upper limit on the value of this parameter; however, ONETEP will refuse to include Bessel functions that oscillate too rapidly for your KE cutoff – you will get a warning and high-numbered Bessel functions will be removed from the basis. The computational cost grows as $\mathcal{O}(q_{\mathrm{max}}^2)$, and so does the memory requirement. Replace ``*metric*`` with `V` to use the electrostatic metric in the SWRI, or with `O` to use the overlap metric. Specifying `VO` or `OV` will generate both metric matrices, although that is not usually done. In general, prefer the electrostatic metric – the error in the energy due to the density fitting approximation is then second-order in the fitting error, while for the overlap metric it is first-order.

As mentioned above, for ONETEP versions $\geq 5.1.5.0$ the electrostatic metric (`V`) is evaluated by default using the more efficient 2Dn-1Da scheme. The overlap metric (`O`) cannot (currently) be evaluated using this scheme, so is evaluated using the more costly 3Dc scheme. Since only a single metric matrix scheme may be used at a time, if both metric matrices are requested (`VO` or `OV`) then ONETEP will fall back to the 3Dc scheme. In this situation, it is worth considering whether your calculation can be run with only the electrostatic metric in order to take advantage of the more efficient 2Dn-1Da scheme.

N_{i} is the number of intervals into which the integration domain will be divided along each axis for the purpose of Chebyshev interpolation. In the 3Dc scheme, this is the localisation sphere of an SW (an NGWF sphere, see [Dziedzic2013] (Sec. II.D.1)), while in the 2Dn-1Da scheme this is a half-disc with the same radius. For 3Dc, 8 is the bare minimum, 10 is crude, 12 is accurate and 14 is extremely accurate. You should avoiding going overboard (recommended value is 12), since the computational cost grows as $\mathcal{O}(N_{\mathrm{i}}^3)$ (only for the SWRI stage, the remainder of the calculation is not sensitive to this value). The memory requirement grows as $\mathcal{O}(N_{\mathrm{i}}^3)$ (only for the SWRI stage). See [Dziedzic2013] (Fig. 5) to see how the accuracy of the metric matrix depends on this parameter when using the 3Dc scheme. For 2Dn-1Da, the computational cost ($\mathcal{O}(N_{\mathrm{i}}^2)$) and memory requirements ($\mathcal{O}(N_{\mathrm{i}}^2)$) are considerably lower, so it is practical to use $N_{\mathrm{i}} = 14$ or larger for routine calculations. In this case, it is recommended to use 12 or greater. In particular, very crude (less than 10) values should be avoided when using 2Dn-1Da. Testing of DMA with the 2Dn-1Da scheme suggests that the 2Dn-1Da scheme is more sensitive than 3Dc to lower values of N_{i} (i.e. larger errors are produced in multipole moments compared to values converged with respect to N_{i} and N_{o}). N_{o} is the order of Chebyshev polynomials used in the interpolation. Just like for N_{i}, for the 3Dc scheme 8 is the bare minimum, 10 is crude, 12 is accurate and 14 is extremely accurate. Again, you should avoid going overboard (recommended value[1] is 12), since the computational cost grows as $\mathcal{O}(N_{\mathrm{o}}^4)$ (only for the SWRI stage, the remainder of the calculation is not sensitive to this value). The memory requirement grows as $\mathcal{O}(N_{\mathrm{o}}^3)$ (only for the SWRI stage). See [Dziedzic2013] (Fig. 5) to see how the accuracy of the metric matrix depends on this parameter when using the 3Dc scheme. For 2Dn-1Da, the computational cost ($\mathcal{O}(N_{\mathrm{o}}^3)$) and memory requirements ($\mathcal{O}(N_{\mathrm{o}}^2)$) are again considerably lower, so it is practical to use $N_{\mathrm{o}} = 14$ or larger for routine calculations. In this case, it is recommended to use 12 or greater. For the reasons outlined above for N_{i}, very crude (less than 10) values of N_{o} should be avoided when using 2Dn-1Da. For DMA, which is performed during a properties calculation[2], crude settings will simply lead to less accurate multipoles. In HFx, on the other hand, settings that are too crude would prevent convergence because the exchange matrix would not be sufficiently symmetric. ONETEP will abort your calculation if the exchange matrix is later found to not be symmetric to at least 3.4 digits. To avoid frustration, do not go below $N_{\mathrm{i}} = 10$, $N_{\mathrm{o}} = 10$.

For the 2Dn-1Da scheme, the cost of evaluating the metric matrix is typically significantly smaller than the overall cost of the subsequent calculation. In this case, it is practical to routinely use higher N_{i} and N_{o} values (e.g. $N_{\mathrm{i}} = N_{\mathrm{o}} = 14$ or 16).

[1] There is a caveat here when using the overlap metric (which you shouldn't be doing anyway). At the boundary of the NGWF localisation region a SW has a derivative discontinuity, while a SWpot is smooth together with its derivative. This discontinuity is poorly approximated with high-order polynomials (Runge effect), and the problem becomes worse when the number of intervals is small. The setting N_{i}=12, N_{o}=12 is a poor choice in this case. When using the overlap metric, always use the lowest possible Chebyshev order (2, a parabola) and a large number of intervals to compensate. A decent setting, and with a comparable cost, would be N_{i}=72, N_{o}=2. Of course you should not be using the overlap metric in the first place!

[2] Unless you're doing QM/MM calculations with polarisable embedding.

Generating the metric matrix can be costly (particularly when using the 3Dc scheme). When restarting calculations that crashed, ran out of walltime, for restarts to do properties, or re-runs with different settings it makes sense to save the metric matrix to a file and re-use it during restarts. A metric matrix can be reused as long as the positions of the atoms and the NGWF radii did not change. ``*flags*`` is a combination of one or more letters or numbers: `W`, `R`, `P`, `Q`, `X`, `E`, `D`, `2`, `3`, controlling the behaviour of ONETEP during SWRI. The following flags instruct ONETEP to perform particular actions:

- `W` – writes the metric matrix to a file, once it has been calculated in its entirety. The file will have the extension `.vmatrix` for the electrostatic metric matrix, and `.omatrix` for the overlap metric matrix. This is highly recommended.
- `R` – reads the metric matrix from a file, instead of calculating it. The file will have the extension `.vmatrix` for the electrostatic metric matrix, and `.omatrix` for the overlap metric matrix. This is highly recommended for restart calculations. ONETEP will not allow you to use this flag when it knows the ions will move (`TASK : GEOMETRYOPTIMIZATION, TRANSITIONSTATESEARCH, MOLECULARDYNAMICS, PHONON, FORCETEST`), as the metric matrix gets invalidated once an ion moves.
- `P` – will instruct ONETEP to print the metric matrix in text form straight to the output. This can be useful for visual inspection and debugging, although be aware that for larger systems the output can be bulky.
- `Q` – will instruct ONETEP to quit immediately after the metric matrix is calculated (and potentially written and/or printed). This can be useful if the SWRI stage is run separately from the main calculation, e.g. on a large number of CPU cores that would be excessive for the main calculation.
- `X` – means "none of the above" and should be used if you don't intend to write, read, print the metric matrix and you don't want ONETEP to quit at this stage.

The remaining flags change how the SWRI is performed:

- `2` – forces use of the 2Dn-1Da metric matrix evaluation scheme, overriding the default selection. Note that the 2Dn-1Da scheme is currently only available for evaluation of the electrostatic metric matrix (`V`) and ONETEP will abort with an error if the `2` flag is used in combination with the overlap metric matrix (`O`).
- `3` – forces use of the 3Dc metric matrix evaluation scheme, overriding the default selection.

The `2` and `3` flags only have effect when computing the metric matrix. When reading the matrix from disk in full (`R` flag), the flags have no effect, as the matrix has already been precomputed. When reading the matrix in part from atomblocks (see below), the flags will only affect atomblocks that are not read from disk (i.e. need to be computed).

By using `W` and `R` you can re-use a fully calculated metric matrix. For large jobs which take many CPU-core-hours you may want to re-use *partial* results simply because you may not have enough walltime to run the SWRI calculation to completion. By default ONETEP writes partial results (metric matrix atomblocks) to files (`*.[vo]matrixblock`) as it churns through the calculation. These matrixblocks will be automatically read from files if they can be found – i.e. before starting to calculate a block, ONETEP will always first look for a corresponding file to try and avoid the calculation, regardless of your ``*flags*``. Thus, if your SWRI calculation is interrupted, retain the matrixblock files to make the next run complete faster. If you wrote the completed matrix to a file, there is no point in keeping the matrixblock files and you should delete them to save disk space. If you would rather not have to delete them manually, specify `E` (for "erase") in ``*flags*`` and they will not be kept (or indeed written to). Each matrixblock file encodes the position of the two atoms between which it is calculated in the filename. This proves useful in TASK PHONON calculations and TASK GEOMETRYOPTIMIZATION calculations with some atoms fixed – atomblocks between pairs of atoms that did not move will not be recalculated needlessly, but rather reloaded from files, unless you specify `E`. Finally, the expert option `D` instructs ONETEP to disassemble the fully calculated metric matrix into atomblocks (best used in combination with `R` and `Q`). This can be useful if you saved the metric matrix to a file, deleted the matrixblock files, and later change your mind.

Examples

This creates an SWRI called `for_hfx`, with an expansion up to l=3, 10 Bessel functions, using the electrostatic metric. Chebyshev interpolation will use 12 intervals and 12-order polynomials. The metric matrix will be written to a file. That would be standard for a HFx calculation.

```
%block swri
  for_hfx 3 10 V 12 12 W
%endblock swri
```

Like above, but will read the metric matrix from a file instead of calculating it.

```
%block swri
  for_hfx 3 10 V 12 12 R
%endblock swri
```

This creates an SWRI called `for_dma`, with an expansion up to l=2, 12 Bessel functions, using the electrostatic metric. Chebyshev interpolation will use 10 intervals and 12-order polynomials. The metric matrix will not be written to a file. That would be standard for a DMA calculation.

```
%block swri
  for_dma 2 12 V 10 12 X
%endblock swri
```

This creates an SWRI called `high_qual`, with an expansion up to l=4, 16 Bessel functions (extremely large and accurate SW basis set), using the overlap metric (not the best choice). Chebyshev interpolation will use 60 intervals and 2-order polynomials (parabolas). The metric matrix will be written to a file, printed out in text form, the matrixblock files will be erased, and ONETEP will quit.

```
%block swri
  high_qual 4 16 O 60 2 WPEQ
%endblock swri
```

This creates an SWRI called `for_hfx_and_dma`, with an expansion up to l=2, 9 Bessel functions, using the electrostatic metric. Chebyshev interpolation will use 14 intervals and 14-order polynomials. The 2Dn-1Da metric matrix evaluation scheme has been explicitly selected (for versions $\geq$ 5.1.5.0, this would not be necessary, as 2Dn-1Da is the default for the electrostatic metric). The resulting metric matrix will be written to a file and the matrixblock files will be erased.

```
%block swri
  for_hfx_and_dma 2 9 V 14 14 WE2
%endblock swri
```

As above, but with the 3Dc metric matrix scheme explicitly selected. This will likely be very costly compared to using the 2Dn-1Da scheme.

```
%block swri
  for_hfx_and_dma 2 9 V 14 14 WE3
%endblock swri
```

Choosing which species participate in a SWRI

For every SWRI defined like above you need to specify which atomic species participate in it. This allows performing an SWRI for a subsystem, e.g. doing DMA only for atoms of a solute in the presence of a solvent, but not for atoms of the solvent itself. In such a scenario atomblocks only need to be calculated between atoms such that at least one atom belongs to the SWRI. For HFx this is less meaningful, and you will want to list all your species in the block. Note how the block name **includes the name** of the SWRI defined above and may look like this:

```
%block species_swri-for_hfx
H
O
C
%endblock species_swri-for_hfx
```

if your SWRI was called `for_hfx` and your system is composed of species H, O and C.

Advanced SWRI options

`swri_verbose (logical)` – set this to `T` to get detailed information on matrixblock I/O. Useful when you want to know where ONETEP is looking for matrixblock files and whether each file was succesfully loaded or not. This is output from all MPI ranks, so can make the output cluttered. Default: `F`.

`swri_cheb_batchsize (integer)` – sets the size of the batches in which SWs are processed in the calculation of the metric matrix. For the 3Dc scheme, the default is 12. Increasing this value can improve efficiency (by better balancing threads), but will increase memory load. Keep this divisible by the number of OMP threads for best performance. For the 2Dn-1Da scheme, batching has little benefit and can lead to significant load imbalance across MPI processes for larger systems. Thus, the default for 2Dn-1Da is the number of SWs in the auxiliary basis set. For both schemes, if this value is set larger than the number of SWs in the auxiliary basis set, it will be capped accordingly.

`swri_assembly_prefix (string)` – sets the prefix for the matrixblock files that are assembled into the metric matrix. The default is the rootname of your ONETEP input. Adjusting this can be useful if you keep a large number of matrixblock files in one directory and have multiple calculations, in different directories, using these matrixblock files.

`swri_proximity_sort_point (string of three values in bohr)` – metric matrix blocks are evaluated in order, with blocks between atoms closest to a predefined point done first. This is useful if you have a giant SWRI calculation (say for a solute and a few solvation shells) and would like other calculations to start using first matrix blocks as soon as possible (e.g. for calculations on just the solute). Using this keyword you can choose the point for sorting the atomblocks. The default is `0.0 0.0 0.0`. A unit of bohr is implicitly added (do not specify it).

`swri_swop_smoothing, swri_overlap_indirect, swri_improve_inverse` – these are experimental features, do not use these.

3.10.4 Hartree-Fock exchange

Now that you have SWRI set up, a basic HFx (or hybrid functional) calculation should be simple to set up. The following three keywords are mandatory and do not provide defaults:

`hfx_use_ri (string)` – tells HFx which SWRI to use. Specify the name used in the SWRI block, e.g. `hfx_use_ri for_hfx`.

`hfx_max_l (integer)` – specifies the maximum angular momentum in the SW basis. In most scenarios this will be equal to $l_{\max}$ that you specified in the SWRI block. Read the description of $l_{\max}$ (Sec. [lmax]) to understand the meaning of this parameter. You can use a *lower* value than the one specified in the SWRI block if you want to use only a subset of the SW basis set (e.g. for benchmarking, or doing DMA with a lower $l_{\max}$ than you use for HFx), but not for HFx (where you must use the same value that you used in the SWRI block).

`hfx_max_q (integer)` – specifies the number of Bessel functions in the SW basis for each angular momentum channel. In most scenarios this will be equal to $q_{\max}$ that you specified in the SWRI block. Read the description of $q_{\max}$ (Sec. [qmax]) to understand the meaning of this parameter. You can use a *lower* value than the one specified in the SWRI block if you want to use only a subset of the SW basis set (e.g. for benchmarking, or doing DMA with a lower $q_{\max}$ than you use for HFx), but not for HFx (where you must use the same value that you used in the SWRI block).

With the above set up, the last step is to choose a suitable functional through `xc_functional`. The following hybrid functionals use HFx: `B1LYP`, `B1PW91`, `B3LYP`, `B3PW91`, `PBE0`, `X3LYP`. For a pure Hartree-Fock calculation use `HF`.

The following two keywords might be handy:

`hfx_cutoff (physical)` – specifies the distance-based cutoff for all HFx interactions. The default is 1000 bohr, which effectively corresponds to no truncation. In the absence of truncation ONETEP's HFx implementation scales as $\mathcal{O}(N^2)$, so you are advised to use HFx truncation even if you do not use density kernel truncation. Exchange interactions are rather short-ranged, and for systems with a band-gap it should be safe to truncate them at $20a_0$. See [Dziedzic2013] (Figs. 19, 20) for more details. Do not use a value smaller than twice the NGWF radius.

`hfx_metric (string)` – selects the metric actually used for HFx calculations. The default is `electrostatic`. The other option is `overlap`. The appropriate metric matrix must have been included at the SWRI stage (`V` or `0`, respectively).

Other HFx-related keywords (`hfx_nlpp_for_exchange`, `hfx_read_xmatrix` and `hfx_write_xmatrix`) correspond to experimental features and should not be used.

Example

The following is a bare-bones example for a reasonably good-quality HFx calculation on a slightly distorted water molecule. That should converge in 11 NGWF iterations within 1.5 minute on a desktop machine (2 MPI ranks, 4 OMP threads each), requiring about 4 GiB of RAM.

```
xc_functional B3LYP
cutoff_energy 800 eV

%block swri
  for_hfx 3 10 V 10 10 WE
%endblock swri

%block species_swri-for_hfx
O
H
%endblock species_swri-for_hfx

hfx_use_ri for_hfx
hfx_max_l 3
hfx_max_q 10

%block lattice_cart
  25.00     0.00     0.00
   0.00    25.00     0.00
   0.00     0.00    25.00
%endblock lattice_cart
```

(continues on next page)

(continued from previous page)

```
%block positions_abs
ang
O 5.79564200 7.40742600 6.63194300
H 5.19938100 8.05407400 6.24141400
H 5.16429100 6.74016800 6.88482600
%endblock positions_abs

%block species
O O 8 4 8.0
H H 1 1 8.0
%endblock species

%block species_pot
O "oxygen.recpot"
H "hydrogen.recpot"
%endblock species_pot
```

3.10.5 DMA

DMA (Distributed Multipole Analysis) is a technique for partitioning charge density into single-atom contributions and finding a set of point multipoles that most accurately represent this charge density. The point multipoles are usually, although not universally, atom-centered – this is the case in ONETEP. DMA was proposed by Rein [Rein1973] and has been pioneered and popularised by Stone [Stone1981] and Alderton [Stone1985]. It is typically performed in a Gaussian basis set [Stone1998], [Stone2005], but ONETEP uses a version adapted to the NGWF basis. More details on our approach can be found in Refs. [Dziedzic2016], [Vitale2015].

DMA in ONETEP first uses SWRI (cf. earlier Sections) to expand NGWF-NGWF overlaps (not exactly atom-pair densities, because there is no density kernel there) in an auxiliary SW basis set. Depending on the metric, this density fitting will strive to either minimise the difference in electronic density between the original density and the fit (for the overlap metric), or the electrostatic energy of the difference in densities interacting with itself (for the electrostatic metric). The use of electrostatic metric is preferred. Once the NGWF-NGWF overlaps are expressed in an SW basis, owing to certain properties of SWs and to the fact that in ONETEP they are chosen to be atom-centered, it becomes easy to find atom-centered point multipoles that yield the (exactly) equivalent potential. This stage is termed spherical wave expansion (SWX) and its result are atom-centered point multipoles that are the best fit to the original electronic density (under the assumed metric). Apart from calculating electronic multipoles, ONETEP's DMA also calculates total (electronic + ionic core) atom-centered multipoles. Also calculated are the total multipoles of the system (e.g.. the molecular dipole or quadrupole), suitably averaged over all the atoms that were part of the SWRI. For non-neutral molecules the value of the dipole depends on the point where it is calculated (similarly for higher multipoles), and so the total multipoles are calculated *at a reference point* of your choosing.

DMA in ONETEP is performed in two contexts. The most common is during a `task properties` calculation ("properties-DMA"). The other use of DMA is in QM/MM calculations using ONETEP and tinker (tinktep approach, cf. [Dziedzic2016], "polemb-DMA"). Some DMA keywords pertain to both contexts, and some pertain only to one of them – this will be carefully highlighted. It is possible to mix the two to a reasonable degree (i.e. to perform QM/MM with one set of DMA parameters, and properties-DMA at the end of the run, with another set of parameters). By "reasonable degree" I mean that some of the parameters are shared.

DMA: minimal set-up

To use DMA, first set up SWRI (cf. earlier Sections). Now that you have SWRI set up, a basic calculation using DMA should be simple to set up. First, specify `dma_calculate T` to enable DMA, as it is off by default. Once you've done that, the following keywords **become mandatory**:

`dma_use_ri (string)` – tells DMA which SWRI to use. Specify the name used in the corresponding SWRI block, e.g. `dma_use_ri for_dma`.

`dma_max_l (integer)` – specifies the maximum angular momentum in the SW basis used in DMA. In most scenarios this will be equal to $l_{\max}$ that you specified in the SWRI block. Read the description of $l_{\max}$ (Sec. [lmax]) to understand the meaning of this parameter. You can use a lower value than the one specified in the SWRI block if you want to use only a subset of the SW basis set (e.g. for benchmarking). This keyword only affects properties-DMA, the equivalent for polemb-DMA is `pol_emb_dma_max_l`. This keyword needs to be specified even if you do not plan to use properties-DMA (in that case, specify 0). If you only care about atom-centered charges, specify 0. If you care about atom-centered charges and dipoles, specify 1. If you care about atom-centered charges, dipoles and quadrupoles, specify 2.

`dma_max_q (integer)` – specifies the number of Bessel functions in the SW basis for each angular momentum channel to be used in DMA. In most scenarios this will be equal to $q_{\max}$ that you specified in the SWRI block. Read the description of $q_{\max}$ (Sec. [qmax]) to understand the meaning of this parameter. You can use a lower value than the one specified in the SWRI block if you want to use only a subset of the SW basis set (e.g. for benchmarking). This keyword only affects properties-DMA, the equivalent for polemb-DMA is `pol_emb_dma_max_q`. This keyword needs to be specified even if you do not plan to use properties-DMA (in that case, specify 0).

Non-mandatory keywords affecting both properties-DMA and polemb-DMA

`dma_metric (string)` – selects the metric used for DMA calculations. The current default is `electrostatic`. The other option is `overlap`. The appropriate metric matrix must have been included at the SWRI stage (`V` or `0`, respectively). This keyword affects both properties-DMA and polemb-DMA.

`dma_bessel_averaging (boolean)` – specifies whether all DMA-based multipoles are to be averaged over an even-odd pair of $q_{\max}$. Multipoles obtained with DMA display an oscillatory behaviour when plotted as a function of $q_{\max}$. This has to do with how the Bessel functions sample the radial profile of NGWFs. In essence, for all even $q_{\max}$ a particular multipole will be overestimated, while for all odd $q_{\max}$ the same multipole will be underestimated (or the other way round). Other multipoles will be affected similarly (except in reverse) to compensate. This oscillatory behaviour decays as the quality of the SW basis is increased, but the decay is slow. Much more stable multipoles are obtained by averaging the results of two SWX runs – one with the $q_{\max}$ the user specified in `dma_max_q`, and one with $q_{\max}$ that is less by one. This even-odd averaging can be performed automatically by specifying `dma_bessel_averaging T`, and this is done by default. When this option is enabled, output files include multipoles obtained with both SWX qualities, followed by the average, except for the `.dma_multipoles_gdma_like.txt` file, which will contain only the final, averaged multipoles. There is no extra effort associated with this option at the SWRI stage, and the effort of the SWX stage (which is usually much, much lower) is practically doubled (two separate SWXs have to be performed). This keyword affects both properties-DMA and polemb-DMA.

`dma_scale_charge (boolean)` – specifies DMA charge-scaling is to be performed (default) or not. This an important option. The multipoles obtained with DMA are always approximate. The total DMA monopole (charge) will be close to, but not exactly equal to, the total charge of the system (or its subset, if DMA's SWRI did not encompass all atoms). This means that the total DMA monopole of a nominally neutral system will not be exactly zero, but typically a very small fraction of an electron. This is inconvenient, because it formally breaks the translational invariance of the total dipole, which begins to depend, very slightly, on the reference point where it is calculated. The easiest workaround is to scale, *a posteriori*, the DMA monopole by the ratio of expected charge to the obtained DMA charge. This scaling is factor will be very close to zero (e.g. 0.9998), unless your SW basis set is very crude (single-digit $q_{\max}$, etc.). The "expected" (electronic) charge either obtained automatically (by default), or can be specified manually using `dma_target_num_val_elec`. When not specified manually, the expected electronic charge is determined as follows. If DMA's SWRI encompasses all atoms ("full-system DMA"), it is equal to the total number of valence electrons in the system (obtained from $\mathrm{Tr}\,[\mathbb{KS}]$). If DMA's SWRI does not encompass all atoms ("subsystem DMA"), Mulliken

analysis is performed every time charge-scaling needs to be done (essentially at every LNV step, or twice per LNV step when Bessel averaging is used), and Mulliken charges of all atoms within DMA's SWRI are summed to obtain the expected electronic charge. Using DMA charge-scaling is recommended (hence it's on by default), but care must be taken when using it with polemb-DMA (there are no issues with properties-DMA). The following issues and limitations arise. (1) In polemb-DMA the DMA multipoles enter LNV and NGWF gradient expressions. The quantity Tr $[\mathbb{KS}]$ is not strictly a constant, and has non-zero DKN and NGWF derivatives, leading to additional terms in LNV and NGWF gradients when charge-scaling is used. These extra terms have been implemented for LNV gradients, but *not* for NGWF gradients, where they become really hairy (these are under development). Hence the threefold combination of polemb-DMA, charge-scaling and NGWF optimisation is not permitted (will refuse to run). (2) The threefold combination of polemb-DMA, charge-scaling and `dma_target_num_val_elec` is not permitted (will refuse to run), regardless of whether NGWF optimisation is used or not. This is because the expected number of electrons becomes constant (user-specified value) in this scenario, which is incompatible with the charge-scaling corrections accounting for Tr $[\mathbb{KS}]$. Long story short: use DMA charge-scaling for properties-DMA, but not for polemb-DMA. This keyword affects both properties-DMA and polemb-DMA.

`dma_target_num_val_elec (integer)` – specifies the expected number of valence electrons for DMA. This keyword should only be used when DMA charge-scaling is in effect (see above), and only if the automatic determination of the expected number of electrons (see above) in the part of your system seen by DMA is not satisfactory. The default is for this keyword to be omitted. This keyword affects both properties-DMA and polemb-DMA.

`polarisation_simcell_refpt (real real real)`. The default is `0.0 0.0 0.0`. Specifies the reference point in the simulation cell (in bohr) at which total DMA multipoles are calculated. This is mostly useful if your system (strictly speaking: your DMA subsystem) is not charge-neutral and you are interested in the value of the total dipole. When the system is non-neutral, the total dipole is not translation invariant, and a reference point for calculating it needs to be specified. This keyword specifies this reference. Also note that when a simcell full-density polarisation calculation is performed (via `task properties` and `polarisation_simcell_calculate`), this keyword also adjusts this calculation's reference point. This keyword affects both properties-DMA and polemb-DMA.

`dma_precise_gdma_output (boolean)`. The default is on (`T`). One of the files output by DMA is the `.dma_multipoles_gdma_like.txt` file, which is formatted as to be compatible with the output generated by Stone's GDMA program. This output can be directly used by other programs expecting input in this format. The original GDMA format is fixed-form, meaning the precision of the output multipoles is rather restricted by the number of digits that can be output. In polemb-DMA mode this precision is insufficient to accurately drive the MM calculation performed by an external program (tinker). Specifying `dma_precise_gdma_output T` instructs ONETEP to output multipoles with the additional necessary precision, but breaks strict compatibility with the GDMA format. If the external program you use to parse the GDMA file is aware of that (e.g. tinker can be suitably patched), this is fine. If you have no control over the external program and need ONETEP to adhere strictly to the GDMA format, use `dma_precise_gdma_output F`.

Expert, non-mandatory keywords affecting both properties-DMA and polemb-DMA

Just don't. These are used for experimental purposes, particularly in QM/MM.

`dma_multipole_scaling (real)` – causes all DMA multipoles to be scaled by a constant. This affects both the output multipoles and the multipoles used internally in polemb expressions. Whenever necessary (during charge-scaling, in gradients) this scaling is temporarily undone internally for consistency. This keyword affects both properties-DMA and polemb-DMA.

`dma_dipole_scaling (real)` – causes all DMA dipoles to be scaled by a constant. This affects both the output dipoles and the dipoles used internally in polemb expressions. Whenever necessary (essentially in gradients) this scaling is temporarily undone internally for consistency. This keyword affects both properties-DMA and polemb-DMA.

`dma_quadrupole_scaling (real)` – causes all DMA quadrupoles to be scaled by a constant. This affects both the output quadrupoles and the quadrupoles used internally in polemb expressions. Whenever necessary (essentially in gradients) this scaling is temporarily undone internally for consistency. This keyword affects both properties-DMA and polemb-DMA.

Non-mandatory keywords affecting only properties-DMA

`dma_output_potential (boolean)`. The default is off (`F`). When turned on (`T`), during properties-DMA the electrostatic potential due to all DMA *electronic* multipoles is calculated on the $z = 0$ and $z = z_{\max}$ faces of the simulation cell (on all fine-grid points lying on those faces). This potential is output to text files. This is useful for assessing the quality of the DMA approximation to the full, distributed charge density. If DMA's SWRI does not span the entire system, the output potential is only due to those atoms included in DMA's SWRI. This keyword affects only properties-DMA.

`dma_output_potential_reference (boolean)`. The default is off (`F`). When turned on (`T`) *and* `dma_output_potential` *is also on*, during properties-DMA the reference electrostatic potential due to the full, distributed *electronic* density is calculated on the $z = 0$ and $z = z_{\max}$ faces of the simulation cell (on all fine-grid points lying on those faces). This potential is output to text files. This is useful to obtain a reference for assessing the quality of the DMA approximation to the full, distributed charge density (see above). Regardless of whether DMA's SWRI spans the entire system or not, the output potential is due to *all* electrons in the system. Thus, the two sets of potentials are only comparable in full-system DMA. The reference potential is calculated by a pointwise integration over the entire volume (fine-grid) *for every point on the two faces*, which is a time-consuming process, so use sparingly. This keyword affects only properties-DMA.

Non-mandatory keywords affecting only polemb-DMA

Refer to the separate documentation for polarisable embedding in ONETEP.

Example

The following is a bare-bones example for a reasonably good-quality properties-DMA calculation on a slightly distorted water molecule. That should converge in 11 NGWF iterations within 1 minute on a desktop machine (2 MPI ranks, 4 OMP threads each), requiring about 3 GB of peak RAM.

```
xc_functional PBE
cutoff_energy 800 eV

do_properties T

%block swri
  for_dma 2 14 V 12 12 WE
%endblock swri

%block species_swri-for_dma
O
H
%endblock species_swri-for_dma

dma_calculate T
dma_use_ri for_dma
dma_metric electrostatic
dma_max_l 2
dma_max_q 14

dma_scale_charge T
dma_bessel_averaging T

%block lattice_cart
```

(continues on next page)

(continued from previous page)

```
  25.00     0.00     0.00
   0.00    25.00     0.00
   0.00     0.00    25.00
%endblock lattice_cart

%block positions_abs
ang
O 5.79564200 7.40742600 6.63194300
H 5.19938100 8.05407400 6.24141400
H 5.16429100 6.74016800 6.88482600
%endblock positions_abs

%block species
O O 8 4 8.0
H H 1 1 8.0
%endblock species

%block species_pot
O "oxygen.recpot"
H "hydrogen.recpot"
%endblock species_pot
```

Expected results:

Description	Dipole (au)	Dipole (debye)
Full density (cores + NGWFs)	0.7564	**1.9226**
DMA (point multipoles, $q_{\max}$=14)	0.7477	1.9005
DMA (point multipoles, $q_{\max}$=13)	0.7680	1.9519
DMA (point multipoles, Bessel-averaged)	0.7578	**1.9261**

Table: Dipole moment of distorted water molecule. Comparison of accuracy: full density vs. DMA point multipoles.

3.10.6 Advanced options

Making Hartree-Fock exchange faster or less memory-hungry

Hartree-Fock exchange is not fast, although we've made great improvements in v5.3.4.0. For small systems ($<$ 200 atoms), with a bit of luck, it will be an order of magnitude slower than GGA calculations. For large systems ($\approx$ 1000 atoms) expect it to be two orders of magnitude slower.

The main way to improve performance is by using more RAM – this is because there are plenty of opportunities for caching some results that would otherwise have to be recomputed. If HFx was to cache everything, it would quickly exhaust all available RAM, even on well-equipped machines. Therefore, there are limits in place for each of the caches. These limits are expressed in MiB (1048576 bytes) and are **per MPI rank**.

Remember that OMP threads can share memory, while MPI ranks cannot. This means that the key to obtaining high performance with HFx is to **use as many OMP threads as possible**. In most HPC settings this will mean using only 2 MPI ranks per node (one MPI rank per NUMA region, most HPC nodes have two NUMA regions). For example on Iridis5, with 40 CPU cores on each node, best performance is obtained by using 2 MPI ranks, with 20 OMP threads each, on every node. This is in contrast to non-HFx calculations, which typically achieve peak performance for 4-5 OMP threads. HFx is well-optimised for high thread counts. By reducing the number of MPI ranks, you allow each

rank to use more RAM. This is the key to success. Don't worry about the drop in performance of the non-HFx part, it will be dwarfed by the gain in HFx efficiency.

The easiest way to control how much RAM HFx can use is via the parameter `hfx_memory_limit`. The default value is 4096, meaning HFx will not ask for more than 4 GiB of RAM **per MPI rank**. This is *in addition* to any memory use from the rest of ONETEP. If you can spare more RAM, definitely tell this to the HFx engine by saying e.g.

```
hfx_memory_limit 16384! I have 16 GiB per MPI rank to spare.
```

The HFx engine will automatically distribute this RAM across the three main caches. Or, more specifically, it will first consume the amount of RAM absolutely needed for core HFx functionality, and *then* distribute the rest to the three caches. You will get a banner informing you about how much memory went into satisfying the minimum requirements:

```
+-----------------------------------------------------+
|  HFx TEFCI engine: minimum requirements             |
|  Estimated memory requirement per MPI rank          |
+-----------------------------------------------------+
|  Radial Bessel lookup                :   30.52 MB   |
|  Dd NGWFs hash table                 :    1.66 MB   |
|  All remote NGWFs hash table         :   37.38 MB   |
|  dlists hash table                   :    6.53 MB   |
|  coeffs hash table (estimate)        :   26.93 MB   |
|  V metric matrix hash table          :  377.59 MB   |
|  f auxiliary term                    :  131.78 MB   |
|  P term in NGWF gradient             :  131.78 MB   |
|  Q term in NGWF gradient             :  238.88 MB   |
|  My kets in NGWF grad. (estimate)    :   78.09 MB   |
|  Local kets in NGWF grad. (estim.)   :   43.20 MB   |
|  K^{CD} hash table                   :    3.88 MB   |
|  K^{AB} hash table                   :    3.60 MB   |
|  tcK^A_B hash table                  :    3.60 MB   |
|  tcK^B_A hash table                  :    3.60 MB   |
+-----------------------------------------------------+
|  Estimated peak total per MPI rank   :    1.09 GB   |
+-----------------------------------------------------+
```

In the event that the memory limit specified with `hfx_memory_limit` is below even the minimum requirement (1.09 GB in the example above), you will get an error message explaining how much more RAM you would need to continue. Be aware of two things: (1) calculations with not much (or no) memory above the minimum requirement will be very slow, (2) the above is only an estimate. Under some circumstances HFx may consume slightly more memory, but not much. If you run out of memory, it is usually the NGWF gradient calculation (its non-HFx part) that breaks the camel's back.

Another banner informs you about how the remaining RAM is divided across the three caches ("hash tables"). It may look like this:

```
HFx: - Adjusting cache sizes according to weights: 0.6250, 0.3125, 0.0625.
+-----------------------------------------------------+
|  HFx TEFCI engine: user-adjustable requirements     |
|  Estimated memory requirement per MPI rank          |
+-----------------------------------------------------+
|  SWOP hash table                     :    3.61 GB   |
|  Expansions hash table               :    1.81 GB   |
|  AD NGWF products hash table         :  369.00 MB   |
+-----------------------------------------------------+
|  Estimated peak total per MPI rank   :    5.77 GB   |
```

(continues on next page)

(continued from previous page)

```
+-------------------------------------------------+
HFx: - Peak memory use capped at 6998.2 MB per MPI rank.
```

Here the user specified `hfx_memory_limit 7000` and HFx distributed the remaining 5.77 GB across the three caches with a default set of weights that is 10:5:1 ($= \frac{10}{16} : \frac{5}{16} : \frac{1}{16} = 0.6250 : 0.3125 : 0.0625$). This is an empirically determined near-optimal default for valence calculations. For conduction calculations the default is to give all remaining RAM to the SWOP hash table, because in conduction calculations expansions are never re-used and the number of NGWF products is so large, that it's faster to give up on storing them entirely.

The three caches store, respectively:

- Spherical waves or potentials thereof ("SWOPs"). Generating SWOPs is typically the main bottleneck of any HFx calculation, and increasing the size of this cache will lead to significant improvements, with returns diminishing after hit ratios exceed 90-95%.
- Spherical wave expansions (potentials of linear combinations of spherical waves on a centre). These can help performance too, but their size quickly becomes unwieldy.
- NGWF products. Less useful than the above, but cheap to store, except in conduction calculations.

In general, it is best to rely on the default division and to specify only `hfx_memory_limit`. However, if you feel you can do better, you can manually set the maximum for any number of caches, using the directives `cache_limit_for_swops`, `cache_limit_for_expansions`, `cache_limit_for_prods`. This might be useful if you specifically want to disable one or more of the caches (by specifying 0). Remember that `hfx_memory_limit` is still in effect by default, even if you do not specify it, and it will interact with the above. If you want to turn off the automatic balancing of memory limits, specify `hfx_memory_limit -1`. Once you do this, the specified cache limits will be used (with a default of 1024). Finally, if you keep the automated balancing, `hfx_memory_weights`, which accepts three real numbers, can be used to set the desired weights, if you are not satisfied with the default. The weights do not need to add to 1, they will be automatically rescaled. They cannot all be zero, but some of them can be zero (which then turns off the associated cache).

The utilisation of each cache is reported at some stage of the calculation (assuming `hfx_output_detail` is at `NORMAL` or higher). You will see banners like this:

```
+-------------------------------------------------------------------------------+
|    MPI |                            |              |           |              |
|   rank |       SWOP cache capacity | SWOPs needed | Cacheable |    Hit ratio |
+-------------------------------------------------------------------------------+
|      0 |       3691 MiB (24189 el) |    109140 el |    22.16% |       58.70% |
|      1 |       3691 MiB (24189 el) |    109140 el |    22.16% |       58.85% |
+-------------------------------------------------------------------------------+
```

This is a breakdown over all MPI ranks (only two in this case), informing you that you devoted about 3.7 GB per MPI rank to the SWOP cache, which enables caching 24189 elements, whereas 109140 elements could be stored, if you had more RAM. You were thus able to cache about 22% of all elements, but because HFx stores the most reusable ones first, the cache hit ratio will be about 59% – that is, in 59% of cases when HFx will be looking for a SWOP, it will find it in the cache. Different SWOPs will be needed on different nodes, hence the hit ratios are not exactly equal. Looking up SWOPs in the cache is at least an order of magnitude faster than recaculating them, so you should aim for a hit ratio of at least 90%.

Banners for the expansion and NGWF product caches will be printed after the first LNV (or EDFT) iteration (for `hfx_output_detail VERBOSE`) or after every energy evaluation (for `hfx_output_detail` at `PROLIX` or higher). They look like this:

```
HFx: +-------------------------------------------------------------------+
HFx: |    MPI |                          Expansion cache                  |
```

(continues on next page)

(continued from previous page)

```
HFx: |   rank |           hits |         misses |          total | hit ratio |
HFx: +-------------------------------------------------------------------------+
HFx: |      0 |        1982298 |       13369145 |       15351443 |   12.91 % |
HFx: |      1 |        1947916 |       13512601 |       15460517 |   12.60 % |
HFx: +-------------------------------------------------------------------------+
HFx: +-------------------------------------------------------------------------+
HFx: |    MPI |                    AD NGWF product cache                        |
HFx: |   rank |           hits |         misses |          total | hit ratio |
HFx: +-------------------------------------------------------------------------+
HFx: |      0 |        1947809 |        7499676 |        9447485 |   20.62 % |
HFx: |      1 |        1992291 |        7737078 |        9729369 |   20.48 % |
HFx: +-------------------------------------------------------------------------+
```

and show you a per-MPI-rank breakdown of how many hits and misses were recorded in accessing the cache, and what the hit ratio was. You will be able to achieve 100% only for the smallest of systems,

Changing cache limits only affects the tradeoff between RAM and CPU time, it has absolutely no effect on results, only on the time and memory it will take to arrive at them. If you are very pressed for RAM, you can set all the above cache sizes to 0. This will stop caching altogether, conserving memory, but will vastly increase run time, by a factor of several.

A simple, perhaps surprising, way of increasing performance (slightly) is by using PPDs that are not flat. By default ONETEP initialises the third dimension of a PPD to 1, making them flat. This makes sense in the absence of HFx. With HFx the book-keeping of the caching machinery will be faster when the PPDs are slightly larger. Preferably use `ppd_npoints` to make the PPDs $5 \times 5 \times 5$ or $7 \times 7 \times 7$. It might be necessary to explicitly set `psinc_spacing` and carefully choose the box dimensions. An easy solution is to choose `psinc_spacing 0.5 0.5 0.5` which corresponds to a kinetic energy cutoff of 827 eV, and to make your box dimensions divisible by $2.5\,a_0$ (for $5 \times 5 \times 5$ PPDs).

Finally, you can try omitting some terms in the expansion, if the expansion coefficients are below a certain threshold. This will affect the accuracy of your results, and so by default nothing is thrown away. This can be done via the keyword `swx_c_threshold` which takes a real number as an argument. Whenever an NGWF-pair expansion coefficient is below this value, potentials from this particular SW for this pair of NGWFs will not even be generated. This can be used in conjunction with a distance-based truncation. A value like 1E-5 will throw away maybe 2-3% of the terms. 1E-3 will throw away about 10-15% (so, little gain), and this will be enough to impair your NGWF convergence. I do not recommend changing this setting.

Other advanced options

`swx_output_detail (string)` – controls the verbosity of the spherical wave expansion. Allowed options: `BRIEF`, `NORMAL`, `VERBOSE`. Defaults to `NORMAL`. At `VERBOSE` you might feel overwhelmed by the output from all MPI ranks letting you know which atom they are working on (lines looking like "`+ A: 1`" or "`- B: 1`"). This is useful for big systems, where it takes a while to get from one LNV iteration to the next one, with no output otherwise.

`hfx_output_detail (string)` – controls the verbosity of Hartree Fock exchange. Allowed options: `BRIEF`, `NORMAL`, `VERBOSE`, `PROLIX`, `MAXIMUM`. Defaults to the value of `output_detail`. At `VERBOSE` you might feel overwhelmed by the output from all MPI ranks letting you know which atom they are working on (lines looking like "`- B: 1 [0] (1)`"). This is useful for big systems, where it takes a while to get from one LNV iteration to the next one, with no output otherwise. At `PROLIX` there is even more feedback. At `MAXIMUM` even the X matrix is printed, which will make your output file extremely big. This is recommended only when debugging. The recommended setting is `VERBOSE`.

`hfx_bessel_rad_nptsx (integer)` – specifies how many points are used in the radial interpolation of Bessel functions. The default is 100000 and should be sufficient. Increasing this value (perhaps to 250000 or so) improves accuracy,

particularly if your simulation cell is large, but there is an associated linear memory cost (typically in tens of MB per MPI rank).

`use_sph_harm_rot (logical)` – Manually activate the `sph_harm_rotation` (spherical harmonic rotation) module (used to evaluate the metric matrix in the 2Dn-1Da scheme). In normal operation this is not necessary, since the module will be activated if it is detected that spherical harmonic rotation is required. Setting this is to false has no effect, since the option will be overridden if ONETEP detects that the module is needed, anyway.

`swx_dbl_grid` – experimental functionality, please do not use.

devel_code values

`SHROT:DEBUG=[T/F]:SHROT` – Activate debug mode for the `sph_harm_rotation` module

`SHROT:UNIT_TEST=[T/F]:SHROT` – Activate unit testing for the `sph_harm_rotation` module

3.10.7 Frequently asked questions

What hybrid functionals are available in ONETEP?

`B1LYP`, `B1PW91`, `B3LYP`, `B3PW91`, `PBE0`, `X3LYP`. For a pure Hartree-Fock calculation use `HF`.

How big can my system be when using HFx?

Up to 200 atoms should be a breeze on a desktop machine (64 GB RAM, 16 cores). About 500-600 atoms will be a limit for a desktop machine, but it might take a week or two. Larger systems will be off-limits because you will either run out of memory (if using > 1 MPI rank), or hit `sparse_mod` integer overflows (if using 1 MPI rank).

On a HPC cluster (say, 640 cores) up to 1000 atoms should not be too difficult (3-4 days). Current record is 4048 atoms (640 cores, 20 days, June 2020). With significant resources ($5000+$ cores) you should be able to do 10000 atoms, but this has not been tried.

How do I make HFx faster?

Use as many OMP threads as possible, without crossing NUMA regions. On a typical HPC system this will mean using only 2 MPI ranks per node, and on a desktop machine – only 1 MPI rank. This will minimise the number of MPI ranks, allowing you to give much more memory to each of them. Increase `hfx_memory_limit` from the default value of 4096 to however much you can spare. This is the maximum RAM consumption of HFx (on top of the rest of ONETEP) per MPI rank. Here it is always the higher the better, except you don't want to run out of memory. Use `ppd_npoints 5 5 5` or `ppd_npoints 7 7 7` for a slight efficiency gain.

I'm running out of memory

Symptoms: you get error messages like "Killed", or "Out of memory: Kill process *nnn* (onetep.exe)" in `dmesg` output, or your system starts *thrashing* (swapping memory to HDD). What you can do:

- Reduce the number of MPI ranks per node. In the worst case scenario, undersubscribe (leave some cores idle), even to the point of having only 1 MPI rank per node.
- Reduce `hfx_memory_limit` from the default value of 4096 to something smaller.
- Reduce the quality of the SW expansion ($l_{\max}$, $q_{\max}$) (this will affect the quality of results).

- If you're running out of memory at the NGWF gradient stage, reduce `threads_num_fftboxes`. The default is equal to the number of OMP threads. Reduce it to 1. This is a component of ONETEP that consumes quite a bit of RAM irrespective of whether HFx is used or not.
- Use high-memory nodes. Many HPC facilities provide these.
- Increase the number of compute nodes used in the calculation. Some of the necessary data will be distributed across nodes, reducing per-node load. With a large number of OMP threads, you can easily use much more CPUs than you have atoms in your system.

The HFx engine does not use all the memory I asked it it to use. Why?

If you devote too much memory to one of the caches, only as much will be used as is needed to store all that is needed. Perhaps adjust `hfx_memory_weights` to give this memory to where it is needed more.

My calculation aborts with `Error in sparse_count_ss: Integer overflow in sparse matrix index detected.` What now?

Your sparse matrices are too large, overflowing integer indices in ONETEP's sparse matrix machinery. Essentially you are trying to run a calculations with too many atoms on too few MPI ranks. Increase the number of MPI ranks or decrease `hfx_cutoff`. The latter will impact results.

My calculation crashes with a `SIGSEGV` and the last line of output is "`KSKS matrix filling:`". What now?

Same as above, except the integer overflow has not been detected.

My calculation aborts with `Exchange matrix not deemed accurate enough for a stable calculation.` What now?

One of the approximations broke down. You turned the "speed vs. accuracy" knob too far towards "speed".

- Is your SW expansion quality too low? The minimum reasonable quality is about $l_{\max} = 2$, $q_{\max} = 8$, see [Dziedzic2013] (Fig. 8). For some systems this might not be enough, particularly in pure Hartree-Fock calculations (`xc_functional HF`). Try $l_{\max} = 3$, $q_{\max} = 10$.
- Is your KE cutoff too low? In general, don't go below 800 eV.
- Is your Chebyshev interpolation quality too low? N_{i} should be at least 10, preferably 12. N_{o} should also be at least 10, preferably 12.
- Is the number of points in the radial Bessel interpolation (`hfx_bessel_rad_nptsx`) too low? Don't go below 100000. For larger simulation cells you might want to use a higher value (say, 250000).

Can I do conduction calculations with HFx?

Yes!

Just make sure all NGWF radii are equal for all species and across `species` and `species_cond` blocks. You may reuse the metric matrices between the valence and conduction calculation. Have a lot of CPU power available. It shouldn't be too difficult up to 500 atoms, then difficulty ramps up. 1000+ atoms will require considerable resources ($\approx 1000+$ cores) and patience (weeks). Current record is 1108 atoms (June 2020).

Can I do LR-TDDFT calculations with HFx?

Yes, but this is at an experimental stage at the moment.

3.10.8 Further questions?

General questions should be directed to Jacek Dziedzic, `J.Dziedzic[-at-]soton.ac.uk`.

Questions relating to the 2Dn-1Da metric matrix evaluation scheme or to hybrid LR-TDDFT should be directed at James C. Womack, `J.C.Womack[-at-]bristol.ac.uk`.

[Hine2011] N. D. M. Hine, J. Dziedzic, P. D. Haynes, and C.-K. Skylaris, *J. Chem. Phys.* **135**, 204103 (2011)

[Dziedzic2013] J. Dziedzic, Q. Hill, and C.-K. Skylaris, *J. Chem. Phys.* **139**, 214103 (2013)

[Rein1973] R. Rein, *On Physical Properties and Interactions of Polyatomic Molecules: With Application to Molecular Recognition in Biology*, in Advances in Quantum Chemistry, ed. P. Lowdin, Academic Press (1973)

[Stone1981] A. J. Stone, *Chem. Phys. Lett.* **2**, 233 (1981)

[Stone1985] A. J. Stone and M. Alderton, *Mol. Phys.* **5**, 56 (1985)

[Stone1998] A. J. Stone, *GDMA: distributed multipoles from Gaussian98 wavefunctions* (technical report), University of Cambridge (1998)

[Stone2005] A. J. Stone, *J. Chem. Theory Comput.* **6**, 1128 (2005)

[Dziedzic2016] J. Dziedzic, Y. Mao, Y. Shao, J. Ponder, T. Head-Gordon, M. Head-Gordon, and C.-K. Skylaris, *J. Chem. Phys.* **145**, 124106 (2016)

[Vitale2015] V. Vitale, J. Dziedzic, S. M.-M. Dubois, H. Fangohr, and C.-K. Skylaris, *J. Chem. Theory Comput.* **11**, 3321 (2015)

3.11 Cut-off Coulomb

Author
Nick Hine, University of Warwick (implementation)

Author
Gabriel Bramley, University of Southampton (documentation)

Date
July 2019

3.11.1 Theory

The plane wave approach inherently involves the use of periodic boundary conditions (PBC). In order to model isolated molecules, the supercell technique was developed, which involves adding large quantities of vacuum to the simualtion cell in order to separate their periodic images. Although this method is adequate for neutral molecules, charged systems and systems with significant multipoles require additional considerations. The potential of the monopole and multipole moments decay in accordance to the power law, where point charges decay with $\frac{1}{r^1}$, dipoles with $\frac{1}{r^2}$ etc.. Using the supercell method for systems with a net charge or significant dipoles require large volumes of vacuum to eliminate the electrostatic interactions between the system of the unit cell and its periodic images. However, as traditional plane wave codes extend across the entire cell, adding large quantities of vacuum is computationally costly. Various dipole corrections such as the cut-off Coulomb (CC) [Jarvis1997], Continuum Screening Method [Otani2006] and Gaussian

Counter Charge model [Dabo2008] have been developed in order to isolate the electrostatic interactions of the unit cell from its periodic images.

The cut-off Coulomb (or Coulomb cut-off) approach, as implemented in ONETEP by Nick Hine [Hine2011], achieves this by only allowing Coulombic interactions within a specified region, thereby emulating the electrostatics of an isolated system with a periodic representation of the charge density [Jarvis1997]. By selecting an appropriate region, one can eliminate spurious electrostatic interactions between charges in the periodic replicas and the home cell, while also retaining the correct description of the potential between charges in the isolated system. One can choose to maintain periodic Coulombic interactions along specified axes, thereby representing electrostatics in systems with either 1D (wire), 2D (slab) or 0D (sphere) periodicity.

3D Periodic Coulomb Interaction

In a standard periodic calculation, the electrostatic potential is defined as through the Hartree potential:

$$V_H(\mathbf{r}) = \int_{space} \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' \tag{3.24}$$

Where $n(\mathbf{r}')$ describes the charge density of the system. This can be equivalently represented as:

$$V_H(\mathbf{r}) = \int_{space} v(\mathbf{r}, \mathbf{r}') n(\mathbf{r}') d\mathbf{r}'$$

Where $v(\mathbf{r})$ represents the Coulomb interaction, $\frac{1}{|\mathbf{r}-\mathbf{r}'|}$. This is more conveniently calculated in reciprocal space, which is achieved through a Fourier transformation of the V_H in accordance with convolution theory:

$$V_H(\mathbf{G}) = v(\mathbf{G}) n(\mathbf{G})$$

In the fully periodic case, the above expressions are taken over all reciprocal space (∞ to $-\infty$), which yields a Coulomb interaction of:

$$v(\mathbf{G}) = \frac{4\pi}{|\mathbf{G}|^2}$$

Coulomb Cut-off in 3D

The standard periodic approach takes the integration of the Coulomb interaction over all space, which allows interaction between the periodic images and the original unit cell. In contrast, Coulomb cut-off sets the Coulomb interaction between charges to zero beyond a specified radius. In the simplest case, one can define this cut-off as a sphere, which assumes a system without periodicity. By selecting an appropriate cut-off radius, R_C, one can retain the correct electrostatic interactions of between charges in the unit cell, while eliminating spurious interactions the periodic images:

$$v^{3D}(\mathbf{r}, \mathbf{r}') = \begin{cases} \frac{1}{|\mathbf{r}-\mathbf{r}'|} & \text{for } R_C < |\mathbf{r} - \mathbf{r}'| \\ 0 & \text{for } R_C > |\mathbf{r} - \mathbf{r}'| \end{cases}$$

Performing an analytic Fourier Transformation of $v(\mathbf{r}, \mathbf{r}')$ with modified boundaries yields the following reciprocal space representation of the Coulomb interaction:

$$v^{3D}(\mathbf{G}) = \frac{4\pi}{\mathbf{G}^2}(1 - \cos(\mathbf{G}R_C))$$

Coulomb Cut-off in 1D

Although this approach is satisfactory for systems with no periodicity, additional considerations must be made if periodicity is required in either 1D (ie. a wire) or in 2D (ie. an infinitely extended plane). In the 1D case, the Coulomb cut-off is defined as a cylinder, where periodicity is retained in the z-axis and the Coulomb interaction is applied in the xy-direction. In theory, this redefines the Coulomb interaction as:

$$v^{1D}(\mathbf{G}_\perp, \mathbf{G}_x) = \frac{4\pi}{\mathbf{G}^2}[1 + \mathbf{G}_\perp R_C J_1(\mathbf{G}_\perp R_C) K_0(\mathbf{G}_x R_C) - \mathbf{G}_x R_C J_0(\mathbf{G}_\perp R_C) K_1(\mathbf{G}_x R_C)] \tag{3.25}$$

Where J and K are modified Bessel functions, $\mathbf{G}_x$, $\mathbf{G}_y$ and $\mathbf{G}_z$ the reciprocal lattice vectors in x, y and z and $\mathbf{G}_\perp = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_z^2}$.

However, a divergence occurs at $G_x = 0$, which is handled by re-casting Equation (3.25) into an analytical expression solved through :

$$v^{1D}(\mathbf{G}_\perp, \mathbf{G}_x = 0) = -4\pi \int_0^R r J_0(\mathbf{G}_\perp) \ln(\mathbf{r}) d\mathbf{r}$$

Coulomb Cut-off in 2D

For systems where periodicity is maintained in 2D, the Coulomb cut-off must only be applied in the out-of-plane direction, while retaining PBC in the xy-plane. Originally, this was implemented in ONETEP from the formulation of *Rozzi et al.* [Rozzi2006], where the Coulomb interaction $v^{3D}(\mathbf{G})$ is re-cast to the following expression:

$$v^{2D}(\mathbf{G}_\parallel, \mathbf{G}_z) = \frac{4\pi}{\mathbf{G}^2}\left[1 + e^{-\mathbf{G}_\parallel R_C}\frac{\mathbf{G}_z}{\mathbf{G}_\parallel}\sin(\mathbf{G}_z R_C) - e^{-\mathbf{G}_\parallel R_C}\cos|\mathbf{G}_z|R_C)\right]$$

Where $\mathbf{G}_\parallel = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$ and $\mathbf{G}_z$ represent the in-plane and out-of-plane reciprocal space vectors respectively. However, as described by *Sohier et al.* [Sohier2017], if $R_C = \frac{L}{2}$, where L represents the length of the simulation cell, this expression simplifies to:

$$v^{2D}(\mathbf{G}_\parallel, \mathbf{G}_z) = \frac{4\pi}{\mathbf{G}^2}\left[1 - e^{-\mathbf{G}_\parallel R_C}\cos(|\mathbf{G}_z|R_C)\right]$$

Where G_z is a multiple of $\frac{2\pi}{L}$. As with the Coulomb interaction under periodic boundary conditions, this term diverges at $\mathbf{G} = 0$, and is therefore treated separately and $v^{2D}(\mathbf{G} = 0) = 0$ as argued by *Sohier et al.* [Sohier2017].

3.11.2 Performing a Calculation with Coulomb Cut-off

To use Coulomb cut-off, the keyword `COULOMB_CUTOFF_TYPE` must be inserted, with the input specifying the periodicity of the system:

- 1D - `COULOMB_CUTOFF_TYPE: WIRE`
- 2D - `COULOMB_CUTOFF_TYPE: SLAB`
- 3D - `COULOMB_CUTOFF_TYPE: SPHERE`

In addition, the length/radius of the cut-off must be specified with either `COULOMB_CUTOFF_RADIUS` or `COULOMB_CUTOFF_LENGTH`:

- 1D & 2D - `COULOMB_CUTOFF_LENGTH`
- 3D - `COULOMB_CUTOFF_RADIUS`

As part of the Coulomb cut-off in ONETEP, the electron density $n(\mathbf{r})$ in the original cell is placed into a larger, padded cell in which $n(\mathbf{r}) = 0$. This is determined in a similar way as the original lattice block through a new block `%BLOCK PADDED_LATTICE_CART`, which determines the size and dimensions of the larger cell: `%BLOCK PADDED_LATTICE_CART a11 a21 a31 a21 a22 a23 a31 a32 a33 %ENDBLOCK PADDED_LATTICE_CART`

This is automatically specified, so adjusting this block is not recommended. The recommended set-ups for calculations of each dimensionality are summarized in the table below:

Coulomb Cut-off Type	Cut-off Length/Radius	Cell Dimensions
		$a_{11}^{pad} = 2a_{11}^{cell}$
Sphere*	$R_C = \sqrt{3}a_{33}^{cell}$	$a_{22}^{pad} = 2a_{22}^{cell}$
		$a_{33}^{pad} = 2a_{33}^{cell}$
		$a_{11}^{pad} = 2a_{11}^{cell}$
Wire**	$R_C = \sqrt{2}a_{33}^{cell}$	$a_{22}^{pad} = 2a_{22}^{cell}$
		$a_{33}^{pad} = a_{33}^{cell}$
		$a_{11}^{pad} = a_{11}^{cell}$
Slab***	$R_C = a_{33}$	$a_{22}^{pad} = a_{22}^{cell}$
		$a_{33}^{pad} = 2a_{33}^{cell}$

Table: The recommended calculation parameters for each periodicity of the Coulomb cut-off, where a_{ii}^{cell} and a_{ii}^{pad} represent the diagonal components of the original simulation cell specified in `% BLOCK_LATTICE_CART` and the padded cell respectively. Assumed orthogonal cell in all cases.

* Assuming $a_{11}^{cell} = a_{22}^{cell} = a_{33}^{cell}$.
** Assuming $a_{11}^{cell} = a_{22}^{cell}$. a_{33} being the periodic direction.
*** a_{33} defined as the non-periodic direction.

These choices of both the padded cell dimension and the cut-off length/radius ensure two conditions are satisfied:

1. Charges within the original unit cell correctly interact with one another.
2. The interaction between charges of the periodic image and the original simulation cell are set to zero.

The first condition is satisfied by setting the cut-off distance equal to or greater than the distance between any two non-zero charges within the original unit cell. For the 3D case, this is typically satisfied by $R_C > \sqrt{3}L_{cell}$, while in 2D and 1D, this is satisfied by $R_C = \frac{L_{cell}}{2}$, where L_{cell} is the cell dimension in the non-periodic axis. The second condition requires that the distance between non-zero charges of the simulation cell and the periodic image must be greater than or equal to the cut-off length. In ONETEP, this is achieved by placing the unit cell inside a larger padded cell, in which the charge density $\rho(\mathbf{r}) = 0$. The second condition is satisfied when the total cell length, $L_{total} = L_{cell} + L_{pad} \geq R_C + L_{cell}$.

[Jarvis1997] M. R. Jarvis, I. D. White, R. W. Godby, and M. C. Payne, *Supercell technique for total-energy calculations of finite charged and polar systems*, Phys. Rev. B, **56**, (1997).

[Otani2006] M. Otani, and O. Sugino, *First-principles calculations of charged surfaces and interfaces: A plane-wave nonrepeated slab approach*, Phys. Rev. B, **73**, (2006).

[Dabo2008] I. Dabo, B. Kozinsky, N. E. Singh-Miller, N. Marzari, *Electrostatics in periodic boundary conditions and real-space corrections*, Phys. Rev. B, **77**, (2008).

[Hine2011] N. D. M. Hine, J. Dziedzic, P. D. Haynes, and C.-K. Skylaris, *Electrostatic interactions in finite systems treated with periodic boundary conditions: Application to linear-scaling density functional theory*, J. Chem. Phys. **135** (2011).

[Rozzi2006] C. A. Rozzi, D. Varsano, A. Marini, E. K. U. Gross, and A. Rubio, *Exact Coulomb cutoff technique for supercell calculations*, Phys. Rev. B **73** (2006).

[Sohier2017] T. Sohier, M. Calandra, and F. Mauri, *Density functional perturbation theory for gated two-dimensional heterostructures: Theoretical developments and application to flexural phonons in graphene*, Phys. Rev. B **96** (2017).

3.12 Species Dependent Scissor Shifts

Author
Nelson Yeung, University of Warwick

Author
Nicholas Hine, University of Warwick

3.12.1 Scissor Hamiltonian

The scissor hamiltonian allows one to apply species-dependent and subspace-dependent energy-level shifts to the hamiltonian, which has the effect of shifting eigenvalues associated with specific layers of the material. One can separately shift the valence and conduction subspaces associated with each layer. This also affects the total energy, so must be applied with great care if you are using the total energy for any purpose. The idea is that a band-alignment correction can be applied to the individual layers of non-covalently-bonded layered materials, though there may well be many other applications as well. It would seem "unwise" at best to apply this approach to different regions of the same molecule or solid which are strongly bonded: results would be unpredictable and likely unphysical. One ideal use would be to correct the alignment of the band-edges of a layered material heterobilayer so that the appropriate heterostructure type was realised, for example straddled-gap rather than broken-gap, using shifts chosen by reference to beyond-DFT accuracy calculations of the individual materials, or from experimental techniques such as ARPES.

In order to apply this shift, we define a scissor Hamiltonian operator as follows:

$$\hat{H}_{\text{scissor}} = |\phi_\eta\rangle K^{\eta\delta}_{\text{shifted}} \langle\phi_\delta| \,,$$

where K_{shifted} is the sum of the species-dependent shifted valence and conduction density kernel, which is defined as

$$K_{\text{shifted}} = \sum_L \left(\sigma_{v,L} K_L + \sigma_{c,L}(S_L^{-1} - K_L)\right) \,.$$

The σ_v and σ_c are the shifts for valence and conduction states, respectively, and the sum is over layer L. The scissor shifted eigenvalues are simply

$$H_{\text{shifted}} = H + SK_{\text{shifted}}S \,.$$

The gradient of the scissor energy with respect to the NGWFs can be calculated using

$$\begin{aligned}
\frac{\partial E_{\text{scissor}}}{\partial \phi^*_\gamma(\mathbf{r})} &= K_n^{\beta\alpha} \frac{\partial}{\partial \phi^*_\gamma(\mathbf{r})} \langle\phi_\alpha|\hat{H}_{\text{scissor}}|\phi_\beta\rangle \\
&= K_n^{\beta\alpha} \frac{\partial}{\partial \phi^*_\gamma(\mathbf{r})} \langle\phi_\alpha|\phi_\eta\rangle K^{\eta\delta}_{\text{shifted}} \langle\phi_\delta|\phi_\beta\rangle \\
&= K^{\beta\delta}_{\text{shifted}} S_{\delta\eta} K_n^{\eta\alpha} \left(\phi_\beta + \sum_{ij} \tilde{p}^i(\mathbf{r}) O_{ij} R^j{}_\alpha\right) \,.
\end{aligned}$$

The approach has been reasonably well tested in the context of LNV calculations. As of April 2019 it has not been validated for EDFT, conduction NGWF optimisation, TDDFT etc, but these would be reasonably expected to work as well.

3.12.2 Performing a Species Dependent Scissor Calculation

To activate the shift, the `species_scissor` block must be present. The user must specify (on separate lines) groups of atom types, with each line finishing with two numbers representing the valence and conduction shifts to be applied to that group of species.

For example, for a MoS_2 / $MoSe_2$ heterobilayer, we might use the following to correct the energies of the individual layers:

```
%block species_scissor
Mo1 S  -0.5 0.5
Mo2 Se -0.1 0.8
%endblock species_scissor
```

This would have the effect of opening the gap of the MoS_2 layer by 1eV and opening the gap of the $MoSe_2$ layer by 0.9 eV, and shifting the alignment of the valence bands by 0.4 eV.

3.13 Embedded Mean-Field Theory

Author
Robert J. Charlton, Imperial College London

Author
Joseph C.A. Prentice, Imperial College London

Date
January 2019

Date
Updated by J.C.A. Prentice October 2021

3.13.1 Embedded Mean-Field Theory (EMFT)

Often in simulations of materials we wish to consider the impact of a host environment on a system of interest, such as chromophores in solvent or doped molecular crystals. While the interesting physics or chemistry may be associated with the subsystem, the effects of the environment can be significant and warrant description at the quantum level of theory. However, the cost of applying accurate quantum methods such as hybrid functionals to potentially large environments can be restricted by the cost of such methods. Quantum embedding [Huang2008], [Gomes2012] methods are intended to combine an accurate, high-level description of the subsystem of interest (active) with a cheaper, low-level method for the host environment.

Embedded mean field theory (EMFT) [Fornace2015] is an approach to quantum embedding based on the one-electron density matrix. To begin, we partition the density matrix into subsystem components,

$$\rho = \begin{pmatrix} \rho_{\mathrm{AA}} & \rho_{\mathrm{AB}} \\ \rho_{\mathrm{BA}} & \rho_{\mathrm{BB}} \end{pmatrix}, \tag{3.26}$$

where A is the active region and B is the inactive environment. The total energy can be written as

$$E[\rho] = \mathrm{tr}[\rho H_0] + G[\rho],$$

where H_0 contains the one-electron terms of the Hamiltonian and $G[\rho]$ contains all two-electron terms (local, Hartree and exchange-correlation effects). In embedded mean-field theory (EMFT), the two-electron interaction for the active subsystem A is constructed at a higher level of theory to the rest of the system,

$$E^{\mathrm{EMFT}}[\rho] = \mathrm{tr}[\rho H_0] + G^{\mathrm{low}}[\rho] + \left(G^{\mathrm{high}}[\rho_{\mathrm{AA}}] - G^{\mathrm{low}}[\rho_{\mathrm{AA}}]\right), \tag{3.27}$$

where G^{low} and G^{high} are the two-electron interaction energies at the lower and higher levels of theory, respectively. For example, the low level theory could be LDA while the higher level uses a hybrid functional such as B3LYP. We assume here that the core Hamiltonian H_0 is the same at both levels of theory, though this need not necessarily be the case. The ground state of the embedded system can thus be obtained by minimising (3.27) with respect to the elements of the density matrix.

Block orthogonalisation

Normalisation is maintained provided the trace of the density matrix with the overlap matrix satisfies

$$\text{Tr}\left[\rho\mathbf{S}\right] = N.$$

EMFT partitioning can result in unrealistic charge spillover from the low-level to the high-level region, producing large negative results for the off-diagonal terms $\text{Tr}\left[\rho_{\text{AB}}\mathbf{S}_{\text{BA}}\right]$ and $\text{Tr}\left[\rho_{\text{BA}}\mathbf{S}_{\text{AB}}\right]$. One possible remedy is to impose a block-orthogonalisation (BO) between the subsystem orbitals [Ding2017],

$$\begin{aligned}
|\tilde{\phi_i^{\text{B}}}\rangle &= \left(1 - \hat{P}^{\text{A}}\right)|\phi_i^{\text{B}}\rangle, \\
\hat{P}^{\text{A}} &= \sum_{j,k\in\text{A}} |\phi_j^{\text{A}}\rangle \left(\mathbf{S}^{\text{AA}}\right)^{\text{-1}}_{jk} \langle\phi_k^{\text{A}}|.
\end{aligned}$$

By construction $\text{Tr}\left[\rho_{\text{AB}}\mathbf{S}_{\text{BA}}\right]$ and $\text{Tr}\left[\rho_{\text{BA}}\mathbf{S}_{\text{AB}}\right]$ are strictly zero and all electrons are associated with the diagonal blocks.

3.13.2 Implementation in ONETEP

Quantum embedding as implemented in ONETEP is based around EMFT [Prentice2020]. Here we denote the active system NGWFs as $|\chi_i^{\text{A}}\rangle$ and the environment NGWFs as $|\phi_j^{\text{B}}\rangle$. The fundamental quantity of interest is the Hamiltonian,

$$\mathbf{H}^{\text{EMFT}} = \begin{pmatrix} \mathbf{H}^{\text{high}}_{\text{AA}} & \mathbf{H}^{\text{low}}_{\text{AB}} \\ \mathbf{H}^{\text{low}}_{\text{BA}} & \mathbf{H}^{\text{low}}_{\text{BB}} \end{pmatrix},$$

where the high- and low-level Hamiltonian operators are given as

$$\begin{aligned}
\hat{H}^{\text{high}} =& \hat{T} + \hat{V}_{\text{local}} + \hat{V}_{\text{Hartree}} + \hat{V}^{\text{high}}_{\text{XC}}, \\
\hat{H}^{\text{low}} =& \hat{T} + \hat{V}_{\text{local}} + \hat{V}_{\text{Hartree}} + \hat{V}^{\text{low}}_{\text{XC}}.
\end{aligned}$$

The total energy can thus be found by minimising the quantity

$$E^{\text{EMFT}} = \min_{\{K^{\alpha\beta}\},\{\chi_\alpha\}} \text{Tr}\left[\mathbf{K}\mathbf{H}^{\text{EMFT}}\right],$$

with respect to the NGWFs and elements of the density kernel $\mathbf{K}$, using the conventional methods available in ONETEP. The Hamiltonian is constructed as follows,

1. The total electron density $n(\mathbf{r})$ is constructed from the full system NGWFs and kernel, from which $V^{\text{low}}_{\text{XC}}(\mathbf{r})$ is calculated.
2. The active subsystem density $n^{\text{AA}}(\mathbf{r})$ is constructed using the subsystem terms and the subsystem XC potentials $V^{\text{low,A}}_{\text{XC}}(\mathbf{r})$ and $V^{\text{high,A}}_{\text{XC}}(\mathbf{r})$ calculated.
3. Final EMFT potential can be written as

$$V^{\text{high}}_{\text{XC}}(\mathbf{r}) = V^{\text{low}}_{\text{XC}}(\mathbf{r}) + \left(V^{\text{high,A}}_{\text{XC}}(\mathbf{r}) - V^{\text{low,A}}_{\text{XC}}(\mathbf{r})\right).$$

   with which we can construct the high-level Hamiltonian.

Although block orthogonalisation is found to work when just the density kernel is being optimised, it does not do so generally for the optimisation of the NGWFs. Because of this, there is an option to optimise the NGWFs at the lower level of theory first (in all regions), and then fix them for an optimisation of the kernel under EMFT.

If you would like to use hybrid functionals with embedding, there are two things to bear in mind. Firstly, only hybrid-in-semi local DFT calculations are currently supported – hybrid-in-hybrid calculations are not possible. Secondly, the species in the `species_swri-[swri name]` block must match the species in the active region exactly. Anything else will give incorrect results. Otherwise, the set-up of the hybrid functional calculation is identical to a normal ONETEP calculation.

LR-TDDFT calculations can be performed with embedding (TD-EMFT) [Ding2017-2], and this is also implemented within ONETEP [Prentice2022]. If you would like to perform a TD-EMFT calculation, it may be advisable to restrict the excitations to the active region, using the `species_tddft_kernel` block.

It is also possible to place the quantum embedding system within implicit solvent, giving multi-level embedding capability [Prentice2022]. This should work very similarly to standard implicit solvent calculations, although there are a couple of additional keywords (see below). The main difference is whether the cavity is constructed using the density kernel optimised solely at the low level of theory, or optimised using EMFT. This only makes a difference if the active region is close to the edge of the cavity.

3.13.3 Keywords

- `species_ngwf_regions` (block): This block defines which species are in which region. Each line of the block corresponds to a distinct region. The species within each region do not necessarily have to be physically next to one another. If this block is not defined, it is assumed that there is only one region, containing all the species in the system.
- `do_fandt` (logical): Controls whether a freeze-and-thaw (F+T) optimisation of the NGWFs is performed or not. This is a cruder form of embedding, where all regions are treated at the same level of theory, but each region's NGWFs are optimised in turn, with the others frozen. Default `F`.
- `freeze_switch_steps` (integer): How many NGWF CG optimisation steps should be spent on each region before moving onto the next in a F+T calculation. `maxit_ngwf_cg` represents the total number of NGWF optimisation steps across all regions. A value less than 0 means that all NGWFs are optimised together i.e. no F+T takes place. Default `-1`.
- `use_emft` (logical): Controls whether an EMFT calculation is performed, as described above. Default `F`.
- `active_region` (integer): Defines which region is the active region – 1 means the species on the first line in the `species_ngwf_regions` block constitute the active region, 2 means the second line, and so on. Default `1`.
- `active_xc_functional` (string): Defines what functional is used as the higher level of theory within EMFT. Default is the value of `xc_functional` i.e. no difference between the regions.
- `freeze_envir_ngwfs` (logical): Controls whether the environment NGWFs should ever be optimised or not. Default `F`.
- `use_emft_follow` (logical): Controls whether the EMFT calculation is only performed after a regular calculation, so the NGWFs are optimised at the lower level of theory first, before applying EMFT. Default `F`.
- `use_emft_lnv_only` (logical): Controls whether only the kernel is optimised within EMFT, with the NGWFs optimised at the lower level of theory and then fixed. Usually used in conjunction with `use_emft_follow`. Default `F`.
- `emft_lnv_steps` (integer): Controls the number of LNV kernel optimisation steps to be used in conjunction with `use_emft_lnv_only`. Default `10`.
- `block_orthogonalise` (logical): Controls whether the environment NGWFs are orthogonalised with respect to the active region NGWFs, as described above. Default `F`.

- parallel_scheme (string): Defines the parallel scheme used for the calculation. See Appendix for more information. Default NONE.
- read_sub_denskern (logical): Controls whether only diagonal blocks of the density kernel are read in when restarting. This is useful for starting an embedding calculation from two separate calculations on the individual regions, so you only have the diagonal blocks of the density kernel. Default F.
- embed_debug (logical): Turns on verbose printing for debugging of embedding functionalities. Default F.
- is_restart_vac_from_vac (logical): Decides whether the vacuum calculation in an autosolvation implicit solvent calculation should be restarted from the vacuum_ files or not. Useful for restarting autosolvation calculations if they time-out or similar. Default F.
- is_emft_cavity (logical): Decides whether the cavity used in implicit solvent calculations is determined using the low-level density kernel (F), or the EMFT-optimised density kernel (T). Default F.

The most reliable way to run EMFT calculations is to have use_emft, use_emft_follow, use_emft_lnv_only and block_orthogonalise all set to T. These can be set to F (most sensibly in reverse order i.e. block_orthogonalise first), but the calculation may become more unstable, depending on the system, the regions chosen and the functionals chosen.

3.13.4 Example input file

```
!=====================================================!
! Input for calculation with the ONETEP program      !
!                                                     !
! O2 and H2 form the embedded system to be treated   !
! at the higher level of theory, O1 and H1 are the   !
! environment treated at the low-level.              !
!=====================================================!

%block species_ngwf_regions
O2 H2
O1 H1
%endblock species_ngwf_regions

task: SINGLEPOINT
cutoff_energy 1000 eV
write_forces: T
xc_functional: LDA
active_xc_functional: PBE

use_emft: T
use_emft_follow: T
use_emft_lnv_only: T
block_orthogonalise : T
parallel_scheme: HOUSE

%block species_atomic_set
H1 "SOLVE"
O1 "SOLVE"
H2 "SOLVE"
O2 "SOLVE"
%endblock species_atomic_set
```

(continues on next page)

(continued from previous page)

```
%block species
H1 H 1 1 7.0
O1 O 8 4 7.0
H2 H 1 1 7.0
O2 O 8 4 7.0
%endblock species

%block species_pot
H1 "pseudo/hydrogen.recpot"
O1 "pseudo/oxygen.recpot"
H2 "pseudo/hydrogen.recpot"
O2 "pseudo/oxygen.recpot"
%endblock species_pot

%block lattice_cart
     30.000000000       0.000000000       0.000000000
      0.000000000      30.000000000       0.000000000
      0.000000000       0.000000000      30.000000000
%endblock lattice_cart

%block positions_abs
O1       16.203224001     15.100000000     11.536063353
H1       15.100000000     15.100000000     10.100000000
H1       15.100000000     15.100000000     12.991451046
O2       12.600158789     15.100000000     17.306583960
H2       13.051873252     13.656398529     18.308114239
H2       13.051873252     16.543601471     18.308114239
%endblock positions_abs
```

3.13.5 Interaction with other functionalities

Fully tested

- Energy and forces calculations
- Hybrid-in-semi local DFT
- Restarting calculations
- LR-TDDFT
- Implicit solvent

Should work, not thoroughly tested

- Geometry optimisation
- Finite displacement phonons
- Molecular dynamics
- Conduction NGWF optimisation
- Ensemble DFT
- Kernel DIIS
- QNTO
- NAO
- Cutoff Coulomb
- Spin polarised calculations
- Some properties calculations (eigenstates, Mulliken charges, plotting, DoS)

Not compatible with embedding

- Hubbard calculations
- DMFT
- PAW
- cDFT
- Bandstructure calculations
- DMA
- EDA
- Electronic transport
- Hybrid-in-hybrid DFT
- NEB
- EELS
- Polarisable embedding
- Transition state searching
- DDEC

Any functionalities missed above are likely to not work with embedding.

3.13.6 Appendix: Parallel strategies with embedding

In a normal ONETEP calculation, atoms are distributed across the available MPI processes according to a 'parallel strategy'. This determines how resources such as matrix elements will be spread across the MPI environment in order to reduce the communication between nodes and maximise the efficiency of the calculation. Details on maximising parallel efficiency are available via the ONETEP documentation and website.

As part of the embedding infrastructure, each subsystem is given its own parallel strategy. This contains all information relating to the distribution of resources across the MPI nodes available to the calculation, which are determined by the parameter `PARALLEL_SCHEME`. There are three settings for the distribution of resources during an embedding calculation:

- `NONE`: All subsystems are treated completely independently, with atoms distributed across all available processors as though the other subsystems do not exist. The number of MPI processes cannot be greater than the number of atoms in the smallest subsystem. For example, if there are 8 processors available then each will hold atoms and data from all subsystems, though the calculation will fail if any subsystem has less than 8 atoms (or possibly slightly more if the space-filling curve is in use). This is the default setting for testing but is not recommended for practical calculations due to the constraint on the number of processors.
- `SENATE`: Nodes are partitioned evenly between all subsystems. For example, if there are 8 processors and 2 subsystems, then each will be allocated 4 processors, regardless of the number of atoms in each subsystem. Unlike the `NONE` setting, there is no upper bound on the number of processors which may be used, so user discretion is advised.
- `HOUSE`: Divides the processors proportionally between all subsystems, with a minimum of 1 processor per subsystem. For example, if we have two subsystems consisting of 15 and 5 atoms each, then with 8 processors each subsystem will be allocated 6 and 2 nodes respectively. At a minimum all subsystems are granted 1 processor — if we had two subsystems with 1 and 100 atoms in our 8 processor example, then they will receive 1 and 7 processor respectively. Like `SENATE`, there is no upper bound on the number of processors that can be allocated and finding a sensible setting is left to the user.

`HOUSE` is the recommended setting for running calculations, the others are mainly of use for testing. Since they should all produce the same results, any significant differences may be a sign of an underlying problem, so comparing them is a useful consistency check.

[Ding2017-2] F. Ding, T. Tsuchiya, F. R. Manby and T. F. Miller, *J. Chem. Theory Comput.*, **13**, 4216–4227, (2017).

[Prentice2020] J. C. A. Prentice, R. J. Charlton, A. A. Mostofi and P. D. Haynes, *J. Chem. Theory Comput.*, **16**, 354–365, (2020).

[Prentice2022] J. C. A. Prentice, *J. Chem. Theory Comput.*, **18**, 1542-1554 (2022).

[Huang2008] P. Huang and E. M. Carter, *Annu. Rev. Phys. Chem.*, **59**, 261–290, (2008).

[Gomes2012] A. S. P. Gomes and C. R. Jacob, *Annu. Rep. Prog. Chem., Sect. C: Phys. Chem.*, **108**, 222–277, (2012).

[Fornace2015] M. E. Fornace, J. Lee, M. Kaito, F. R. Manby, T. F. Miller, *J. Chem. Theory Comput.*, **11**, 568–580, (2015).

[Ding2017] F. Ding, F. R. Manby and T. F. Miller, *J. Chem. Theory Comput.*, **13**, 1605–1615, (2017).

CHAPTER

FOUR

CORRELATION AND CONSTRAINED DFT

4.1 DFT+U(+J)

Author
David D. O'Regan, Trinity College Dublin

Date
July 2015

DFT+U is fully and self-consistently implemented in ONETEP, together with a number of advanced ancillary functionalities. The method is linear-scaling with respect to system size, exhibiting no systematic tendency to slow convergence to the ground-state. DFT+U in its conventional fixed-projector form introduces only a small increase in computational pre-factor with respect to the underlying exchange-correlation functional [O-Regan2012].

PLEASE NOTE: Seven columns are now required in the Hubbard block in order to allow for $+J$ calculations. Older input files with six columns will not yield incorrect results, but the code will exit.

4.1.1 A very short introduction to DFT+U

DFT+U [Anisimov1991], [Anisimov1997], [Dudarev1998], also known as LDA+U or LSDA+U, is a method used to improve the description of so-called strongly correlated materials offered by DFT within conventional approximations for exchange-correlation (XC) such as the LSDA and σ-GGA, quantitatively and even qualitatively. These functionals, based on the locally-evaluated density and its gradients, can sometimes fail to reproduce the physics associated with localised orbitals of $3d$ and $4f$ character characteristic of conventionally-classed strongly correlated materials, a category consisting of not only first-row transition metals and their oxides, but also lanthanoid oxide materials, and other materials such as certain magnetic semiconductors and organometallic molecules.

Typically, the LDA and its extensions underestimate local magnetic moments and the tendency to favour high-spin ground-states in such materials, and the insulating gap in cases where it is related to electron localisation. Underestimation of the gap due to the absence, in the LDA, of the derivative discontinuity with respect to orbital occupancy in the exact XC-funtional may be confounded by an underestimation of the exchange splitting induced by local magnetic moments.

The DFT+U correction term is usually thought of as an explicit mean-field treatment of the exchange-correlation energy contributed by the correlated sites (subspaces projected out with functions of $3d$ and or $4f$ character) within the Hubbard model, including a double-counting correction for that contribution already included in the LDA term. The flavour implemented in ONETEP is the basis-set independent, rotationally invariant quadratic penalty functional of Ref [Cococcioni2005], defined by the additive energy correction

$$E_{DFT+U}\left[n^{(I)(\sigma)}\right] = \sum_{I\sigma} \frac{U^{(I)}}{2} \mathrm{Tr}\left[\mathrm{n}^{(\mathrm{I})(\sigma)}\left(1-\mathrm{n}^{(\mathrm{I})(\sigma)}\right)\right].$$

Here, U is an estimate of the scalar screened density-density Coulomb repulsion between localised orbitals. The occupancy matrix of the correlated site I, for spin channel σ, is defined, in the case of orthonormal projector functions $\{|\varphi_m^{(I)}\rangle\}$, and density-matrix $\hat{\rho}^{(\sigma)}$, by

$$n_{mm'}^{(I)(\sigma)} = \langle\varphi_m^{(I)}|\hat{\rho}^{(\sigma)}|\varphi_{m'}^{(\sigma)}\rangle.$$

Put simply, if the system under study comprises open $3d$ or $4f$ sub-shells, then there is a good chance that the LDA will find a minimum energy by partly occupying and leaving degenerate the Kohn-Sham orbitals strongly overlapping with these states, rather than splitting them into occupied and virtual Hubbard bands. This leads to an underestimation of the insulating gap and any associated magnetic order. In this case, the DFT+U method can be used to penalise the non-integer occupancy of these orbitals, tending to fill states with occupancy greater than 0.5 and empty states with occupancy less than 0.5, as can be seen from the expression for the DFT+U potential

$$\hat{V}_{DFT+U}^{(\sigma)} = \sum_I U^{(I)}|\varphi_m^{(I)}\rangle \left(\frac{1}{2}\delta_{mm'} - n_{mm'}^{(I)(\sigma)}\right)\langle\varphi_{m'}^{(I)}|.$$

The DFT+U term may be considered as a correction which cancels the contribution to the energy arising due to the spurious self-interaction of a partially occupied orbital [Cococcioni2005]. In this case, the U parameter is the curvature of the total energy with respect to the occupancy of the correlated manifold - which should be a piece-wise linear curve were Janak's theorem satisfied [Janak1978] – which can be computed using linear-response theory (among other methods such as constrained DFT) according to the prescription given in Refs. [Cococcioni2005], [Kulik2006].

4.1.2 How to activate DFT+U in ONETEP

In order to activate the DFT+U functionality, the **hubbard** block is added to the input file. For example, in the case of a system containing iron and cerium atoms incorrectly described by the exchange-correlation functional, which we suspect could benefit from the DFT+U correction to improve the description of localisation, we might use the `hubbard` block:

```
% block hubbard
  Fe1   2   4.0   0.0  -10.0   0.00   1.0
  Fe2   2   4.0   0.0  -10.0   0.00  -1.0
  Ce1   3   6.0   0.0  -10.0   0.50   0.0
% endblock hubbard
```

The columns of the `hubbard` block are described as follows:

1. The species label, e.g. $Fe1$ for iron atoms of a first type in the cell, $Fe2$ for iron atoms of a second kind, etc. Only the species to which orbitals to be corrected by DFT+U are assigned should be listed in this block.
2. The angular momentum channel of the projectors used to delineate the strongly correlated sites on Hubbard atoms this type, e.g. $l = 2$ for $Fe1$. Conventionally, the radial quantum number $r = l + 1$ is used to generate atom-centred atomic projectors, so that $l = 2$ gives $3d$ orbitals, $l = 3$ gives $4f$ orbitals etc. (please get in contact if you need to use a $r \neq l + 1$ combination, or multiple sub-shells per atom).
3. The value of the Hubbard U for this sub-shell, in electron-volts. Most users will simply work with the value for U that they find corrects the band-gap or bond-lengths in the system they wish to study. Methods do, however, exist to estimate its value, for example the linear-response technique [Cococcioni2005], [Kulik2006], which is implemented in ONETEP.
4. The value of the Hund's exchange J for this sub-shell, in electron-volts. The rotationally invariant exchange corrective term described in detail in Ref. [Himmetoglu2011] is fully implemented in ONETEP (including forces etc), and activated for any $J \neq 0$.
5. This number Z selects how the radial part of the projector functions used to describe the $1s$, $2p$, $3d$ or $4f$ atomic orbitals entering the DFT+U functional are defined. In the case that $\mathbf{Z} < \mathbf{0}$, a subset of the orbitals generated by

solving the atomic problem subject to the pseudopotential for the species in question are chosen (in which case the projectors form a subset of the initial guesses for the ONETEP NGWFs); here the magnitude of the negative Z makes no difference. In the case that $\mathbf{Z} > \mathbf{0}$, for more advanced users, this number is the effective charge divided by the ratio of effective masses used to generate projectors in the form of solutions to the hydrogenic Schrödinger equation. A good guess for this number might be the Clementi-Raimondi effective charge, tabulated in Refs. [Clementi1963], [Clementi1967], and the choice of radial profile does matter [O-Regan2010]. In both cases, the projectors are effectively renormalised within an atom-centred sphere with the same radius as the NGWFs on that atom.

6. An additional potential acting on the subspace in question, the prefactor α is here entered in electron-volts. This is needed, for example, in order to locally vary the potential in order to determine the value of U which is consistent with the screened response in the system with linear-response theory [Cococcioni2005], [Kulik2006], or to break a spatial symmetry, such as in a mixed-valence system. In the example given, we are additionally penalising the occupancy on cerium $4f$ atomic orbitals.

7. The spin-splitting factor, in electron-volts, which is deducted from the α factor for the spin-up channel and added to α for the spin-down channel. In the example shown here we're promoting spin-up magnetisation for iron atoms $Fe1$, and spin-down for $Fe2$. This can be very useful for appropriately breaking magnetic symmetries in antiferromagnetic solids or open-shell singlet molecules, or for estimating the magnetic susceptibility or exchange coupling.

   N.B. Users may find the DFT+U functionality useful in cases of systems even when the DFT+U correction is not needed (setting the all U parameters to zero does not disable the functionality). The implementation offers a very inexpensive method for carrying out carefully-defined atom-centred atomic population analysis, or breaking symmetries in spin or charge ordered systems.

4.1.3 Compatibility

The DFT+U functionality is fully compatible with almost all other parts of the ONETEP code, such as listed below, since it simply involves an additional term in the Hamiltonian and ionic forces. Please get in touch first if you would like to use a more exotic combination of these functionalities:

1. Total-energy minimisation and ionic forces
2. Geometry optimisation, molecular dynamics and phonon calculations
3. All other functionals including hybrids and Van der Waals functionals
4. Implicit solvation
5. The PAW formalism and ultrasoft pseudopotentials
6. Constrained DFT
7. Local density of states (including a correlated subspace decomposition)
8. Natural bond orbital calculations
9. Conduction-band optimisation and Fermi's Golden Rule spectra
10. Calculations of changes in electric polarisation
11. Time-dependent DFT
12. Electronic transmission calculations

The extension of the DFT+U implementation to cluster Dynamical mean-field theory has also been implemented in ONETEP; for an example of its capabilities see Ref. [Weber2012].

4.1.4 Using NGWFs and projector self-consistency

Any reasonable set of localised atomic-like functions may, in principle, be used for the projectors defining the correlated subspaces in DFT+U; the choice is somewhat arbitrary and the description "atomic orbitals" does not uniquely define them. One possible approach is to use Wannier functions for the Kohn-Sham orbitals, so that the correlated subspaces are proper subspaces of the Kohn-Sham Hilbert space. Indeed, there is numerical evidence to suggest that Maximally Localised Wannier Functions (MLWFs) [Marzari1997], [Souza2001], in particular, provide a basis that maximises a particular measure of the on-site Coulomb repulsion [Miyake2008], and MLWFs are in common use as a minimal basis with which to construct tight-binding models from first-principles.

In ONETEP, a set of variationally-optimised nonorthogonal generalised Wannier functions (NGWFs) are generated as a by-product of total-energy minimisation. NGWFs exhibit some similar properties to MLWFs and other flavours of localised Wannier functions, and, for example, can be used to calculate finite-difference response properties in a similar way [O-Regan2012-2]. As they are conveniently available in ONETEP, we have made it possible to re-use the NGWFs from the end of a ground-state calculation as a set of Hubbard projectors with which to define the DFT+U correction. For this, it was necessary to develop a tensorially-consistent formulation of DFT+U in order to accommodate nonorthogonal projector functions [O-Regan2011]; projector nonorthogonality for a given subspace is automatically compensated for.

In order to ensure that NGWFs with appropriate symmetry are chosen as Hubbard projectors for a given atom, those n NGWFs $|\phi_\alpha\rangle$ that maximise $\sum_{m,\alpha}^{n}\langle\varphi_m|\phi^\alpha\rangle\langle\phi_\alpha|\varphi_m\rangle$, for a given set of n hydrogenic orbitals $|\varphi_m\rangle$, defined in the `hubbard` block, are selected for the task. The keyword `hubbard_max_iter`, (defaulting to 0), sets the task to `HUBBARDSCF`, which performs a self-consistency cycle over the Hubbard projectors, demonstrated in Refs. [O-Regan2010], [O-Regan2011]. The density from one minimisation is re-used at the beginning of the next, and setting `hubbard_max_iter` to 2 one can carry out a DFT+U calculation using the LDA NGWFs as projectors.

The keywords `hubbard_energy_tol`, `hubbard_conv_win`, and `hubbard_proj_mixing` are used to manage the Hubbard projector self-consistency cycle. For convergence, the ground state energy must deviate less than `hubbard_energy_tol` (defaulting to $10^{-8}Ha$) from one `HUBBARDSCF` iteration to the next, over `hubbard_conv_win` (defaulting to 2) iterations. A fraction `hubbard_proj_mixing` (defaulting to 0.0) of the previous Hubbard projectors may be mixed with the new ones in order to accelerate the procedure, although this has never been found to be necessary. Setting `hubbard_proj_mixing` to a negative value causes the projectors to be read in from a `.tightbox_hub_projs` file, for restarting a `HUBBARDSCF` calculation or for a variety of post-processing tasks.

[O-Regan2012] D. D. O'Regan, N. D. M. Hine, M. C. Payne and A. A. Mostofi, Phys. Rev. B **85**, 085107 (2012).

[Anisimov1991] J. Z. V. I. Anisimov and O. K. Andersen, Phys. Rev. B **44**, 943 (1991).

[Anisimov1997] V. I. Anisimov, F. Aryasetiawan, and A. I. Liechtenstein, J. Phys.: Condens. Matter **9**, 767 (1997).

[Dudarev1998] S. L. Dudarev, Phys. Rev. B **57**, 3 (1998).

[Cococcioni2005] M. Cococcioni and S. de Gironcoli, Phys. Rev. B **71**, 035105 (2005).

[Janak1978] J. F. Janak, Phys. Rev. B **18**, 12 (1978).

[Kulik2006] H. J. Kulik, M. Cococcioni, D. A. Scherlis and N. Marzari, Phys. Rev. Lett. **97**, 103001 (2006).

[Himmetoglu2011] B. Himmetoglu, R. M. Wentzcovitch, and M. Cococcioni, Phys. Rev. B,**84**, 115108 (2011).

[Clementi1963] E. Clementi and D.L. Raimondi, J. Chem. Phys. **38**, 2686 (1963).

[Clementi1967] E. Clementi, D.L. Raimondi, and W.P. Reinhardt, J. Chem. Phys. **47**, 1300 (1967).

[O-Regan2010] D. D. O'Regan, N. D. M. Hine, M. C. Payne and A. A. Mostofi, Phys. Rev. B **82**, 081102 (2010).

[Weber2012] C. Weber, D. D. O'Regan, N. D. M. Hine, M. C. Payne, G. Kotliar and P. B. Littlewood, Phys. Rev. Lett. **108**, 256402 (2012).

[Marzari1997] N. Marzari and D. Vanderbilt, Phys. Rev. B **56**, 12847 (1997).

[Souza2001] I. Souza, N. Marzari and D. Vanderbilt, Phys. Rev. B **65**, 035109 (2001).

[Miyake2008] T. Miyake and F. Aryasetiawan, Phys. Rev. B **77**, 085122 (2008).

[O-Regan2012-2] D. D. O'Regan, M. C. Payne, and A. A. Mostofi, Phys. Rev. B **85**, 193101 (2012).

[O-Regan2011] D. D. O'Regan, M. C. Payne and A. A. Mostofi, Phys. Rev. B **83**, 245124 (2011).

4.2 Constrained Density Functional Theory (cDFT)

Author
Gilberto Teobaldi, University of Liverpool (g.teobaldi@liv.ac.uk)

Date
November 2013

4.2.1 cDFT input check-list

This is a short check-list meant to help the setting up of a constrained-DFT (cDFT) simulation. ONETEP implements some rather extensive check of the input (as specified in .dat file). In spite of this, users may be still capable of creating erroneous inputs which I could not think of. If you experience disaster (commutators larger than 1.E-3, NWGFs- and cDFT-optimisation stuck at the same value of the gradient for many iterations, or NaN (Not A Number) and '***' outputs from ONETEP), please do get in touch.

Can I start by running a cDFT geometry-optimisation and/or Molecular Dynamics straightaway?

Currently, this is **not** possible. ONETEP will not allow you to run a geometry-optimisation/molecular dynamics simulation unless a .cdft file (containing the cDFT-potentials, Uq/s) is present. This is meant to force you to first make sure that your cDFT-settings let you obtain a good convergence in a single-point calculation for the system and constrained solution you are interested in.

What projectors am I telling ONETEP to use?

Is cdft_read_projector=F (this is the default) and cdft_multi_proj=F (this is the default) in your input?

If so, you will be using as cDFT-projectors the orbitals, of angular momentum *L-projectors*, of a hydrogenic atom of atomic number *Z-projector*, provided *Z-projector* is positive in the %block constrained_dft (Z>0). Conversely, if you have entered a negative *Z-projector* value in the %block constrained_dft (Z<0), you will be using the numerical orbital (of angular momentum *L-projectors*) for the given cDFT-atom as cDFT-projectors.

Is cdft_multi_proj=T (this is NOT the default) in your input?

If so, you will be using as cDFT-projectors the numerical orbitals, obtained from the pseudo-atom solver. You will be using as many cDFT-projectors as NGWFs for the given cDFT-site. Thus, to know the number and angular momentum of your cDFT-projectors, you need to refer to the specific settings of the cDFT-site in the %block species_atom_set and the relevant output in the standard .out file.

Is cdft_read_projector=T in your input?

If so, you will use the projectors stored in the .tightbox_hub_projs file (or experience a crash if this file is not present). Mind that the file will contain different classes of projectors (hydrogenic orbitals, numerical orbitals, self-consistent projectors) depending on the settings you used for the cDFT-run which generated the .tightbox_hub_projs file. This means that, unless one wants to experience a crash, the projectors in the .tightbox_hub_projs file should have been generated with the same entries for *Z-projector and L-projector* in %block constrained_dft **OR** cdft_multi_proj and %block species_atom_set as in the current calculation.

Is this the first single-point cDFT submission or do I want to restart a single-point calculation?

For submission from scratch the user can chose between starting the cDFT-run from previously obtained DFT-NGWFs and Density Kernel (DKN), by setting (read_denskern=T and read_tightbox_ngwfs=T), **or not** (read_denskern=F and read_tightbox_ngwfs=F).

To restart a cDFT-single point run (from the latest Uq/s-potentials) or to run cDFT-geometry optimisations/Molecular Dynamics it is necessary to activate cdft_continuation=T. **Mind** that if, instead of hydrogenic or numerical orbitals as cDFT-projectors (see above) one wants to use/keep using pre-optimised projectors (stored in .tightbox_hub_projs file) it is necessary to activate also cdft_read_projector=T in a restart.

Are the columns of the %block constrained_dft requesting the right targets for the given cDFT-flavour?

MIND that, as explained in the description of the keywords, different columns of the %block constrained_dft are used depending on the selected cDFT-modes (cdft_atom_charge, cdft_atom_spin, cdft_group_charge/spin_acceptor/donor, cdft_group_charge/spin_diff). Are you requesting the right targets in terms of atomic-population, group-populations, atomic magnetic-moments and group magnetic-moments?

Mind also that the cdft_group_charge/spin_acceptor/donor, cdft_group_charge/spin_diff cDFT-modes require that targeted population, magnetic-moment and differences are entered explicitly with the corresponding _TARGET keyword. If you have forgotten entering the relevant _TARGET keyword in your input, the simulation will stop and ONETEP will tell you about it.

Have I set CDFT_GROUP_CHARGE_UP/DOWN_ONLY=T?

Remember that, for CDFT_GROUP_CHARGE_UP/DOWN_ONLY=T, only the corresponding spin-channel will be constrained in cdft_group_charge_acceptor/donor and cdft_group_charge_diff runs. Accordingly, you should target populations for one-spin channel only (**not** for the UP+DOWN channels).

Is the sign of $U_{q/s}$ correct for the group_charge/spin_acceptor/donor/diff run?

Remember that atoms in acceptor- and donor-group are identified by mean of the sign of the Uq/s in the %block constrained_dft. Thus, **acceptor-group atoms** need to be assigned negative (e-attractive) $U_{q/s}$ (**U:sub:`q/s`<0**), whereas **donor-group atoms** need to be assigned positive (e-repulsive) $U_{q/s}$ (**U:sub:`q/s`>0**).

Are my constraints compatible with the spin (=N_{UP}-N_{DOWN}) keyword in the input (.dat) file?

Remember that ONETEP optimises the Density Kernel keeping the number of UP (N_{UP}) and DOWN (N_{DOWN}) electrons fixed. As a result, the net magnetization of the system (N_{UP}-N_{DOWN}) is also fixed. Have you introduced any incongruence between the spin=0,1,2,etc keyword and the cDFT-ones in your input? If your targeted cDFT-solution results in a high-value (i.e. > 0 μ_B) magnetization of the **total** system, the spin keyword should reflect it.

What values of min/maxit_lnv, maxit_ngwf_cg and maxit_cdft_u_cg should I use?

Based on the tests run so far, and considering that the cDFT-potentials (Uq/s) are iteratively optimised at each (1:sup:*st*) step of the NGWFs-optimisation line-search, the following choices seem to be reasonable:

```
maxit_palser_mano : -1
kerfix : 2
maxit_pen : 0
minit_lnv : 5
maxit_lnv : 10
maxit_ngwf_cg : 60
lnv_check_trial_steps : T
lnv_threshold_orig : 1.0e-10
maxit_cdft_u_cg : 5
```

Increasing maxit_cdft_u_cg above 5 is hardly going to accelerate the convergence of the cDFT-run (the NGWFs will change at the following step of NGWFs-optimisation), decreasing minit_lnv below 5 might be risky.

To what value should I initialise the cDFT-potentials (Uq/s)?

In the current implementation, unless cdft_guru=T, in which case the Uq/s entered in the %block constrained_dft will be used, the absolute value of the Uq/s cDFT-potentials are internally initialised to 1 eV (their original sign is, of course, maintained). For spin-excitation |Us|=1eV may be too large, and initialising the |Us| with 0.1-0.3 eV may accelerate convergence (this has been tested only on triplet excitations in benzene dimers).

What is the difference between a cdft_atom_charge run and a cdft_group_charge_acceptor/donor one with one-atom group?

Whereas the cdft_atom_charge=T mode allows independent (i.e. potentially different) constraining potentials to be applied to the UP and DOWN spin-channels, for **one-atom** cdft_group_charge_acceptor/donor=T the same Uq will be applied to both the UP and DOWN spin-channel. Activation of cdft_group_charge_up/down_only=T in cdft_group_acceptor/donor modes allows to optimise Uq for only a spin-channel, leaving the other spin-channel unconstrained.

Have I chosen a meaningful cdft_cg_max for the cDFT-mode I wish to use?

For cDFT-runs with only one cDFT-group in the system (cdft_group_charge/spin_acceptor/donor modes) and group_charge/spin_diff runs, **only one** cDFT-potential (Uq/s) will be optimised in the cDFT-loop. Accordingly, for these cases it is recommended to perform the cDFT-optimisation via a steepest descendent algorithm (cdft_cg_max=1).

How do I obtain the population of the cDFT-sites for a standard DFT-run?

Performing a single-point (task=singlepoint) fixed-Uq/s (maxit_cdft_u_cg=0) cDFT-run using very small (e.g. 1.E-60) Uq/s-potentials in the %block constrained_dft and setting output_detail=VERBOSE will result in the cDFT-population of all the cDFT-sites being printed in the standard output (.out) file. To obtain atom-specific (instead of atomic_species-specific) information on the population of the cDFT-sites, it is necessary to set CDFT_PRINT_ALL_OCC=T.

Mind. The population of a given cDFT-site depends critically on the projector used. Make sure you decide your cDFT-targets from the DFT-populations obtained with the same set of projectors!

How do I optimise self-consistently the projectors for a given geometry?

In analogy with DFT+U simulations, self-consistent optimisation of the cDFT-projectors (for a fixed geometry) is activated by setting task=HUBBARDSCF in the .dat file. It is recommended to start the task=HUBBARDSCF run from pre-optimised PAO-cDFT projectors (Z<0 in %block constrained_dft), cDFT-potentials (Uq/s), NGWFs and DKN. This is accomplished, regardless of the keywords specific to the chosen cDFT-flavour, by making sure the input file (.dat) contains:

```
task : HUBBARDSCF
hubbard_max_iter : 40 # perform 40 Hubbard-SCF iterations
read_denskern : T
read_tightbox_ngwfs : T
cdft_read_projectors : T
cdft_continuation : T
```

The percentage of the latest-optimised cDFT-NGWFs to be used as new cDFT-projectors is controlled by the hubbard_proj_mixing keyword [0 < |hubbard_proj_mixing| < 1], with hubbard_proj_mixing=1 meaning that the latest optimised-NGWFs are used entirely as new cDFT-projectors.

Mind. Provided cdft_read_projector=T, the cDFT-projectors in the .tightbox_hub_projs file are used (entirely) as new projector during the 1st HUBBARDSCF iteration.

Is self-consistent optimisation of the cDFT-projectors at the DFT-geometry a good idea?

From the tests so far, this is **not a good idea**. For tightly constrained systems (for instance an hypothetical $N^{(+)}$=$N^{(-)}$ excitation) a cDFT task=HUBBARDSCF run at the DFT-optimised geometry may result in very slow convergence. The recommended procedure is to **first** optimise the cDFT-geometry using numerical orbitals (PAO) as projectors (Z-projector <0 in %block constrained_dft) and **then** optimise the cDFT-projectors at the PAO-cDFT optimised geometry.

4.2.2 cDFT Keywords

Intermediate Keywords

CDFT_ATOM_CHARGE

Syntax: CDFT_ATOM_CHARGE [Logical]

Description: Activate atom charge-constrained-DFT mode. This mode is incompatible with any other cDFT-mode.

Default: False

Example: CDFT_ATOM_CHARGE T

CDFT_ATOM_SPIN

Syntax: CDFT_ATOM_SPIN [Logical]

Description: Activate atom magnetic-moment-constrained-DFT mode. This mode is incompatible with any other cDFT-mode.

Default: False

Example: CDFT_ATOM_SPIN T

CDFT_CG_MAX

Syntax: CDFT_CG_MAX [Real]

Description: Specifies the maximum number of constraining potential (Uq/s) conjugate gradient iterations between resets.

Default: Number of independent Uq/s for cdft_guru=F

Example: CDFT_CG_MAX 1 #Perform steepest descents optimisation

CDFT_CG_MAX_STEP

Syntax: CDFT_CG_MAX_STEP [Real]

Description: Maximum length of trial step for the constraining potential (Uq/s) optimisation line search.

Default: 50.0

Example: CDFT_CG_MAX_STEP 10.0

CDFT_CG_THRESHOLD

Syntax: CDFT_CG_THRESHOLD [Real]

Description: Specifies the convergence threshold for the RMS gradient of the constraining potentials (Uq/s).

Default: 1.0E-3

Example: CDFT_CG_THRESHOLD 0.01

CDFT_CG_TYPE

Syntax: CDFT_CG_TYPE [Text]

Description: Specifies the variant of the conjugate gradients algorithm used for the optimization of the constraining potentials (Uq/s), currently either NGWF_FLETCHER for Fletcher-Reeves or NGWF_POLAK for Polak-Ribiere.

Default: NGWF_FLETCHER

Example: CDFT_CG_TYPE NGWF_POLAK

CDFT_CHARGE_ACCEPTOR_TARGET

Syntax: CDFT_CHARGE_ACCEPTOR_TARGET [Real]

Description: Targeted acceptor-group electron population for acceptor-group charge-constrained-DFT mode [CDFT_GROUP_CHARGE_ACCEPTOR=T].

Default: 0.

Example: CDFT_CHARGE_ACCEPTOR_TARGET 17 #Constrain Nup+Ndown=17 e in subspace

CDFT_CHARGE_DONOR_TARGET

Syntax: CDFT_CHARGE_DONOR_TARGET [Real]

Description: Targeted donor-group electron population for donor-group charge-constrained-DFT mode [CDFT_GROUP_CHARGE_DONOR=T].

Default: 0.

Example: CDFT_CHARGE_DONOR_TARGET 17 #Constrain Nup+Ndown=17 e in subspace

CDFT_CONTINUATION

Syntax: CDFT_CONTINUATION [Logical]

Description: Continue a constraining potential (Uq/s) optimisation from a previous run using the .cdft file with the latest cDFT-potentials. CDFT_CONTINUATION=T allows also to perform single-point cDFT runs (MAXIT_CDFT_U_CG=0) reading atom-specific constraining potentials from .cdft file (instead of species-specific ones from the CONSTRAINED_DFT block). For cdft_continuation=T, the constraining potentials (Uq/s) are read from the .cdft file no matter the setting of cdft_guru.

Default: False

Example: CDFT_CONTINUATION T

CDFT_GROUP_CHARGE_ACCEPTOR

Syntax: CDFT_GROUP_CHARGE_ACCEPTOR [Logical]

Description: Activate acceptor-group charge-constrained-DFT mode. This mode is compatible with CDFT_GROUP_CHARGE_DONOR and CDFT_GROUP_SPIN_ACCEPTOR/DONOR cDFT-modes, and incompatible with CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_ACCEPTOR T

CDFT_GROUP_CHARGE_DIFF

Syntax: CDFT_GROUP_CHARGE_DIFF [Logical]

Description: Activate group charge-difference constrained-DFT mode. This mode is compatible with CDFT_GROUP_SPIN_DIFF cDFT mode only. Thus, it is incompatible with any other CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_ACCEPTOR/DONOR cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_DIFF T

CDFT_GROUP_CHARGE_DIFF_TARGET

Syntax: CDFT_CHARGE_DIFF_TARGET [Real]

Description: Targeted electron population difference between acceptor and donor group for -group charge-difference constrained-DFT mode [CDFT_GROUP_CHARGE_DIFF=T].

Default: 0.

Example: CDFT_CHARGE_ACCEPTOR_TARGET 2

#Constrain [Nup+Ndown]_ACC - [Nup+Ndown]_DON to 2 e.

CDFT_GROUP_CHARGE_DONOR

Syntax: CDFT_GROUP_CHARGE_DONOR [Logical]

Description: Activate donor-group charge-constrained-DFT mode. This mode is compatible with CDFT_GROUP_CHARGE_ACCEPTOR and CDFT_GROUP_SPIN_ACCEPTOR/DONOR cDFT-modes, and incompatible with CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_DONOR T

CDFT_GROUP_CHARGE_DOWN_ONLY

Syntax: CDFT_GROUP_CHARGE_DOWN_ONLY [Logical]

Description: Constrain only SPIN-DOWN channel in CDFT_GROUP_CHARGE_ACCEPTOR, CDFT_GROUP_CHARGE_DONOR and CDFT_GROUP_CHARGE_DIFF modes. To avoid disaster, make sure the specified CDFT_CHARGE_ACCEPTOR/DONOR_TARGET or CDFT_CHARGE_DIFF_TARGET keywords are consistent with the fact only one spin channel is being constrained. This functionality is NOT compatible with CDFT_GROUP_CHARGE_UP_ONLY, CDFT_ATOM_CHARGE/SPIN, and CDFT_GROUP_SPIN_ACCEPTOR/DONOR and CDFT_GROUP_SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_DOWN_ONLY T

CDFT_GROUP_CHARGE_UP_ONLY

Syntax: CDFT_GROUP_CHARGE_UP_ONLY [Logical]

Description: Constrain only SPIN-UP channel in CDFT_GROUP_CHARGE_ACCEPTOR, CDFT_GROUP_CHARGE_DONOR and CDFT_GROUP_CHARGE_DIFF modes. To avoid disaster, make sure the specified CDFT_CHARGE_ACCEPTOR/DONOR_TARGET or CDFT_CHARGE_DIFF_TARGET keywords are consistent with the fact only one spin channel is being constrained. This functionality is NOT compatible with CDFT_GROUP_CHARGE_DOWN_ONLY, CDFT_ATOM_CHARGE/SPIN, and CDFT_GROUP_SPIN_ACCEPTOR/DONOR and CDFT_GROUP_SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_UP_ONLY T

CDFT_GROUP_SPIN_ACCEPTOR Syntax: CDFT_GROUP_SPIN_ACCEPTOR [Logical]

Description: Activate acceptor-group magnetic-moment constrained-DFT mode. This mode is compatible with CDFT_GROUP_SPIN_DONOR and CDFT_GROUP_CHARGE_ACCEPTOR/DONOR cDFT-modes, and incompatible with CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_SPIN_ACCEPTOR T

CDFT_GROUP_SPIN_DIFF

Syntax: CDFT_GROUP_SPIN_DIFF [Logical]

Description: Activate group magnetic-moment-difference constrained-DFT mode. This mode is compatible with CDFT_GROUP_CHARGE_DIFF cDFT mode only. Thus, it is incompatible with any other CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_ACCEPTOR/DONOR cDFT modes.

Default: False

Example: CDFT_GROUP_CHARGE_DIFF T

CDFT_GROUP_SPIN_DIFF_TARGET

Syntax: CDFT_SPIN_DIFF_TARGET [Real]

Description: Targeted magnetic-moment difference between acceptor and donor group for group magnetic-moment-difference constrained-DFT mode [CDFT_GROUP_SPIN_DIFF=T].

Default: 0.

Example: CDFT_CHARGE_ACCEPTOR_TARGET 2

#Constrain [Nup-Ndown]_ACC - [Nup-Ndown]_DON to 2 e.

CDFT_GROUP_SPIN_DONOR

Syntax: CDFT_GROUP_SPIN_DONOR [Logical]

Description: Activate donor-group magnetic-moment constrained-DFT mode. This mode is compatible with CDFT_GROUP_SPIN_ACCEPTOR and CDFT_GROUP_CHARGE_ACCEPTOR/DONOR cDFT-modes, and incompatible with CDFT_ATOM_CHARGE/SPIN and CDFT_GROUP_CHARGE/SPIN_DIFF cDFT modes.

Default: False

Example: CDFT_GROUP_SPIN_DONOR T

CDFT_GURU

Syntax: CDFT_GURU [Logical]

Description: Tell ONETEP you are a cDFT-expert and prevent it from initialising the active |Uq/s| to failsafe value of 1 eV overwriting the values entered in the %block constrained_dft (Uq/s).

Default: False

Example: CDFT_GURU T

CDFT_HUBBARD

Syntax: CDFT_HUBBARD [Logical]

Description: Activate the constrained-DFT+U functionality. It requires specifications of a positive value for the Hubbard correction (Uh) in the CONSTRAINED_DFT Block.

Default: False

Example: CDFT_HUBBARD T

CDFT_MAX_GRAD

Syntax: CDFT_MAX_GRAD [Real]

Description: Specifies the convergence threshold for the maximum value of the constraining-potential (Uq/s) gradient at any cDFT-site

Default: 1.0E-3

Example: CDFT_MAX_GRAD 0.01

CDFT_MULTI_PROJ

Syntax: CDFT_MULTI_PROJ [Logical]

Description: Activate the "as many cDFT-projectors as NGWFs" cDFT-mode. In this mode, the number of cDFT-projectors for a given cDFT-atom equals the number of NWGFs for that atom as specified in the %block species. Both the cDFT-projectors and the NGWFs are localised within spheres of the same radius. When activated, this mode overwrites the L-projectors and Z-projectors settings in %block constrained_dft, and the cDFT-projectors are built according to the settings in %block species_atomic_set for that atom=cDFT-site.

Default: False

Example: CDFT_MULTI_PROJ T

CDFT_PRINT_ALL_OCC

Syntax: CDFT_PRINT_ALL_OCC [Logical]

Description: Print detailed information of occupancies for al the cDFT-sites, for OUTPUT_DETAIL = VERBOSE.

Default: False

Example: CDFT_PRINT_ALL_OCC T

CDFT_READ_PROJ

Syntax: CDFT_READ_PROJ [Logical]

Description: Read cDFT-projectors from .tightbox_hub_proj file. Activation of this keyword overwrites any Z-projector setting in %block constrained_dft. It also makes not necessary to set hubbard_proj_mixing<0 to have task=HUBBARDSCF runs with projectors read in from file.

Default: False

Example: CDFT_READ_PROJ T

CDFT_SPIN_ACCEPTOR_TARGET

Syntax: CDFT_SPIN_ACCEPTOR_TARGET [Real]

Description: Targeted group magnetic-moment for acceptor-group magnetic-moment constrained-DFT mode [CDFT_GROUP_SPIN_ACCEPTOR=T].

Default: 0.

Example: CDFT_SPIN_ACCEPTOR_TARGET -2 #Constrain Nup-Ndown=-2 in subspace

CDFT_SPIN_DONOR_TARGET

Syntax: CDFT_SPIN_DONOR_TARGET [Real]

Description: Targeted group magnetic-moment for donor-group magnetic-moment constrained-DFT mode [CDFT_GROUP_SPIN_DONOR=T].

Default: 0.

Example: CDFT_SPIN_DONOR_TARGET -2 #Constrain Nup-Ndown=-2 in subspace

CDFT_TRIAL_LENGTH

Syntax: CDFT_TRIAl_LENGTH [Real]

Description: Specifies initial trial length for first step of constraining-potential (Uq/s) conjugate gradients optimisation.

Default: 0.1

Example: CDFT_TRIAL_LENGTH 1.0

CI_CDFT

Syntax: CI_CDFT [Logical]

Description: Perform a Configuration Interaction calculation based on constrained-DFT configurations.

Default: False

Example: CI_CDFT T

CI_CDFT_NUM_CONF

Syntax: CDFT_MAX_GRAD [Integer]

Description: Specifies the number of constrained-DFT configuration available for a CI_CDFT=T simulation

Default: 0

Example: CI_CDFT_NUM_CONF 4

CONSTRAINED_DFT

Syntax: CONSTRAINED_DFT [Block]

Syntax: %BLOCK CONSTRAINED_DFT

S1 L1 Z1 Uh1 Uq1(UP) Uq1(DOWN) Us1 N1(UP) N1(DOWN) [N1(UP)-N1(DOWN)] S2 L2 Z2 Uh2 Uq2(UP) Uq2(DOWN) Us2 N2(UP) N2(DOWN) [N2(UP)-N2(DOWN)] . . . . . .

. . . . . .

SM LM ZM UhM UqM(UP) UqM(DOWN) UsM NM(UP) NM(DOWN) [NM(UP)-NM(DOWN)]

%ENDBLOCK CONSTRAINED_DFT

Description: Manages constrained-DFT simulations. Provided cdft_multi_proj=F, for species S and subspace of angular momentum channel L (with principal quantum number n=L+1) we apply charge spin-specific [Uq(UP), Uq(DOWN)] or magnetic-moment-specific (Us) constraining potentials (eV). For cdft_atom_charge=T, N(UP) and N(DOWN) indicate the targeted e-population for spin-channel UP and DOWN, respectively. For cdft_atom_spin=T, [N1(UP)-N1(DOWN)] indicates the targeted e-population difference (i.e. local magnetic moment). Uh indicates the

optional Hubbard parameter (U, eV) to be applied for cdft_hubbard=T. An effective nuclear charge Z defines the hydrogenic orbitals spanning the subspace unless a negative value is given, e.g., Z=-10, in which case the NGWFs initial guess orbitals (numerical atomic orbitals) are used. Depending on the activated cDFT-mode, different columns of the block are used. These are:

S, L, Z, (Uh), Uq(UP), Uq(DOWN), N(UP), N(DOWN) for cdft_atom_charge=T

S, L, Z, (Uh), Us, [N(UP)-N(DOWN)] for cdft_atom_spin=T

S, L, Z, (Uh), Uq(UP), Uq(DOWN) for cdft_group_charge_acceptor=T, cdft_group_charge_donor=T, or cdft_group_charge_diff=T. In this case, Uq(UP) must be equal to Uq(DOWN). Acceptor and donor atoms are differentiated by mean of negative [Uq(UP/DOWN)<0] and positive [Uq(UP/DOWN)>0] constraining-potentials, respectively. Setting Uq=0 in the %block constrained_dft will result in the given cDFT-atom being excluded from the list of the atoms in a given cdft_group_charge_donor/acceptor/diff group.

S, L, Z, (Uh), and Us for cdft_group_spin_acceptor=T, cdft_group_spin_donor=T, or cdft_group_spin_diff=T. In this case, Acceptor and donor atoms are differentiated by mean of negative (Us<0) and positive (Us>0) constraining-potentials, respectively. Setting Us=0 in the %block constrained_dft will result in the given cDFT-atom being excluded from the list of the atoms in a given cdft_group_spin_donor/acceptor/diff group.

cdft_group_spin_acceptor=T, cdft_group_spin_donor=T, cdft_group_charge_acceptor=T and cdft_group_charge_donor=T are all compatible one with another. Charge- and magnetic-moment acceptor- and donor-groups may or may not be the same group. Thus, besides simultaneously constraining the charge and magnetic-moment on a given group, it is also possible (by setting the appropriate sign of Uq and Us in the %block constrained_dft) to create, within the same input and system, a charge_acceptor group-A, a charge_donor group-B, a spin_acceptor group-C and a spin_donor group-C. Similar considerations apply also for simultaneous activation of group_charge_diff and group_spin_diff cDFT-modes. In sum,

Activation of cdft_group_charge_up(down)_only=T for cdft_group_charge_acceptor/donor or cdft_group_charge_diff modes leads to optimisation of the Uq potentials only for the selected spin-channel i.e. Uq(UP) only for cdft_group_charge_up_only=T, and Uq(DOWN) only for cdft_group_charge_down_only=T, leaving the other spin channel unconstrained.

For cdft_multi_proj=T the L-projector and Z-projector columns in the %block constrained_dft are read but NOT used. The cDFT-projectors are set on the basis of the %block species_atomic_set and taken as the NGWFs initial guess (numerical atomic orbital). This leads to as many cDFT-projectors as NGWFs for the cDFT-atom being used. In the current implementation, the same Uq/s is applied to all the projectors of a given cDFT-atom regardless of their principal quantum number and angular momentum.

For all the cDFT-modes, unless maxit_cdft_u_cg=0, and depending of the specific cDFT-mode, the constraining potentials (Uq,Us) will be automatically optimised. Note that, unless cdft_guru=T, the constraining potentials (Uq/s) will be initialised to 1 eV. Thus, to perform fixed-Uq/s cDFT-runs or to initialise Uq/s with values different from 1 eV (useful for low-energy spin-excitation), it is necessary to set cdft_guru=T.

The CONSTRAINED_DFT Block is incompatible with the HUBBARD Block. To perform a constrained-DFT+U simulation with Hubbard (Uh) correction applied to the subspace in addition to the constraining potentials (Uq/s) it is necessary to set cdft_hubbard=T. For cdft_hubbard=F (which is the default), the Hubbard correction will NOT be applied to the subspace.

Example: %BLOCK CONSTRAINED_DFT

L Z Uh Uq(UP) Uq(DOWN) Us N(UP) N1(DOWN) [N1(UP)-N1(DOWN)]

N1 1 -5. 0.0 11.0 11.0 0.0 2.3 1.3 0.

N2 1 -5. 0.0 -26.0 -26.0 0.0 2.7 2.7 0.

%ENDBLOCK CONSTRAINED_DFT

MAXIT_CDFT_U_CG

Syntax: MAXIT_CDFT_U_CG [Integer]

Description: Specifies the maximum number of iterations for the constraining potentials (Uq/s) conjugate gradients optimisation.

Default: 60

Example: MAXIT_CDFT_U_CG 5

HUBBARD_TENSOR_CORR

Syntax: HUBBARD_TENSOR_CORR [Integer]

Description:

1: Correct tensorially for the slight nonorthogonality between DFT+U or constrained DFT (cDFT) projectors of numerical pseudoatomic orbital form, individually on each atom, which arise due to finite psinc sampling. See details see Phys. Rev. B 83, 245124 (2011).

2: Use the full simulation-cell overlap matrix of the initial numerical pseudoatomic orbitals (whether or not they are selected as projectors) to form the nonorthogonality correction, in the vein of Mulliken analysis. Experimental, non-Hermitian at present, and not recommended.

3: Do not correct for the slight nonorthogonality between DFT+U or constrained DFT (cDFT) projectors on a given atom. This is standard in many codes, and currently necessary to choose when using USP/PAW.

4: This is a reasonable, but not necessary choice, when using cDFT with constraints based on atom group populations. The non-negligible nonorthogonality between projectors on different atoms in the group is accounted for tensorially. This also activates multi-site Pulay force terms in constrained DFT (cDFT) that account for varying inter-atom nonorthogonality. For details see Phys. Rev. B 97, 205120 (2018).

5: This is also an arguably reasonable, but not necessary choice, when using cDFT with constraints based on, e.g. the difference of atom or atom group populations, that is source-drain cDFT. The non-negligible nonorthogonality between projectors on different atoms in a group is accounted for tensorially, and also the possible nonorthogonality between the source and drain atoms or atom-groups. For an application see Phys. Rev. B 93, 165102 (2016). This also activates (in principle) multi-site Pulay force terms in constrained DFT (cDFT) that account for varying inter-subspace nonorthogonality. This also activates multi-site Pulay force terms in constrained DFT (cDFT) that account for varying inter-atom nonorthogonality. This also activates multi-site Pulay force terms in constrained DFT (cDFT) that account for varying inter-atom nonorthogonality. For details see Phys. Rev. B 97, 205120 (2018).

The HUBBARD_TENSOR_CORR functionality is not activated in the rare case that analytical hydrogenic projectors are instead of the default numerical pseudoatomic ones.

Default: 1

Example: HUBBARD_TENSOR_CORR 4

CHAPTER

FIVE

DYNAMICS

5.1 Born-Oppenheimer Molecular Dynamics

Author
Simon M.-M. Dubois, University of Cambridge

Author
Valerio Vitale, University of Southampton

This document is intended as a guide to the molecular dynamics (MD) functionality in ONETEP (v4.4) [Skylaris2005]. Though some theoretical concepts are reviewed, it is not meant to be a stand-alone introduction to Born-Oppenheimer Molecular Dynamics (BOMD) simulations. The reader is referred to the textbook of Frenkel and Smit [Frenkel2001] for a review of the field.

5.1.1 Integrating the equations of motion

The MD functionality implemented in ONETEP is founded on the Born-Oppenheimer approximation which states that the electrons are much lighter than nuclei, the dynamics of electrons is much faster compared to the dynamics of the nuclei. As a consequence, the former can be considered to react instantaneously to the motion of the latter. The forces acting on the nuclei are derived from the ground state electronic configuration by means of the Hellmann-Feynamn theorem. The motion of the nuclei is described by the laws of classical mechanics

$$\frac{\partial H}{\partial \mathbf{r}} = -\dot{\mathbf{p}} \quad \text{and} \quad \frac{\partial H}{\partial \mathbf{p}} = \dot{\mathbf{r}}$$

where H is the Hamiltonian (or the total energy) of the system and $\mathbf{r}$, $\mathbf{p}$ are the nuclei positions and conjugate momenta. At each MD steps, the forces on the particles are computed, and the particles positions and momenta are updated according to Newton's equations of motion. Though this is an excellent approximation for many materials, it is important to keep in mind that classical dynamics does not account for quantum phenomena such as zero point motion, tunneling, or quantum fluctuations which may play a significant role in the dynamics of some systems.

In a BOMD simulation, the classical laws of motion are integrated using a finite difference scheme (that usually preserves the symplectic structure of phase space, e.g. the Velocity-Verlet algorithm [Verlet1967], [Swope1982]). For small enough time steps, the particle trajectory becomes independent of the discretization and the total energy of the system is conserved. At room temperature and in situation close to equilibrium, a time step Δt of a fraction of a femtosecond is usually adopted.

The Velocity-Verlet algorithm corresponds to the following set of four operations:

$$1 : \mathbf{v}_{n+1/2} = \mathbf{v}_n + \frac{\Delta t}{2m} * \mathbf{F}_n \tag{5.1}$$

$$2 : \mathbf{r}_{n+1} = \mathbf{r}_n + \Delta t * \mathbf{v}_{n+1/2} \tag{5.2}$$

$$3 : \text{Compute ionic forces } \mathbf{F}_{n+1} \tag{5.3}$$

$$4 : \mathbf{v}_{n+1} = \mathbf{v}_{n+1/2} + \frac{\Delta t}{2m} * \mathbf{F}_{n+1} \tag{5.4}$$

where subscripts are used to label the MD time steps. This approach yields a reversible integrator that weights correctly the phase space and conserves the phase space volume.

The velocities in eqs. (5.1)-(5.4), are the internal (or peculiar) velocities and not the atomic velocities. Internal velocities are used to properly take into account the internal motion of the system, for which the total linear momentum must vanish. When using open boundary conditions, the use of internal velocities ensures that also the total internal angular momentum vanishes. By setting the total linear (angular) momentum to zero at the beginning of a simulation while employing atomic velocities in eqs. (5.1)-(5.4), does not guarantee to keep the linear (angular) momentum conserved. This is due to numerical errors that unavoidably modify the initial values. One of the possible drawback is the well-known "flying ice cube effect". The interested reader is referred to Ref. [Hunenberger2005] for a comprehensive description. However, before printing out the trajectory info to the `rootname.md` file, the internal velocities are transformed back to the atomic velocities for visualization and post-processing. In the limit of very long time, the ergodic hypothesis is invoked which allows us to derive ensemble averages from the molecular trajectories.

5.1.2 Basic input parameters

The Molecular Dynamics functionality is activated by setting the input parameter `TASK` to `MOLECULARDYNAMICS`. If a fresh calculation is started, the initial nuclear positions are read from the `POSITIONS_ABS` block while the nuclear velocities are obtained from the `VELOCITIES` block. If the latter is not specified, the velocities are drawn from a maxwell-boltzmann distribution at a (user defined) temperature set in the `THERMOSTAT` block (see Thermostats section). The values of Δt is determined by the parameter `MD_DELTA_T`. The number of integration steps is fixed by `MD_NUM_ITER`.

For example, the following set of input parameters instructs the code to run a 4 ps long BOMD calculation with $\Delta t = 0.8$ fs.

```
TASK          :  MOLECULARDYNAMICS
MD_DELTA_T    :  0.8 fs
MD_NUM_ITER   :  5000
MD_PROPERTIES :  T
MD_RESTART    :  F
```

The flag `MD_PROPERTIES` instructs the code to enter the properties module at each MD steps. During the calculation a file `rootname.md` is generated that contains a summary of the trajectory, such as temperature, energies, nuclear positions, velocities and forces at each MD steps. Additionally, the latest phase space coordinates are stored in the unformatted file `rootname.md.restart`. The flag `MD_RESTART` enables to restart an MD calculation from the phase space coordinates stored in `rootname.md.restart`. It is important to stress here that `MD_NUM_ITER` is an incremental counter. This means that the when starting a fresh calculation the number of MD steps corresponds to `MD_NUM_ITER`, while for a restart calculation the actual number of MD steps is calculated as the difference between `MD_NUM_ITER` and the total number of MD steps completed up to that point. Therefore, if we want to continue the 4 ps long calculation of the previous example for other 4 ps, we would have to set

```
TASK          :  MOLECULARDYNAMICS
MD_DELTA_T    :  0.8 fs
MD_NUM_ITER   :  10000
MD_PROPERTIES :  T
MD_RESTART    :  T
```

5.1.3 Thermostats

The `THERMOSTAT` block must be defined for any MD calculation, even when performing microcanonical runs. For equilibration purposes or to extract thermodynamical averages, it is often desirable to sample the canonical ensemble (constant-NVT) rather than the microcanonical one (constant-NVE). In order to achieve this, there needs to be a mechanism (i.e. a thermostat) by which the system can exchange energy with the rest of the universe. Several thermostats, Andersen, Langevin, Nose-Hoover chains, Berendsen and Bussi, are available in ONETEP.

Andersen thermostat

One of the simplest constant temperature algorithm has been proposed by Andersen [Andersen1980]. The system is thermally coupled with a bath of fictitious particles at temperature T. Practically this coupling acts by replacing the momentum of a number of atoms by a new momentum derived from the appropriate Boltzmann distribution. The strength of the coupling can be adjusted by fixing the characteristic time (τ) at which the momentum rescaling occurs and the amplitude (γ) of the rescaling. Eventually, the probability that collision occurs during a time step Δt is given by,

$$q_{col} = 1 - e^{-\Delta t/\tau} \tag{5.5}$$

and the collision on atom i acts as,

$$p^{new} = \sqrt{(1-\gamma^2)}\, p + \gamma\, p^{boltzmann}, \tag{5.6}$$

where p^{new} is the momentum rescaled by Andersen thermostat, and $p^{boltzmann}$ is a random variable with appropriate Boltzmann distribution.

Langevin thermostat

The Langevin thermostat accounts for the motion of the atoms in the presence of a fictitious viscous solvent [Grest1986]. As they have to be pushed away, the solvent particles create a friction force damping the momentum of the atoms. Besides random perturbations of the ionic forces arise from the collisions between the atoms and the solvent particles. Langevin dynamic corresponds to the modified equation of motion,

$$\dot{p_\alpha} = F_\alpha - \gamma \frac{p_\alpha}{m_\alpha} + f_\alpha \tag{5.7}$$

where greek superscripts label the nuclei, F_α are the conservative forces acting on the nuclei, γ is the damping factor associated with the solvent viscosity and f_α are the random forces accounting for the collisions. In order to guarantee NVT statistics, the random forces and the damping factor are chosen so as to fulfill the fluctuation-dissipation theorem. Eventually, the update of the nuclei momenta p_α and forces F_α is given by,

$$\begin{aligned} p_\alpha^{new} &= p_\alpha \ * \ e^{-\gamma \Delta t} \\ F_\alpha^{new} &= F_\alpha \ * \ \frac{1}{\gamma}(1 - e^{-\gamma \Delta t}) + f_\alpha \\ f_\alpha &= \sqrt{\frac{m_\alpha k_B T (1 - e^{-2\gamma \Delta t})}{\Delta t^2}} * \xi_\alpha \end{aligned}$$

where $\{\xi_\alpha\}$ is a set of mutually uncorrelated random Gaussian variables with a zero mean and unit variance.

Nosé-Hoover thermostat and Nose-Hoover chains

In the Andersen and Langevin approaches, the constant temperature is achieved by stochastic collisions with fictitious particles. The approach of Nosé is different and allows to perform deterministic MD at constant temperature [Nose1984], [Hoover1985]. To achieve isothermal MD, an additional coordinate associated with an effective mass is introduced in the Lagrangian ruling the dynamics of the nuclei. For a derivation of the equations of motion, the reader is referred to the textbook of Berend and Smith. Provided the center of mass of the system remains fixed, the Nosé-Hoover thermostat leads to a canonical distribution of positions and momenta. To alleviate this restriction on the center of mass, the nuclei are coupled to a Nosé-Hoover thermostat whose fluctuations are determined by another thermostat (i.e. the so called Nosé-Hoover chains). In ONETEP, the effective masse of the thermostats (Q_{th_i}) is chosen, following the prescription of Martyna and Tuckerman [Martyna1996], as

$$Q_{th_1} = 3N\frac{k_B T}{\omega^2} \tag{5.8}$$

$$Q_{th_i} = \frac{k_B T}{\omega^2}, \tag{5.9}$$

where N is the number of nuclei and $\omega = 2\pi/\tau$ is the characteristic frequency of the thermostats. That parameter τ has to be chosen so as to guarantee a good coupling with the atomic system. E.g. when water is used as solvent in the system, a value of 9.4 fs is appropriate as it corresponds to the first asymmetric stretching mode of water molecules.

Berendsen thermostat

In the Berendsen thermostat, the ionic equation of motions are supplemented by a first order equation for the kinetic energy,

$$dK = \frac{K_t - K}{\tau} dt\,, \tag{5.10}$$

where K_t stands for the target kinetic energy. The weak coupling of the system with the heat bath is determined by the time constant τ. This thermostat does not generate a canonical ensemble but is vary efficient for thermalization of large systems.

Canonical velocity scaling

An extension of the Berendsen thermostat allows to recover the canonical distribution of the kinetic energy. In this approach, the instantaneous kinetic energy is propagated using an auxiliary stochastic dynamics. The equation of motion for the kinetic energy is defined as,

$$dK = \frac{K_t - K}{\tau}\, dt\, + 2\sqrt{\frac{K K_t}{3N\tau}}\, dW\,, \tag{5.11}$$

where K_t stands for the target kinetic energy and dW is a Wiener noise. For a complete derivation of the equations of motion, the reader is referred to G. Bussi et al. [Bussi2007]. In the same way as for the Berendsen thermostat, the coupling of the system with the heat bath is determined by the characteristic time τ.

5.1.4 Thermostat definition

The parameters related to constant-NVE or constant-NVT sampling are determined by means of the `THERMOSTAT` block. For a constant-NVE calculation, the thermostat block is needed to specify the initial temperature for the maxwell-boltzmann distribution, from which initial velocities are drawn. For constant-NVT sampling, different thermostats can be associated with different groups of atoms.

```
%block thermostat
  start_iter & end_iter & thermo_name & temp & ! First thermostat definition
      option_1 = value ! Optional parameter 1
      option_2 = value ! Optional parameter 2
  start_iter & stop_iter & thermo_name & temp & ! Second thermostat definition
      option_1 = value ! Optional parameter 1
      option_2 = value! Optional parameter 2
%endblock thermostat
```

A thermostat definition contains four mandatory parameters and several optional parameters. The mandatory parameters are : the starting and stopping MD steps (these must be set bearing in mind the global counter logic), the type of thermostat (i.e. `none`, `andersen`, `langevin`, `nosehoover`, `berendsen`, or `bussi`,) and the temperature. The line containing the mandatory parameters may be followed by one or more of optional parameter definition (one per line).

Let us set an NVT calculation at 300K with Langevin thermostat for the equilibration (3000 steps) and Nosé-Hoover thermostat for the thermodynamical sampling (10000 steps). The input parameters could look like

```
%block thermostat
0  3000  langevin  300.0 K
   damp = 0.2
3001  13000  nosehoover  300.0 K
   nchain = 4
   nsteps = 10
   tau = 100fs
%endblock thermostat
```

If both `MD_RESTART` and `MD_RESTART_THERMO` flags are set to true, the thermostat internal parameters are initialized from the values found in the unformatted file named`rootname.thermo.restart` (`rootname.thermo.global.restart` if `MD_GLOBAL_RESTART = .true.`, see section on MD history). This is particularly useful when using Nosé-Hoover thermostat as it avoids any disruption in the trajectories of the thermostat coordinates. A formatted report on the thermostat trajectories is outputted in the file `rootname.thermo`.

Thermostat optional parameters

tgrad

(Physical) (default = 0K)
Discrete variation of temperature T per MD step.

group

(Integer) (default = 0)
Index of the group of atoms (as defined in `positions_abs`) to wich the thermostat is coupled. If no group of atoms is specified the thermostat is applied to the full system (i.e. group index 0).

tau

(Physical) (default = 10*`MD_DELTA_T`)
Characteristic time scale of the thermostat. Depending on the type of thermostat, it may relate either to the average collision frequency (see Eq. (5.5)) or the thermostat fluctuation frequency (see Eqs. (5.8) and (5.9)) or to the coupling with the heat bath (see Eqs. (5.10) and (5.11)).

mix

(real) (default = 1.0)
Collision amplitude of the Andersen thermostat (see Eq. (5.6)).

damp

(real) (default = 0.2)
Damping factor in the Langevin equation of motion (see Eq. (5.7)).

nchain

(integer) (default = 0)
Number of thermostats in the Nosé Hoover chain.

nsteps

(integer) (default = 20)
Number of substep used to integrate the equation of motion of the Nosé-Hoover coordinates.

update

(logical) (default = .false.)
Impose to update the effective masses of the Nosé-Hoover coordinates when the temperature is modified.

5.1.5 Using MD history

In order to predict sensible trajectories and ensemble averages, BOMD requires to solve the self-consistent field (SCF) equations that determines the ground-state electronic structure at each MD steps. Solving the SCF equations therefore dominates the computational effort. The number of SCF cycles required to reach a given level of self-consistency can be substantially reduced by using a good initial guess for the electronic degrees of freedom. Various schemes have been proposed that enable to make a good use of the MD history in order to build efficient initial guesses.

Extrapolation of NGWFs and density kernel

In ONETEP [Skylaris2005], the Kohn-Sham SCF equations are formulated in terms of the single-particle density matrix $\rho(\mathbf{x}, \mathbf{x}')$,

$$\rho(\mathbf{x}, \mathbf{x}') = \phi_\alpha(\mathbf{x})\ K^{\alpha\beta}\ \phi^*_\beta(\mathbf{x}')\ , \tag{5.12}$$

where Einstein's notation for repeated indices has been used. $\{\phi_\alpha(\mathbf{x})\}$ is a set of localised support functions, hereafter named Non-orthogonal Generalized Wannier Functions (NGWFs), and $\mathbf{K}$ is the kernel representing the density operator. At each MD step, the total energy is minimized with respect to both the density kernel and the support functions. Here below, we briefly review various algorithms that allows to initialise the density kernel and NGWFs by extrapolation from previous time steps.

Hereafter, χ_i^{init} and χ_i^{scf} are used to represent respectively the initial guess and SCF solution for either the density kernel or a given NGWF at time $t = i\Delta t$.

One-dimensional linear extrapolation

The simplest attempt at a trial configuration for the electronic degrees of freedom is the linear extrapolation,

$$\chi^{\text{init}}_{(i+1)} = 2\chi^{\text{scf}}_{i} - \chi^{\text{scf}}_{(i-1)}. \tag{5.13}$$

Multi-dimensional linear extrapolation

The idea of multi-dimensional linear extrapolation was first proposed by Arias, Payne and Joannopoulos for the generation of trial wavefunctions [Arias92]. The one-dimensional linear extrapolation scheme creates an acceptable initial configuration for the ionic coordinates $\mathbf{r}' = 2\mathbf{r}_i - \mathbf{r}_{(\mathbf{i-1})}$. However, the actual coordinates $\mathbf{r}_{i+1}$ are in general different. In order to account for the non-linear propagation of the coordinates, the extrapolation can be generalized as follow,

$$\chi^{\text{init}}_{(i+1)} = \chi^{\text{scf}}_{i} + \sum_{n=0}^{N} c_n \left(\chi^{\text{scf}}_{(i-n)} - \chi^{\text{scf}}_{(i-(n+1))} \right) \tag{5.14}$$

where the $N+1$ coefficients $\{c_n\}$ are chosen by minimizing the norm,

$$\left\| (\mathbf{r}_i - \mathbf{r}_{i+1}) + \sum_{n=0}^{N} c_n \left(\mathbf{r}_{(i-n)} - \mathbf{r}_{(i-(n+1))} \right) \right\|. \tag{5.15}$$

This insures that the extrapolated degrees of freedom are in close correspondence to the BOMD trajectory.

Generalized multi-dimensional linear extrapolation

The multi-dimensional extrapolation of NGWFs can be further generalized in order to account for the local characteristics of the ionic trajectories. By introducing a localization function $F(r - r_{cut})$ within Eq.[multixtpol2], the coefficients $\{c_n\}$ can be further optimized with respect to the local environment. In practice, a set of coefficients $\{c_n\}_\alpha$ is derived for each ion (α) by minimizing the modified norm,

$$\left\| \left(\mathbf{r}'(\alpha)_i - \mathbf{r}'(\alpha)_{(i+1)}\right) + \sum_{n=0}^{m} c(\alpha)_n \left(\mathbf{r}'(\alpha)_{(i-n)} - \mathbf{r}'(\alpha)_{(i-(n+1))} \right) \right\|, \tag{5.16}$$

where $\mathbf{r}'(\alpha)_i$ refers to a local projection of the ionic coordinates at time t_i,

$$r'(\alpha)_{\beta,i} = F(r_{\alpha,i} - r_{\beta,i} - r_{cut})\, r_{\beta,i} \tag{5.17}$$

This way, the extrapolated NGWFs associated with a given ion are in better correspondence to the BOMD trajectory of its local environment.

One-dimensional polynomial extrapolation

Another way to extrapolate the density kernel and NGWFs is to assume that each element of the density kernel ($K^{\alpha\beta}$) and component of the NGWFs on the grid ($\phi_\alpha(\mathbf{x})$) can be represented as a polynomial in the time t. Applied to the density kernel, this gives,

$$K^{\alpha\beta}(t) = \sum_{m=0}^{N} c_m^{\alpha\beta}\, t^m \tag{5.18}$$

where the $N+1$ extrapolation coefficients $c_m^{\alpha\beta}$ are dertemined by fitting the polynomial expression to the last $N+1$ values of $K^{\alpha\beta}$.

Density kernel transformations

The extrapolation schemes, as described above, illustrates a point of view in which the density kernel and the support functions are considered on the same footing, either as a functional of the ionic coordinates or as an oscillatory function in time. This is ignoring the close link between the support functions and the density kernel (see Eq.[dkn]). There is a broader point of view, where the density kernel ($K^{\alpha\beta}$) is considered as the representation of the density operator in the time-dependent basis formed by the NGWFs. If one assume that the BOMD propagation of the electronic degrees of freedom is more or less adiabatic, it is tempting to rely on the schemes described above for the extrapolation of the support functions and to transform the latest density kernel in order to account for the modification of the basis set. In ONETEP, this can be done in two ways.

Projection of the density kernel

The simplest attempt at transforming the density kernel in order to adapt it to the new support functions is to project the density kernel onto the extrapolated NGWFs. This transformation reads,

$$\mathbf{K}^{\text{init}}_{(i+1)} = \left(\mathbf{S}^{\text{init}}_{i+1}\right)^{-1} \mathbf{T}_{(i+1),i}\, \mathbf{K}^{\text{scf}}_{i}\, \mathbf{T}_{i,(i+1)} \left(\mathbf{S}^{\text{init}}_{i+1}\right)^{-1} ,$$

where $\mathbf{K}_i$ and $\mathbf{S}_i$ stand for the density kernel and overlap matrix at MD step i; and $\mathbf{T}_{i,(i+1)}$ is the overlap between the NGWFs at MD step i and the *extrapolated* NGWFs at step $(i+1)$.

Christoffel correction to the density kernel

While projecting the density kernel onto the extrapolated support functions is appealing because of its conceptual simplicity, it does not fully account for the tensorial character of the density operator. As the support functions are extrapolated, the metric of the representation manifold changes giving rise to non-vanishing Christoffel symbols. In order to preserve tensorial integrity and idempotency to first order, contributions from the Christoffel symbols should be accounted for in the transformation of the density kernel. The correction to the density kernel then reads,

$$\Delta\mathbf{K}^{\text{init}}_{(i+1)} = -\left(\mathbf{S}^{\text{scf}}_{i}\right)^{-1} \mathbf{D}_{(i+1),i}\, \mathbf{K}_i \; - \mathbf{K}_i\, \mathbf{D}_{i,(i+1)} \; \left(\mathbf{S}^{\text{scf}}_{i}\right)^{-1}$$

with

$$\left(\mathbf{D}_{(i+1),i}\right)_{\alpha\beta} = \left\langle (\phi^{\text{init}}_{(i+1)})_\alpha - (\phi^{\text{scf}}_{i})_\alpha \,\middle|\, (\phi^{\text{scf}}_{i})_\beta \right\rangle .$$

Extended Lagrangian propagation of density kernel schemes

Extended Lagrangian naïve approach

The number of SCF iterations needed to reach a given threshold at each step of the BOMD calculation can be significantly reduced by the extrapolation schemes presented in sections [xtpol] and [dkn]. However, those methods come with a caveat that has to be kept in mind. While, with a perfect SCF optimization, the SCF ground-state electronic structure is independent from the initial guess, in practice, self-consistence is only achieve up to a given threshold. The consequence of this *incomplete* convergence is that the extrapolation schemes introduce a *memory* effect in the simulation and break the time-reversibility of the BOMD algorithm. As a consequence, the resulting trajectories suffers from systematic error and a significant energy drift may appear on time scales of a few picoseconds. A simple way to restore energy conservation is to impose tighter SCF convergence thresholds. However, this may results in a considerable increase of the computational cost. Another solution has been proposed by Niklasson et al. (see Ref. [Niklasson2006]). This scheme restores the time-reversibility of BOMD by extending the BO Lagrangian with auxilliary degrees of freedom directly associated with χ^0, the initial guess of the electronic degrees of freedom. The user is referred to Refs. [Niklasson2006], [Niklasson2009] for a complete introduction to this formalism.

Extended Lagrangian with dissipation, dEL/SCF

A more stable propagation scheme for the density kernel has also been proposed by Niklasson [Niklasson2009]. In this scheme, the numerical errors arising from an incomplete convergence are averaged out via a dissipative term in the extended BO Lagrangian. Following Bowler [Arita2014], we propagate the orthogonal representation **P** of the auxiliary density kernel, i.e. **P** has the sparsity pattern of **KS** rather than of **K**, to avoid the extra intricacies of propagating tensors in a space with non-unitary metric. The dissipative term is defined in terms of a linear combination of previous density kernels, which using the symplectic Verlet algorithm, yields the following equation of motion

$$\mathbf{P}_{i+1} = 2\mathbf{P}_i - \mathbf{P}_{i-1} + \kappa[(\mathbf{KS}^{\text{scf}})_i - \mathbf{P}_i] + \alpha \sum_{m=0}^{M} c_m \mathbf{P}_{i-m}.$$

where κ, α and c_m's are optimized coefficients obtained from Ref. [Niklasson2009]. The initial guess for the density kernel is given by

$$\mathbf{K}_{i+1}^{\text{init}} = \text{sym}(\mathbf{P}\mathbf{S}_{i+1}^{-1}) = \frac{1}{2}[(\mathbf{P}\mathbf{S}^{-1})_{i+1} + (\mathbf{S}^{-1}\mathbf{P})_{i+1}]$$

The problem with the above mentioned scheme lays in the use of a dissipative term that unavoidably breaks the time-reversibility, which in turn will generate, over long simulation time, a drift in the energy. However, for simulation time accessible at the moment in AIMD, this issue is of little concern.

Extended Lagrangian with thermostat, inertial iEL/SCF

Recently, a similar scheme that overcomes the issue of the time breaking symmetry has been proposed [Albaugh2015]. The idea is to control the dynamics of the auxiliary degrees of freedom through a thermostat. One of the simplest yet efficient thermostats around is the Berendsen thermostat. Here, we also propagate the orthogonal representation of the auxiliary density kernel for the same reasons listed in the previous section. The equation of motion for the auxiliary density kernel, using a velocity-Verlet integrator, reads

$$\begin{aligned}
\mathbf{P}_{i+1} &= \mathbf{P}_i + \dot{\mathbf{P}}_i \Delta t + \omega^2 \Delta t^2 [(\mathbf{KS})_i^{\text{scf}} - \mathbf{P}_i] \\
\dot{\mathbf{P}}_{i+1} &= \gamma_i \tilde{\dot{\mathbf{P}}}_{i+1} \\
&= \gamma_i \{\dot{\mathbf{P}}_i + \omega^2 \Delta t/2 [((\mathbf{KS})_{i+1}^{\text{scf}} - \mathbf{P}_{i+1}) + \\
&\quad ((\mathbf{KS})_i^{\text{scf}} - \mathbf{P}_i)]\}
\end{aligned}$$

with γ_i given by

$$\gamma_i = \sqrt{1 + \frac{\tau}{\Delta t}\left(\frac{T_{\text{K}}}{\langle \dot{\mathbf{P}}_i{}^2 \rangle} - 1\right)}$$

where T_{K} is the target temperature, τ is the characteristic time of the thermostat, and $\langle \dot{\mathbf{P}}_i{}^2 \rangle$ is the instantaneous temperature of the auxiliary degrees of freedom. The key parameter is the target temperature and much care must be done in assigning a value for it.

5.1.6 Extrapolation and propagation of NGWFs

The main input parameters that determine the extrapolation and propagation of NGWFs are `mix_ngwfs_type` and `mix_ngwfs_num`. The localization function $F(r - r_{cut})$ used in the generalized version of the multi-dimensional linear extrapolation (see Eq. [genxtpol2].) is characterized by the input parameters `mix_local_length` and `mix_local_smear`.

`mix_ngwfs_type` (String) (default = `none`)

`none` : No use of MD history. Initial NGWFs are built according to `species_atomic_set` block.

`reuse` : No mixing of NGWFs. NGWFs at previous MD step are used as initial guess.

`linear` : One dimensional linear extrapolation from NGWFs at two previous MD steps (see Eq. (5.13)).

`multid` : Multi-dimensional linear extrapolation from NGWFs at previous MD steps (see Eqs. (5.14) and (5.15)). The dimension of the extrapolation space is determined by input parameter `mix_ngwfs_num`.

`poly` : One-dimensional polynomial extrapolation from NGWFs at previous steps (see Eqs. (5.16)). The degree of the extrapolation polynom is determined by input parameter `mix_ngwfs_num`.

`local` : Generalized multi-dimensional linear extrapolation from NGWFs at previous steps (see Eqs. (5.16)). The dimension of the extrapolation space is determined by input parameter `mix_ngwfs_num`. The localization radius is determine by input parameter `mix_local_length`. Optionnally, the localization radius can be smeared out by using non-zero values for `mix_local_smear`

`trprop` : Time-reversible propagation of auxiliary NGWFs. See section on extended Lagrangian and references therein.

`mix_ngwfs_num` (Integer) (default depends on `mix_ngwfs_type`)

Number of previous MD steps required to build the initial guess for the density kernel

`mix_loc_length` (Physical) (default = `10.0 bohr`)

Cutoff radius of the localization function $F(r - r_{cut})$, see Eq. (5.17)

`mix_loc_smear` (Physical) (default = `5.0 bohr`)

When `mix_loc_smear` is non-vanishing, the localization function $F(r - r_{cut})$ is assumed to be Fermi-Dirac like with a characteristic smearing of `mix_loc_smear`.

5.1.7 Extrapolation and transformation of density kernel

The main input parameters that determine the extrapolation, transformation and propagation schemes for the density kernel and NGWFs are respectively `mix_dkn_type` and `mix_dkn_num`.

`mix_dkn_type` (String) (default = `none`)

`none` : No use of MD history. Initial density kernel is built according to `coreham_denskern_guess` parameter.

`reuse` : No kernel mixing. SCF density kernel at previous MD step is used as initial guess.

`linear` : One dimensional linear extrapolation from density kernel at two previous MD steps (see Eq. (5.13)).

`multid` : Multi-dimensional linear extrapolation from density kernel at previous MD steps (see Eqs. (5.14) and (5.15)). The dimension of the extrapolation space is determined by `mix_dkn_num`.

`poly` : One-dimensional polynomial extrapolation from density kernel at previous steps (see Eqs. (5.16)). The degree of the extrapolation polynom is determined by `mix_dkn_num`.

`proj` : Projection of the previous SCF density kernel onto the set of extrapolated NGWFs. This option requires `mix_ngwfs_type` $\neq$ none.

`tensor` : Correction of the previous SCF density kernel in order to preserve tensorial integrity. This option requires `mix_ngwfs_type` $\neq$ none.

`trprop` : Naïve time-reversible propagation of auxiliary density kernel. See section on extended Lagrangian and references therein.

`dissip` : Dissipative propagation of auxiliary density kernel. See section on extended Lagrangian and references therein. The number of previous MD steps used for the derivation of the dissipative force is determined by `mix_dkn_num`.

`berendsen` : Thermostatted propagation of auxiliary density kernel with Berendsen thermostat. See section on extended Lagrangian and references therein. The target temperature for the thermostat is set by `md_aux_dkn_t` and the characteristic time constant τ by `md_aux_beren_tc`.

`mix_dkn_num` (Integer) (default depends on `mix_dkn_type`)

Number of previous MD steps required to build the initial guess for the density kernel.

`mix_aux_dkn_t` (Physical) (default = `1e-8`)

Target temperature of the auxiliary degrees of freedom to use in the Berendsen propagation of the density kernel.

`mix_aux_beren_tc` (Physical) (default = `0.1 ps`)

Characteristic time constant for the Berendsen thermostat to use in the Berensen propagation of the density kernel.

5.1.8 Additional notes on extrapolation and propagation

Most of the extrapolation and propagation schemes suffer from restricted stability under incomplete SCF convergence. Depending on the convergence parameters, significant discrepancies between the MD trajectories and the Born-Oppenheimer surface may arise during the first few MD iterations. In this case, it is recommended not to use the extrapolation and propagation schemes until a good level of SCF convergence is reached. The input parameters `mix_ngwfs_init_type` and `mix_ngwfs_init_num` allows to set up a smooth initialization phase. It is also possible to have a different (usually tighter) thresholds during this initialization phase. In fact, the usual `lnv_threshold_orig` and `ngwf_threshold_orig` are used to set the LNV and NGWFs gradient thresholds during the intialization phase, while the two keywords `md_lnv_threshold` and `md_ngwf_threshold` determine the LNV threshold and NGWFs gradient threshold for the remaining MD calculation.

It is also possible to periodically reset the MD history using `mix_ngwfs_reset`, `mix_dkn_reset` and `md_reset_history`, although this is not recommended if one wants to avoid jumps into the energy profile, i.e. avoid discontinuities in energy plots.

`md_reset_history` (Integer) (default = `100`)

Every n MD steps, new initial guesses for the electronic degrees of freedom are built according to `coreham_denskern_guess` and `species_atomic_set`.

`mix_ngwfs_reset` (Integer) (default = `50`)

Every n MD steps, the NGWFs mixing/extrapolation scheme is reset and a new initial guess for the NGWFs is built according to `species_atomic_set`.

`mix_dkn_reset` (Integer) (default = `50`)

Every n MD steps, the density kernel mixing/extrapolation scheme is reset and a new initial guess for the kernel is built according to `coreham_denskern_guess`.

`mix_ngwfs_init_num` (Integer) (default = `0`)

Length of the initialization phase. Number of MD steps before the activation of the extrapolation/propagation scheme for building NGWF initial guesses.

`mix_ngwfs_init_type` (String) (default = `none`)

none : During the initialization phase, initial NGWFs are built according to species_atomic_set block.

reuse : During the initialization phase, NGWFs at last MD step is used as initial guess.

mix_dkn_init_num (Integer) (default = 0)

Length of the initialization phase. Number of MD steps before the activation of the extrapolation/propagation scheme for building density kernel initial guesses.

mix_dkn_init_type (String) (default = none)

none : During the initialization phase, initial density kernels are built according to coreham_denskern_guess.

reuse : During the initialization phase, density kernel at last MD step is used as initial guess.

md_lnv_threshold (Double) (default = lnv_threshold_orig)

LNV threshold for the MD calculation. This can be set to be different from the initial LNV threshold lnv_threshold_orig of the first n steps (set by mix_ngwfs_init_num / mix_dkn_init_num).

md_ngwf_threshold (Double) (default = ngwf_threshold_orig)

NGWF gradient threshold for the MD calculation. This can be set to be different from the initial NGWF gradient threshold ngwf_threshold_orig of the first n steps (set by mix_ngwfs_init_num / mix_dkn_init_num).

5.1.9 Additional notes on restart when using a propagation scheme

If a "history" of NGWFs/density kernels is generated during a MD calculation it can be periodically saved into external files through the keyword md_write_history. More precisely, when using md_write_history = T all the information about the dynamical state (positions, velocities and accelerations), the thermostat state, and the propagation scheme is saved to external files as well. To restart a MD calculation by reading in the history from the last save the md_global_restart keyword must be set to true in the restart input file. On a restart, one can either use the thermostat state stored in rootname.thermo.global.restart or start with a new thermostat block. This is achieved by setting the md_restart_thermo keyword.

md_write_history (Integer) (default = -1)

Every n MD steps the history of auxiliary density kernels is written into external files rootname.history.dkn#.scf/init/vel (one of each kind for any element in the history). The info on the dynamical state, the thermostat, the propagation scheme and the composition method are saved into rootname.md.global.restart, rootname.md.thermo.restart, rootname.history.info and rootname.history.var respectively.

md_global_restart (Logical) (default = F)

MD global restart. This allows to restart a calculation by reading in a history of density kernels if present. md_restart is set to false.

md_restart_thermo (Logical) (default = T)

Read thermostat info from file. If set to false, the thermostat is set according to the thermostat block in the input file.

WARNING: Restarting a calculation with md_global_restart = T comes with a caveat: depending on the value of md_write_history the last batch of NGWFs/density kernels saved to files may not correspond to the NGWFs/density kernels history of the last MD step completed. However, the calculation restarts using the information stored in the rootname.history.info and rootname.history.var. As a result, there might be duplicated entries in the rootname.md file which has to be deleted manually by the user.

For example, let's consider the following scenario

```
MD_NUM_ITER      : 124
MD_WRITE_HISTORY : 10
```

where we save a history of density kernels every 10 MD steps and the simulation stops after 124 steps. The last batch of density kernels (together with all the other MD info) is saved at step 120, but the summary of the trajectory from step 121-124 is still appended to `rootname.md`. When restarting with `md_global_restart = T`, the code reads in the files `rootname.history.info` and `rootname.history.var` containing the info corresponding to step 120 and starts to append the trajectory info to `rootname.md`. As a result, the summary of the trajectory from step 121-124 in the `rootname.md` is duplicated.

[Skylaris2005] C.-K. Skylaris et al., J. Chem. Phys. **122**, 084119 (2005).

[Frenkel2001] *Understanding Molecular Simulation*, 2nd Ed. D. Frenkel and B. Smit,Academic Press (2001)

[Verlet1967] L. Verlet, Phys. Rev. **159**, 98 (1967).

[Swope1982] W. C. Swope et al., J. Chem. Phys. **76**, 637 (1982).

[Andersen1980] H. C. Andersen, J. Chem. Phys. **72**, 2384 (1980).

[Grest1986] G. S. Grest and K. Kremer, Phys. Rev. A, **33** 3628 (1986).

[Nose1984] S. Nose, J. Chem. Phys., **81** 511 (1984).

[Hoover1985] W. G. Hoover, Phys. Rev. A, **31** 1695 (1985).

[Bussi2007] G. Bussi et al., J. Chem. Phys., **126** 014101 (2007).

[Martyna1996] G. J. Martyna, M.E. Tuckerman, et al., Molecular Physics **87**, 1117 (1996).

[Arias1992] T. A. Arias et al., Phys. Rev. B, **45**, 1538 (1992).

[Niklasson2006] A. M. N. Niklasson et al., Phys. Rev. Lett. **97**, 123001 (2006).

[Niklasson2009] A. M. N. Niklasson et al., J. Chem. Phys. **130**, 214109 (2009).

[Hunenberger2005] P. H. Hünenberger, Advanced Computer Simulation **173**, 105-109 (2005).

[Arita2014] M Arita et al., J. Chem. Theory Comput., **10**, 5419-5425 (2014)

[Albaugh2015] A. Albaugh et al, J. Chem. Phys., **143**, 174104 (2015)

5.2 Phonon calculations

Author
Fabiano Corsetti, Imperial College London

Date
September, 2013

5.2.1 Theory

We make use of the harmonic approximation, in which the total energy of the system E^{tot} is expanded to quadratic order in the displacement of the ions about their equilibrium positions:

$$E^{\mathrm{tot}} = E^{\mathrm{eq}} + \frac{1}{2} \sum_{a,\alpha,\kappa,a',\alpha',\kappa'} u_{a,\alpha,\kappa} \phi^{\alpha,\alpha'}_{\kappa,\kappa'} (a,a')\, u_{a',\alpha',\kappa'},$$

where E^{eq} is the equilibrium energy and $u_{a,\alpha,\kappa}$ denotes a small displacement of ion α belonging to unit cell a in the Cartesian coordinate direction κ from its equilibrium position; $\boldsymbol{\phi}(a,a')$ is known as the force constants matrix, defined as

$$\phi^{\alpha,\alpha'}_{\kappa,\kappa'} (a,a') = \frac{\partial^2 E}{\partial u_{a,\alpha,\kappa} \partial u_{a',\alpha',\kappa'}}.$$

It can be shown that the phonon frequencies $\omega_{\mathbf{q},n}$ at wavevector $\mathbf{q}$ are the eigenvalues of the dynamical matrix $\mathbf{D}(\mathbf{q})$, which can be calculated from the Fourier transform of the force constants matrix:

$$D^{\alpha,\alpha'}_{\kappa,\kappa'} (\mathbf{q}) = \frac{1}{\sqrt{M_\alpha M_{\alpha'}}} \sum_a \phi^{\alpha,\alpha'}_{\kappa,\kappa'} (a,0)\, \mathrm{e}^{-\mathrm{i}\mathbf{q}\cdot\mathbf{R}_a}, \tag{5.19}$$

where M_α is the mass of ion α and $\mathbf{R}_a$ is the lattice vector displacement for unit cell a. The vibrational free energy for the unit cell is then given by

$$F(T) = \frac{1}{2} \sum_{\mathbf{q},n} \omega_{\mathbf{q},n} + k_{\mathrm{B}} T \sum_{\mathbf{q},n} \ln\left(1 - \mathrm{e}^{-\omega_{\mathbf{q},n}/k_{\mathrm{B}}T}\right), \tag{5.20}$$

where the first term is the zero-point energy of the system, and the second term is the temperature-dependent part of the free energy. In the limit of an infinite periodic system the sum over $\mathbf{q}$ should be replaced by an integral of the phonon dispersion curves over the first Brillouin zone.

The phonon module in onetep uses the *finite-displacement* method to calculate the phonon frequencies of the system; for molecules (the default), only the Γ-point frequencies $\omega_{\mathbf{0},n}$ are calculated, while for supercells of bulk crystal, any arbitrary q point $\omega_{\mathbf{q},n}$ can be calculated. The elements of the force constants matrix are calculated by a central-difference formula, using either 2 (the default) or 4 displacements:

$$\begin{cases} \phi^{\alpha,\alpha'}_{\kappa,\kappa'} \approx \frac{F^{+}_{\alpha,\kappa} - F^{-}_{\alpha,\kappa}}{2d} & \texttt{phonon\_sampling 1} \text{ (2 disps.)} \\ \phi^{\alpha,\alpha'}_{\kappa,\kappa'} \approx \frac{-F^{2+}_{\alpha,\kappa} + 8F^{+}_{\alpha,\kappa} - 8F^{-}_{\alpha,\kappa} + F^{2-}_{\alpha,\kappa}}{12d} & \texttt{phonon\_sampling 2} \text{ (4 disps.)} \end{cases}, \tag{5.21}$$

where $F^{\pm}_{\alpha,\kappa}$ is the force on ion α in direction κ caused by a displacement $\pm d$ of ion α' in direction κ', and $F^{2\pm}_{\alpha,\kappa}$ is the same for a displacement $\pm 2d$. Therefore, $6N$/$12N$ calculations are needed in total, where N is the number of atoms in the system. However, each of these calculations is simply a small perturbation on the equilibrium configuration. Therefore, the converged set of NGWFs $\{\xi_\beta(\mathbf{r})\}$ and density kernel $\mathbf{K}$ that are obtained from a preliminary ground-state calculation on the equilibrium structure are used as the starting guess for each of the displacement calculations.

5.2.2 Overview of the phonon module

A phonon calculation in onetep is divided into three stages:

1. A ground-state calculation is performed for the unperturbed configuration, as specified in the input file. The forces on the ions are then calculated, and the code checks that the magnitude of the force on every ion is smaller than the value specified by phonon_fmax, as the starting configuration must correspond to a minimum in the energy landscape for the phonon calculation to be meaningful. If this requirement is not met, the calculation is interrupted.

2. Each ion is displaced in turn in the +ve and −ve x-, y-, and z-directions by a distance d (and, optionally, $2d$). For each displacement a separate ground-state calculation is performed. The initial description of the electronic structure is read in each time from the converged files <seedname>.dkn and <seedname>.tightbox_ngwfs for the unperturbed structure that have been obtained from Stage 1; the overwriting of these files is therefore disabled at the start of Stage 2. After each set of +ve/−ve displacements, one row of the force constants matrix is calculated and written to the file <seedname>.force_consts_<i>, where <i> is the number identifier of the row (going from 1 to $3N$ for an N-atom system). It is important to note that *not all rows are necessarily computed*, if some vibrational degrees of freedom are switched off (see section on selecting degrees of freedom below), and/or a supercell calculation of bulk crystal is being performed (see section on supercell calculations below); however, <i> retains the same value as it would have if all $3N$ rows were to be used.

3. The rows of the force constants matrix are read back in from the files <seedname>.force_consts_<i>, and the full force constants matrix is constructed. The dynamical matrix can then be calculated for the desired q points and diagonalized to find the phonon frequencies. First, the phonon frequencies are calculated on a regular grid of q points (Γ only for a molecule); in either case, the Γ-point frequencies only are printed to standard output. Then, the following thermodynamic quantities are calculated on the full grid and printed to standard output: the zero-point energy, and the free energy, entropy, internal energy, and specific heat within a user-specified temperature range. The phonon DOS is also calculated on the full grid and written to the file <seedname>.qdos. Additionally, the user can specify a list of arbitrary q points, for which the phonon frequencies (and, optionally, the corresponding eigenvectors) are calculated and written to the file <seedname>.phonon_freqs. Finally, a list of Γ-point modes <j> can be specified for which animation files <seedname>.phonon_<j>.xyz are written.

This division in stages is done so as to allow for a *task farming* approach (see section on task farming for details).

5.2.3 Selecting degrees of freedom

phonon_vib_free	x	y	z
0	F	F	F
1	T	F	F
2	F	T	F
3	T	T	F
4	F	F	T
5	T	F	T
6	F	T	T
7	T	T	T

Table: Allowed options for keyword phonon_vib_free.

The input file allows the user to select only a subset of the complete vibrational degrees of freedom of the entire system, as well as to specify different finite-displacement options for each (α, κ) pair. This is controlled through two keywords: phonon_vib_free and phonon_exception_list.

phonon_vib_free is an integer parameter controlling the global default of which Cartesian directions are 'switched on' for all ions. The options are listed in Table [table:free]. The default option is 7, corresponding to all three Cartesian directions being switched on (i.e., all vibrational degrees of freedom are allowed).

phonon_exception_list is a block in which the user can list specific (α, κ) pairs with options differing from the global defaults defined by phonon_vib_free, phonon_sampling, and phonon_finite_disp. An example of doing so is as follows:

```
phonon_vib_free 3
phonon_sampling 1
phonon_finite_disp 1.4e-1 bohr
```

(continues on next page)

(continued from previous page)

```
%block phonon_exception_list
10 3 1 2 0.9
15 1 0 1 1.0
36 2 0 1 1.0
%endblock phonon_exception_list
```

Here, we have first defined the global defaults; `phonon_vib_free 3` corresponds to only the x- and y-directions being selected for the calculation, and the z-direction being switched off. Then, in the `phonon_exception_list` block, we list three exceptions:

- displacement of ion `10` in the z-direction (`3`) is switched on (`1`), with a value of `phonon_sampling` of `2`, and a value of `phonon_finite_disp` of `0.9` × the global value (i.e., `1.26e-1 bohr`);
- displacement of ion `15` in the x-direction (`1`) is switched off (`0`), with the last two parameters not being read;
- displacement of ion `36` in the y-direction (`2`) is switched off (`0`), with the last two parameters not being read.

5.2.4 Bulk crystal supercell calculations

Phonon calculations for crystalline systems can be performed in onetep using a supercell approach, with either a real-space truncation of the force constants matrix, or a Slater-Koster style interpolation. The size of the supercell is chosen by the user; obviously, larger supercells will produce more accurate results at arbitrary q points.

It is the responsability of the user to provide the correct supercell lattice vectors and atomic coordinates in the usual way. Additionally, the `supercell` block must be specified to inform the `phonon` module that the system is a supercell of bulk material; otherwise, it will be assumed to be a molecule. An example of doing so is as follows:

```
%block lattice_cart
ang
5.3938105 5.3938105 0.0000000
5.3938105 0.0000000 5.3938105
0.0000000 5.3938105 5.3938105
%endblock lattice_cart

%block positions_abs
ang
Si 0.000000000 0.000000000 0.000000000
Si 0.000000000 2.696905250 2.696905250
Si 2.696905250 0.000000000 2.696905250
Si 2.696905250 2.696905250 5.393810500
Si 2.696905250 2.696905250 0.000000000
Si 2.696905250 5.393810500 2.696905250
Si 5.393810500 2.696905250 2.696905250
Si 5.393810500 5.393810500 5.393810500
Si 1.348452625 1.348452625 1.348452625
Si 1.348452625 4.045357875 4.045357875
Si 4.045357875 1.348452625 4.045357875
Si 4.045357875 4.045357875 6.742263125
Si 4.045357875 4.045357875 1.348452625
Si 4.045357875 6.742263125 4.045357875
Si 6.742263125 4.045357875 4.045357875
Si 6.742263125 6.742263125 6.742263125
```

(continues on next page)

(continued from previous page)

```
%endblock positions_abs

%block supercell
2 2 2
1
9
%endblock supercell
```

Within the `supercell` block, the first line gives the shape of the supercell ($2\times2\times2$), and subsequent lines list the ions in the `positions_abs` block that belong to the 'base' unit cell (of course, this supercell is too small to give sensible results for a phonon calculation, and is probably too small to run in onetep anyway; a 1000-atom cubic supercell of Si gives excellent results however!)

When a supercell calculations is specified, only the ions within the unit cell are displaced, although the forces on all ions in the system are used to calculate the elements of the dynamical matrix from Eq. eq:*dynamical_mat*. It is also possible to specify `phonon_vib_free` and `phonon_exception_list` in a supercell calculation, although only the ions listed in the `supercell` block can be included in the `phonon_exception_list` block.

5.2.5 Task farming

The most efficient way of performing a phonon calculation is by task farming, as the full force constants matrix is built up from many perturbed-structure calculations, each of which is completely independent. This can be done with the following steps:

1. Run `phonon_farming_task 1` as a single job; this is essentially a standard single-point energy-and-force onetep calculation. Find the line in the main output file which gives the `Number of force constants` needed for the phonon calculation you have specified (this will be between 1 and $3N$).
2. Divide the total number of force constants that need to be calculated between the desired number of jobs. Prepare the onetep input file for each job specifying `phonon_farming_task 2` and a subset of the force constant calculations in the `phonon_disp_list` block. Make sure every job has access to the files `<seedname>.dkn` and `<seedname>.tightbox_ngwfs` obtained from the unperturbed calculation in the previous step.
3. Collect all the `<seedname>.force_consts_<i>` files and place them in the same directory. Finally, run `phonon_farming_task 3` as a single job, to construct the full force constants matrix and perform the post-processing calculations.

5.2.6 Keywords

The phonon calculation is selected by specifying `task phonon`. All other keywords related to the module are optional. They are:

- `phonon_farming_task` [Integer]
  Select which stage to perform (as described in Sec. [sec:farm]). Can be either `1`, `2`, `3` for a single stage, or `0` for all stages. Default is `0`.
- `phonon_sampling` [Integer]
  Finite-difference formula to use (see Eq. eq:*fd*). Default is `1`.
- `phonon_finite_disp` [Physical]
  Ionic displacement distance d. Default is `1.0e-1 bohr`.
- `phonon_fmax` [Physical]
  Maximum ionic force allowed in the unperturbed system. Default is `5.0e-3 ha/bohr`.

- phonon_energy_check [Logical]
  Perform a sanity check that the total energy doesn't decrease upon ionic displacement. Default is F.
- phonon_vib_free [Integer]
  Default allowed vibrational degrees of freedom for all ions (see Sec. [sec:free] for details). Default is 7.
- phonon_exception_list [Block]
  List of exceptions to the global defaults defined by phonon_vib_free, phonon_sampling, and phonon_finite_disp (see Sec. [sec:free] for details). Default is unspecified.
- supercell [Block]
  Definition of the supercell used for crystalline material (see Sec. [sec:supercell] for details). Default is unspecified.
- phonon_disp_list [Block]
  List of force constant calculations to perform for Stage 2. Note that the total number of force constant calculations is given in the main output file in the line Number of force constants; this will be less than or equal to $3N$. The numbers listed in the phonon_disp_list block should go from 1 to this number; *they can only be equated to the label* <i> *if all* $3N$ *force constants are calculated.* If unspecified, all displacements are performed. Default is unspecified. Example:

```
%block phonon_disp_list
1
3
5
%endblock phonon_disp_list
```

- phonon_grid [Block]
  Definition of the regular grid of q points used for the computation of thermodynamic quantities and the phonon DOS. Default is 1 1 1 (i.e., Γ point only). Example:

```
%block phonon_grid
10 10 10
%endblock phonon_grid
```

- phonon_SK [Logical]
  Use a Slater-Koster style interpolation for q points instead of a real-space cutoff of the force constants matrix elements. Default is F.
- phonon_tmin [Physical]
  Lower bound of the temperature range for the computation of thermodynamic quantities, expressed as an energy ($k_{\mathrm{B}}T$). Default is 0.0 hartree.
- phonon_tmax [Physical]
  Upper bound of the temperature range for the computation of thermodynamic quantities. Default is 2.0e-3 hartree ($\simeq 632$ K).
- phonon_deltat [Physical]
  Temperature step for the computation of thermodynamic quantities. Default is 1.5e-5 hartree ($\simeq 5$ K).
- phonon_min_freq [Physical]
  Minimum phonon frequency for the computation of thermodynamic quantities, expressed as an energy ($\hbar\omega$); frequencies lower than this are discarded. Default is 3.6e-6 hartree ($\simeq 5$ cm^{-1}).
- phonon_DOS [Logical]
  Calculate the phonon DOS and write to file. Default is T.
- phonon_DOS_min [Real]

Lower bound of the phonon DOS range (in cm^{-1}). Default is `0.0`.

- `phonon_DOS_max` [Real]
  Upper bound of the phonon DOS range (in cm^{-1}). Default is `1000.0`.
- `phonon_DOS_delta` [Real]
  Frequency step for the phonon DOS calculation (in cm^{-1}). Default is `10.0`.
- `phonon_qpoints` [Block]
  List of additional q points for which to calculate the phonon frequencies, in fractional coordinates of the reciprocal unit cell vectors. For non-supercell calculations only the Γ point can be specified. Default is unspecified. Example:

```
%block phonon_qpoints
0.0 0.0 0.0
0.0 0.0 0.1
0.0 0.0 0.2
0.0 0.0 0.3
0.0 0.0 0.4
0.0 0.0 0.5
%endblock phonon_qpoints
```

- `phonon_write_eigenvecs` [Logical]
  Write the eigenvectors as well as the phonon frequencies to file for the additional q points. Default is `F`.
- `phonon_animate_list` [Block]
  List of Γ-point modes (where 1 is the lowest) for which to write xyz animation files. Default is unspecified. Example:

```
%block phonon_animate_list
2
6
33
34
%endblock phonon_animate_list
```

- `phonon_animate_scale` [Real]
  Relative scale of the amplitude of the vibration in the xyz animation. Default is `1.0`.

5.2.7 Additional notes

Phonon calculations are quite sensitive to the accuracy of the ionic forces calculated for the perturbed structures. Therefore, it is advisable to make sure that the forces are well-converged with respect to the usual parameters: cut-off energy, number and radius of NGWFs, and spatial cut-off of the density kernel.

Furthermore, it is also important to make sure that for a given set of parameters the forces are properly converged at the end of the energy minimization procedure, and that the numerical noise is reduced to a minimum; the code will not check this automatically, and the forces generally converge slower than the total energy. To ensure an accurate result, therefore, the following values for the convergence threshold parameters are suggested:

- `ngwf_threshold_orig 1.0e-7`.
- `lnv_threshold_orig 1.0e-11`.

CHAPTER

SIX

TRANSITION STATES AND NEB

6.1 Nudged Elastic Band Transition State Searching and the Image-Parallel Running Mode

Author
Kevin Duff, University of Cambridge

Date
2018

6.1.1 NEB Method

The Nudged Elastic Band (NEB) method is a systematic approach to transition state searching. A brief overview will be provided here; for a more detailed description see [Jonsson1998].

Overview

Tools such as geometry optimization make determining the properties and relative energies of products and reactants relatively straightforward. Of equal importance to understanding a reaction or diffusion problem is the transition state (TS) - the highest-energy point on the minimum energy path (MEP) connecting the reactant and product in configuration space. Unfortunately determining the TS is not a simple local minimization problem, but instead it requires a determination of the MEP from the set of paths that connect the reactant and product. Several approaches that attempt to approximate the TS given a set of assumptions about the energy landscape exist, such as LST/QST. NEB attempts to offer a systematic determination of the TS through a local optimization of the MEP from an initial guess.

Theory

An elastic band method connects the reactant and product with a chain of beads in configuration space connected to their immediate neighbors with springs of natural length 0. This chain is then relaxed with the reactant and product held fixed and each bead on the chain feeling the forces from the potential energy surface as well as the springs.

Unfortunately, this approach runs into two immediate issues. The spring force perpendicular to the path works to pull the chain away from the correct MEP, leading to a poor approximation of the TS, and the force from the potential energy surface tangent to the path pushes beads to lower energy areas, whereas the goal is to sample the highest-energy point on the path. Varying the spring constant can reduce one issue while exacerbating the other and there is no systematic way to sample the highest point of the correct MEP.

The nudged elastic band approach is to project out the problematic components of the spring and PES force, 'nudging' the chain back onto the MEP. That is, each bead feels only the component of the force due to the potential energy surface perpendicular to the path tangent and that of the spring force parallel to the path tangent. The former relaxes

the chain onto the MEP and the latter evenly distributes the images along the path. Using this approach the beads will lie strictly on the MEP regardless of bead count assuming an accurate approximation to the path tangent at each point.

As with any local minimization, the final path depends on the initial guess and is not guaranteed to be the global MEP. Care should be taken in complicated reactions or diffusions - in particular a reaction passing through several stable intermediates might be broken into a number of separate NEB calculations. More control over the initial path guess may be implemented in the future.

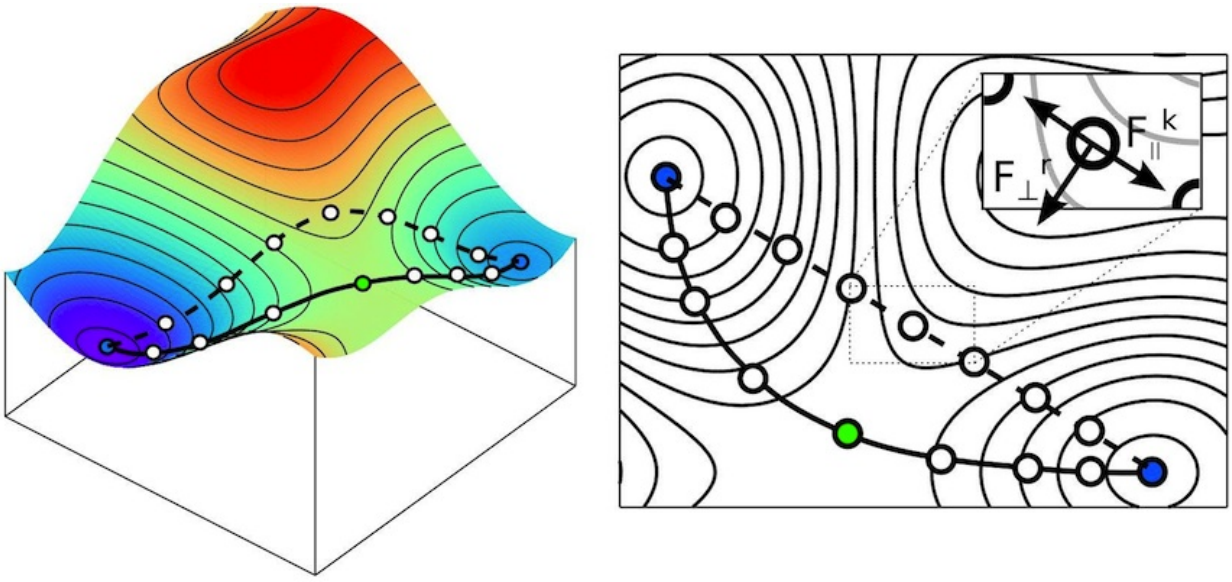

Fig. 6.1: Cartoon of a NEB path initialization by linear interpolation and final sampling of the MEP. Each image only feels the component of the spring force parallel to the path tangent and the real force perpendicular to the path tangent. Image source & copyright [Cordier2018].

Tangent Approximation

Because the MEP is approximated by a series of beads, the path tangent must be approximated. A number of valid approximations exist - ONETEP uses an improved-stability approximation described in [Henkelman2000]. This approach reduces kinks in the path that arise in some systems and is generally stable. Equation (6.1) describes the tangent $\boldsymbol{\tau}_i$ approximated at bead i with energy E_i, where $\boldsymbol{\tau}_i^+ = \mathbf{R}_{i+1} - \mathbf{R}_i$ and $\boldsymbol{\tau}_i^- = \mathbf{R}_i - \mathbf{R}_{i-1}$, in cases where the neighboring beads' energies make a strictly increasing or decreasing series. If that's not the case, Equation (6.2) gives the tangent approximation, where $\Delta V_i^{\max} = \max\left(|V_{i+1} - V_i|, |V_i - V_{i-1}|\right)$ and $\Delta V_i^{\min} = \min\left(|V_{i+1} - V_i|, |V_i - V_{i-1}|\right)$.

$$\boldsymbol{\tau}_i = \begin{cases} \boldsymbol{\tau}_i^+, & V_{i+1} > V_i > V_{i-1} \\ \boldsymbol{\tau}_i^-, & V_{i-1} > V_i > V_{i+1} \end{cases} \tag{6.1}$$

$$\boldsymbol{\tau}_i = \begin{cases} \boldsymbol{\tau}_i^+ V_i^{\max} + \boldsymbol{\tau}_i^- V_i^{\min}, & V_{i+1} > V_{i-1} \\ \boldsymbol{\tau}_i^+ V_i^{\min} + \boldsymbol{\tau}_i^- V_i^{\max}, & V_{i-1} > V_{i+1} \end{cases} \tag{6.2}$$

Climbing-Image NEB

NEB tries to ensure that the beads are equally spaced along the path. This doesn't guarantee good sampling of the transition state, which is the most important part of the path. The TS energy can be interpolated if there are enough beads on the path, but the climbing image addition to NEB (CI-NEB) works to move a selected bead near the TS to exactly sample the TS. Once enabled, the highest-energy bead doesn't feel the NEB spring force and moves in a modified potential energy surface, where the components perpendicular to the path are essentially mirrored. This transforms a saddle point region of the PES into a basin with a minimum at the transition state. That bead is then minimized in this potential and the rest of the chain in the normal way. This is generally useful when the MEP is being sampled well, though it does make assumptions about the shape of the PES near the saddle point. More details can be found in [Henkelman2000-2].

6.1.2 Image-Parallel Implementation in ONETEP

Each NEB iteration each bead requires a local energy and force calculation, as well as knowledge of the locations and relative energies of its neighbors for spring force calculation and tangent approximation. For this reason, and for other simulations that involve multiple communicating but largely independent subsystems, an alternate running mode was developed for Onetep that allows multiple simulations to exist in the same MPI world. Each simulation, or image, can progress independently but special communicators have been set up to allow communication between them. In the case of NEB, each Onetep image controls one bead in the chain.

Important to note is that when running in image-parallel mode the default communicator for things like comms operations is changed from `mpi_comm_world` to `comms` mod's `pub_image_comm`, a communicator between all processes in one Onetep image. `comms` mod's `pub_imroots_comm` is a communicator between the root processes of each image, allowing images to communicate and allowing tasks to give each image something different to do. Additionally, each image opens its own new file `{rootname}{image_num}.onetep` to write `stdout` to, with the original `stdout` being available through `image_comms` mod's `orig_stdout`. Each image similarly maintains its own set of restart files, properties files, etc, and nothing special has to be done to restart image-parallel calculations.

Normally the number of MPI processes specified at runtime must be divisible by the number of images requested. Advanced configuration allows images to be different sizes, in case a task needs to be able to perform calculations that aren't necessarily comparable in cost.

6.1.3 Commands

NEB Keywords

NEB can be enabled by setting `task : tssearch` and `tssearch_method : neb` in the ONETEP input file. ONETEP must be executed with enough MPI processes to support the number of images requested. Several geometry optimization keywords will apply to NEB as the chain optimization is threaded through the geometry optimizer.

The reactant is taken from the atomic positions specified in the input file. A product section must also be provided through e.g. `%block positions_abs_product`. A guess intermediate can also be provided, in which case the NEB chain will place beads on the linear interpolation from reactant to intermediate and intermediate to product. This can be specified with e.g. `%block positions_abs_intermediate`.

Basic Usage

- `num_images: n` [Intermediate integer, default `1`]. Defines the number of Onetep instances that should run in the simulation and enables image-parallel mode. In NEB, this is also the number of beads in the chain.
- `{reactant,product}_energy` [Intermediate real physical, default `N/A`] and `{reactant, product}_rootname` [Intermediate string, default `NONE`]. Both the reactant and product energies must be known at the start of the calculation. The energy can be specified either as a raw total energy or as a rootname from which Onetep can read the tightbox NGWF and density kernel (and, in EDFT, Hamiltonian) files from a previous calculation, or they can be calculated from scratch if neither is specified. The reactant and product energies needn't be specified in the same way.

Additional Controls

- `neb_ci_delay: n` [Intermediate integer, default `-1`]. Defines the number of BFGS steps the chain should take before enabling a climbing image. Negative numbers disable the climbing image entirely.
- `neb_print_summary` [Intermediate boolean, default `.true.`]. If `.true.`, Onetep will print NEB convergence information as well as a summary of the reduced reaction coordinate and relative energy of each bead after each NEB step to the original stdout.

Convergence

Currently the calculation is considered converged when each bead is individually converged. These tolerances are used instead of the geomopt ones.

- `tssearch_energy_tol` [Expert real physical, default `1.0e-5 Ha`]. Convergence tolerance on change in bead energy in one NEB step.
- `tssearch_force_tol` [Expert real physical, default `0.005 Ha/bohr`]. Convergence tolerance on max force on any atom.
- `tssearch_disp_tol` [Expert real physical, default `0.01 bohr`]. Convergence tolerance on displacement of any atom in one NEB step.

Other Image-Parallel Keywords

- `image_sizes` [Expert string, default `DEFAULT`]. If specified in the input file, a string of the format `i|j|k|l|m|...` can be used to individually size the images in an image-parallel run. The number of sections specified should be equal the number of images in the run and the sum of the image sizes should be equal the number of MPI processes specified at runtime.

[Jonsson1998] H. Jónsson, G. Mills, and K. W. Jacobsen. *Chapter 16: Nudged elastic band method for finding minimum energy paths of transitions*, Classical and Quantum Dynamics in Condensed Phase Simulations Part II.

[Henkelman2000] G. Henkelman, and H. Jónsson. *Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points*. J. Chem. Phys. **113**, 9978 (2000).

[Henkelman2000-2] G. Henkelman, B. P. Uberuaga, and H. Jónsson. *A climbing image nudged elastic band method for finding saddle points and minimum energy paths*. J. Chem. Phys. **113**, 9901 (2000).

[Cordier2018] Copyright P. Cordier http://umet.univ-lille.fr/Projets/RheoMan/en/to-learn-more-about/nudged-elastic-band.php Accessed 21 April 2018.

CHAPTER

SEVEN

SPECTROSCOPY AND TRANSPORT

7.1 Calculating the Local/Partial Density of States and Angular Momentum Projected Density of States

Author
Nicholas D.M. Hine, University of Warwick (originally Imperial College London)

Author
Jolyon Aarons, University of Warwick

Date
June 2019 (Updated by Jolyon Aarons to add angular momentum PDOS information).

Date
Originally written by Nicholas D.M. Hine April 2012.

7.1.1 What is being calculated?

The local density of states (LDOS) provides a description encompassing both the spatial and energetic distribution of the single-particle eigenstates simultaneously. The angular momentum projected density of states (PDOS) decomposes the density of states energetic distribution into angular momentum components. Both decompositions may be combined into an LPDOS. The LDOS and PDOS can thus be two valuable sources of information for understanding and interpreting electronic structure calculations.

In the local-orbital framework of ONETEP [Skylaris2005], both decompositions are achieved by first performing a diagonalisation of the Hamiltonian matrix in the NGWF basis. This is a post-processing step performed at the end of the calculation, once NGWF and density kernel convergence have been achieved. While this comes with a $O(N^3)$ computational cost, the prefactor is low because the NGWF basis is generally quite small. Therefore, such a diagonalisation remains fast up to quite large system sizes, particularly if a parallel eigensolver such as ScaLAPACK is used.

The generalised eigenproblem that needs to be solved to provide the eigenvalues and eigenvectors is:

$$\sum_{\beta} H_{\alpha\beta} M^{\beta}{}_{n} = \epsilon_n \sum_{\beta} S_{\alpha\beta} M^{\beta}{}_{n} \tag{7.1}$$

The matrix $M^{\beta}{}_{n}$ describes the eigenvectors, which take the form $|\psi_n\rangle = \sum_{\beta} |\phi_\beta\rangle M^{\beta}{}_{n}$. In ONETEP, Eq. (7.1) can be solved using either the LAPACK routine DSYGVX or (preferably) the ScaLAPACK routine PDSYGVX, depending on whether -DSCALAPACK has been specified at compile time, and ScaLAPACK libraries have been provided.

The result is the eigenvalues $\{\epsilon_n\}$ and eigenvectors $M^{\beta}{}_{n}$. From these, the total density of states can be obtained as

$$D(\epsilon) = \sum_{n} \delta(\epsilon - \epsilon_n) \ . \tag{7.2}$$

In practice the delta function is replaced with a Gaussian with broadening σ, typically of the order of $0.1\,\mathrm{eV}$:

$$\delta(\epsilon - \epsilon_n) \approx \sqrt{\frac{\log(2)}{\pi\sigma^2}} * \exp\left(\frac{-\log(2)(\epsilon-\epsilon_n)^2}{\pi\sigma^2}\right).$$

Local Density of States

The local density of states in a given region I is calculated by projecting each eigenstate onto the local orbitals of region I, as

$$D_I(\epsilon) = \sum_n \delta(\epsilon - \epsilon_n)\ \langle\psi_n| \sum_{\alpha\in I} \left(|\phi^\alpha\rangle\langle\phi_\alpha|\right) |\psi_n\rangle. \tag{7.3}$$

Here, the non-orthogonality of the NGWFs when used as projectors means that the ket must be contravariant. We are therefore implicitly using the contravariant dual of the NGWF (as in DFT+U [O-Regan2011], [O-Regan2012]). Fortunately, the functions $|\phi^\alpha\rangle$ need not be explicitly constructed in real space: the relationship $\langle\phi_\alpha|\phi^\beta\rangle = \delta_{\alpha\beta}$ implies that we can re-write Eq. (7.3) as:

$$\begin{aligned} D_I(\epsilon) = &\sum_n \delta(\epsilon-\epsilon_n) \sum_{\beta,\gamma,\alpha\in I} (M^\dagger)_n{}^\gamma \langle\phi_\gamma| \left(|\phi^\alpha\rangle\langle\phi_\alpha|\right) |\phi_\beta\rangle M^\beta{}_n \\ = &\sum_n \delta(\epsilon-\epsilon_n) \sum_{\alpha\in I} (M^\dagger)_n{}^\alpha \left(\sum_\beta S_{\alpha\beta} M^\beta{}_n\right) \end{aligned}$$

What is therefore obtained is a series of functions $D_I(\epsilon)$ for each of the chosen regions I, which may be the NGWFs of a single atom, or those of a group of atom types.

Angular Momentum Projected Density of States

To calculate the PDOS, we need to insert an additional resolution of the identity into equation (7.4) using a basis of angular momentum resolved functions on which to project our NGWFS, $|\chi_{l,m}\rangle$:

$$D_{l,I}(\epsilon) \approx \sum_n \delta(\epsilon-\epsilon_n) \sum_{\alpha,l\in I} (M^\dagger)_n{}^\alpha \sum_{m\in l} \langle\phi_\alpha|\chi'_{\alpha lm}\rangle \sum_{l'm'} \Lambda^{lm,l'm'} \sum_\beta \left(\langle\chi'_{l'm'}|\phi_\beta\rangle M^\beta{}_n\right), \tag{7.4}$$

where we need to include the overlap matrix of angular momentum resolved functions, Λ, since this basis is also non-orthogonal.

We have considerable scope in which basis we choose for the angular momentum resolved functions. Effectively, this is a set of spherical harmonics multiplied by some radial term. In ONETEP, we currently have two options implemented for the radial term: either spherical waves, or pseudo-atomic functions, as used to initialise the NGWFs, before optimisation in the NGWF SCF loop. More details about the theory behind these options as well as tests and comparisons can be found in our paper [Aarons2019].

7.1.2 Performing an LDOS Calculation

An LDOS calculation is performed as part of the optional post-processing activated using `do_properties: T` or using `task: PROPERTIES`. To activate LDOS we then need to specify the Gaussian broadening, such as `dos_smear : 0.1 eV`. The default value of `dos_smear : -0.1 eV` disables LDOS.

Then, we need to specify the groups of atom types. This is done via a block, with each line listing a group of atoms. For example, in a benzene ring, we might use the following to find the contributions of the carbon and hydrogen atoms respectively:

```
%block species_ldos_groups
  C
  H
%endblock species_ldos_groups
```

A more complex example would be for a GaAs nanorod with hydrogen termination on the faces. If we wished to see the LDOS varying over 5 layers, labelled 1-5, we could use:

```
%block species_ldos_groups
  Ga1 As1 H1
  Ga2 As2 H2
  Ga3 As3 H3
  Ga4 As4 H4
  Ga5 As5 H5
%endblock species_ldos_groups
```

Examples of the use of LDOS analysis, including example plots, can be found in several recent papers employing ONETEP [Avraam2011], [Avraam2012], [Hine2012].

7.1.3 Performing a PDOS Calculation

The default settings in ONETEP for PDOS calculations are to use the pseudo-atomic states as the angular momentum resolved projection basis with a Löwdin orthogonalisation. For most applications, the spilling parameter associated with this basis will be sufficiently small. PDOS calculations are enabled as part of the optional post-processing by writing `do_properties : T` into the ONETEP input file, along with a Gaussian smearing width, such as `dos_smear : 0.1 eV` and a maximum angular momentum in the angular momentum resolved projection basis, such as `pdos_max_l : 2` to include up to d-states (when using the default pseudo-atomic basis, this will also be limited by the maximum angular momentum state in each species, as calculated by the pseudo-atomic solver).

Local PDOS (LPDOS) Calculations

If you intend to calculate an angular momentum projected DOS on a subset of atoms, this can be achieved by specifying a block in the input file, in the same way as for LDOS. The block can be set up by using the `species_pdos_groups` keyword. For example, in a benzene ring calculation, if you want to find the contributions to the PDOS coming from solely carbon atoms and solely hydrogen atoms, you could write:

```
%block species_pdos_groups
  C
  H
%endblock species_pdos_groups
```

If you also want the combined contribution from carbon and hydrogen to each angular momentum channel, you should add a line for this:

```
%block species_pdos_groups
  C
  H
  C H
%endblock species_pdos_groups
```

This will calculate PDOS histogram data up to `pdos_max_l` for each line. As many or as few combinations of species as you require can be calculated by adding extra lines.

If you instead want a specific subset of atoms of a particular species, this can be achieved easily by labelling this subset differently to the others in its species. For example, if you have the following species specification:

```
%block species
  Pt Pt 78 9 9.0
%endblock species

%block species_atomic_set
  Pt SOLVE conf=5d9 6s1 6p0
%endblock species_atomic_set

%block species_pot
  Pt platinum.paw
%endblock species_pot

%block species_pdos_groups
  Pt
%endblock species_pdos_groups

%block positions_abs
  Pt 8.8292 12.2847 8.7330
  Pt 9.2819 11.1839 11.1325
%endblock positions_abs
```

Then you may wish to duplicate the platinum species definitions to label a subset of the atoms in the `positions_abs` block, as shown here for example:

```
%block species
  Pt Pt 78 9 9.0
  Pt1 Pt 78 9 9.0
%endblock species

%block species_atomic_set
  Pt SOLVE conf=5d9 6s1 6p0
  Pt1 SOLVE conf=5d9 6s1 6p0
%endblock species_atomic_set

%block species_pot
  Pt platinum.paw
  Pt1 platinum.paw
%endblock species_pot

%block species_pdos_groups
  Pt
  Pt1
  Pt Pt1
%endblock species_pdos_groups

%block positions_abs
  Pt  8.8292 12.2847 8.7330
  Pt1 9.2819 11.1839 11.1325
```

(continues on next page)

(continued from previous page)

```
%endblock positions_abs
```

and hence calculate PDOS contributions for subsets of atoms of a single species.

Expert PDOS Options

Further options are available in the ONETEP PDOS functionality to control the quality of the projection. These will be unneeded in most cases, but if, for instance, you are observing larger spilling parameters than your requirements permit, you may wish to enable some of these options.

The most reliable way we have found to reduce the spilling parameter is to use a spherical-wave basis rather than the pseudo-atomic basis as the angular momentum resolved projection basis. To do this in ONETEP, add `pdos_pseudoatomic : F` to your input file. By default, this will create a set of contracted spherical waves by fitting spherical waves to your converged NGWFs, via the contraction coefficients.

The spherical wave basis is contracted by default to reduce the memory requirements of the code. You may, however, not see an improvement in the spilling parameter by using this set. To be certain of reducing the spilling parameter, you should also opt to use the full, non-contracted spherical wave basis, by setting `pdos_reduce_sws` : T in your input file, along with an adequately large `pdos_max_l`. For `pdos_max_l` you can start by running with 2 and increase to 3 if required. If you choose to take this approach, beware of the memory requirements, which can be *up to* 10 times greater.

If you choose to use a contracted set, then you almost certainly want to use the default fitting coefficients (fitted to NGWFs). These can be changed to unity by setting `pdos_construct_basis : F`, however this is not likely to improve your results, and is likely to be removed in a future version of ONETEP due to it being mainly of use for debugging purposes. To reduce the spilling parameter with the contracted set, we recommend increasing the `pdos_max_l` parameter.

In specialised cases, you may also wish to *not* sum over the magnetic quantum number. This can be achieved by setting `pdos_sum_mag : F`. This will give histogram data for every magnetic quantum number of every angular momentum channel of each atom group.

Interpreting Outputs

ONETEPs PDOS outputs are written to several files as well as to stdout, which is itself usually redirected to the main log/output file. The PDOS output to stdout will look something like the following (for an input file with 3 `pdos_groups` and `pdos_max_l=2`):

```
================ Projected Density of States (pDOS) calculation ================

Constructing AM resolved functions  ...... done

Performing overlap integrals ...  done

Computing pDOS weights ...  done

All bands spilling parameter =   2.16 %
Occupancy-weighted spilling parameter =   0.30 %

 => Outputting data for OptaDOS <=

Writing pDOS weights to file "Pt30.val_pdos_bin" ... done
```

(continues on next page)

(continued from previous page)

```
Writing band gradients to file "Pt3O.val_dome_bin" ... done

Writing Castep output cell file to "Pt3O-out.cell" ... done

 => Computing Gaussian smeared pDOS <=
Writing "Pt3O_PDOS.txt" ...  done

 => Computing Occupancy-weighted Gaussian smeared pDOS <=
Writing "Pt3O_occ_PDOS.txt" ...  done
  => Band centres:
 S band centre of group 1:  -10.784858 eV
 P band centre of group 1:   -6.380333 eV
 D band centre of group 1:   -1.992269 eV
 S band centre of group 2:   -3.492084 eV
 P band centre of group 2:   -5.494629 eV
 D band centre of group 2:   -1.992269 eV
 S band centre of group 3:  -20.033217 eV
 P band centre of group 3:   -6.607254 eV
  Band centres done. <=
  => Integrated number of electrons in each AM band:
 S num electrons of group 1:    3.769061
 P num electrons of group 1:    5.624284
 D num electrons of group 1:   26.497454
 S num electrons of group 2:    2.107330
 P num electrons of group 2:    1.147080
 D num electrons of group 2:   26.497454
 S num electrons of group 3:    1.661731
 P num electrons of group 3:    4.477204
  Integrated number of electrons done. <=
================================================================================
```

First, we can see the spilling parameters – effectively how well the angular momentum resolved basis is able to represent the NGWFs. A lower value is better; if you are running production calculations, you should want a value lower than a few percent. If not, consider making some of the changes suggested in the expert options section above. The all-bands value includes un-occupied bands as well as valence states, which will be the same as the occupancy-weighted version unless you are using EDFT with a finite electronic temperature.

Following this are the files for use with **OptaDOS**. OptaDOS is a freely available piece of software for plotting various DOS projections. If you intend to use it, then please follow the CASTEP section of the OptaDOS manual with these files, as they are compatible.

The histogram files are then written, "*_PDOS.txt". These also come in all-bands and occupancy weighted flavours (the occupancy weighted variant is more reliable for a usual ground state calculation with ONETEP as the conduction states are not well described. Only if you are doing LPDOS on the output of a ONETEP conduction calculation the occupancy un-weighted LPDOS outputs will be meaningful). The order of columns is firstly the energy column, followed by the angular momentum component columns (i.e. s,p,d...) for each pdos group. This can be plotted trivially with xmgrace, or any other plotting tool.

Finally ONETEP reports the energy and occupancy weighted averages of the PDOS, so called-band centres, useful in catalysis (e.g. the value "d-band centre" is a very useful decsriptor about the ability of a metal surface to bind atomic oxygen and other types of adsorbates) and the integrated number of electrons in each component.

[Skylaris2005] C.-K. Skylaris, P. D. Haynes, A. A. Mostofi, and M. C. Payne, J. Chem. Phys. **122**, 084119 (2005).

[O-Regan2011] D. D. O'Regan, M. C. Payne and A. A. Mostofi, Phys. Rev. B **83**, 245124 (2011).

[O-Regan2012] D. D. O'Regan, N. D. M. Hine, M. C. Payne and A. A. Mostofi, Phys. Rev. B **85**, 085107 (2012).

[Avraam2011] P. W. Avraam, N. D. M. Hine, P. Tangney, and P. D. Haynes, Phys. Rev. B **83**, 241402(R) (2011).

[Avraam2012] P. W. Avraam, N. D. M. Hine, P. Tangney, and P. D. Haynes, Phys. Rev. B **85**, 115404 (2012).

[Hine2012] N. D. M. Hine, P. W. Avraam, P. Tangney, and P. D. Haynes, J. Phys. Conf. Ser. (2012).

[Aarons2019] J. A. Aarons, L. G. Verga, N. D. M. Hine, and C.-K. Skylaris, Submitted (2019).

7.2 Linear response time-dependent density-functional theory (LR-TDDFT)

Author
Tim Zuehlsdorff, Imperial College London

7.2.1 Linear Response TDDFT

The linear response TDDFT (LR-TDDFT) functionality in ONETEP allows the calculation of the low energy excited states of a system in linear scaling effort. In contrast to time-evolution TDDFT, where the density matrix of the system is propagated explicitly in time, LR-TDDFT recasts the problem of finding TDDFT excitation energies into an effective non-hermitian eigenvalue equation of the form:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{pmatrix} \begin{pmatrix} \vec{\mathbf{X}} \\ \vec{\mathbf{Y}} \end{pmatrix} = \omega \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \vec{\mathbf{X}} \\ \vec{\mathbf{Y}} \end{pmatrix} \tag{7.5}$$

where the elements of the block matrices **A** and **B** can be expressed in canonical Kohn-Sham representation as

$$\begin{aligned} A_{cv,c'v'} &= \delta_{c,c'}\delta_{v,v'}(\epsilon_c^{\mathrm{KS}} - \epsilon_v^{\mathrm{KS}}) + K_{cv,c'v'} \\ B_{cv,c'v'} &= K_{cv,v'c'} \end{aligned}$$

Here, c and v denote Kohn-Sham conduction and valence states and **K** is the coupling matrix with elements given by

$$\begin{aligned} K_{cv,c'v'} = 2 \int \mathrm{d}^3r \mathrm{d}^3r' \left[\frac{1}{|\mathbf{r}-\mathbf{r}'|} + \left. \frac{\delta^2 E_{\mathrm{xc}}}{\delta\rho(\mathbf{r})\delta\rho(\mathbf{r}')} \right|_{\rho^{\{0\}}} \right] \\ \times \psi_c^{\mathrm{KS}*}(\mathbf{r})\psi_v^{\mathrm{KS}}(\mathbf{r})\psi_{v'}^{\mathrm{KS}*}(\mathbf{r}')\psi_{c'}^{\mathrm{KS}}(\mathbf{r}'). \end{aligned}$$

with E_{xc} being the exchange-correlation energy. Its second derivative, evaluated at the ground-state density $\rho^{\{0\}}$ of the system, is normally referred to as the exchange-correlation kernel.

The above equation can be understood as an effective 2-particle Hamiltonian consisting of a diagonal part of conduction-valence eigenvalue differences and a coupling term $K_{cv,c'v'}$ connecting individual Kohn-Sham excitations.

In ONETEP, LR-TDDFT is implemented both in terms of the full TDDFT eigenvalue equation (Eqn. [full_tddft]) and in the Tamm-Dancoff approximation, a commonly used simplification to the full non-hermitian eigenvalue equation, where the off diagonal elements **B** are set to zero. The problem of calculating the TDDFT excitation energies thus becomes equivalent to solving the hermitian eigenvalue equation

$$\mathbf{A}\vec{\mathbf{X}} = \omega\vec{\mathbf{X}}$$

The Tamm-Dancoff approximation violates time-reversal symmetry and oscillator strength sum rules and can blue-shift strong peaks in the spectrum by up to 0.3 eV, however, dark states are typically left almost unaltered from their corresponding states in the Tamm-Dancoff approximation.

In the ONETEP code, the Tamm-Dancoff eigenvalue equation is re-expressed in terms of two sets of NGWFs, one optimised for the valence space (denoted as $\{\phi_\alpha\}$) and one optimised for a low energy subspace of the conduction manifold (denoted as $\{\chi_\beta\}$, see the documentation of the conduction NGWF optimiation functionality). Furthermore, the eigenvalue equation is solved iteratively for the lowest few eigenvalues using a conjugate gradient method. In order to do so we define the action **q** of operator **A** acting $\vec{\mathbf{X}}$ in conduction-valence NGWF space as

$$(q^{\chi\phi})^{\alpha\beta} = (P^{\{\mathrm{c}\}}H^{\chi}P^{\{1\}} - P^{\{1\}}H^{\phi}P^{\{\mathrm{v}\}})^{\alpha\beta} + (P^{\{\mathrm{c}\}}V_{\mathrm{SCF}}^{\{1\}\chi\phi}P^{\{\mathrm{v}\}})^{\alpha\beta}. \tag{7.6}$$

where $\mathbf{H}^{\chi}$ and $\mathbf{H}^{\phi}$ are the Hamiltonians in conduction and valence NGWF representation respectively, $\mathbf{P}^{\{c\}}$ and $\mathbf{P}^{\{v\}}$ denote the conduction and valence density matrices and $\mathbf{P}^{\{1\}}$ is the response density matrix, a representation of the trial vector $\vec{\mathbf{X}}$ in conduction-valence NGWF space. $V_{\mathrm{SCF}}^{\{1\}}$ is the first order response of the system due to the density $\rho^{\{1\}}(\mathbf{r})$ associated with $\mathbf{P}^{\{1\}}$. Under this redefinition of the action **A** in conduction-valence NGWF space, finding the lowest N_ω excitation energies is equivalent to minimising

$$\Omega = \sum_{i}^{N_\omega}\omega_i = \sum_{i}^{N_\omega}\left[\frac{\mathrm{Tr}\left[\mathbf{P}_i^{\{1\}\dagger}\mathbf{S}^{\chi}\mathbf{q}_i^{\chi\phi}\mathbf{S}^{\phi}\right]}{\mathrm{Tr}\left[\mathbf{P}_i^{\{1\}\dagger}\mathbf{S}^{\chi}\mathbf{P}_i^{\{1\}}\mathbf{S}^{\phi}\right]}\right]$$

with respect to $\left\{\mathbf{P}_i^{\{1\}}\right\}$ under the constraint

$$\mathrm{Tr}\left[\mathbf{P}_i^{\{1\}\dagger}\mathbf{S}^{\chi}\mathbf{P}_j^{\{1\}}\mathbf{S}^{\phi}\right] = \delta_{ij}. \tag{7.7}$$

If all density matrices involved in the above expressions, ie. $\mathbf{P}^{\{1\}}$, $\mathbf{P}^{\{c\}}$ and $\mathbf{P}^{\{v\}}$ are truncated and thus become sparse, the algorithm scales as $O(N)$ with system size for a fixed number of excitation energies N_ω and as $O(N_\omega^2)$ with the number of excitation energies required.

A similar algorithm can be derived for the full TDDFT eigenvalue equation, where we make use of the change of variables $\mathbf{p} = \vec{\mathbf{X}} + \vec{\mathbf{Y}}$ and $\mathbf{q} = \vec{\mathbf{X}} - \vec{\mathbf{Y}}$. Each TDDFT excitation then has two effective density matrices, $\mathbf{P}^{\{p\}}$ and $\mathbf{P}^{\{q\}}$, associated with it that have the same structure as $\mathbf{P}^{\{1\}}$ in the Tamm-Dancoff approximation. The density matrices do obey an updated orthonormality constraint of the form

$$\frac{1}{2}\left(\mathrm{Tr}\left[\mathbf{P}_i^{\{p\}\dagger}\mathbf{S}^{\chi}\mathbf{P}_j^{\{q\}}\mathbf{S}^{\phi}\right] + \mathrm{Tr}\left[\mathbf{P}_i^{\{q\}\dagger}\mathbf{S}^{\chi}\mathbf{P}_j^{\{p\}}\mathbf{S}^{\phi}\right]\right) = \delta_{ij} \tag{7.8}$$

and an analogous expression for the total energy Ω in full TDDFT can be derived.

7.2.2 Performing a LR-TDDFT calculation

The LR-TDDFT calculation in ONETEP is enabled by setting the task flag to TASK=LR_TDDFT. The LR-TDDFT calculation mode reads in the density kernels and NGWFs of a converged ground state and conduction state calculation, so the .dkn, .dkn_cond, .tightbox_ngwfs and .tightbox_ngwfs_cond files all need to be present. The most important keywords in a TDDFT calculation are:

- `lr_tddft_RPA`: T/F.
  Boolean, default `lr_tddft_RPA`=F. If set to T, the code performs a full TDDFT calculation without relying on the simplified Tamm-Dancoff approximation.
- `lr_tddft_num_states`: n
  Integer, default `lr_tddft_num_states` = 1.
  The keyword specifies how many excitations we want to converge. If set to a positive integer n, the TDDFT algorithm will converge the n lowest excitations of the system.
- `lr_tddft_cg_threshold`: x
  Real, default `lr_tddft_cg_threshold` = 10^{-6}.

The keyword specifies the convergence tolerance on the sum of the n TDDFT excitation energies. If the sum of excitation energies changes by less than x in two consecutive iterations, the calculation is taken to be converged.

- `lr_tddft_maxit_cg`: n
  Integer, default `lr_tddft_maxit_cg` = `60`.
  The maximum number of conjugate gradient iterations the algorithm will perform.
- `lr_tddft_triplet`: T/F
  Boolean, default `lr_tddft_triplet` = `F`.
  Flag that decides whether the `lr_tddft_num_states` = `n` states to be converged are singlet or triplet states.
- `lr_tddft_write_kernels`: T/F
  Boolean, default `lr_tddft_write_kernels` = `T`.
  If the flag is set to T, the TDDFT response density kernels are printed out at every conjugate gradient iteration. These files are necessary to restart a LR_TDDFT calculation.
- `lr_tddft_restart`: T/F
  Boolean, default `lr_tddft_trestart` = `F`.
  If the flag is set to T, the algorithm reads in `lr_tddft_num_states` = `n` response density kernels in .dkn format and uses them as initial trial vectors for a restarted LR_TDDFT calculation.
- `lr_tddft_restart_from_TDA`: T/F
  Boolean, default `lr_tddft_trestart_from_TDA` = `F`.
  If the flag is set to T and `lr_tddft_RPA`: T, the code will read in already converged density kernels $\left\{\mathbf{P}_i^{\{1\}}\right\}$ and use them as a starting guess for a full TDDFT calculation such that $\mathbf{P}_i^{\{p\}} = \mathbf{P}_i^{\{q\}} = \mathbf{P}^{\{1\}}$. In many cases, the full TDDFT results are similar to the Tamm-Dancoff results and this strategy of starting the full TDDFT calculation leads to a rapid convergence.
- `lr_tddft_init_random` T/F
  Boolean, default `lr_tddft_init_random` T.
  By default, the initial TDDFT eigenvector guesses are initialised to random matrices. This yields an unbiased convergence of the TDDFT algorithm but can mean that one starts the optimisation relatively far away from the minimum. If `lr_tddft_init_random`=F, the code instead computes the n minimum energy pure Kohn-Sham transitions in linear-scaling effort and initialises the n TDDFT response density matrices to the pure Kohn-Sham transition density matrices. In many small to medium sized systems this leads to initial states much closer to the TDDFT minimum and rapid convergence. In large extended systems this can yield states that are spurious charge transfer states that are not ideal, especially if a more advanced density matrix truncation scheme is used. In this case it is possible to set the keyword `lr_tddft_init_max_overlap` T, in which, rather than choosing the lowest Kohn-Sham transitions, the code picks the lowest few transitions that also have a maximum overlap of electron and hole densities.
- `lr_tddft_kernel_cutoff`: x
  Real, default `lr_tddft_kernel_cutoff` = `1000a`$_0$.
  Keyword sets a truncation radius on all response density kernels in order to achieve linear scaling computational effort with system size.

While the LR_TDDFT calculation can be made to scale linearly for a fixed number of excitations converged, it should be kept in mind that the algorithm needs to perform orthonormalisation procedures and thus scales as $O(N^2)$ with `lr_tddft_num_states`.

7.2.3 Truncation of the Response density matrix

To run a fully linear scaling TDDFT calculation the response density matrix has to be truncated by setting `lr_tddft_kernel_cutoff`. This truncation introduces numerical errors into the calculation, which mainly manifest themselves in the form that the response density matrices do no longer exactly obey a first order idempotency constraint that is placed on them. The idempotency constraint can be written in form of an invariance equation:

$$\mathbf{P}^{\{1\}'} = \mathbf{P}^{\{c\}}\mathbf{S}^{\chi}\mathbf{P}^{\{1\}}\mathbf{S}^{\phi}\mathbf{P}^{\{v\}} = \mathbf{P}^{\{1\}}$$

To measure the degree to which the invariance relation is violated we make use of a penalty functional $Q\left[\mathbf{P}^{\{1\}}\right]$ given by:

$$Q\left[\mathbf{P}^{\{1\}}\right] = \mathrm{Tr}\left[\left(\mathbf{P}^{\{1\}\dagger}\mathbf{S}^{\chi}\mathbf{P}^{\{1\}}\mathbf{S}^{\phi} - \mathbf{P}^{\{1\}'\dagger}\mathbf{S}^{\chi}\mathbf{P}^{\{1\}'}\mathbf{S}^{\phi}\right)^2\right].$$

For truncated $\mathbf{P}^{\{1\}}$, $Q\left[\mathbf{P}^{\{1\}}\right] \neq 0$ which can lead to problems in the convergence of the conjugate gradient algorithm. In order to avoid these issues, the TDDFT routines perform the minimisation of the energy in an analogous form to the LNV method in ground-state calculations: The auxiliary density kernel $\mathbf{P}^{\{1\}'}$ is used instead of $\mathbf{P}^{\{1\}}$ for the minimisation of Ω. While $\mathbf{P}^{\{1\}'}$ is much less sparse than $\mathbf{P}^{\{1\}}$ it preserves idempotency to the same degree as the conduction and valence density kernel, yielding a stabilised convergence.

However, should $Q\left[\mathbf{P}^{\{1\}}\right]$ diverge significantly from 0 during the calculation, there are routines in place similar to the kernel purification schemes in ground state DFT that force the kernel towards obeying its idempotency constraint. The keyword controlling these routines are given below:

- `lr_tddft_penalty_tol`: x
  Real, default `lr_tddft_penalty_tol` $= 10^{-8}$.
  Keyword sets a tolerance for the penalty functional. If $Q\left[\mathbf{P}^{\{1\}}\right]$ is larger than `lr_tddft_penalty_tol` the algorithm will perform purification iterations in order to decrease the penalty value and force $\mathbf{P}^{\{1\}}$ towards the correct idempotency behaviour.
- `lr_tddft_maxit_pen`: n
  Integer, default `lr_tddft_maxit_pen` $= 20$.
  The maximum number purification iterations performed per conjugate gradient step.

7.2.4 More advanced TDDFT kernel truncation schemes

There are many situations where physical intuition allows one to specify a more sophisticated sparsity pattern than a uniform spherical kernel cutoff on $\mathbf{P}^{\{1\}}$ (or $\mathbf{P}^{\{p\}}$ and $\mathbf{P}^{\{q\}}$ for full TDDFT). For example, in pigment-protein complexes the excitations of interest retain a relative localisation on the pigment and one would ideally converge these states directly, without obtaining any spurious charge transfer states from the pigment to far away regions of the protein, that can arise due to failures in semi-local exchange correlation functionals. This can be achieved by introducing a new block into the input file of the form

```
%block species_tddft_kernel
  label1 label 2 label3 ...
  label5 ...
  ...
%endblock species_tddft_kernel
```

where the labels refer to atom labels. As an example, consider a pigment protein complex, where the pigment atoms are labelled H1, C1 etc. while the protein atoms are labelled H, C, etc. Then we can force the excitations of the system to be fully localised on the pigment by including

```
%block species_tddft_kernel
  C1 H1 ...
%endblock species_tddft_kernel
```

This has the effect of setting all elements of $\mathbf{P}^{\{1\}}$ to zero that correspond to conduction or valence NGWFs centered on atoms of the environment. In this way the electrostatic effects of the environment are treated fully quantum mechanically, while no delocalisation into the protein is allowed. If one would like to introduce a coupling to the environment but wants to suppress any charge transfer coupling between the pigment and its environment, it is possible to specify

```
%block species_tddft_kernel
  C1 H1 ...
  C H ...
%endblock species_tddft_kernel
```

It is possible to specify an arbitrary number of subregions in the system in this way. It is also possible to list the same species in different lines, allowing for charge transfer interactions between some atom types of two regions but not others.

Rather than having the off-diagonal charge-transfer blocks defined in `%block species_tddft_kernel` set exactly to zero, it is also possible to give these blocks a more realistic sparsity pattern, for example that of the overlap matrix. While this process still suppresses any significant amount of charge transfer between TDDFT regions, it can be used to allow overlapping NGWFs from different TDDFT regions to contribute to the TDDFT transition density. In order to do so, set the block

```
%block species_tddft_ct
  C1 H1 ...
  C2 H2 ...
%endblock species_tddft_ct
```

and set `lr_tddft_ct_length` to a chosen cutoff length for the charge-transfer interaction between the specified blocks. For example, if the off-diagonal blocks of the response density matrix (corresponding to charge-transfer excitations between TDDFT regions) should have the same sparsity pattern as the overlap matrix, set `lr_tddft_ct_length` to twice the NGWF localisation radius.

7.2.5 Preconditioning

The TDDFT eigenvalue problem is generally ill-conditioned, which can lead to a relatively slow convergence. For this reason, it is possible to precondition the eigenvalue problem, which is achieved by solving a linear system iteratively to a certain tolerance at each conjugate gradient step. Solving the linear system only requires matrix-matrix multiplications and is very cheap for small and medium sized systems, however, it can get more costly for very large systems, especially when no kernel truncation is used. In these cases, it can be necessary to reduce the number of default iterations of the preconditioner. The main keywords controlling the preconditioner are

- `lr_tddft_precond`: T/F
  Boolean, default `lr_tddft_precond = T`.
  Flag that decides whether the preconditioner is switched on or off.
- `lr_tddft_precond_iter`: n
  Integer, default `lr_tddft_precond_iter = 20`.
  Maximum number of iterations in the linear system solver applying the preconditioner.
- `lr_tddft_precond_tol`: x
  Real, default `lr_tddft_precond_tol` $= 10^{-8}$.
  The tolerance to which the linear system is solved in the preconditioner. Choosing a large tolerance means that the preconditioner is only applied approximately during each iteration.

7.2.6 Representation of the unoccupied subspace

In the LR_TDDFT method as implemented in ONETEP, the user has two options regarding the representation of the unoccupied subspace. The first option is to define the active unoccupied subspace of the calculation to only contain the Kohn-Sham states that were explicitly optimised in the COND calculation. The other is to make use of a projector onto the entire unoccupied subspace, where we redefine the conduction density matrix as:

$$\mathbf{P}^{\{c\}} = \left((\mathbf{S}^{\chi})^{-1} - (\mathbf{S}^{\chi})^{-1} \mathbf{S}^{\chi\phi} \mathbf{P}^{\{v\}} \left(\mathbf{S}^{\chi\phi} \right)^{\dagger} (\mathbf{S}^{\chi})^{-1} \right).$$

The first option has the advantage that we only include states for which the NGWFs are well optimised, but has the drawback that some excitations converge very slowly with the size of the unoccupied subspace and thus a good convergence with the number of conduction states optimised is hard to reach. The second method implicitly includes the entire unoccupied subspace (to the extent that it is representable by a small, localised NGWF representation), but has the disadvantage that now states are included in the calculation for which the NGWFs are not optimised. Furthermore, the density matrix defined above is no longer strictly idempotent, leading to violations of the idempotency condition and thus a non-vanishing penalty functional $Q\left[\mathbf{P}^{\{1\}}\right]$, requiring kernel purification iterations as described in the previous section.

The problem of loss of idempotency can be avoided by using the joint NGWF set to represent the conduction space when using the projector. While this increases the computational cost of the LR_TDDFT calculation by a factor of 2, it preserves the idempotency of $\mathbf{P}^{\{c\}}$ and is the recommended option when using the projector onto the unoccupied subspace.

The keywords controlling the use of the projector are

- `lr_tddft_projector`: T/F
  Boolean, default `lr_tddft_projector = T`.
  If the flag is set to T, the conduction density matrix $\mathbf{P}^{\{c\}}$ is redefined to be a projector onto the entire unoccupied subspace.
- `lr_tddft_joint_set`: T/F
  Boolean, default `lr_tddft_joint_set = T`.

If the flag is set to T, the joint NGWF set is used to represent the conduction space in the LR_TDDFT calculation.

7.2.7 Calculations in implicit solvent

A TDDFT calculation in implicit solvent is performed in an analogous way to the implicit solvent calculation in combination with a conduction optimisation (see the documentation of the conduction optimisation for further details). By default, the implicit solvent only acts on the ground state of the system and thus influences the conduction and valence Kohn-Sham states mixed into the TDDFT calculation. However, a screening of the response density due to a dynamic dielectric constant is not included in the calculation. In order to activate dynamic screening effects in TDDFT, the user can set the keyword `lr_optical_permittivity` to the effective dynamic dielectric constant ϵ_∞ of the system in question.

7.2.8 Outputs

The LR_TDDFT calculation will produce a number of outputs. At the end of the calculation, the individual excitation energies and oscillator strengths will be computed and printed in the main ONETEP output file. Furthermore, the energies and oscillator strengths are used to generate a excitation spectrum written to the textfile rootname.tddft_spectrum. The peaks in the spectrum are Gaussians of a width controlled by `lr_tddft_spectrum_smear`. Furthermore, by default, density cube files of the response density, the electron and the hole density for each excitation are printed out. The LR_TDDFT code can also perform an analysis of individual excitations, where the response density matrix is decomposed into dominant Kohn-Sham transitions. Since this analysis requires the Kohn-Sham eigenstates and thus a diagonalisation of the Hamiltonian, it scales as $O(N^3)$ and should not be performed for very large system sizes.

The keywords controlling these outputs are:

- `lr_tddft_write_densities`: T/F
  Boolean, default `lr_tddft_write_densities` = `T`.
  If the flag is set to T, the response density, electron density and hole density for each excitation is computed and written into a .cube file.
- `lr_tddft_analysis`: T/F
  Boolean, default `lr_tddft_analysis` = `F`.
  If the flag is set to T, a full $O(N^3)$ analysis of each TDDFT excitation is performed in which the response density is decomposed into dominant Kohn-Sham transitions.

7.2.9 Good practices and common problems

- The quality of the TDDFT excitation energies critically depends on the representation of the conduction space manifold. Any excitation that has a large contribution from an unoccupied state that is not explicitly optimised in the COND calculation is not expected to be represented correctly in the LR_TDDFT calculation. In general it is advisable to optimise as many conduction states as possible. However, high energy conduction states are often very delocalised and only representable if the conduction NGWF radius is increased significantly, thus leading to poor computational efficiency. In practice, there is a tradeoff between computational efficiency and the representation of the conduction state manifold (see also the documentation on conduction state optimisation on this issue). Generally, TDDFT excitations should be converged with respect to both the conduction NGWF radius and the number of conduction states explicitly optimised.
- Since the ground state and conduction density kernels are used as projectors onto the occupied and unoccupied subspace in LR_TDDFT, one often finds that the inner loop of the SINGLEPOINT and COND optimisation has to be converged to a higher degree of accuracy to achieve well behaved TDDFT results. It is therefore recommended to increase MAXIT_LNV and MINIT_LNV from their default value in the SINGLEPOINT and COND calculation. If no density kernel cutoff is used, the penalty functional value in the LR_TDDFT calculation

should be vanishingly small. If the number increases significantly during a calculation or if the code begins to perform penalty optimisation steps, that is a clear sign that the initial conduction and valence density kernels are not converged well enough.

- In order to perform a LR_TDDFT calculation that scales fully linearly with system size, all density matrices involved have to be sparse and thus a KERNEL_CUTOFF has to be set for both the SINGLEPOINT and COND calculation. Using a density matrix truncation on the conduction states can sometimes be difficult depending on how the subspace of optimised conduction states is chosen and care has to be taken to prevent unphysical results.
- When running calculations in full linear scaling mode, the ground state and conduction density kernels are no longer strictly idempotent, which means that the penalty functional in LR_TDDFT will no longer be strictly zero. The code might perform penalty functional optimisation steps to keep the idempotency error small. However, these idempotency corrections can cause the conjugate gradient algorithm to stagnate and can even cause the energy to increase. If this happens, it is an indication that the minimum energy and maximum level of convergence for this truncation of the density kernel has been reached.
- When placing a truncation onto the the response density kernels it should be kept in mind that this may cause the optimisation to miss certain low energy excitations completely. Very long range charge-transfer type excitations cannot be represented by a truncated response density kernel and will thus be missing from the spectrum of excitations converged. However, well localised excitations should be unaffected. In a similar way, if the TDDFT kernel is limited to a certain region, it should be checked whether increasing the region leads to a smooth convergence of the energy of the localised state within the region.

7.2.10 Reference

For further background regarding the theory behind the LR_TDDFT method in ONETEP, as well as a number of benchmark tests, see

- Linear-scaling time-dependent density-functional theory in the linear response formalism, T. J. Zuehlsdorff, N. D. M. Hine, J. S. Spencer, N. M. Harrison, D. J. Riley, and P. D. Haynes, J. Chem. Phys. **139**, 064104 (2013)

7.3 Electron Energy Loss Calculations

Author
Edward Tait, University of Cambridge

Author
Nicholas Hine, University of Warwick

Date
September 2015

Date
Updated and Converted by NDMH September 2022

7.3.1 Theory

Computation of Electron Energy Loss (EEL) spectra in the Kohn-Sham formal- ism relies on the application of Fermi's Golden Rule to compute the imaginary part of the dielectric function,

$$\epsilon_2(\omega) = \frac{1}{\Omega} \sum_c \sum_i |\langle \psi_i | \exp(i\mathbf{r} \cdot \mathbf{q}) | \psi_c \rangle|^2 \delta(E_i - E_c - \omega) \,,$$

Here ω is the transition energy, Ω the unit cell volume, the ψ_i are (all electron) conduction band states, the ψ_c are core states, with respective energies E_i and E_c. $\mathbf{r}$ is the position operator (defined as the displacement from the nucleus whose core electrons are being excited). The δ-function conserves energy. ONETEP relies on the external tool OptaDoS[1] for the computation of ϵ_2 and only needs to supply matrix elements in a compatible form. The rest of this section discusses the calculation of these elements.

Projector Augmented Wave

In common with many plane wave codes onetep uses pseudopotentials to increase computational efficiency. A drawback of pseudopotentials is poor representation of the all-electron Kohn-Sham wavefunction close to the nucleus, this however is exactly the region in which the matrix elements used for EELS simulation are computed. To overcome this obstacle the projector augmented wave (PAW) formalism of Blochl[2] is adopted, which permits reconstruction of all-electron states ψ (and thus matrix elements) from pseudo-wavefunctions $\widetilde{\psi}$:

$$\langle \phi_\alpha | \hat{O} | b \rangle = \underbrace{\langle \phi_\alpha | \hat{O} | b \rangle}_{\text{Cartesian Grid}} + \sum_i \langle \phi_\alpha | \widetilde{p}_\text{i} \rangle \underbrace{\left(\langle \varphi_\text{i} | \hat{O} | b \rangle - \langle \widetilde{\varphi}_\text{i} | \hat{O} | b \rangle \right)}_{\text{Radial Grids}} \tag{7.9}$$

This process is accomplished by decomposing the pseudo-wavefunction in the augmentation region into a sum of pseudo partial waves, $\widetilde{\varphi}_i$, the weights in this sum are determined using the projectors $\widetilde{p}_i$. These pseudo partial waves are subtracted off and all electron partial waves, φ_i, added in their place. Simply put, within the augmentation regions (close to the nucleus) the pseudised part of pseudo-wavefunction is subtracted off and the all electron part is added back on.

Several PAW data sets are freely available (subject to the caveat that they should be thoroughly tested for a specific use case). onetep accepts PAW pseudopotentials in the abinit file format. Core wavefunctions, as produced by the pseudopotential generator are also required, these are less frequently available for download, but your PAW library should provide input files for a pseudopotential generator which can be used to produce core wavefunction data sets (again in the abinit format).

Generation of Position-Core Kets

To calculate the PAW matrix elements we must compute terms of the form: $\langle \widetilde{\psi} | \mathbf{r} | \psi_\text{c} \rangle$

As an intermediate we compute expressions of the form: $\langle \phi_\alpha | \mathbf{r} | \psi_\text{c} \rangle$

with ϕ_α an NGWF. The integral implied by this bra-ket must be computed on the grid. The bra term, an NGWF, is readily available in this form but the ket $|\psi_\text{c}\rangle$ must be constructed.

A Fourier space method is used, as it was found to offer superior numerical performance and correctly treats periodic systems without further modification. This method exploits the fact that differentiation in Fourier space is equivalent

to multiplication by the position vector in real space:

$$\psi_{\mathrm{c}}(\mathbf{r}) = \sum_{\mathbf{G}}^{\mathbf{G}_{\max}} \exp(i\mathbf{G}\cdot\mathbf{r})$$

$$\rightarrow \sum_{\mathbf{G}}^{\mathbf{G}_{\max}} (\nabla_{\mathbf{G}}(\psi_{\mathrm{c}}(\mathbf{G}))\exp(i\mathbf{G}\cdot\mathbf{r}) = \sum_{\mathbf{G}}^{\mathbf{G}_{\max}} (\nabla_{\mathbf{G}}(\psi_{\mathrm{c}}(\mathbf{G})\exp(i\mathbf{G}\cdot\mathbf{r}))$$

$$- \sum_{\mathbf{G}}^{\mathbf{G}_{\max}} \psi_{\mathrm{c}}(\mathbf{G})\nabla_{\mathbf{G}}\exp(i\mathbf{G}\cdot\mathbf{r})$$

$$= - \sum_{\mathbf{G}}^{\mathbf{G}_{\max}} i\mathbf{r}\psi_{\mathrm{c}}(\mathbf{G})\exp(i\mathbf{G}\cdot\mathbf{r}))$$

$$= -i\mathbf{r}\psi_{\mathrm{c}}(\mathbf{r})$$

Conduction Optimisation

Core loss calculations rely on an accurate description of conduction band states, in onetep it is necessary to perform a conduction optimisation calculation in order to obtain a second NGWF set and kernel which correctly represent the conduction manifold. Detail of the process may be found in the document "Conduction NGWF optimisation and optical absorption spectra in ONETEP"[3] and is published[4].

Two main parameters should be supplied: `cond_energy_range` which sets the energy window above the HOMO in which conduction states will be optimised. The second parameter cond energy gap specifies the energy gap between the highest optimised state and the lowest un- optimised state, this parameter is used to prevent attempts to optimise only part of a set of degenerate states

7.3.2 Practical Example

In this section we will discuss the procedure for running an EELS calculation on a toy system: silene (the silicon equivalent of ethene). You will need to obtain PAW pseudopotentials for Silicon and Hydrogen and the associated core wavefunction data for Silicon. The author used the JTH pseudopotentials[5], though can't offer any guarantee of their suitability for any particular use. A boilerplate input file for a onetep EELS calculation is provided in Appendix B.

This input file can also be downloaded from the tutorials section of the onetep site. There are a few differences between this input and the one for a single point calculation:

- PAW is mandatory
- We specify a second species for the atom whose core electrons we're exciting
- Because a conduction calculation is being performed we must provide a species cond block
- We must provide a `species_core_wf` block and every species must be listed there.

The input file can be run swiftly on a single node and should produce a large number of output files. Most of these files are `.cube`'s of wavefunctions produced by default during the properties calculations. The files of interest are the `.elnes_bin` files, which contain OptaDoS compatible matrix elements.

A little more setup is needed before we can run OptaDoS (using the silene example):

- A dummy castep `silene-out.cell` file must be produced, and it must contain a symmetry block
- By default two `.elnes bin` files are produced, one based on Kohn-Sham wavefunctions represented using only the valence NGWFs (`silene_val_grad.elnes_bin`) and a second which makes use of the joint basis of valence and conduction NGWFs (`silene_joint_grad.elnes_bin`)

- As per the discussion above, you should choose the latter and copy it to `silene.elnes bin`.
- A `silene.bands` file must be produced, this is best done by copying `silene.joint bands` to `silene.bands`.
- An OptaDoS input file, `silene.odi` is needed.

To assist in these tasks a utility script, `prep_optados_eels`, is provided in the utils folder of the onetep distribution. Run it with the calculation seed name as its argument and the steps listed above will be completed automatically.

The .odi file produced should be regarded as a basic template, consult the OptaDoS documentation[6] if you wish to use more advanced features. Note that at the moment only fixed broadening is supported by onetep. When you are satisfied with your OptaDoS input file, execute OptaDoS with your calculation seed name as the argument. All being well, you should see a .dat file which you can plot with your favorite tool. Individual edges are listed sequentially in the file, so a little post processing with awk or python is needed to separate the edges for individual plotting

7.3.3 OptaDoS

The OptaDoS code provides a single tool to compute densities of states and optical spectra of various sorts. OptaDoS provides a number of smearing schemes for evaluating integrals over the Brillouin Zone including fixed and adaptive schemes. At present onetep only supports simple fixed broadening schemes. The prep optados eels utility script provides a template .odi input file:

```
TASK : CORE

# Method OptaDoS should use to work out
# the Fermi energy in the material .
# the insulator method relies on electron
# counting and is best for the systems
# onetep is most commonly used to study .
EFERMI : insulator

# Smearing scheme and width
BROADENING : fixed
DOS_SPACING : 0.01

# Use an average of x , y and z components of
# the position vector matrix elements
CORE_GEOM : polycrystalline

# Parameters below control lifetime and
# instrument broadening
CORE_LAI_BROADENING : true
LAI_GAUSSIAN_WIDTH : 0.6
LAI_LORENTZIAN_WIDTH : 0.2
LAI_LORENTZIAN_SCALE : 0.1
```

7.3.4 References

[1] RJ Nicholls, AJ Morris, CJ Pickard, and JR Yates. Optados-a new tool for eels calculations. In Journal of Physics: Conference Series, volume 371, page 012062. IOP Publishing, 2012.

[2] Peter E Blochl. Projector augmented-wave method. Physical Review B, 50(24):17953, 1994.

[3] Laura E. Ratcliff. Conduction NGWF optimisation and optical absorption spectra in onetep. http://www2.tcm.phy.cam.ac.uk/onetep/pmwiki/ uploads/Main/Documentation/conduction.pdf, 2011. [Online; accessed 22-Sept-2015].

[4] Laura E Ratcliff, Nicholas DM Hine, and Peter D Haynes. Calculating op- tical absorption spectra for large systems using linear-scaling density func- tional theory. Physical Review B, 84(16):165131, 2011.

[5] Francois Jollet, Marc Torrent, and Natalie Holzwarth. Generation of projec- tor augmented-wave atomic data: A 71 element validated table in the XML format. Computer Physics Communications, 185(4):1246 – 1254, 2014.

[6] Andrew J. Morris, Rebecca J. Nicholls, Chris J. Pickard, and Jonathan R. Yates. OptaDoS user guide. http://www.cmmp.ucl.ac.uk/~ajm/ optados/files/user_guide_1.0.pdf, 2014. [Online; accessed 22-Sept- 2015]

7.4 Electronic transport calculations

Author
Robert A. Bell, University of Cambridge (rab207@cam.ac.uk)

7.4.1 Introduction

This document describes the use of the electronic transport functionality that is implemented in the ONETEP code [Bell2014]. The implementation computes the ballistic Landauer-Büttiker conductance at zero bias through a device and associated properties using electronic structure derived from density functional theory (DFT). The calculation proceeds as a post-processing step after a ground-state single-point calculation.

This document focuses on the practical aspects of setting up calculations; a detailed technical explanation of the method can be found in [Bell2014]. For a detailed discussion of the Landauer-Büttiker formalism and conductance derived from DFT see [DiVentra2008], [Datta1995].

I first briefly outline what is being calculated, I then give an example calculation and suitable input parameters, and then finally a full input parameter listing is given.

7.4.2 What is being calculated?

The transport calculation determines the ballistic conductance through a device. Current can flow between two leads via the connection made by the central scattering region. We will refer to this geometry as the *device* geometry.

Under the Landauer formalism, the contribution to the conductance between two leads (indexed i and j) from electrons with energy E is given by

$$G_{ij}(E) = \frac{2e^2}{h} T_{ij}(E)$$

where $T_{ij}(E)$ is the transmission function. The transmission function is the central quantity to the Landauer-Büttiker conductance and is calculated using a Green's function technique:

$$T_{ij}(E) = \mathrm{tr}\Big[\Gamma_i(E)\, G_\mathrm{d}(E)\, \Gamma_j(E)\, G_\mathrm{d}^\dagger(E)\Big], \tag{7.10}$$

$$G_\mathrm{d}(E) = \big[(E + i\eta)S_\mathrm{d} - H_\mathrm{d} - \Sigma(E)\big]^{-1}, \tag{7.11}$$

where H_d, S_d the Hamiltonian and overlap matrices for the device, and η is an infinitessimal positive energy that selects the retarded response of the Green's function G_d. The interaction with the semi-infinite leads accounted for through the lead self-energies Σ_j, with $\Sigma = \sum_j \Sigma_j$. The coupling matrices are defined as $\Gamma_j = i(\Sigma_j - \Sigma_j^\dagger)$.

From the Green's function, one may also calculate the density of states within the device region as

$$\mathcal{N}(E) = -\frac{1}{\pi}\mathrm{Im}\mathrm{tr}\big[G_\mathrm{d}(E)S_\mathrm{d}\big].$$

Further post processing steps may be performed to determine the eigenchannels. [Bell2014], [Paulsson2007]

7.4.3 Building the device

The principal quantities required in Eqns. (7.10) and (7.11) are the device Hamiltonian and overlap matrices, and the lead self energies. The approach adopted in this implementation is to extract these matrix elements from the ground-state electronic structure obtained after a standard single-point calculation. This approach corresponds to a non-self consistent calculation of the transmission spectrum and device electronic structure. Note that this procedure differs from self consistent approaches, such as e.g. [Brandbyge2002], where the open boundary conditions of the leads are treated explicitly. At zero bias, however, both methods produce equivalent results yet the non-self consistent approach is often far less computationally expensive.

A detailed description of the matrix element extraction procedure, as implemented in the ONETEP code, is given in [Bell2014] which should be referred to in the first instance. Summarising briefly, to calculate the transmission through any device geometry the user must construct another *auxilliary simulation* structure that contains this device. The *only* electronic structure that is calculated is that of the auxilliary simulation geometry.

The structure of the auxilliary simulation geometry must take a specific form, consisting of the central region that connects the leads and a bulk-like region for each lead. Each lead region must have a local electronic structure that is bulk-like, and be large enough that it contains at least two principal layers of the bulk lead structure.[1] The device geometry is then defined as all the atoms contained within the central region and the two lead principal layers for each lead.

For a device connecting two leads, the device Hamiltonian then looks like

$$H_\mathrm{d} = \begin{pmatrix} \mathbf{h}_{00,L} & \mathbf{h}_{0,L} & \cdot & \cdot & \cdot \\ \mathbf{h}_{01,L}^\dagger & \mathbf{h}_{00,L} & \mathbf{h}_{LC} & \cdot & \cdot \\ \cdot & \mathbf{h}_{LC}^\dagger & \mathbf{h}_{C} & \mathbf{h}_{CR} & \cdot \\ \cdot & \cdot & \mathbf{h}_{CR}^\dagger & \mathbf{h}_{00,R} & \mathbf{h}_{01,R} \\ \cdot & \cdot & \cdot & \mathbf{h}_{01,R}^\dagger & \mathbf{h}_{00,R} \end{pmatrix}$$

where $\mathbf{h}_{00,L}$ is the on-site Hamiltonian block for the left lead principal layer, $\mathbf{h}_{01,L}$ is the coupling Hamiltonian between principal layers, $\mathbf{h}_{LC}$ is the coupling block between the left lead principal layer and the central region, and $\mathbf{h}_C$ is the on-site block for the central region. As a result of the localisation of the NGWFs the device Hamiltonian takes the block tri-diagonal form, with zero matrix blocks denoted by a dot. An equivalent form is found for the device overlap matrix.

The lead self energies are computed using the $\mathbf{h}_{00,L}, \mathbf{h}_{01,L}$ blocks via the method of Ref. [Lopez-Sancho1985]. Note that these blocks are contained within the device Hamiltonian H_d.

The device matrix elements are extracted from the electronic structure of the auxilliary simulation geometry. This procedure is entirely automated, however, and the user need only supply a list of atoms that define the device, and the subsets of these atoms that form the two principal layers for each lead. *i.e.* for an n-lead device, the user must define $2n + 1$ sets of atoms.

[1] A lead principal layer is defined as the minimum number of primitive unit cells for that lead such that Hamiltonian and overlap matrix elements between atoms in non-adjacent principal layers are exactly zero. This is guaranteed to be true if the periodic length of the principal layer is larger than twice the maximum NGWF radius.

A single limitation of the implementation is that the two principal layers of any lead must be the exact same structure *with atoms in the same relative ordering*. This ensures that the lead self energies are computed correctly. The implementation will not do this automatically, however a check is performed to ensure that it is true before the calculation is started.

For a more detailed discussion of the procedure for extracting matrix elements see [Bell2014].

7.4.4 Setting up a calculation

The best way to explain how to set up an input is through an example, and here I give an explanation of the parameters required to calculation the transmission between two semi-infinite organic wires. Specifically, I will explain the setup for calculating the tunnelling current between a polyacetylene wire and polyyne wire where each wire is semi-infinite and terminated with a hydrogen atom. I will, however, assume that the user knows how to successfully converge the standard ONETEP single-point calculation.

A suitable auxiliary simulation geometry is shown in schematic form in Fig. Fig. 7.1, consisting of molecular fragments of the polyacetylene and polyyne wires located in vacuum. This entire geometry contains 52 atoms. I will assume that these atoms are ordered in the input file by their position from left to right.

Suppose that I wish to calculate the transmission spectrum in a $\pm 2eV$ window about the Fermi energy with a resolution of $0.01eV$. The transport specific input parameters are as follows:

```
etrans_lcr               : T
etrans_bulk              : T
etrans_emin              : -2.0 eV
etrans_emax              :  2.0 eV
etrans_enum              :  401
etrans_calculate_lead_mu : T

%block etrans_setup
  10  47
%endblock etrans_setup

%block etrans_leads
  10  17  18  25  unit_cells=2
  44  47  40  43  unit_cells=2
%endblock etrans_leads

! rest of input file...
```

The first six lines indicate that we wish to calculate the transmission between the two leads (the LCR transmision, see [Bell2014]), and the maximum transmission that could be injected by each lead (the bulk transmission), and the energy range the calculation will be performed over. I am also indicating that I want to calculate the lead band structures.

The first block after these six lines is the `etrans_setup` block which states that all atoms between atom 10 and 47 (inclusive, and in the order found in the input file) are defined as the device geometry. This region is given by the long-dashed box in . Atoms outside this set will not be included in the transmission calculation, but will be used when calculating the ground-state electronic structure.

The final block is the `etrans_leads` block which gives the subset of these device atoms that define the two principal layers for each lead. Each lead is given on a separate line, with the first pair of indices defining the principal layer furthest from the central region, and the second pair the principal layer closest to the central region. Note that this ordering of the two principal layers is important. Finally, I have indicated that each principal layer is in fact two primitive unit cells of the lead structure by using `unit_cells=2`. This tells ONETEP to symmetrise the lead matrix elements to reflect this periodicity. The atoms contained within each principal layer are indicated in by dotted boxes.

This choice of auxilliary simulation geometry tries to ensure that the lead principal layers are sufficiently far from the

ends of the molecular fragments that the local electronic structure within the lead is bulk-like. This has required the use of buffer atoms (atoms 1–9 and 48–52) that are not included in the transport calculation. Some buffer has also been used between the principal layers and the central tunnelling gap (atoms 26–34 and 35–39) for the same reason. In practice, the size of this buffer is probably too small and the lead principal layers will not be well converged to the bulk. This can be tested by comparing the band structures of each lead to the corresponding bulk band structure for that lead, which can be calculated separately. If the lead band structure is not converged, the size of the buffer region should be increased.

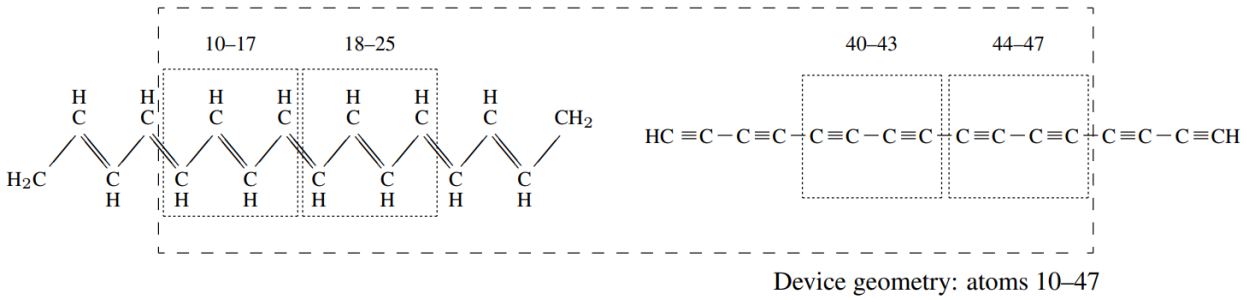

Fig. 7.1: A possible auxilliary simulation geometry for calculating the tunnelling transmission between a polyacetylene wire and a polyyne wire. The atoms appear ordered in the input file by their position from left to right.

7.4.5 Computational scaling

The dominant computationally-intensive task in the transport routines is the computing of the device Green's function. This calculation is performed using an efficient block tri-diagonal Gaussian-elimination algorithm [Petersen2008]. This algorithm results in memory usage and operation counts that scale linearly in the number of atoms. The pre-factor to this scaling depends on the precise geometry of the device: the more one-dimensional a device (i.e. the fewer NGWFs that overlap), the lower this pre-factor.

7.4.6 Information on parallelisation

The calculation of transmission coefficients is parallelised over the energy points, with each MPI process performing the calculation in serial. No internal communications are necessary making the routines scale perfectly with the number of MPI processes. However, as all matrices must be replicated on each MPI process, the memory requirements can be large. On entering the transport routines but prior to starting the calculation, an estimate of the additional memory required by the transport routines is printed. Note that this estimate does not include the memory required by the rest of the ONETEP routines.

Parallelisation using OpenMP threading is not currently available. However, if the code is linked against Intel's Math Kernel Library (MKL), the linear algebra routines can make use of multi-threading within this library. To do this, the flag `-DMKLOMP` must be included during compilation, and the number of threads can be set from the input file using `threads_num_mkl`.

The calculation of the eigenchannels can only be parallelised if linking against the ScaLAPACK library to make use of parallel dense algebra. To enable this, the flag `-DSCALAPACK` must be included at compilation.

7.4.7 Calculations using the joint NGWF basis

Transmission calculations using the joint (valence + conduction) NGWF basis set have often been found to be numerically unstable and produce qualitively incorrect results when compared to calculations using the valence NGWF basis alone. This is a result of ill-conditioning of the joint basis when the valence and conduction NGWFs are very similar in character. This is a known issue, however there is currently no fix.

For some systems it has been found that the valence NGWFs alone are capable of describing low-energy conduction states with good accuracy, and therefore the joint basis is not needed. It is advised that the valence basis is used in the first instance.

If you are certain that the joint basis is required, then proceed with caution and always compare the results generated with the joint basis to those generated using the valence basis. For energies below the Fermi energy, the two calculations should coincide. If they do not, or if errors (e.g. complaining about computing the transfer matrix/self energy, or lead band structures) are found then it may not be possible to use the joint basis.

If using the joint NGWF basis, all output files have `.joint` prepended to the extension.

7.4.8 Warnings and fixing errors

A useful check is to ensure that the lead band structure is as expected, and that the lead occupancy is correct. It is also useful to check the lead/device `.xyz` files to ensure that the leads have the expected geometry.

The following are the main warning messages that may be encountered, and how they may be tackled.

`Inversion of (eS-H)_lcr failed`

> **Cause**: Failed to calculate the Green's function as it is singular at this energy. This can happen if etrans_ecmplx is too small, or if localised states are present near this energy.
>
> **Severity**: Low
>
> **Fix**: It is safe to ignore this warning; the transmission coefficients and associated values are not written to file. If it occurs over a wide energy range, or in an energy range of interest, try increasing `etrans_ecmplx`.

`Warning in compute_transfer:  Failed to compute transfer matrix.`

> **Cause**: Failed to compute the lead self energy. This can happen if `etrans_ecmplx` is too small, or if localised states are present near this energy.
>
> **Severity**: Low
>
> **Fix**: It is safe to ignore this warning; the transmission coefficients and associated values are not written to file. If it occurs over a wide energy range, or in an energy range of interest, try increasing `etrans_ecmplx`. If using the joint NGWF basis, this may indicate that the basis is ill-conditioned and that the calculation is numerically unstable.

`Lead electronic occupancy is significantly different ...`

> **Cause**: Large discrepancy between the ionic and electronic charge in the lead. Likely due to under-converged buffer region between this lead and the scattering region.
>
> **Severity**: high
>
> **Fix**: Check that the lead band structure is what is expected. Increase buffer region between the lead and the scattering region. Check the ground state DFT calculation is converged and that the density kernel occupancies are 0 or 1 (if using the default LNV algorithm).

Unable to determine lead potential for lead ...

Cause: lead potential calculation for this lead failed: could not determine the lead band structure. This may indicate that the leads have been defined incorrectly. Check the lead structure.

Severity: possibly high

Fix: Check the lead structure. Check the matrix elements contained in the lead .hsm output file are resonable, that the main weight is along the matrix diagonal and that the values are sensible. If using joint NGWF basis, try using the valence basis only (`task = properties` not `properties_cond`). The calculation will attempt to continue, but check carefully the results are as expected.

7.4.9 Full input parameter description

Main parameters

`etrans_lcr: T/F` [Logical, default `etrans_lcr: F`]

Calculate the transmission spectrum between all pairs of leads in the system.

`etrans_bulk: T/F` [Logical, default `etrans_bulk: F`]

Calculate the bulk transmission spectrum for the leads.

`etrans_setup: [block]`

```
%block etrans_setup
     start end
%endblock etrans_setup
```

The block defining the atoms to be used in the transport calculation. `start/end` are integers giving the indices (from the input file) of the first/last atoms to be included as part of the system. All atoms between `start` and `end` are used in the calculation; atoms outside this range are buffer regions and ignored. The atoms consituting the leads must be contained within this region.

`etrans_leads: [block]`

```
%block etrans_leads
     start0 end0 start1 end1
     start0 end0 start1 end1 unit_cells=<n> file=<lead2.hsm>
     ...
%endblock etrans_leads
```

The block defining the atoms to be used as the leads. Each line defines a new lead. `start0/end0` are integers giving the indices (from the input file) of the first/last atoms of the lead principal layer farthest from the central region; `start1/end1` are integers giving the indices of the first/last atom of the principal later closest to the central region.

The first principal layer geometry must be a periodic repeat of the lead geometry (i.e. same number of atoms, in the same relative ordering in the input file).

For each lead, two optional tags can be defined. The option `unit_cells=<n>` forces the translational symmetry of the principal layer matrix elements, where `n` is the number of primitive unit cells in the principal layer. The option `file=<lead1.hsm>` allows for the lead matrix elements to be read in from a `.hsm` file. Reading in matrix elements is generally unnecessary, however it may be used effectively by taking the `.hsm` of a separate pristine/bulk transport calculation to ensure that the matrix elements are exactly at their bulk values. The positions and ordering of atoms in the lead principal layers in both calculations must be identical: this is not checked in the calculation.

etrans_enum: n [Integer, default etrans_enum: 50]

Number of transmission energy points calculated for. Energies are distributed uniformly.

etrans_emax : $E_{\max}$ [Physical, default etrans_emax: -0.2 Hartree]

The maximum energy above the reference energy transmission is calculated for.

etrans_emin : $E_{\min}$ [Physical, default etrans_emin: 0.2 Hartree]

The minimum energy below the reference energy transmission is calculated for.

etrans_eref_method: LEADS|REFERENCE|DIAG [String, default etrans_eref_method: LEADS]

The method used to determine the reference energy E_{ref} for the transmission calculation.

If etrans_eref_method: LEADS, the reference energy is set as the average lead chemical potential.

If etrans_eref_method: DIAG, the reference energy is set as the Fermi energy of the original DFT system.

If etrans_eref_method: REFERENCE, the reference energy is set with etrans_eref.

If unset, this will be determined by calculating the average chemical potential of the leads, or if that calculation fails, the Fermi energy of the original DFT system will be used.

etrans_eref : E_{ref} [Physical, default etrans_eref: Determined automatically]

Reference energy around which the transmission will be calculated. Energies are distributed uniformly in range $E_{\text{ref}} + E_{\min} \leq E \leq E_{\text{ref}} + E_{\max}$.

This value is only used if etrans_eref_method : REFERENCE, otherwise it is determined automatically by the method given by etrans_eref_method.

etrans_num_eigchan : n_{chan} [Integer, default etrans_num_eigchan: 0]

The number of eigenchannel transmissions calculated. The default does not decompose into eigenchannel transmissions.

etrans_plot_eigchan : T/F [Logical, default etrans_plot_eigchan: F]

Whether to plot the transmission eigenchannels of the LCR system.

etrans_plot_eigchan_energies : [block]

```
%block etrans_plot_eigchan_energies
    {Ha|eV} ! optional energy unit
    E_1
    E_2
    ...
%endblock etrans_plot_eigchan_energies
```

The energies at which the eigenchannels are calculated and plotted. The first line may, optionally, define the energy unit; if undefined, the energy unit is Hartree.

Plotting the eigenchannels uses an algorithm that scales with the cube of the number of atoms in the LCR system. Compiling with the ScaLAPACK library is strongly recommended to reduce memory requirements.

etrans_write_xyz : T/F [Logical, default etrans_write_xyz : T]

Whether to write, separately, the lead and LCR geometries to file in the .xyz file format. This is useful for ensuring that the correct atoms have been chosen for the leads/LCR region.

Tweaking/optional parameters

`etrans_ecmplx` : η [Physical, default `etrans_ecmplx: 1e-6 Hartree`]

The small complex energy added to calculate the retarded Green's function.

Ideally this should be infinitesimally small, however too small values create numerical instabilities. Large values improve numerical stability, but creates artificial scattering, reducing transmission.

`etrans_calculate_lead_mu` : `T/F` [Logical, default `etrans_calculate_lead_mu: T`]

Determine the chemical potential of the leads using a tight-binding approach. If `etrans_eref` is unset, the average lead chemical potential will be used as the reference energy. The band structure for each lead is saved to file `basename_lead<nn>.bands`.

`etrans_num_lead_kpoints` : n_k [Integer, default `etrans_num_lead_kpoints: 32`]

The number of lead band structure k-points sampled to determine the lead chemical potential. The k-points are distributed uniformly between Γ and X.

`etrans_same_leads` : `T/F` [Logical, default `etrans_same_leads: F`]

Reuse the self energy from one lead for all leads. If all leads are identical, this may lead to a small computational saving. The saving is typically negligible however.

`etrans_write_hs` : `T/F` [Logical, default `etrans_write_hs: F`]

Write the lead Hamiltonian and overlap matrices to disk.

`etrans_lead_disp_tol` : Δr [Physical, default `etrans_lead_disp_tol : 1.0 bohr`]

The geometries of each lead and corresponding first principal layer should be identical; this parameter determines how strictly this is enforced. This should only be done if you are clear of the consequences.

This may be useful in the case that the lead unit cell is much larger than the NGWF radius. If the distance between the lead and the first principal layer furthest from the lead is much larger than 2 NGWF radii, these atoms do not directly influence the non-zero matrix elements between the principal layer and the lead, and the coupling matrix h_L is (approximately) independent of the structure of these atoms. Therefore, it may be possible to use a much smaller buffer region between the lead and the first principal layer by including part of the first principal layer in that buffer. The first principal layer and lead geometry need not be identical, however both regions must contain the same number of atoms and orbitals.

For each lead, the translation vector between the i^{th} atoms of the lead and first principal layer is calculated: $\mathbf{R}_i = \mathbf{r}_{i,\mathrm{lead}} - \mathbf{r}_{i,\mathrm{PL}}$. If the geometries of the lead and first principal layer are identical, all of these translation vectors are identical. The maximum allowed difference between these displacement vectors is $\Delta r > |\mathbf{R}_i - \mathbf{R}_j|$.

If the lead crosses the supercell boundary, this check will fail. This can be overridden by increasing Δr to some very large value.

`etrans_lead_size_check` : `T/F` [Logical, default `etrans_lead_size_check: T`]

If true, a check is performed to ensure that each lead forms complete principal layer. The run will abort if the periodic length of the lead unit cells, as defined in the input file, are smaller than twice the maximum NGWF radius of the species in that lead.

Turning off this check is not advised as the lead band structure and self energies can be inaccurate. This can also be a problem in situations where the left lead can interact with the principal layer of the right lead, allowing current to flow between the leads in the wrong direction, bypassing the central scattering region.

`etrans_seed_lead` [Integer, default `etrans_seed_lead: 1`]

The lead used to seed the block tri-diagonal partitioning. This should only be relevant for devices with more than two leads where one lead is much larger than the other leads.

`threads_num_mkl` [Integer, default `threads_num_mkl: 1`]

If the code is linked against Intel's Math Kernel Library (MKL), this defines the number of threads used in the linear algebra routines. The flag `-DMKLOMP` must be set during compilation to enable this parameter, otherwise it is ignored.

7.4.10 Output file description

Output files contain self-explanatory headers for each column. For spin polarised calculations, a separate file is outputted for each spin channel.

`basename_LCR.TRC`

The LCR transmission coefficients between different pairs of leads of the system.

`basename_LCR_channels_lead<nn>.TRC`

The LCR transmission coefficients decomposed into eigenchannels. The lead number defines which lead acts as the source.

`basename_LCR_E<nn>_lead<mm>_chan<ll>_{real\|imag}.cube`

The plotted LCR eigenchannel. nn is the the energy index from `%block etrans_plot_eigchan_energies`; mm is the lead that acts as the source; ll is the eigenchannel number. The real and imaginary part are printed separately.

`basename_BULK.TRC`

The bulk transmission coefficients for each lead of the system.

`basename_LCR.DOS`

The density of states for the LCR system.

`basename_BULK.DOS`

The density of states for each lead of the system.

`basename_lead<nn>.bands`

Bandstructure of the lead in the .bands format. The number of k-points is set with etrans_num_lead_kpoints.

`basename_device.xyz` `basename_lead<nn>.xyz`

The geometries of the device and leads in .xyz file format.

`basename_lead<nn>.hsm`

The lead matrix elements in Fortran (unformatted) binary format. Note that all energies are in Hartree. It is planned, but currently not possible, to be able to access the device matrix elements. Please contact the developers if you are interested in using this functionality. The format of this file is

```
character(len=*) :: block_type  ! always lead
character(len=*) :: ham_type    ! valence or joint
integer          :: nspin       ! number of spin channels
integer          :: orbs(4)     ! NGWF indices corresponding
                                ! to the atoms defined
                                ! in block_etrans_leads
```

(continues on next page)

(continued from previous page)

```
integer          :: norb        ! matrix sizes
real(kind=8)     :: eref        ! the lead chemical potential

real(kind=8)   :: h00(norb,norb,nspin)
real(kind=8)   :: h01(norb,norb,nspin)
real(kind=8)   :: s00(norb,norb)
real(kind=8)   :: s01(norb,norb)
```

[Bell2014] R.A. Bell, S.M.-M. Dubois, M.C. Payne, A. A. Mostofi, *in preparation* (2014)

[DiVentra2008] M. Di Ventra, *Electrical Transport in Nanoscale Systems*, (Cambridge University Press, Cambridge 2008)

[Datta1995] S. Datta, *Electronic Transport in Mesoscopic System*, 2nd ed. (Cambridge University Press, Cambridge 1995)

[Paulsson2007] M. Paulsson, M. Brandbyge, Phys. Rev. B. **76** 115117 (2007)

[Lopez-Sancho1985] M.P. Lopez-Sancho, J.M. Lopez-Sancho, J. Rubio, *J. Phys. F: Met. Phys.* **15**, 851 (1985)

[Petersen2008] D.E. Petersen *et al.*, *J. Comp. Phys* **227**, 3174 (2008)

7.5 Bandstructure (spectral-function) unfolding

Author
Gabriel Constantinescu, University of Cambridge

As described in the Supplementary Information in the work of Constantinescu and Hine [Constantinescu2015], spectral function unfolding is the means through which one can study the bandstructure of the primitive cell from simulations involving (often complicated) supercells. For details theoretical descriptions, one should visit the aforementioned article [Constantinescu2015].
In this documentation, I will give a short explanation of the procedures required to unfold the bandstructure in a ONETEP "Properties" calculation. We strongly suggest that one reads the entire document carefully before attempting any calculations, as there are some stringent requirements along the way. For further details, please contact Gabriel Constantinescu, the code author. Essentially, all one will need is the following group of keywords and blocks:

```
BSUNFLD_CALCULATE : T
BSUNFLD_TRANSFORMATION : t11 t12 t13 t21 t22 t23 t31 t32 t33

%block SPECIES_BSUNFLD_GROUPS
Species_name-1 Species_name-2 ... Species_name-N
%endblock SPECIES_BSUNFLD_GROUPS

BS_PERTURBATIVE_SOC : F
BSUNFLD_NUM_ATOMS_PRIM : nat_prim
BSUNFLD_RESTART : F
BSUNFLD_NUM_EIGENVALUES : num_eigvl

%block BSUNFLD_KPOINT_PATH
 fraction_G1_kpt-1 fraction_G2_kpt-1 fraction_G3_kpt-1
```

(continues on next page)

(continued from previous page)

```
 fraction_G1_kpt-2 fraction_G2_kpt-2 fraction_G3_kpt-2
 ...
 fraction_G1_kpt-N fraction_G2_kpt-N fraction_G3_kpt-N
%endblock BSUNFLD_KPOINT_PATH

BSUNFLD_NUM_KPTS_PATH : num_kpts_path
```

For each in turn:

- BSUNFLD_CALCULATE setting this to True will enable the calculation of the unfolded spectral function. Default: False
- BSUNFLD_TRANSFORMATION is a list of 9 integers representing the flattened transformation matrix from the primitive cell to the supercell: $\mathbf{T} = \begin{bmatrix} t11 & t12 & t13 \\ t21 & t22 & t23 \\ t31 & t32 & t33 \end{bmatrix}$, where $\mathbf{T} \begin{bmatrix} \mathbf{a_1} \\ \mathbf{a_2} \\ \mathbf{a_3} \end{bmatrix} = \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \\ \mathbf{A_3} \end{bmatrix}$, with $\mathbf{a_i}$ ($\mathbf{A_i}$) being the lattice vectors of the primitive cell (supercell). For instance, BSUNFLD_TRANSFORMATION : 4 0 0 0 4 0 0 0 1 would correspond to an implicit 4x4x1 supercell. Default: 1 0 0 0 1 0 0 0 1, corresponding to a 1x1x1 supercell.
- The block SPECIES_BSUNFLD_GROUPS contains the atoms on which we are projecting; one needs to specify the of the atoms species (can be different from the chemical symbol) on which the spectral function is projected. Currently, only one group at a time is allowed. Example:

  ```
  %block SPECIES_BSUNFLD_GROUPS
  Mo Se1
  %endblock SPECIES_BSUNFLD_GROUPS
  ```

  this projects onto the $MoSe_2$ monolayer in a $MoSe_2$/WSe_2 heterostructure, where the Se atoms belonging to the $MoSe_2$ monolayer have been denoted as *Se1*.
- The logical keyword BS_PERTURBATIVE_SOC controls the inclusion (True) or exclusion (False) of perturbative spin-orbit coupling in our calculation. Note that if set to True, the eigenvalues are no longer spin-degenerate and one will have twice the number of states. Since this option essentially quadruples the size of the Hamiltonian and overlap matrix, the calculation will be significantly slower. Default: False.
- The integer keyword BSUNFLD_NUM_ATOMS_PRIM indicates the number of atoms in the primitive cell onto which you are unfolding. This brings us to a major requirement of the code: while the projected atoms can be split across the list of all input atoms, **the former must be grouped by primitive cells, always maintaining the same order of the atoms in the primitive cells!** Example: if atoms **At1** and **At2** form a primitive cell, the input ordering "At3 **At1 At2** At4 **At1 At2** A3" is correct, while "At3 **At1 At2** At4 **At2** At3 **At1**" or "At3 **At1 At2** At4 **At2 At1** At3" are incorrect.
- The logical keyword BSUNFLD_RESTART controls whether one wishes to reuse previously calculated values for the spectral function. The restart procedure works as follows: the code writes a restart file ("unfolded_specfunc_red.dat") where it stores the calculated values for the unfolded spectral function at unique k-points of the monolayer - the values that have not been calculated yet are filled with zeros. If this file is present and the restart keyword is set to true, the code will read in the previously calculated values and skip them in the current computations. The final output file ("unfolded_specfunc.dat") will only be written once all the k-points have been dealt with.

  Each line in final output file or the restart file contains 8 numbers: the eigenvalue (in eV), the real part of the unfolded spectral function (at that k-point and eigenvalue), the imaginary part of the unfolded spectral function, the eigenvalue count (from 1 to 2· BSUNFLD_NUM_EIGENVALUES, the x, y, and z component of the current primitive-cell k-point ($^{-1}$), and the index of the k-point (from 1 to the total number of considered k-points).

After the final output file has been obtained, one can use a discretisation script (should be found on the ONETEP webside, in the utilities section), in order to obtain a file that is ready to plot with gnuplot.

- The integer keyword `BSUNFLD_NUM_EIGENVALUES` controls the number of eigenvalues (above and below the Fermi level) for which the spectral function is calculated. If set to negative values, `BSUNFLD_NUM_EIGENVALUES` will internally be set to the number of nondegenerate occupied eigenstates. Thus, in total, for each k-point, the spectral function will be calculated at $2\cdot$ `BSUNFLD_NUM_EIGENVALUES`.
- The block `BSUNFLD_KPOINT_PATH` indicates the fractional coordinates of the different k-points that mark endpoints of desired paths through the **primitive-cell Brillouin zone. The fractional coordinates are with respect to the implied reciprocal lattice vectors of the primitive cell, not the simulation cell (supercell).**
- `BSUNFLD_NUM_KPTS_PATH` represents the number of k-points calculated along each path from the `BSUNFLD_KPOINT_PATH` block. This number includes the endpoints of each path. Default: 2 (the endpoints only)

[Constantinescu2015] Constantinescu, G. C.; Hine, N. D. M. *Phys. Rev. B* **2015**, *91*, 195416

7.6 Configuration Interaction (CI)

Author
David Turban, University of Cambridge

7.6.1 Introduction

The aim of the CI functionality is to evaluate the electronic Hamiltonian with respect to a set of reference configurations obtained from constrained DFT (cDFT) and LR-TDDFT. This makes it possible to obtain transition rates between different electronic states using Fermi's Golden Rule (e.g. rate of charge transfer at donor/acceptor-interface in an organic solar cell). Also, CI can help to construct eigenstates of a system in situations where the basic ground- and excited state methods fail due to deficiencies of the approximate exchange-correlation functional.

A classic example for such a case is the binding curve of H_2^+. The LDA functional gives an incorrect dissociation limit with a binding energy that is significantly too small. The reason is that there is a spurious self-interaction of the single electron with itself, such that a delocalisation of the electron over both atoms will always yield a lower energy, even at infinite separation. Even more advanced hybrid functionals (like B3LYP) do not solve this problem since they only include part of the exact Hartree-Fock exchange which would cancel the self-interaction. In this scenario CI offers a way forward. By choosing the two states with the electron fully localised on either atom as references one ensures the correct long-range limit. Finally, CI is used to evaluate the Hamiltonian in the reference basis and obtain approximate eigenstates. This gives a very good match with LDA at short range and also retains the physical dissociation limit.

7.6.2 Theory

As a first step we assume that one intends to find the Hamiltonian matrix element $\langle\Psi_B|\hat{H}|\Psi_A\rangle$ between two cDFT states. In cDFT the constrained solutions are obtained as ground states of the electronic Hamiltonian augmented with a constraining potential that pushes charge (and/or spin) around the system:

$$(\hat{H}+\hat{V}_c)|\Psi_c\rangle = (\hat{H}+V_c\hat{w}_c)|\Psi_c\rangle = F|\Psi_c\rangle.$$

Here the potential is written as the product of its magnitude V_c and a weighting operator $\hat{w}_c$ which specifically acts on the donor and acceptor regions of the system with appropriate signs. In ONETEP the weighting operator is built from

local orbitals (like PAOs or NGWFs) that define the donor and acceptor regions. The eigenvalue F is the energy E of the constrained solution plus a correction due to the (unphysical) constraining potential:

$$F = \langle \Psi_c | \hat{H} + V_c \hat{w}_c | \Psi_c \rangle = E[\rho_c] + V_c \int d\mathbf{r} w_c(\mathbf{r}) \rho_c(\mathbf{r}) = E + V_c N_c.$$

The magnitude of the potential V_c takes the role of a Lagrange multiplier that is chosen such that the amount of displaced charge/spin matches the population target N_c. It should be noted that cDFT states are generally not eigenstates of the electronic Hamiltonian $\hat{H}$.

Using the cDFT potentials we now obtain

$$\begin{aligned} \langle \Psi_B | \hat{H} | \Psi_A \rangle = & \quad \langle \Psi_B | \hat{H} + \hat{V}_A - \hat{V}_A | \Psi_A \rangle \\ = & \quad F_A \langle \Psi_B | \Psi_A \rangle - \langle \Psi_B | \hat{V}_A | \Psi_A \rangle, \end{aligned}$$

which reduces the problem to calculating overlaps of states and matrix elements of the cDFT potentials. At this point the expression still contain the full many-body wave functions Ψ. To obtain a practical computational scheme we replace them with Kohn-Sham (KS) determinants Φ and approximate

$$\langle \Psi_B | \Psi_A \rangle \approx \langle \Phi_B | \Phi_A \rangle, \;\; \langle \Psi_B | \hat{V} | \Psi_A \rangle \approx \langle \Phi_B | \hat{V} | \Phi_A \rangle.$$

We will assume that all orbitals are chosen to be real functions, and not worry about complex conjugation in inner products. The standard result for the overlap of two Slater determinants is given by

$$\begin{aligned} \langle \Phi_B | \Phi_A \rangle = & \int d\mathbf{r}_1 \ldots d\mathbf{r}_N \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1^B(\mathbf{r}_1) & \cdots & \psi_1^B(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^B(\mathbf{r}_1) & \cdots & \psi_N^B(\mathbf{r}_N) \end{vmatrix} \times \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1^A(\mathbf{r}_1) & \cdots & \psi_1^A(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^A(\mathbf{r}_1) & \cdots & \psi_N^A(\mathbf{r}_N) \end{vmatrix} \\ = & \int d\mathbf{r}_1 \ldots d\mathbf{r}_N \psi_1^B(\mathbf{r}_1) \ldots \psi_N^B(\mathbf{r}_N) \begin{vmatrix} \psi_1^A(\mathbf{r}_1) & \cdots & \psi_1^A(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^A(\mathbf{r}_1) & \cdots & \psi_N^A(\mathbf{r}_N) \end{vmatrix} \\ = & \begin{vmatrix} \langle \psi_1^A | \psi_1^B \rangle & \cdots & \langle \psi_1^A | \psi_N^B \rangle \\ \vdots & & \vdots \\ \langle \psi_N^A | \psi_1^B \rangle & \cdots & \langle \psi_N^A | \psi_N^B \rangle \end{vmatrix} = \det\left(S_{AB}\right), \end{aligned}$$

where the functions ψ denote KS orbitals. The result is simply the determinant of the matrix S_{AB} of overlaps between KS orbitals of states A and B. In a similar fashion we can evaluate matrix elements of a potential operator:

$$\begin{aligned} \langle \Phi_B | \hat{V} | \Phi_A \rangle = & \int d\mathbf{r}_1 \ldots d\mathbf{r}_N \psi_1^B(\mathbf{r}_1) \ldots \psi_N^B(\mathbf{r}_N) \left[\sum_i V(\mathbf{r}_i) \right] \times \begin{vmatrix} \psi_1^A(\mathbf{r}_1) & \cdots & \psi_1^A(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^A(\mathbf{r}_1) & \cdots & \psi_N^A(\mathbf{r}_N) \end{vmatrix} \\ = & \sum_i \begin{vmatrix} \langle \psi_1^A | \psi_1^B \rangle & \cdots & \langle \psi_1^A | \hat{V} | \psi_i^B \rangle & \cdots & \langle \psi_1^A | \psi_N^B \rangle \\ \vdots & & \vdots & & \vdots \\ \langle \psi_N^A | \psi_1^B \rangle & \cdots & \langle \psi_N^A | \hat{V} | \psi_i^B \rangle & \cdots & \langle \psi_N^A | \psi_N^B \rangle \end{vmatrix} \\ = & \sum_{ij} \langle \psi_j^A | \hat{V} | \psi_i^B \rangle \cdot C_{j,i} \\ = & \det\left(S_{AB}\right) \times \mathrm{tr}\left(V_{AB} \cdot S_{AB}^{-1}\right). \end{aligned}$$

In the third line the determinant is expanded along the i-th column. $C_{j,i}$ denotes cofactors of S_{AB} which are sign-adapted determinants of the submatrices formed by deleting the j-th row and i-th column of S_{AB}. A well-known theorem in linear algebra states that the matrix of cofactors of an invertible matrix is equal to the transpose of the inverse of the matrix times its determinant.

Next, we discuss how a constrained reference state can be coupled to an excited state from LR-TDDFT. In LR-TDDFT the excited states are represented as superpositions of single-particle excitations from an occupied to an unoccupied orbital. This information is contained in the response density matrix R_{jb}. A particular non-zero entry indicates that a transition from valence orbital ψ_j to conduction orbital ψ_b contributes to the excited state. In the following indices $i, j, k, \ldots$ will denote valence orbitals and indices $b, c, d, \ldots$ conduction orbitals. A natural choice for a DFT wave function of such an excitation that retains the response density by construction is

$$|\Phi\rangle = \sum_{jb} R_{jb} |\Phi_j^b\rangle.$$

$|\Phi_j^b\rangle$ denotes a Slater determinant constructed from the valence orbitals, except for the single valence orbital j replaced with conduction orbital b. For the following we assume that state B was obtained as a LR-TDDFT excitation, and A is a (constrained) ground state as before. For the overlap we calculate

$$\begin{aligned}
\langle \Phi_B | \Phi_A \rangle = & \int d\mathbf{r}_1 \ldots d\mathbf{r}_N \sum_{jb} R_{jb} \psi_1^B(\mathbf{r}_1) \ldots \psi_b^B(\mathbf{r}_j) \ldots \psi_N^B(\mathbf{r}_N) \times \begin{vmatrix} \psi_1^A(\mathbf{r}_1) & \cdots & \psi_1^A(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^A(\mathbf{r}_1) & \cdots & \psi_N^A(\mathbf{r}_N) \end{vmatrix} \\
= & \sum_{jb} R_{jb} \begin{vmatrix} \langle \psi_1^A | \psi_1^B \rangle & \cdots & \langle \psi_1^A | \psi_b^B \rangle & \cdots & \langle \psi_1^A | \psi_N^B \rangle \\ \vdots & & \vdots & & \vdots \\ \langle \psi_N^A | \psi_1^B \rangle & \cdots & \underset{\uparrow j}{\langle \psi_N^A | \psi_b^B \rangle} & \cdots & \langle \psi_N^A | \psi_N^B \rangle \end{vmatrix} \\
= & \sum_{ijb} R_{jb} \langle \psi_i^A | \psi_b^B \rangle \cdot C_{i,j} \\
= & \det(S_{AB}) \times \mathrm{tr}\left(T_{AB}^{vc} \cdot R^\top \cdot S_{AB}^{-1} \right).
\end{aligned}$$

The matrix T_{AB}^{vc} represents the overlap of the valence orbitals of state A with the conduction orbitals of state B. The derivation of the overlap with a potential operator is a bit more involved but proceeds along similar lines:

$$\begin{aligned}
\langle \Phi_B | \hat{V} | \Phi_A \rangle = & \int d\mathbf{r}_1 \ldots d\mathbf{r}_N \sum_{jb} R_{jb} \psi_1^B(\mathbf{r}_1) \ldots \psi_b^B(\mathbf{r}_j) \ldots \psi_N^B(\mathbf{r}_N) \left[\sum_i V(\mathbf{r}_i) \right] \begin{vmatrix} \psi_1^A(\mathbf{r}_1) & \cdots & \psi_1^A(\mathbf{r}_N) \\ \vdots & & \vdots \\ \psi_N^A(\mathbf{r}_1) & \cdots & \psi_N^A(\mathbf{r}_N) \end{vmatrix} \\
= & \sum_{i \neq j} \sum_b R_{jb} \begin{vmatrix} \langle \psi_1^A | \psi_1^B \rangle & \cdots & \langle \psi_1^A | \hat{V} | \psi_i^B \rangle & \cdots & \langle \psi_1^A | \psi_b^B \rangle & \cdots & \langle \psi_1^A | \psi_N^B \rangle \\ \vdots & & \vdots & & \vdots & & \vdots \\ \langle \psi_N^A | \psi_1^B \rangle & \cdots & \langle \psi_N^A | \hat{V} | \psi_i^B \rangle & \cdots & \langle \psi_N^A | \psi_b^B \rangle & \cdots & \langle \psi_N^A | \psi_N^B \rangle \end{vmatrix} \\
& + \sum_{jb} R_{jb} \begin{vmatrix} \langle \psi_1^A | \psi_1^B \rangle & \cdots & \langle \psi_1^A | \hat{V} | \psi_b^B \rangle & \cdots & \langle \psi_1^A | \psi_N^B \rangle \\ \vdots & & \vdots & & \vdots \\ \langle \psi_N^A | \psi_1^B \rangle & \cdots & \langle \psi_N^A | \hat{V} | \psi_b^B \rangle & \cdots & \langle \psi_N^A | \psi_N^B \rangle \end{vmatrix} \\
= & \sum_{ijb} R_{jb} \sum_{kl} \langle \psi_k^A | \hat{V} | \psi_i^B \rangle \langle \psi_l^A | \psi_b^B \rangle \cdot \epsilon_{kl} \epsilon_{ij} C_{kl,ij} \quad + \quad \sum_{ijb} R_{jb} \langle \psi_i^A | \hat{V} | \psi_b^B \rangle \cdot C_{i,j}.
\end{aligned}$$

In the first determinant two columns are distinct from the overlap S_{AB}, we therefore expand along both. This leads to an expression including the second cofactors $C_{kl,ij}$. It follows from Jacobi's theorem that

$$\epsilon_{kl} \epsilon_{ij} C_{kl,ij} = \det(S_{AB}) \times \left[\left(S_{AB}^{-1}\right)_{ik} \left(S_{AB}^{-1}\right)_{jl} - \left(S_{AB}^{-1}\right)_{il} \left(S_{AB}^{-1}\right)_{jk} \right].$$

Putting everything together we finally obtain

$$\begin{aligned}
\langle \Phi_B | \hat{V} | \Phi_A \rangle = & \det(S_{AB}) \times \left[\mathrm{tr}\left(V_{AB} \cdot S_{AB}^{-1}\right) \mathrm{tr}\left(T_{AB}^{vc} \cdot R^\top \cdot S_{AB}^{-1}\right) - \mathrm{tr}\left(V_{AB} \cdot S_{AB}^{-1} \cdot T_{AB}^{vc} \cdot R^\top \cdot S_{AB}^{-1}\right) \right] \\
& + \det(S_{AB}) \times \mathrm{tr}\left(W_{AB}^{vc} \cdot R^\top \cdot S_{AB}^{-1}\right),
\end{aligned}$$

where W_{AB}^{vc} refers to matrix elements of $\hat{V}$ between valence orbitals of state A and conduction orbitals of state B.

We note that in general the Hamiltonian matrix obtained in the way shown is not symmetric due to the approximations inherent in the DFT formalism. Hence, the Hamiltonian must be symmetrised before eigenstates can be obtained.

7.6.3 Implementation

The CI functionality is implemented in `couplings_mod`. For each reference state the density kernel and NGWFs are read from the corresponding files. Additionally, the cDFT-potentials are read from file for a cDFT reference state. For an excited state from LR-TDDFT, conduction kernel, conduction NGWFs and the response kernel are read. A set of orthonormal orbitals representing the valence space is obtained from the NGWF representation by solving the eigenvalue problem

$$\sum_{\beta\gamma} K^{\alpha\beta} S_{\beta\gamma} x^{\gamma} = n \cdot x^{\alpha},$$

and restricting to the occupied subspace. Here $K^{\alpha\beta}$ is the valence density kernel and $S_{\beta\gamma}$ the overlap matrix of valence NGWFs. Orthonormal conduction orbitals are obtained in an equivalent manner. The actual CI calculations then proceeds in this basis as outlined in the theory section. It should be noted that the orbitals obtained in this way generally do not correspond to the KS orbitals (they do not result from a diagonalisation of the Hamiltonian). However, both are related through an orthogonal transformation. Hence, the determinants are identical, therefore all results are unaffected by this choice of basis.

The transformation to a orthonormal basis comes with an inherent N^3 scaling of the method. The computational effort is expected to be comparable with a properties calculation (which involves a diagonalisation of the Hamiltonian).

7.6.4 Performing a calculation

This section explains how to set up a CI calculation, and points out a couple of important things to look out for.

- First perform calculations for desired reference states. For each state the density kernel and NGWFs have to be written to files (`.dkn` and `.tightbox_ngwfs`). For cDFT reference states the potentials and projectors are required (`.cdft` and `.tightbox_hub_projs`). For LR-TDDFT states conduction kernel and NGWFs are required (`.dkn_cond` and `.tightbox_ngwfs_cond`), as well as the response density matrix.
- It is currently required that all reference states use the same unit cell, grid, geometry and identical atomic species with the same number of NGWFs and the same pseudopotentials. **NOTE:** The current implementation is not compatible with PAW!
- Now set up a new input file for the CI calculation. It is recommended to copy the input file of one of the cDFT reference calculations. This ensures that the setup of the CI run is consistent with the reference calculations (in particular with the correct projectors). If a LR-TDDFT reference state is used, also copy the conduction species block into the file. Set `TASK` to `COUPLINGS`.
- Add the block `couplings_states`. This tells the CI calculation which reference states to use. Here is an example:Each line corresponds to one reference state. The first column is short identifier for the state (currently unused). The second column indicates whether it is a cDFT or LR-TDDFT state, the third column contains the root name (i.e. name of original input file without extension). For a LR-TDDFT state, the fourth column determines the index of the excitation to be used (set to 0 for cDFT states). Finally, the fifth column is the energy in Hartree. It should be made sure that all energies are referenced to the same zero point.
- The output is written to matrix files (using `dense_write`). The names of these files consist of the root name of the CI run with extensions `_ci_ham` and `_ci_ham_sym` for the CI Hamiltonian and its symmetrised version, respectively. Eigenvalues and -states of the (symmetrised) CI Hamiltonian are written to files with extensions `_ci_eigvals` and `_ci_eigstates` (column-wise). If `output_detail : VERBOSE` is chosen, the results are also written to standard output.

7.6.5 References

- Extracting electron transfer coupling elements from constrained density functional theory, Q. Wu and T. Van Voorhis, J. Chem. Phys. **125**, 164105 (2006)
- Exciton/Charge-Transfer Electronic Couplings in Organic Semiconductors, S. Difley and T. Van Voorhis, JCTC **7**, 594 (2011)
- Determinants and matrices, A.C. Aitken, University mathematical texts vol. 20, Oliver and Boyd (1958)

CHAPTER

EIGHT

POPULATION ANALYSIS

8.1 ONETEP to GENNBO `FILE.47` Input Parameters

Author
Louis Lee, University of Cambridge (`lpl24@cam.ac.uk`)

8.1.1 Standalone nbo 5 Program gennbo

The standalone version of the NBO program (GENNBO) [Glendening] accepts parameters from an input ASCII free-format `FILE.47` containing atomic coordinates and matrix information, as printed by ONETEP if the Natural Population Analysis [Reed1985] subroutine is called during a `PROPERTIES` calculation, by specifying the keyword `write_nbo: T`. The NBO formalism allows one transform a converged 1-particle wavefunction in an atom-centred bases into a set of highly-local 'Natural Bond Orbitals', which are one and two (or three)-centred 'lone' and 'bond' pairs recognizable as chemical bonds from a classical Lewis structure standpoint [Reed1988]. Details of the NBO formalism are discussed elsewhere [Reed1985], [Reed1988], [MacKerell1998].

Compiling GENNBO

Compilation of the standalone GENNBO does not involve ONETEP in any way. As of writing, the latest nbo release is version 5.9 [Glendening]. Compilation instructions are listed here for convenience, based on some trial-and-error when the arcane `g77` compiler listed in the nbo manual is unavailable.

To compile GENNBO, first, compile the activator, `enable.f`:

```
gfortran -o enable enable.f
```

then run the `enable` program. Complete the selections to generate the standalone GENNBO source `gennbo.f`.

By default, GENNBO limits the number of atoms and basis in the `FILE.47` input to 200 and 2000 respectively. This can be increased by replacing all instances of `MAXATM = 200` and `MAXBAS = 2000` to a user-specified value, up to a limit of 999 and 9999 respectively (higher values are possible, albeit accompanied by illegible output due to format overflow. In principle one could modify the code even further to remedy this issue.).

The following command should compile GENNBO correctly on x64 architectures, when no modification is made to the `gennbo.f` source:

```
gfortran -fdefault-integer-8 -fno-sign-zero -m64 -o gennbo gennbo.f
ifort -i8 -m64 -f77rtl -o gennbo gennbo.f
```

For the 32-bit version, integer length should be set to 4 bytes instead (e.g. `-i4` in `ifort`). If `MAXATM` and `MAXBAS` have been increased then the memory model should also be set to allow data > 2 GB, by adding a `-mcmodel=medium` flag. For `ifort`, an additional `-shared-intel` flag is most likely necessary.

Then, to run:

```
gennbo < FILE.47 > output.out
```

8.1.2 ONETEP NPA Generation Routine

The Natural Population Analysis [Reed1985] method of computing atomic charges is implemented in ONETEP. The routine transforms the set of non-orthogonal, optimized NGWFs into a set of orthogonal atom-centred 'Natural Atomic Orbitals' (NAOs) via an 'occupancy-weighted symmetric orthogonalization' procedure, which serves to maximise the resemblance of the final orthogonal orbitals to their initial non-orthogonal parents (a la Löwdin orthogonalization), weighted according to the parent orbital occupancies. Therefore, vacant, highly-diffuse orbitals are free to distort to achieve orthogonality with their more highly-preserved occupied counterpart. This ensures that the final NAO population (the 'Natural Population') remains stable with respect to basis set size.

Once in the NAO basis, further transformations such as pair-block density matrix diagonalization produce the final set of NBOs – these procedures are performed by nbo 5 from the `FILE.47` output of ONETEP, which contains relevant matrices in the NAO basis. The NAO routine is performed internally in ONETEP as nbo 5 requires pseudo-atomic orbitals (such as Gaussian-type orbitals) with free-atom symmetries and orthogonality within each atom, a property not rigorously satisfied by the optimized NGWFs.

The NPA module in ONETEP performs at its best for large systems when compiles with the ScaLapack linear algebra package, as it takes advantage of the distributed memory storage of dense global matrices, such as the inverse square root of the overlap matrix that needs to be computed for the 'occupancy-weighted symmetric orthogonalization' step. This has the unfortunate side effect of rendering the NAO transformation a cubic-scaling method. However, this step occurs only once during the routine, and should be comparable to the time needed to generate canonical molecular orbitals.

8.1.3 List of Available Parameters

Table 8.1: NAO Generation (Default)

Keyword	Type	Default	Level	Description
write_nbo	L	F	B	Enables Natural Population Analysis (NPA) and writing of GENNBO input file <seedname>_nao_nbo.47
nbo_init_lclowdin	L	T	E	Performs atom-local Löwdin orthogonalisation on NGWFs as the first step before constructing NAOs
nbo_write_lclowdin	L	F	E	Writes full matrices (all atoms) in the atom-local Löwdin-orthogonalized basis to FILE.47 (For reference/testing/comparison purposes). Output will be <seedname>_lclowdin_nbo.47
nbo_write_npacomp	L	F	B	Writes NAO charges for all orbitals to standard output
nbo_write_dipole	L	F	B	Computes and writes dipole matix to FILE.47
nbo_scale_dm	L	T	E	Scales partial density matrix output to <seedname>_nao_nbo.47 in order to achieve charge integrality
nbo_scale_spin	L	T	E	Scales α and β spins independently to integral chrage when partial matrices are printed and nbo_scale_dm = T. Inevitably means spin density values from GENNBO are invalid and one should calculate them manually from the α and β NPA populations.
nbo_write_species	B	N/A	B	Block of lists of species to be included in the partial matrix output of <seedname>_nao_nbo.47. If not present all atoms will be included. E.g. specified will default to AUTO. E.g.: %block nbo_write_species C1 H1 %endblock nbo_write_species

Table 8.2: NAO Generation (Default) continued

nbo_species_ngwflabel	B	AUTO	I	Optional user-defined (false) *lm*-label for NGWFs according to GENNBO convention. Species not specified will default to AUTO. E.g.: %block nbo_species_ngwflabel C1 "1N 151N 152N 153N" H1 AUTO %endblock nbo_species_ngwflabel -N suffix denotes NMB orbital. If 'SOLVE' orbitals are used, this block should be present as 'AUTO' initialisation assumes orbitals were also initialised as 'AUTO'.
nbo_aopnao_scheme	T	ORIGINAL	E	The AO to PNAO scheme to use. Affects the '*lm*-averaging' and diagonalisation steps in the initial AO to PNAO, NRB *lm*-averaging, and rediagonaliation transformations (the '**N**' transformations in [Reed1985]). For testing purposes only - so far none of the other schemes apart from ORIGINAL works. Possbile values are: ORIGINAL - default, as in [Reed1985] with *lm*-averaging DIAGONALIZATION - Diagonalises entire atom-centred sub-block w/o *lm*-averaging or splitting between different angular channels. NONE - Skips all '**N**' transformations.
nbo_pnao_analysis	L	F	E	Perform s/p/d/f analysis on the PNAOs (analogous to ngwf_analysis)

Table 8.3: Orbital Plotting

Keyword	Type	Default	Level	Description
plot_nbo	L	F	B	Instructs ONETEP to read the relevant orbital transformation output from GENNBO, determined by the flag nbo_plot_orbtype and plots the orbitals specified in %block nbo_list_plotnbo. write_nbo and plot_nbo are mutually exclusive. Scalar field plotting must be enabled (e.g. cube_format = T).
nbo_plot_orbtype	T	N/A	B	The type of GENNBO-generated orbitals to read and plot. Possible values and their associated GENNBO transformation files must be present, as follows: NAO - <seedname>_nao.33 NHO - <seedname>_nao.35 NBO - <seedname>_nao.37 NLMO - <seedname>_nao.39 NLMO is only defined for the full system i.e. partitioned FILE.47 will give meaningless NLMOs. Except for NLMO, adding a 'P' prefix e.g. 'PNAO' to the value of nbo_plot_orbtype causes the non-orthogonalised PNAOs to be used in plotting instead of NAOs. PNAOs are of the normal type, i.e. when RPNAO = F in GENNBO (default).
nbo_list_plotnbo	B	N/A	B	The list of nbo_plot_orbtype orbitals to be plotted, identified by their indices as in the GENNBO output. Specify each index on a new line.

Table 8.4: Output files

<seedname>_nao_nbo.47	Always written. Contains partial matrices according to %block nbo_write_species.
<seedname>_lclowdin_nbo.47	Written if nbo_write_lclowdin = T. Always contains all atoms in the atom-local Löwdin-orthogonalized basis. If nbo_init_lclowdin = T and all atoms are included <seedname>_lclowdin_nbo.47 = <seedname>_nao_nbo.47 except for ordering of atomic centers (will be fixed in newer releases).
<seedname>_nao_atomindex.dat	Contains mapping of atomic indices of the potentially subset of the full system in <seedname>_nao_nbo.47 to the real atomic index of the full system (since labels have to be consecutive). Real atomic index refers to original input order in ONETEP.
<seedname>_inittr_nao_nbo.dat	Raw NGWF to NAO transformation read for plotting (i.e. when plot_nbo = T)
<seedname>_inittr_pnao_nbo.dat	Raw NGWF to PNAO transformation read for plotting (i.e. when plot_nbo = T). PNAOs are of the 'normal' type, i.e when RPNAO = F in GENNBO.
<seedname>_nbo_DEBUG.dat	Contains various debugging info. Only written if compiled in debug mode.

8.1.4 Notes

Orbital labelling with pseudoatomic solver

- GENNBO labels in %block nbo_species_ngwflabel should always be explicitly given when SOLVE is used to initialise the NGWFs. The label string is however limited to 80 characters in ONETEP, which should be fine up to $1s2sp3spd4sp$. This will be fixed later unless it is urgently required.
- Make sure the orbitals selected for plotting are valid. The NPA routine assumes that the appropriate transformation file from GENNBO in the same directory is correct, and only complains if it encounters an EOF, but not if the wrong transformation file is given (e.g. from a different system with a larger basis).
- Do not rename the GENNBO-generated transformation files. ONETEP expects them to have the name <seedname>_nao.xx.

Orbital plotting

- In order to plot the various orbitals, first run the output FILE.47 through GENNBO to obtain the relevant orbital vectors. Refer to the nbo 5 manual for details on how to print these (e.g. to print NBOs in the input FILE.47 basis, set AONBO=W in the $NBO block).
- For some reason, the PLOT keyword itself in GENNBO doesn't work. This might have something to do with the 'ORTHO bug'.

'ORTHO bug'

The nbo 5 program up till circa April/May 2011 had a bug whereby specifying the ORTHO flag causes the program to crash. The nbo 5 developers seem to have fixed most of this and given me the an updated version, but residual bug could remain (have they made the fix a general release yet?). This is of course fixable by running the <seedname>_lclowdin_nbo.47 file through GENNBO instead, albeit this would mean one can't do DM partitioning.

8.1.5 Example Usage

Obtaining 2^{nd}-order Perturbation Estimates of the $n \rightarrow \sigma^*$ Secondary Hyperconjugation in Water Dimer (Hydrogen Bond)

The hydrogen bond stabilization in water dimer can be attributed to the non-classical 'charge transfer' interaction between two water molecules due to delocalization of the electronic charge from the oxygen lone pair n of the donor monomer to the σ^* O–H antibond of the acceptor [Reed1988]. The expansion of the variational space to included non-Lewis, formally vacant antibond NBOs leads to an energetic lowering compared to the ideal Lewis configuration (all Lewis NBO occupancy = 2 e), which can be estimated via 2^{nd}-order perturbation theory as the 'charge transfer' energetic component of the dimer interaction.

From a converged SCF calculation in ONETEP using the reference coordinates below (given in Angstroms):

```
%block  positions_abs
ang
 O  10.6080354926368 12.5000150953008 12.5705695516353
 H  10.4341376488693 12.5000119731552 13.5119746410112
 H1 11.5729802892758 12.5000098564464 12.5000098564464
 H  13.9638274701977 13.2691512541917 12.2000071259853
 O1 13.4760438789916 12.5000098564464 12.5000098564464
 H  13.9638258826660 11.7308667653340 12.2000083960106
%endblock  positions_abs
```

with the pseudoatomic solver employing a minimal NGWF basis (1 NGWF on H, 4 on O) with a 10.0 a_0 NGWF radius cutoff, PBE exchange-correlation functional, norm-conserving pseudopotential with pseudized $1s$ core for O, and a 1200 eV psinc cutoff in a 25.0 a_0 cubic simulation cell, one should run a PROPERTIES calculation with the additional keywords as such:

```
write_nbo: T
%block species_ngwflabel
 H  "1N"
 O  "1N 152N 153N 151N"
 H1 "1N"
 O1 "1N 152N 153N 151N"
%endblock species_ngwflabel
```

where the species_ngwflabel block tells the NPA routine in ONETEP how to label each NGWF. The order of m for each l in the $Y(l,m)$ isn't straightforward, and follows the pattern of e.g. "152 153 151" i.e. $m = \{-1, 0, 1\}$ for $l = 1$, and "251 253 255 252 254" for $l = 2$. I've yet to look at how others are arranged, though this is not very important unless one is interested in 'NHO Directionality and Bond Bending' analysis, as in the NBO scheme, all m of the same l are treated equally. The order of each $Y(l,m)$ should follow that of the pseudoatomic solver, which does them in principal quantum number (n) increments (with multiple-ζ basis, the split-valence set of $Y(l,m)$ probably comes first i.e. $Y^{\zeta 1}(l,m)$ then $Y^{\zeta 2}(l,m)$ before the next n. The "N" suffix denotes valence orbital in the ground state, which in the case of H, "1N" is the $1s$ orbital. Make sure the correct orbitals are marked as valence as they would appear in the ground state (even if the pseudoatomic solver basis was initialized in an excited configuration). In this example, the pseudoatomic solver block would have explicitly been:

```
%block species_atomic_set
 H  "SOLVE conf=1s1"
 O  "SOLVE conf=1sX 2s2 2p4"
%endblock species_atomic_set
```

ONETEP should run and produce an NPA output listing the NPA charges on each atom, and print a <seedname>_nao_nbo.47 file. This .47 file serves as the input for GENNBO.

If we wanted to generate NBOs and visualize them, insert the keyword AONBO=W in the $NBO block of the .47 file before running it through GENNBO. GENNBO will output a report containing NBO information, including the 2^{nd}-order perturbation estimates, and a .37 file containing the NBO vectors in terms of the .37 input basis (don't change any of the .47, .37 etc. filenames).

First, we can see that the 2^{nd}-order perturbation report shows one prominenet interaction, namely one between the occupied lone pair of oxygen from one H_2O unit (LP (2) O 5) to the O–H antibond of the other (BD*(2) O 1- H 3) with an estimate of 15.32 kcal/mol, corresponding to the hydrogen bond in water dimer:

```
SECOND ORDER PERTURBATION THEORY ANALYSIS OF FOCK MATRIX IN NBO BASIS

    Threshold for printing:   0.50 kcal/mol
   (Intermolecular threshold: 0.05 kcal/mol)
                                                          E(2)  E(j)-E(i) F(i,j)
     Donor NBO (i)              Acceptor NBO (j)       kcal/mol   a.u.    a.u.
================================================================================

within unit  1
      None above threshold

from unit  1 to unit  2
  2. BD ( 1) O 1- H 3       11. BD*( 1) H 4- O 5         0.08    0.67    0.007
  2. BD ( 1) O 1- H 3       12. BD*( 1) O 5- H 6         0.08    0.67    0.007

from unit  2 to unit  1
  3. BD ( 1) H 4- O 5       10. BD*( 1) O 1- H 3         0.10    0.83    0.008
  4. BD ( 1) O 5- H 6       10. BD*( 1) O 1- H 3         0.10    0.83    0.008
  7. LP ( 1) O 5            10. BD*( 1) O 1- H 3         0.18    0.49    0.008
  8. LP ( 2) O 5            10. BD*( 1) O 1- H 3        15.32    0.60    0.085

within unit  2
      None above threshold
```

Noting down the orbital numbers, we can then proceed to plot them by running another properties calculation in ONETEP with the following block:

```
write_nbo   : F
plot_nbo    : T
cube_format : T
nbo_plot_orbtype : NBO
%block nbo_list_plotnbo
  8
 10
%endblock nbo_list_plotnbo
```

where write_nbo needs to be set to F. ONETEP will then read the <seedname>_inittr_nao_nbo.dat file printed during the first run and the .37 file to plot the orbitals specified in the nbo_list_plotnbo block into Gaussian cube

files.

An example result is displayed in a figure available in the published paper.

Notes on Selectively Passing sub-region sub-matrices into GENNBO

To circumvent the limitations on system size in GENNBO, and for convenience, we could output only matrix elements corresponding to atoms within a selected sub-region of a large system. To do so, during an NPA analysis run (not plotting) within a properties run in ONETEP, the following should be specified:

```
%block nbo_write_species
 O1
 H1
 C1
 ...
%endblock nbo_write_species
```

ONETEP would then print only matrix elements belongning to species specified by the labels in the `%block nbo_write_species` block to `<seedname>_nao_nbo.47`. Due to GENNBO insisting on integral charges, the density matrix in the `.47` file is re-scaled downwards to the nearest lowest integral number, to avoid the possibility of orbitals having occupancies > 2 *e*, which also annoys GENNBO. To minimize the impact of this technical re-scaling to the NBO results, a sufficiently-sized partition should be chosen in `%block nbo_write_species` so that $1/N_e << 1$, where N_e is the number of electrons in the partition.

The final results fron NBO analysis that depend on the density matrix will then need to be de-scaled to arrive at the correct value (e.g. NPA charges, NBO occupancies, 2^{nd}-order perturbation estimates, while orbital energies don't require de-scaling).

Note that the region included in `%block nbo_write_species` should have buffer atoms, which minimally should include the next-nearest neighbour atom bonded to the last atom in the selection – that way, the severing of a bond would only affect NBOs centred on the buffer atom, and not anywhere else.

As a final note, there is a possibility that during an NBO search, slightly different NBO pictures are obtained when passing only part of the matrix as compared to analyzing the full system – this can be caused by the fact that during an NBO search, the nbo 5 program iterates through different occupancy thresholds (n_{min}) for deciding upon whether an orbital is a lone pair/NBO. If one is pedantic about this, then n_{min} can be fixed by specifying the `THRESH` $=n_{min}$ keyword manually in the `$NBO` block in the `.47` file, where n_{min} is defined by the user.

[Glendening] E. D. Glendening, J. K. Badenhoop, A. E. Reed, J. E. Carpenter, J. A. Bohmann, C. M. Morales, F. Weinhold; NBO 5.9 (http://www.chem.wisc.edu/~nbo5) & the NBO 5.9 Manual, Theoretical Chemistry Institute, University of Wisconsin, Madison, WI.

[Reed1985] A. E. Reed, R. B. Weinstock, F. Weinhold *J. Chem. Phys.* **1985,** *83,* 735-746.

[Reed1988] A. E. Reed, L. A. Curtiss, F. Weinhold *Chem. Rev.* **1988,** *88,* 899-926.

[MacKerell1998] A. D. MacKerell, Jr., B. Brooks, C. L. Brooks III, L. Nilsson, B. Roux, Y. Won, M. Karplus, in Encyclopedia of Computational Chemistry; R. Schleyer et al. Eds.; John Wiley & Sons, Chichester, **1998**; Vol. 3, Chapter 'Natural Bond Orbital Methods', pp 1792-1811.

8.2 Density derived electrostatic and chemical (DDEC) electron density partitioning

Author
Louis P. Lee, University of Cambridge

Author
Daniel J. Cole, University of Cambridge

Atoms-in-molecule electron density partitioning is a useful post-processing analysis tool for computing atomic charges (as well as higher order atomic multipoles) from the total electron density. ONETEP uses the DDEC3 method [1,3] for this purpose, as the computed charges are both chemically meaningful and reproduce the electrostatic potential of the underlying QM calculation. Options are also available for computing *Hirshfeld* and *iterated stockholder atoms* (ISA) charges [3,4].

A DDEC3 calculation to partition the electron density and output atomic charges, multipoles and volumes is performed by specifying:

```
ddec_calculate : T
ddec_multipole : T
ddec_moment : 3
```

along with the ddec_rcomp block for your system (see below). Iterated stockholder atoms (ISA) partitioning may be performed instead by additionally specifying:

```
ddec_IH_fraction : 0.00
```

Classical Hirshfeld partitioning may be performed instead by additionally specifying:

```
ddec_classical_hirshfeld : T
ddec_IH_fraction : 1.00
ddec_maxit : 1
```

The reference ion densities for use with DDEC3 are read in from an external library kindly provided by Thomas A. Manz and Nidia Gabaldon Limas (please cite Refs. [1,2]), and are available for download from the ONETEP website:
http://www.onetep.org/pmwiki/uploads/Main/Utilities/ddec_atomic_densities.tar.gz
The paths to the reference densities are specified in the block `ddec_rcomp`. Specify one core and one total density file for each species in your system (except for hydrogen and helium which do not require a core density file). The example below is for methanol:

```
%block ddec_rcomp
  H ALL “H_c2.refconf”
  O ALL “O_c2.refconf”
  O CORE “O_c2.coreconf”
  C ALL “C_c2.refconf”
  C CORE “C_c2.coreconf”
%endblock ddec_rcomp
```

8.2.1 References

For the development of the DDEC method:

[1] T.A. Manz and D.S. Sholl, "Improved Atoms-in-Molecule Charge Partitioning Functional for Simultaneously Reproducing the Electrostatic Potential and Chemical States in Periodic and Non-Periodic Materials," J. Chem. Theory Comput. 8 (2012) 2844-2867.

[2] T. A. Manz and D. S. Sholl, "Chemically Meaningful Atomic Charges that Reproduce the Electrostatic Potential in Periodic and Nonperiodic Materials", J. Chem. Theory Comput. 6 (2010) 2455-2468.

And its implementation in ONETEP:

[3] L. P. Lee, N. Gabaldon Limas, D. J. Cole, M. C. Payne, C.-K. Skylaris, T. A. Manz, "Expanding the Scope of Density Derived Electrostatic and Chemical Charge Partitioning to Thousands of Atoms", J. Chem. Theory Comput., 10 (2014) 5377.

[4] L. P. Lee, D. J. Cole, C.-K. Skylaris, W. L. Jorgensen, M. C. Payne, "Polarized Protein-Specific Charges from Atoms-in-Molecule Electron Density Partitioning", J. Chem. Theory Comput., 9 (2013), 2981.

CHAPTER

NINE

GPU ACCELERATED CODE

9.1 GPU Accelerated Implementation

Author
Karl A. Wilkinson, University of Cape Town[1]

9.1.1 Introduction

An OpenACC implementation of ONETEP is available to allow execution on machines containing graphic processing units based accelerators (GPUs). GPUs are highly parallel and are well suited to algorithms such as the fast fourier transforms (FFTs) within ONETEP during the calculation of properties such as the local potential integrals and the charge density.

However, the connection of the accelerators to the host machine through the peripheral component interconnect express (PCIe) bus introduces a bottleneck when large amounts of data are transferred. Currently, this is an issue when moving the fine grid FFT boxes from the accelerator to the host machine but future generations of hardware, and developments within ONETEP are expected to reduce this issue and improve performance significantly.

This work has been published in the Journal of Computational Chemistry. More detailed information is available in this publication: http://onlinelibrary.wiley.com/doi/10.1002/jcc.23410/abstract It should be noted that this feature of the ONETEP package is under development and that significant performance improvements have achieved since the publication of this article.

9.1.2 Compilation

Compilation of the OpenACC implementation of ONETEP is only currently supported by the compilers from the Portland Group International (PGI). Relatively few changes are required in order to perform the compilation: The flag:

```
-DGPU_PGI
```

should be used and additional variable describing the flags and libraries need to be defined:

```
ACCFLAGS = -ta=nvidia -Mcuda=6.5
ACCLIBS = -lcufft -lcudart
```

Here, we are utilising the CUDA 6.5 runtime libraries as they are the most up to date version available on the TITAN supercomputer at the Oak Ridge National laboratories, your local machine may have a more up to data version available.

Further examples of complete config files for the desktops at the University of Southampton and the Wilkes cluster at the University of Cambridge follow:

[1] karl.wilkinson@uct.ac.za

```
################  Southampton desktop ################
F90 = pgf90
MPIROOT=/local/scratch/kaw2e11/software/openmpi_1.6.4/pgi/
FFTWROOT=/local/scratch/kaw2e11/software/fftw/pgi/
FFLAGS = -DGPU_PGI -DFFTW3 -DMPI -I$(MPIROOT)include -I$(FFTWROOT)include -I
↪$(MPIROOT)lib/
OPTFLAGS = -O3 -fast
DEBUGFLAGS = -g -C
MPILIBS= -L$(MPIROOT)lib/ -lmpi_f90 -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,
--export-dynamic -lnsl -lutil -ldl
ACCFLAGS = -ta=nvidia -Mcuda=6.5
ACCLIBS = -L/usr/lib64/nvidia -L$(CUDAROOT)/lib64/ -lcufft -lcudart
LIBS = $(MPILIBS) -llapack -lblas -L$(FFTWROOT)lib/ -lfftw3_omp -lfftw3 -lm
```

```
################ WILKES ################
FC := mpif90
F90 := $(FC)
FFLAGS = -DGPU_PGI -DFFTW3_NO_OMP -DMPI -DNOMPIIO -Mdalign
MKLPATH=${MKLROOT}/lib/intel64
LIBS=  -L$(MKLROOT)/lib/intel64 -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lpthread -
↪lm
OPTFLAGS = -O3 -m64
WARNINGFLAGS = -Wall -Wextra
DEBUGFLAGS =
COMPILER = PORTLAND-pgf90-on-LINUX
ACCFLAGS = -acc -ta=nvidia:cc35 -Mcuda=6.5
ACCLIBS = -lcufft -lcudart
```

Unfortunately, attention should be paid to to the version of the compilers and libraries used as, due to the speed at which the OpenACC approach is evolving, it is a common for functionality to break. As such, this document will be regularly updated with details of combinations of compiler and library versions that are known to be stable.

Known Working Configurations

The following combinations of machine, PGI compiler and CUDA libraries have been tested successfully.

Machine	Compiler	CUDA library
Wilkes	PGI 15.3	6.5
Wilkes	PGI 15.9	7.5
Titan	Cray	6.5

9.1.3 Execution

Use of the OpenACC implementation of ONETEP does not require any changes to the ONETEP input files. However, job submission does change significantly in some platforms.

CUDA Multi Process Service

The CUDA Multi Process Service (MPS) daemon controls the way MPI processes see GPUs and allows multiple MPI processes to use a single GPU wherein the hyperqueue scheduler is used to utilise the hardware much more efficiently than when a single process is used per GPU. As, in the case of a single MPI process does not provide sufficient computation to fully utilize a GPU, it is critical to use this technology to achieve optimal performance.

However, attention must be paid to ensure that GPU memory is not exhausted. Currently, the usage is reported but these safety checks need to be extended to allow a graceful exit should the total memory be exhausted.

Below are examples for the usage of MPS during job submission on Wilkes and TITAN:

Wilkes

On Wilkes, job submission is performed using: sbatch slurm_submit.tesla

where: slurm_submit.tesla is:

```
#!/bin/bash
#SBATCH -J MPS_test
#SBATCH -A SKYLARIS-GPU
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH -p tesla

. /etc/profile.d/modules.sh
module purge
module load default-wilkes
module unload intel/impi intel/cce intel/fce cuda
module load pgi/14.7
module load mvapich2/2.0/pgi-14
ulimit -s unlimited

numnodes=$SLURM_JOB_NUM_NODES
numtasks=$SLURM_NTASKS
mpi_tasks_per_node=$(echo "$SLURM_TASKS_PER_NODE" | sed -e  's/^\([0-9][0-9]*\).*$/\1/')
JOBID=$SLURM_JOB_ID

cd $SLURM_SUBMIT_DIR

application="onetep.wilkes.gpu.cuda55"

echo "JobID: $JOBID"
echo "Time: `date`"
echo "Running on master node: `hostname`"
echo "Current directory: `pwd`"
```

(continues on next page)

(continued from previous page)

```
if [ "$SLURM_JOB_NODELIST" ]; then
       #! Create a machine file:
       export NODEFILE=`generate_pbs_nodefile`
       cat $NODEFILE | uniq > machine.file.$JOBID
       echo -e "\nNodes allocated:\n================"
       echo `cat machine.file.$JOBID | sed -e 's/\..*$//g'`
fi

echo -e "\nnumtasks=$numtasks, numnodes=$numnodes, \
mpi_tasks_per_node=$mpi_tasks_per_node (OMP_NUM_THREADS=$OMP_NUM_THREADS)\n"

# Start MPS deamons...
srun -N$SLURM_JOB_NUM_NODES -n$SLURM_JOB_NUM_NODES ./run_MPS.sh

echo -e "\nExecuting program:\n==================\n\n"

mpirun -np ${SLURM_NTASKS} -ppn ${mpi_tasks_per_node} --genvall \
-genv MV2_RAIL_SHARING_LARGE_MSG_THRESHOLD 1G -genv MV2_ENABLE_AFFINITY 1 \
-genv MV2_CPU_BINDING_LEVEL SOCKET -genv MV2_CPU_BINDING_POLICY SCATTER \
-genv MV2_SHOW_CPU_BINDING 1 ./run_app.sh ../${application} onetep.dat 2>&1 \
| tee onetep.out


echo -e "\n\n>>> Program terminated! <<<\n"
echo -e "Time: `date` \n\n"

# Kill MPS deamons
srun -N$SLURM_JOB_NUM_NODES -n$SLURM_JOB_NUM_NODES ./kill_MPS.sh
```

This file, and the following files, were obtained from the Wilkes systems administrators. It is advisable to contact system administrators if you have any questions regarding the submission process.

Here, the files: run_MPS.sh and kill_MPS.sh manage the initialisation and termination of the MPS deamon and the run_app.sh controls the allocation of MPI processes to the correct GPUs. For reference, the contents of those files are as follows, again, it is advisable to speak with your systems administrator about equivalent scripts for other machines (For example, run_app.sh assumes the use of MVAPICH2).

```
#####run_MPS.sh
#!/bin/bash

# Number of gpus with compute_capability 3.5  per server
NGPUS=2

# Start the MPS server for each GPU
for ((i=0; i< $NGPUS; i++))
do
 echo "[CUDA-PROXY] Setting MPS on `hostname` for GPU $i..."
 mkdir /tmp/mps_$i
 mkdir /tmp/mps_log_$i
 export CUDA_VISIBLE_DEVICES=$i
 export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_$i
 export CUDA_MPS_LOG_DIRECTORY=/tmp/mps_log_$i
```

(continues on next page)

(continued from previous page)

```
 nvidia-cuda-mps-control -d
done

exit 0
```

```
###/run_app.sh
#!/bin/bash

# Important note: it works properly when MV2_CPU_BINDING_LEVEL=SOCKET &&
# MV2_CPU_BINDING_POLICY=SCATTER

lrank=$MV2_COMM_WORLD_LOCAL_RANK
grank=$MV2_COMM_WORLD_RANK

case ${lrank} in
0|2|4|6|8|10)
  export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_0
  export MV2_NUM_HCAS=1
  export MV2_NUM_PORTS=1
  export MV2_IBA_HCA=mlx5_0
  echo "[CUDA-PROXY] I am globally rank $grank (locally $lrank ) on \
  `hostname` and I am using GPU 0"
  "$@"
  ;;
1|3|5|7|9|11)
  export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_1
  export MV2_NUM_HCAS=1
  export MV2_NUM_PORTS=1
  export MV2_IBA_HCA=mlx5_1
  echo "[CUDA-PROXY] I am globally rank $grank (locally $lrank ) on \
  `hostname` and I am using GPU 1"
  "$@"
  ;;
esac
```

```
##kill_MPS.sh
#!/bin/bash

echo "[CUDA-PROXY] Kill nvidia-cuda-mps-control on `hostname`..."
killall -9 nvidia-cuda-mps-control

# this waiting time is to let killall have effect...
sleep 3

echo "[CUDA-PROXY] Clean /tmp on `hostname`..."
rm -rf /tmp/mps_*
rm -rf /tmp/mps_log_*

exit 0
```

TITAN

Job submission on TITAN is somewhat more straightforward and the following script may be used directly. The important line is: `export CRAY_CUDA_PROXY=1` which enables the use of MPS.

```
#!/bin/bash
#PBS -A CODENAME
#PBS -N MgMOF74_111_SP
#PBS -j oe
#PBS -l walltime=1:30:00,nodes=XNUMNODES
#PBS -l gres=atlas1%atlas2

PROJECT=chm113

source $MODULESHOME/init/bash
module load cudatoolkit
#module swap PrgEnv-pgi/5.2.40 PrgEnv-intel/5.2.40

export CRAY_CUDA_PROXY=1

EXEDIR=/lustre/atlas/scratch/kaw2e11/chm113/binaries
#EXE=onetep.4313.titan.cpu.intel
EXE=onetep.4313.titan.gpu.pgi

########################################
SOURCEDIR=/ccs/home/kaw2e11/BENCHMARKS/PGI_GPU/benchmark-XTOTALMPI-\
XNUMNODES-XMPIPERNUMANODE
INPUT=G_222_80_D2.dat
INFO=PGI_GPU-XTOTALMPI-XNUMNODES-XMPIPERNUMANODE
########################################

BASENAME=`basename $INPUT`-$INFO
OUTPUT=$BASENAME.out

cd $MEMBERWORK/$PROJECT/
mkdir dir-$BASENAME
cd dir-$BASENAME

cp $SOURCEDIR/* $MEMBERWORK/$PROJECT/dir-$BASENAME

aprun -n XTOTALMPI -S XMPIPERNUMANODE -j 2 $EXEDIR/$EXE $INPUT &> $OUTPUT

cd ..
```

CHAPTER

TEN

QM/MM (TINKTEP) AND POLARISABLE EMBEDDING

10.1 TINKTEP and polarisable embedding: linear-scaling QM/polarisable-MM

Author
Jacek Dziedzic, University of Southampton

Date
May 2018

This manual pertains to ONETEP versions v4.5.18.8 and later, and TINKTEP versions v1.19 and later.

10.1.1 Executive summary

TINKTEP [Dziedzic2016], [Dziedzic2018] is an interface between ONETEP and TINKER (an implementation of the AMOEBA force-field). It enables QM/MM calculations, where ONETEP is used as the QM engine, and TINKER is used as the MM engine. The two main distinguishing features of TINKTEP are the linear scaling of the QM part (subject to usual ONETEP linear scaling caveats), and the fact that the MM side uses a polarisable force field (AMOEBA). There is full mutual self-consistency between the QM and MM subsystem – the QM subsystem polarises the MM subsystem and vice versa. Van der Waals interactions between QM and MM are also taken care of. TINKTEP is work in progress, both on the front of theory (improving the model) and implementation (adding desired features, simplifying use). It is currently not distributed to end users and not fully supported at the level we support mainstream ONETEP features.

10.1.2 Main limitations

Some of these might be deal-breakers for you, so make sure you read this carefully.

- You must obtain and install TINKER separately, and then patch it using the provided patches.
- The filesystem on the node from which you submit parallel jobs must be visible on the compute nodes. This can make set-up tricky on systems like ARCHER, where there is an intermediate layer of MOM nodes between the login node and the compute nodes. Currently it's easiest to run TINKTEP on single-node desktop machines, where you can leverage MPI and OMP parallelism without the hassle of a batch system.
- Only open boundary conditions (OBCs) are supported so far. The QM calculation is performed in OBCs, with the QM subsystem embedded in MM point charges, dipoles, quadrupoles and polarisable dipoles. The MM calculation is performed in OBCs. You will not be able to simulate e.g. a QM subsystem in a periodic box of water, for instance, but using a large sphere of MM water molecules is a decent workaround. We plan to support PBCs in the future.
- QM/MM forces, molecular dynamics, geometry optimisation and transition state search are all not available currently.

- Covalent bonds spanning the QM/MM interface are not supported.
- All QM species must have identical NGWF radii. Without this simplification the numerical methods used in SWRI (refer to "Spherical-wave resolution of identity, distributed multipole analysis (DMA) and Hartree-Fock exchange" manual for more details) would get ugly. Typically this is a non-issue, just increase the NGWF radii to the largest value. This might admittedly be an issue if you have a single species that requires a large NGWF radius. This is checked against and ONETEP will not let you do SWRI unless all NGWF radii are identical.
- TINKTEP is currently incompatible with complex NGWFs and ONETEP will refuse to use both.
- The TINKTEP1 model uses a classical model for QM/MM van der Waals interactions. The TINKTEP2 model uses a quantum-classical model for QM/MM van der Waals interactions, which requires suitable vdW parameters for all MM species that you use. We only provide these parameters for water, K^+ and Cl^-, so applications of TINKTEP2 are restricted to embedding a QM subsystem in water or KCl solutions, unless you wish to determine suitable MM vdW parameters yourself. These are **not** the familiar LJ parameters σ and ϵ .

10.1.3 Basics

Consult [Dziedzic2016] for a detailed exposure of the theory behind the model. Consult [Dziedzic2018] for a description of TINKTEP2 and how it differs from TINKTEP1. This manual takes a hands-on approach, but assumes you are familiar with the theory. This manual should be suitable for both TINKTEP1 and TINKTEP2, but the former will likely be discontinued in the future.

TINKTEP uses Distributed Multipole Analysis (DMA) [Stone1998], [Stone2005], [Vitale2015] to build an auxiliary representation (termed QM*) of the QM subsystem as a set of multipoles. Make sure you are famliar with DMA and SWRI ("Spherical-wave resolution of identity, distributed multipole analysis (DMA) and Hartree-Fock exchange") beforehand.

10.1.4 Setting up your computational environment

1. Obtain a copy of TINKER v7.1.3. This should be available at no cost from Jay Ponder's website. It is imperative that you use v7.1.3 or else the patches supplied with ONETEP will not work.
2. Patch your copy of TINKER using the set of patches located in `utils/tinktep/tinktep_patches`. Make sure they all applied correctly.
3. Compile and install your patched copy of TINKER.
4. Ensure that at least the following executables from the TINKER suite can be found in your `PATH`: `analyze`, `dynamic`, `poledit`.
5. Compile ONETEP v4.5.18.8 or later.
6. Ensure all scripts supplied in `utils/tinktep` can be found in your `PATH`.

10.1.5 Setting up a QM/MM calculation

Input `.xyz` file

First, prepare your complete QM+MM system in the form of a TINKER `.xyz` file. This format differs from the four-column (species, x, y, z) `.xyz` file you might be familiar with. Consult the TINKER manual for the details of the format. In further text "`.xyz` format" will implicitly mean the TINKER `.xyz` format. If your `.xyz` file came from a TINKER MD simulation, no adjustments are necessary. If you are creating the `.xyz` file on your own, make sure you get the atom types and classes right, and the connectivity too – as prescribed by your MM `.prm` file (e.g. `amoeba09.prm`). If you do not know what atom types to choose for what will be your QM atoms, do not worry particularly, just make sure they roughly match their classical equivalents in the `.prm` file in terms of chemical species, hybridisation (sp2,

sp3, etc.) and connectivity. What will only matter will be their polarisabilities (for correct Thole damping of induced QM/MM interactions) and classical vdW parameters (for QM/MM vdW energies (both the repulsive and dispersive term in TINKTEP1, dispersive term only in TINKTEP2).

An example `.xyz` file for a water dimer (one water molecule in QM, one water molecule in MM) could look like this:

```
6
1  O      0.583801    0.000000    0.759932    36     2     3
2  H      0.000000    0.000000    0.000000    37     1
3  H      0.000000    0.000000    1.530090    37     1
4  O     -0.687305    0.000000    2.795684    36     5     6
5  H     -0.448269   -0.763921    3.325671    37     4
6  H     -0.448269    0.763921    3.325671    37     4
```

In the above 36 and 37 are atom types as defined in `amoeba09.prm`, and the last two colums define connectivity. Do not worry about the absolute positioning of the molecule (if it crosses zero in any of the directions) – TINKER will work in OBC mode and will not attempt to wrap your atoms back to the simulation cell, because there is none. ONETEP will work with a suitably translated version of the molecule anyway.

Input `.tag` file

Now we need to designate each atom as part of the QM subsystem or the MM subsystem. This is done via a `.tag` file. This file should contain two or three lines. The first line specifies the indices of atoms belonging to the QM subsystem. The second line specifies the indices of atoms belonging to the MM subsystem, like this:

```
1 2 3
4 5 6
```

The above assigns the first water molecule to the QM subsystem, and the second water molecule to the MM subsystem. In the `.tag` file, all atoms must be accounted for. If you want TINKTEP to ignore some atoms (say, you have `.xyz` file of a large system and want to discard some of it), put their indices in the third line. Normally you would simply omit the third line. It is not permitted for covalent bonds to span the QM/MM interface, and TINKTEP will refuse to proceed if it detects this. For instance this:

```
1 2
3 4 5 6
```

would not be a valid `.tag` file.

If you want *no* atoms in the QM subsystem (for a purely MM calculation) or the MM subsystem (for a purely QM calculation), put `-1` in the corresponding line, rather than leaving it blank.

Do not put any comments or additional information in the `.tag` file, that would make it misformatted.

Rename your `.tag` file to use the same base name as the `.xyz` file (say, `my_molecule.tag` and `my_molecule.xyz`).

Input .key file

Create a .key file with the same base name as the .xyz and .tag files, and with the following contents:

```
digits 9
parameters amoeba09
```

The line with digits 9 is necessary to force TINKER to use extra precision in its outputs. The second line specifies the MM parameter file for TINKER. Adjust it if you want to use a file different from amoeba09.prm.

Input .dat.template file

Create a .dat.template file with the same base name as the .xyz, .tag and .key files. Populate this file with the keywords you want to be passed to ONETEP. Essentially, this file will be slightly modified by TINKTEP (specifically by qm_xyz_to_dat) and will become the .dat that ONETEP will read. The modifications undertaken by TINKTEP are:

- pol_emb_pot_filename will be added[1] to instruct ONETEP to perform a QM/MM calculation and inform it about the name of the file used for communicating between ONETEP and TINKER.
- pol_emb_polscal will be set accordingly if qm_mm_polscal was set in tinktep.config.
- pol_emb_thole_a will be set accordingly if qm_mm_thole_a was set in tinktep.config.
- pol_emb_fixed_charge T will be added if the MM force field is *not* polarisable (e.g. GAFF), to inform ONETEP about this fact.
- A %block positions_abs will be added, containing the species and positions of QM atoms inferred from the .xyz and .tag files, suitably translated.
- If tinktep.config specified qm_thole_polarisability, a %block thole_polarisabilities will be added, containing the Thole polarisabilities of QM atoms inferred from the .prm and .tag files.
- If the scenario of a purely QM calculation with classical atoms ("sparkles") has been selected by specifying classical_atoms in tinktep.config, a %block classical_info will be added, containing the species and positions of MM atoms inferred from the .xyz and .tag files, suitably translated.
- If the .xyz file contained a bounding box (for PBC calculations), a suitable %block lattice_cart will be added to match the MM box size. PBCs are not supported yet, do not rely on this functionality.

Basically, what you put in the .dat.template file should look like a normal ONETEP .dat file, *except for* the positions of atoms. Do not attempt to create a .dat file on your own, leave it to TINKTEP to create it automatically when it is run. For instance, for our water dimer example you could use this bare-bones .dat.template file:

```
! --- usual ONETEP keywords ---
%block species_atomic_set
H  "SOLVE"
O  "SOLVE"
%endblock species_atomic_set

%block species
H  H 1 1 7.0
O  O 8 4 7.0
%endblock species

%block species_pot
```

(continues on next page)

[1] Except for purely QM calculations.

(continued from previous page)

```
H  'H_04.recpot'
O  'O_02.recpot'
%endblock species_pot

%block lattice_cart
30.0  0.0  0.0
 0.0 30.0  0.0
 0.0  0.0 30.0
%endblock lattice_cart

cutoff_energy 1000 eV
charge 0
xc_functional PBE
dispersion 1

! --- cutoff Coulomb to enforce OBCs in ONETEP ---
coulomb_cutoff_type SPHERE
coulomb_cutoff_radius 40.0 bohr
coulomb_cutoff_write_int F

! --- DMA setup, needed for QM/MM. Consult DMA manual for details ---
%block swri
  for_dma 1 12 V 12 12 W
%endblock swri

%block species_swri-for_dma
H
O
%endblock species_swri-for_dma

dma_calculate T
dma_use_ri for_dma
dma_max_l 1
dma_max_q 12
dma_metric ELECTROSTATIC
dma_bessel_averaging T
dma_scale_charge F

! --- Polarisable embedding, needed for QM/MM. See further text. ---
pol_emb_dma_min_l 0
pol_emb_dma_max_l 1
pol_emb_mpole_exclusion_radius 1.00 bohr
pol_emb_repulsive_mm_pot_cutoff 10.0 bohr

%block mm_rep_params
H   35 2.400 ! follows TINKTEP-2 paper
O  550 1.580 ! follows TINKTEP-2 paper
%endblock mm_rep_params
```

TINKER's `.prm` file

Copy the `.prm` file of your choice (typically `amoeba09.prm`) to the same directory where you put the `.xyz`, `.tag`, `.key` and `.dat.template` files. Do not rename it. Ensure its basename is reflected in the `parameters` keyword in the `.key file`.

`tinktep.config` file

This is the main file for controlling the QM/MM calculation. Its name is fixed, do not change it. Here's an example suitable for our water dimer, using the TINKTEP2 model:

```
jobname water_dimer

# *** Computational environment set-up ***
tinker_nthreads 8
onetep_nranks 2
onetep_nthreads 8
onetep_executable ./onetep.RH7
mpirun_executable mpirun

# *** Nuts and bolts of the QM/MM interface ***
qm_mm_polscal 6.0
qm_polarisability
qm_thole_polarisability
renumber_offset 500

# *** Physics ***

# Undamped fixed, permanent multipoles using the full density representation,
# and Thole-damped induced dipoles using the QM* representation. MM repulsive potential.
onetep_perm_mpoles     perm_fix_rep potential_coulombic_smeared energy_from_potential
onetep_induced_dipoles ind_qmstar   potential_thole_damped      energy_from_potential

# TINKER handles all bonded (valence) terms between MM atoms.
tinker_bond_energy 1
tinker_angle_energy 1
tinker_ureybrad_energy 1

# TINKER handles MM electrostatics.
tinker_mm_perm_energy 1
tinker_mm_pol_energy 1

# ONETEP handles QM/MM electrostatics.
tinker_qm_mm_perm_energy 0
tinker_qm_mm_pol_energy 0

# TINKER handles van der Waals for MM, and only the dispersive term for QM/MM.
tinker_mm_vdw_energy 1
tinker_qm_mm_vdw_energy 2
```

All `tinktep.config` keywords will be described later.

10.1.6 Running a QM/MM calculation

Once you have all input files in place, simply type `tinktep` to run the QM/MM calculation. Expect the following to happen:

1. `xyz_split` will be run to split your `.xyz` file into a `qm.xyz` and a `mm.xyz` file, based on the contents of the `.tag` file.
2. `qm_xyz_to_dat` will be run to build a ONETEP `.dat` file from the `.dat.template` file and the `qm.xyz` file, using information from the `.prm` file.
3. A pair of FIFOs (`$QM2MM.lock` and `$MM2QM.lock`) will be created. These will be used for interprocess communication (ONETEP to TINKTEP and TINKTEP to ONETEP).
4. ONETEP will be started in the background (via `mpirun` or equivalent).
5. A watchdog process will be started in the background. It will keep an eye on the `mpirun` process that launched ONETEP and on ONETEP's `.err` and `.error_message` files. It will attempt to clean up gracefully if it decides that ONETEP crashed or was killed.
6. [scf]TINKTEP will block until it ONETEP reaches a point where total energy needs to be evaluated. Then it will resume.
7. TINKTEP will read the `.gdma_like.txt` file produced by ONETEP. This file contains the multipole representation of the QM subsystem. It will run TINKER's `poledit` to process these multipoles.
8. `xyz_process` will be run to process the `qm.xyz` and `mm.xyz` files to a form digestible by TINKER (`qm_mm.xyz` file). This is mostly about renumbering the atom types in the QM subsystem so that they do not clash with the types in the `.prm` file.
9. `key_process` will be run to prepare a suitable `.key` file for TINKER (`qm_mm.key`). This takes the contents of the original `.key` file, and modifies it accordingly so that it is digestible by TINKER. For instance dummy bond and angle parameters will be supplied for the QM atoms, QM sites and multipoles will be renumbered, polarisabilities of QM atoms will be defined, the QM subsystem will be made inactive and "only formally polarisable", etc..
10. TINKER (specifically `analyze` and `dynamic`) will be run to obtain the polarisation response of the MM subsystem, all of MM electrostatics, QM/MM electrostatics, QM/MM van der Waals energies, MM van der Waals and MM bonded interactions. Not all of these terms will necessarily be used in the final energy expression.
11. `mpoles_process` will be run to process TINKER's multipoles and energy terms into a format understandable by ONETEP (`.mpoles_for_onetep` file).
12. ONETEP will resume, after having been pinged via `$MM2QM.lock`.
13. If SCF convergence has been reached, TINKTEP will terminate with a short summary. If not, control will transfer to step [scf].

All in all, the TINKTEP script drives both ONETEP and TINKER. ONETEP is executed once, in the background, and is resumed when necessary. TINKER is started each time ONETEP performs an energy evaluation. TINKER, which does not support MPI parallelism, is run on the local node (possibly using OMP threads). ONETEP is started via `mpirun` or equivalent, and it's the user's responsibility to set the parallel environment in such a way, that ONETEP gets started on the appropriate nodes (e.g. via a hostfile).

All ONETEP output goes to a `qm_mm.out` file with the same base name as the input. All ONETEP error messages go to a `qm_mm.err` file with the same base name as the input. TINKTEP's output goes to `stdout` and `stderr`. TINKTEP attempts to detect a large variety of error conditions and should at least provide an informative error message if something goes wrong. When diagnosing errors, examine `stderr`, ONETEP's `qm_mm.err` file, ONETEP's `qm_mm.error_message` file (if any), and see if there's a file called `error_message` (with no base name) – it is created when more exotic error conditions occur.

All intermediate files are automatically moved to a subdirectory called `intermediate` (at each SCF iteration), they are tagged with an SCF iteration (energy evaluation) number. It is safe to delete this directory after the calculation has run, it's mostly useful when diagnosing problems.

10.1.7 Output from a QM/MM calculation

Your `qm_mm.out` file will contain usual ONETEP output, interspersed with a lot of informative messages from DMA and polarisable embedding. Every time the energy is evaluated, you will get a detailed breakdown of energies associated with the QM/MM interface. Here's what it looks like:

```
/~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
| Polarisable embedding potential from water_dimer_mm.mpoles_for_onetep        |
| Multipole set "perm_fix_rep": 6 sites.                                       |
| Multipole set "ind_qmstar": 6 sites.                                         |
| All in all 12 sites, and 11 external energy terms (out of which 3 #ignored). |
| Energy term                              Energy      Source   Included?      |
| - MMv bond stretch                  0.000013814 Ha  TINKER      YES         |
| - MMv angle bend                    0.000229041 Ha  TINKER      YES         |
| - MMv Urey-Bradley                 -0.000000274 Ha  TINKER      YES         |
| - #QM/MM perm mpole                -0.032818501 Ha  TINKER       NO         |
| - #QM/MM polarisation              -0.000794393 Ha  TINKER       NO         |
| - MM perm mpole                     0.000000000 Ha  TINKER      YES         |
| - MM+ polarisation                  0.000000000 Ha  TINKER      YES         |
| - #QM/MM vdW-rep                    0.050008920 Ha  TINKER       NO         |
| - QM/MM vdW-disp                   -0.013332666 Ha  TINKER      YES         |
| - MM vdW-rep                        0.000000000 Ha  TINKER      YES         |
| - MM vdW-disp                       0.000000000 Ha  TINKER      YES         |
| - QM elec <-> rep MM perm_fix       0.033334141 Ha  ONETEP   REPULS  P Fr    |
| - QM elec <-> MM perm_fix_rep       0.229940160 Ha  ONETEP   COUL-S  P Fr    |
| - QM core <-> MM perm_fix_rep      -0.263586238 Ha  ONETEP   COUL-S  P FR    |
| - QM* elec <-> MM ind_qmstar        0.014117659 Ha  ONETEP    THOLE  I*      |
| - QM core <-> MM ind_qmstar        -0.014912052 Ha  ONETEP    THOLE  I*      |
|------------------------------------------------------------------------------|
|                     External |           Internal |         Difference      |
| Permanent:       -0.032818501 |       -0.033646079 |     0.000827577479 (Ha) |
| Induced:         -0.000794393 |       -0.000794393 |    -0.000000000001 (Ha) |
|------------------------------------------------------------------------------|
| Perm embed. potential  min: -0.3743E+01  max:  0.6338E+01  norm:  0.1776E-01 |
\~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/
```

From the above example you can infer the following:

1. The file from which ONETEP reads the details of the MM embedding is `water_dimer_mm.mpoles_for_onetep`.
2. There are two sets of multipoles, with 6 sites each. The first set entails permanent (`perm`) (as in not induced), fixed (`fix`) (as in not variable in time) multipoles that generate a repulsive (`rep`) potential. The second set entails induced (`ind`) multipoles, which interact not with ONETEP's full electronic density, but with the QM* description (`qmstar`). You probably expected 3, not 6 sites, but the `.mpoles_for_onetep` file also includes QM sites in addition to MM sites. The QM sites are tagged with "#" and subsequently ignored.
3. There are 11 energy terms coming from TINKER, but 3 of these will be ignored by ONETEP in accordance with the user's wishes. The ignored terms are prefixed by "#" and have a "NO" in the "Included?" column. The ignored terms in this case are:

- QM interaction with permanent MM multipoles as calculated by TINKER. This is because we instead use the full density QM representation (as calculated by ONETEP) for this term, excluding TINKER's approximate idea on purpose (compare `tinker_qm_mm_perm_energy 0` earlier).
- QM interaction with induced MM multipoles as calculated by TINKER. This is because we instead use the value calculated by ONETEP for this term, even though the two should be (and are) identical, excluding TINKER's value on purpose (compare `tinker_qm_mm_pol_energy 0` earlier).
- Repulsive part of QM/MM van der Waals interaction as calculated by TINKER. This is because we instead use the repulsive potential model introduced in TINKTEP2, calculated by ONETEP for this term, excluding TINKER's classical value on purpose (compare `tinker_qm_mm_vdw_energy 2` earlier).

The other terms coming from TINKER are included. These are: MM valence terms (`MMv`): bond, angle and Urey-Bradley, MM-MM permanent multipole interactions (zero in this case, as there is only one molecule in MM, and AMOEBA masked the MM permanent interactions within the water molecule), MM polarisation ("+" serves as a reminder that this polarisation is not strictly only due to MM, because of non-additivity) (again zero in this case, because of masking), dispersive part of QM/MM van der Waals interactions, and MM/MM vdW terms (repulsion and dispersion) (also zero, since there is only one molecule in MM).

4. There are 5 energy terms coming from ONETEP (denoted by "ONETEP" in the "Source" column). These are
   - The interaction of QM electrons with the MM repulsive potential attached to one of the multipole sets (`REPULS`).
   - The interaction of QM electrons with the permanent MM multipoles, treated Coulombically with smearing (`COUL-S`).
   - The interaction of QM ionic cores with the permanent MM multipoles, treated Coulombically with smearing (`COUL-S`).
   - The interaction of QM* electrons (i.e. electronic multipoles) with the induced MM multipoles, treated using Thole damping (`THOLE`).
   - The interaction of QM ionic cores (i.e. ionic charges) with the induced MM multipoles, treated using Thole damping (`THOLE`).
5. The symbols to the right of the table inform us about the assumptions ONETEP makes about some energy terms:
   - Column 1: `P` – MM multipole set is permanent, or `I` – MM multipole set is induced. This affects energy expressions – induced multipoles require work to assemble, cf. Ref. [Dziedzic2016].
   - Column 2: `*` – calculation uses the QM* representation of the density, or (blank) – calculation uses the full density.
   - Column 3: `F` – MM multipole set is fixed (its value is time-independent), so its electrostatic potential can be stored and reused, or (blank) – MM multipole set is not fixed (then its electrostatic potential has to be recalculated every time).
   - Column 4: `1` – Energy term has been calculated for the first time, and will either be reused later (e.g. for QM cores interacting with permanent MM multipoles) or at least the MM potential will be reused (e.g. for QM electrons interacting with permanent MM multipoles), or `R` – energy term has just been reused, or `r` – the MM potential has been reused, but the energy has been recalculated, or (blank) – neither of the above.
   - Column 5: `S` – `dma_multipole_scaling` affected this energy term, or `s` – `pol_emb_perm_scaling` affected this energy term. These are expert directives, do not worry about them.
6. This is followed by a summary of QM/MM permanent and QM/MM induced electrostatics. "External" is TINKER's idea of these energy terms, "Internal" is ONETEP's idea, "Difference" is the difference between the two. Unless you do something very exotic, like ignoring polarisation, the row with "Induced" should always match extremely well, because both ONETEP and TINKER use the same model (QM:math:^* interacting with MM dipoles), and their calculations should match (if not, this indicates a bug or a set-up problem, and ONETEP will

abort). Unless you do something exotic, like using the QM* for permanent interactions, the row with "Permanent" will not match, because TINKER uses the QM* model and thus suffers from charge penetration, while ONETEP uses the full density for permanent interactions, arriving at the "right" result. Here, "Difference" is a good estimate of QM/MM charge penetration error.

7. The last row gives some statistics about the permanent MM multipole potential in which QM electrons are embedded.

10.1.8 `tinktep.config` directives

Here is a list of directives understood by TINKTEP that you can put in the `tinktep.config` file. Make sure you spell these right, unlike ONETEP, TINKTEP **silently ignores** directives it does not recognise. You can use "#" to denote comment lines. These will be ignored.

Environment set-up

`jobname (string)` **[mandatory, basic]** – specifies the base name for input (`.xyz`, `.tag`, `.key`, `.dat.template`), files. Example: `jobname qm_tryptophan_in_40_mm_waters`.

`onetep_executable (string)` **[mandatory, basic]** – specifies the name of the ONETEP executable that TINKTEP will pass to `mpirun` (or equivalent). This file must be user-executable.

`onetep_nranks (integer)` **[mandatory, basic]** – specifies the number of MPI ranks that TINKTEP will tell `mpirun` (or equivalent) to start ONETEP on.

`onetep_nthreads (integer)` **[mandatory, basic]** – specifies the number of OMP threads that TINKTEP will tell ONETEP to use (by setting `OMP_NUM_THREADS`. You can always override this with specific ONETEP thread keywords in the `.dat.template` file.

`onetep_args (string)` **[optional, expert]** – specifies additional arguments that you might want to pass to ONETEP. These will go between the ONETEP executable and the ONETEP input file. This only makes sense if your `onetep_executable` actually points to a wrapper script that will know what to do with these arguments.

`tinker_nthreads (integer)` **[optional, intermediate]** – specifies the number of OMP threads that TINKTEP will tell TINKER to use (by adding a keyword to `qm_mm.key`. If left unspecified, this will be left at TINKER's discretion. Caveat: TINKER sometimes carelessly outputs to `stdout` from OMP regions, which can cause a mess. If TINKTEP complains that it cannot parse TINKER's output, try setting this to 1 to disable OMP in TINKER.

`mpirun_executable (string)` **[mandatory, basic]** – specifies the name of the `mpirun` executable that TINKTEP will use to launch ONETEP. Set this to `mpirun`, unless your environment uses something fancier like `aprun`, or you want to specify a full path to select a specific `mpirun` executable.

`mpirun_args (string)` **[optional, intermediate]** – specifies additional arguments that you might want to pass to `mpirun`. These will go between the `mpirun` executable and `-np <onetep_nranks>`. Can be useful for passing a hostfile name.

`watchdog_unfazed_by_stderr (no args)` **[optional, intermediate]** – tells TINKTEP's watchdog not to keep an eye on ONETEP's `.err` file. Normally any output to this file is an indication that something went wrong, and the watchdog then initiates cleanup. In some environments you can get innocuous messages written to a job's `.err` file, e.g. warnings from MPI or the transport layer. Use this directive to immunize the watchdog against these.

QM/MM set-up: basic

`onetep_perm_mpoles (set_name) (potential_mode) (energy_mode)` **[mandatory, basic]** – specifies the treatment of permanent MM multipoles inside ONETEP. All three arguments are strings and are mandatory. These are extremely important and have to be discussed in detail.

- `set_name` – a short, descriptive name that will identify the permanent MM multipole set in ONETEP. Crucially, this name will also be parsed by ONETEP to infer the properties of this set. Thus, certain tokens (substrings) carry very specific meaning in the context of the name. These are:
  - `perm` – the set is permanent (as in "not induced"). This affects energy expressions – induced multipoles require work to assemble, permanent sets do not, cf. Ref. [Dziedzic2016].
  - `ind` – the set is induced (as in "not permanent"). This affects energy expressions – induced multipoles require work to assemble, permanent sets do not, cf. Ref. [Dziedzic2016].
  - `fix` – the set is fixed (as in "not changing in time"). This does not mean "not moving through space" (currently *all* MM multipoles are presumed not to be moving through space). Calculations on fixed sets will be optimised to re-use electrostatic potentials or entire energy terms.
  - `qmstar` – the set uses the QM* representation and not the full QM density when interacting with the QM subsystem. If absent, the full QM density is used by default.
  - `rep` – the set generates an MM repulsive potential (for the TINKTEP2 model). If absent, no MM repulsive potential will be generated by the set. Note: If `rep` is present for more than one set, only the last set set will generate the MM repulsive potential.
- `potential_mode` – describes how ONETEP generates the electrostatic potential coming from this multipole set. Four options are possible:
  - `potential_zero` – the set does not generate any potential (and so is essentially ignored).
  - `potential_coulombic_smeared` – the set generates a Coulombic potential, with a small degree of smearing only very close to the location of each point multipole – this is done to avoid singularities (cf. `pol_emb_mpole_exclusion_radius`, `polemb_smearing_a`).
  - `potential_coulombic_masked` – the set generates a Coulombic potential, which is, however, masked (zeroed) very close to the location of each point multipole – this is done to avoid singularities (cf. `pol_emb_mpole_exclusion_radius`). This is not recommended, except for tests, use `potential_coulombic_smeared` instead.
  - `potential_thole_damped` – the set generates a Thole-damped potential, mimicking AMOEBA polarisation interactions. Thole damping is a classical scheme and is designed to be applied to interactions between two point multipoles. Thus it is best suited to the QM* representation (cf. `qmstar` above), and not to interactions between MM point multipoles and the full, distributed QM density, although this is, in principle, possible. The magnitude of the damping depends on the polarisabilities of the two interacting sites. For MM sites the polarisabilities are determined by the force field (`.prm` file). For QM* sites the polarisabilities either have to be specified in the `.dat.template` file (`%block thole_polarisabilities`), or can be inferred automatically from the `.prm` file. The latter option is preferred, it can be activated via the directive `qm_thole_polarisability` in `tinktep.config`. This instructs TINKTEP (and `qm_xyz_to_dat` in particular) to add a suitable `%block thole_polarisabilities` automatically. When an attempt is made to use Thole damping with a multipole set that does not specify `qmstar`, a Thole-damped potential coming from the set will need to be integrated with the full QM density, and there is no corresponding polarisability that can be used in the Thole damping expression. In this unlikely scenario, the value of `pol_emb_pairwise_polarisability`, with a unit of bohr, is used for the Thole variable A (which is otherwise equal to $\sqrt{\alpha_1 \alpha_2}$). The default value of this parameter corresponds to the average polarisability of all atom types in AMOEBA09.
- `energy_mode` – describes how ONETEP calculates the electrostatic energy of this multipole set interacting with the QM subsystem. Two options are possible:

- `energy_zero` – the set does not contribute to energy (and so is essentially ignored).
- `energy_from_potential` – the set's contribution to energy will be made consistent with the setting for the potential (see above).

Not all combinations of `potential_mode` and `energy_mode` make sense. For instance trying to combine `potential_coulombic_smeared` with `energy_zero` will lead to an inconsistency between the Hamiltonian and the energy expression, breaking LNV and NGWF convergence. Using `energy_from_potential` is generally the safest option, as it guarantees consistency.

`onetep_induced_dipoles (set_name) (potential_mode) (energy_mode)` **[mandatory, basic]** – specifies the treatment of induced MM dipoles inside ONETEP. The meaning of the arguments is the same as for `onetep_perm_mpoles` above.

Some typical settings for the above two keywords:

```
# Usual TINKTEP2 set-up: Permanent multipoles interact Coulombically with full QM␣
↪density,
#                        induced dipoles interact with QM* via Thole damping,
#                        repulsive MM potential in effect (attached to perm. multipoles)
onetep_perm_mpoles      perm_fix_rep  potential_coulombic_smeared  energy_from_potential
onetep_induced_dipoles  ind_qmstar    potential_thole_damped       energy_from_potential

# Usual TINKTEP1 set-up: Permanent multipoles interact Coulombically with full QM␣
↪density,
#                        induced dipoles interact with QM* via Thole damping,
#                        no repulsive MM potential in effect.
onetep_perm_mpoles      perm_fix      potential_coulombic_smeared  energy_from_potential
onetep_induced_dipoles  ind_qmstar    potential_thole_damped       energy_from_potential

# TINKTEP2 set-up for a non-polarisable force-field (eg. GAFF).
onetep_perm_mpoles      perm_fix_rep  potential_coulombic_smeared  energy_from_potential
onetep_induced_dipoles  ind_qmstar    potential_zero               energy_from_potential

# TINKTEP2 set-up, where both permanent and induced interactions use QM* and Thole␣
↪damping
onetep_perm_mpoles      perm_fix_rep_qmstar  potential_thole_damped  energy_from_
↪potential
onetep_induced_dipoles  ind_qmstar           potential_thole_damped  energy_from_
↪potential
```

`qm_thole_polarisability (no args)` **[optional, basic]** – asks TINKTEP to automatically infer the Thole polarisabilities of QM sites from the `.prm` file and to automatically add a suitable `%block thole_polarisabilities` to the `.dat` file. Strongly recommended.

`qm_polarisability (no args)` **[optional, basic]** – forces TINKER to treat QM sites as "formally polarisable", that is, to take their polarisabilities into account in Thole damping expressions, but not to put induced dipoles on them. Treat this directive as mandatory, the alternative scheme is no longer supported.

QM/MM set-up: treatment of energy terms

The following directives control how different energy terms generated by TINKER are taken into account in the QM/MM calculation.

`tinker_bond_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the valence MM term due to bond stretches is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this enabled. Disabling might be necessary if there are no bonds between MM atoms (e.g. for a noble gas).

`tinker_angle_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the valence MM term due to angle bends is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this enabled. Disabling might be necessary if there are no angles between MM atoms (e.g. for a noble or diatomic gas).

`tinker_ureybrad_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the valence MM term due to Urey-Bradley interactions is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this enabled. Disabling might be necessary if there are no Urey-Bradley interactions between MM atoms.

`tinker_mm_perm_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the electrostatic term due to permanent MM-MM interactions is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this enabled.

`tinker_mm_pol_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the electrostatic term due to MM-MM polarisation interactions is included in the QM/MM energy expression. Even though polarisation interactions are not additive, TINKER formally splits them *a posteriori* into QM-MM polarisation and MM-MM polarisation, which add up to total MM polarisation. When omitted, defaults to `0`. In typical scenarios you'd want this enabled.

`tinker_qm_mm_perm_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the electrostatic term due to interactions between permanent MM multipoles and QM is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this **disabled**, because TINKER's idea of this interaction suffers from charge-penetration error. This term would then be taken into account on ONETEP's side, via an `energy_from_potential` setting for permanent multipoles (cf. `onetep_perm_mpoles`).

`tinker_qm_mm_pol_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the electrostatic term due to QM-MM polarisation interactions is included in the QM/MM energy expression. Even though polarisation interactions are not additive, TINKER formally splits them *a posteriori* into QM-MM polarisation and MM-MM polarisation, which add up to total MM polarisation. When omitted, defaults to `0`. In typical scenarios you'd want this **disabled**. This term would then be taken into account on ONETEP's side, via an `energy_from_potential` setting for induced dipoles (cf. `onetep_induced_dipoles`). The two expressions should yield the same result, provided `qmstar` is used for `onetep_induced_dipoles`. Having ONETEP calculate this term permits checking the two results (TINKER's and ONETEP's) for consistency.

`tinker_mm_vdw_energy (0 or 1)` **[optional, intermediate]** – when enabled (`1`), the van der Waals term due to MM-MM non-bonded interactions is included in the QM/MM energy expression. When omitted, defaults to `0`. In typical scenarios you'd want this enabled.

`tinker_mm_vdw_energy (0 or 1 or 2 or 3)` **[mandatory, intermediate]** – controls if and how the van der Waals term due to QM-MM non-bonded interactions is included in the QM/MM energy expression. The following values are possible:

- 0 – omit QM-MM vdW interactions entirely.
- 1 – include QM-MM vdW interactions, both the repulsive and dispersive term, via classical Halgren potential. Recommended for TINKTEP1 model.
- 2 – include QM-MM vdW interactions, but only the dispersive term, via classical Halgren potential. Recommended for TINKTEP2 model, where repulsive QM-MM vdW interactions would be handled via a repulsive MM potential.

- 3 – include QM-MM vdW interactions, but only the repulsive term, via classical Halgren potential. Only included for completeness.

QM/MM set-up: details of the interface

The following directives control how the QM/MM interface behaves.

`qm_mm_polscal (real)` **[optional, intermediate]** – controls scaling of QM/MM polarisation interactions (introduced in TINKTEP2). Defaults to no scaling if omitted. The apparent polarisability of QM sites is scaled (multiplied) by the value of this parameter, thereby increasing damping if it is greater than 1.0, and attenuating damping (increasing the strength of polarisation interactions) if it is between 0.0 and 1.0. Recommended value: 6.0 for TINKTEP2, omit for TINKTEP1.

`renumber_offset (integer)` **[mandatory, intermediate]** – specifies the value by which atom types for QM atoms are offset from their original force field counterparts. This is used to avoid clashes between "fake" QM types presented to TINKER and original atom types. Use a large, three-digit value, like 500, unless you have good reason for doing otherwise.

`coord_xlat (x) (y) (z)` **[optional, intermediate]** – specifies the vector (in bohr) by which all QM atoms are translated in the `.dat` file. All arguments are real numbers and are mandatory. Values will be interpreted as bohr, do not specify any units. This directive becomes useful when the contents of the *qm.xyz* file are positioned unsuitably for ONETEP in OBC mode, e.g. leading to NGWFs not fitting in the simulation cell. In general, TINKER always works with original (untranslated) coordinates, and ONETEP always sees coordinates translated by this vector. All `.xyz` files work with the original coordinates, the `.dat` file works with the translated coordinates. If omitted, this value will be determined automatically: a bounding box will be calculated for the QM subsystem, and the translation vector will be such that the centre of the bounding box will coincide with the centre of the simulation cell. This is handy, but can lead to eggbox errors when comparing energies between different molecules (as they might be translated differently). It is advised to set this translation vector manually, and identically for all molecules whose energies will be compared.

QM/MM set-up: expert options

`classical_atoms (no args)` **[optional, expert]** – when present, TINKTEP will add a `%block classical_info` containing the species and positions of MM atoms inferred from the `.xyz` and `.tag` files, suitably translated, to the `.dat` file. Charges for the classical atoms will be extracted from specially crafted comments in the `.dat.template` file. This is useful when generating reference classical embedding ("sparkles") calculations. You should normally omit this directive. The format for specifying charges of classical atoms is as follows:

```
!$ classical_species_charge H 0.417
!$ classical_species_charge O -0.834
```

These lines are formally comments and will be ignored by ONETEP, but `qm_xyz_to_dat` will know how to interpret them.

`mm_fixed_charge (no args)` **[optional, expert]** – when present, instructs TINKTEP that the MM force field is not polarisable. Leave this directive out by default. Non-polarisable force-fields need a few particular tweaks (TINKER's `multipole` keyword is replaced by `charge` in the `qm_mm.key` file, the `polarizable` keyword needs to be omitted from same, `pol_emb_fixed_charge T` will be automatically added to the `.dat` file to instruct ONETEP not to look for `%block thole_polarisabilities` in the `.dat` file, and TINKER's output will be parsed differently). These tweaks are activated by this directive.

`pure_mm (no args)` **[optional, expert]** – when present, instructs TINKTEP that the calculation is a purely MM calculation, and ONETEP does not need to be invoked.

`pure_qm (no args)` **[optional, expert]** – when present, instructs TINKTEP that the calculation is a purely QM calculation: TINKTEP does not need to be invoked, and `pol_emb_pot_filename T` must **not** be added to the `.dat`

file so that ONETEP runs without polarisable embedding. This is necessary due to a bug in TINKER, which causes it to hang if all atoms are inactive.

`qm_mm_thole_a (real)` **[optional, expert]** – when present, the specified value overrides the value of Thole's a parameter read from the `.prm` file. This value will be used *only* for QM/MM interactions, as TINKER will use the force-field value for MM/MM interactions.

`qm_dummy_atoms (no args)` **[optional, expert]** – experimental functionality for adding dummy DMA sites on MM atoms, do not use.

Command-line options to the tinktep script

`tinktep` is normally run without arguments. The following command-line options are supported:

`-1` – Only the first two stages described in Section [running] are performed, then TINKTEP exits. In essence, all input files are parsed and intermediate inputs are prepared, but neither ONETEP nor TINKER are actually run.

`-2` – All stages described in Section [running], *except* the first two, are performed. In essence, the calculation is run, assuming all intermediate inputs have been prepared in advance. Splitting the two parts of the calculation (preparation and actual execution) enables manual or scripted tweaking of intermediate inputs by the user.

`-dry-run` – equivalent to `-1`, only deprecated.

10.1.9 ONETEP keywords pertaining to polarisable embedding

A number of ONETEP keywords can be used to control the polarisable embedding functionality, which underlies QM/MM calculations.

Keywords mandatory for polarisable embedding

`pol_emb_pot_filename (string)` – specifies the name of the file used for exchanging information between ONETEP and TINKTEP. This file will be read by ONETEP and constructed by TINKTEP at every energy evaluation. This keyword is also used to turn on polarisable embedding (when it is present), and to turn it off (when it is absent). Default: absent.

`pol_emb_dma_max_l (integer)` – specifies the maximum angular momentum in the SW basis used in polarisable-embedding-DMA. In most scenarios this will be equal to $l_{\max}$ that you specified in the SWRI block. Read the description of $l_{\max}$ in the DMA documentation to understand the meaning of this parameter. You can use a lower value than the one specified in the SWRI block if you want to use only a subset of the SW basis set (e.g. for benchmarking). This keyword does not affect properties-DMA (for which `dma_max_l` should be used). This directly affects the quality of the QM* representation – specifying 0 leads to a charge-only representation, specifying 1 leads to a charge-and-dipole representation, specifying 2 leads to a charge-dipole-and-quadrupole representation.

Common optional keywords

`pol_emb_polscal (real)` – specifies the scaling factor applied to QM polarisabilities (cf. `qm_mm_polscal tinktep.config` directive). This keyword will be added automatically by TINKTEP, do not add it on your own. Default: 1.0.

`pol_emb_repulsive_mm_pot_cutoff (physical)` – specifies the cutoff (in length units) for the MM repulsive potential (cf. `rep` token in `tinktep.config` directives `onetep_perm_mpoles` and `onetep_induced_dipoles`). The value of the MM repulsive potential from an MM site will be assumed to be zero beyond this cutoff. This is used to minimise computational effort. The default is 10.0 bohr.

pol_emb_mpole_exclusion_radius (physical) – specifies the distance (in length units) around any multipole below which its Coulombic potential is masked or smeared (cf. potential_coulombic_masked and potential_coulombic_smeared arguments to tinktep.config directives onetep_perm_mpoles and onetep_induced_dipoles. This has no effect on Thole-damped multipole sets. The default is 0.25 bohr.

pol_emb_fixed_charge (logical) – if T, the MM force field will be assumed to be non-polarisable. If F (which is the default), the MM force field will be assumed to be polarisable. When the force field is polarisable and the QM* representation is used (pol_emb_qmstar T), %block thole_polarisabilities becomes mandatory. This keyword will be added automatically by TINKTEP, do not add it on your own.

pol_emb_qmstar (logical) – if T, the QM* representation will be employed for some or all QM-MM interactions. This is automatically set to T once polarisable embedding is activated. Only use F in the rare scenario, where the QM* representation will not be necessary (e.g. for non-polarisable force fields) and you want to elide %block thole_polarisabilities.

Uncommon and expert optional keywords

pol_emb_thole_a (real) – can be used to override the value of Thole's a parameter read from the .prm file. This value will be used *only* for QM/MM interactions, as TINKER will use the force-field value for MM/MM interactions. This keyword will be added automatically by TINKTEP if qm_mm_thole_a is specified in tinktep.config, do not add it on your own. Default: 0.39.

pol_emb_smearing_a (physical) – can be used to control the aggresiveness of Coulombic smearing (cf. potential_coulombic_smeared argument to tinktep.config directives onetep_perm_mpoles and onetep_induced_dipoles. This has no effect on Thole-damped multipole sets. The default is 0.2 bohr.

pol_emb_pairwise_polarisability (physical) – see discussion of potential_coulombic_smeared argument to tinktep.config directives onetep_perm_mpoles and onetep_induced_dipoles. Default: 1.92618 bohr (yields the average AMOEBA09 polarisability). You should not have to use this keyword.

pol_emb_repulsive_mm_pot_write (logical) – if set to T, the repulsive MM potential will be written to a .cube/.dx/.grd file at every energy evaluation. Used for debugging purposes. Default: F.

pol_emb_dbl_grid (logical) – if set to T the polarisable embedding contribution to the NGWF gradient will be computed on the double grid. By default it is computed on the coarse grid, like in HFx. Technically, to prevent aliasing, the double grid version should be used, but it is unbearably inefficient and has not been optimised much, as I plan this to remain an experimental facility at most. To ensure consistency, pol_emb_dbl_grid T should only be used together with swx_dbl_grid T, which ensures NGWF-SWOP overlaps are calculated on the double grid too. This functionality has not been thoroughly tested. I strongly recommend leaving this at default (F).

pol_emb_perm_scaling (real) – all QM permanent multipoles will be scaled by this factor (default: 1.0, so no scaling). This can be used to test the effects of overpolarising/underpolarising QM/MM interactions.

Experimental vacuum-DMA functionality

This is an experimental, unpublished and hardly-tested functionality that allows having two different QM* descriptions – one for the vacuum density (i.e. in the absence of polarisable embedding), and one for the difference between the density with embedding present and without it ("polarisation density"). In principle this permits expanding the vacuum state density e.g. up to quadrupoles, and the difference e.g. only in dipoles, to mimic AMOEBA (by excluding "polarisable quadrupoles" and "fluctuating charges" in QM). Preliminary tests (2017) showed that this hardly matters, but it might be worth studying in more detail.

pol_emb_vacuum_qmstar (logical) – when set to T, it enables the vacuum-DMA functionality. Default: F, unless (all three simultaneously) pol_emb_write_vacuum_restart F, pol_emb_vacuum_dma_max_l is present and pol_emb_qmstar T.

pol_emb_dma_min_l (integer) – specifies the minimum angular momentum in the SW basis used in polarisable-embedding-DMA *for the polarisation density*. In most scenarios this will be equal to 0. This keyword does not affect properties-DMA (for which the corresponding value is always 0). This directly affects the quality of the QM* representation *for the polarisation density*. Setting pol_emb_dma_min_l 1 and pol_emb_dma_max_l 1 in particular will cause the polarisation density to be expanded only in terms of dipoles. Default: 0.

pol_emb_vacuum_dma_min_l (integer) – specifies the minimum angular momentum in the SW basis used in polarisable-embedding-DMA *for the in-vacuum density*. In most scenarios this will be equal to 0. This keyword does not affect properties-DMA (for which the corresponding value is always 0). This directly affects the quality of the QM* representation *for the in-vacuum density*. Default: 0.

pol_emb_vacuum_dma_max_l (integer) – specifies the maximum angular momentum in the SW basis used in polarisable-embedding-DMA *for the in-vacuum density*. In most scenarios this will be equal to 1 (up to dipoles) or 2 (up to quadrupoles). This keyword does not affect properties-DMA (for which dma_max_l should be used). This directly affects the quality of the QM* representation *for the in-vacuum density*. Default: -1 (indicating "unset").

pol_emb_write_vacuum_restart (logical) – if set to T, restart files (.pol_emb_vac_tightbox_ngwfs and .pol_emb_kdenskern) will be written every time the NGWF gradient is calculated. These restart files can be later used in the polarisation calculation. Default: F

Block keywords used in polarisable embedding

The following blocks are used to control polarisable embedding in ONETEP:

- ```
  %block thole_polarisabilities
  <QMatom1> <polarisability1>
  <QMatom2> <polarisability2>
  ...         ...
  <QMatom<n>> <polarisability<n>>
  %endblock thole_polarisabilities
  ```

  This block specifies Thole polarisabilities for all QM atoms, in units of bohr^3. Normally (i.e. if qm_thole_polarisability is specified in tinktep.config), this block will be automatically added by TINKTEP, which infers the QM polarisabilities from the .prm and .tag files. If you prefer to specify polarisabilities manually, add this block and omit qm_thole_polarisability from tinktep.config. For example for water TINKTEP would generate:

  ```
  %block thole_polarisabilities
  O 5.648355971436 ! Water O
  H 3.347173908999 ! Water H
  H 3.347173908999 ! Water H
  %endblock thole_polarisabilities
  ```

- ```
  %block mm_rep_params
  <MMspecies1> <A!1> <zeta1>
  <MMspecies2> <A!2> <zeta2>
  ...         ...
  <MMspecies<n>> <A!<n>> <zeta<n>>
  %endblock mm_rep_params
  ```

  This block specifies the MM repulsive potential parameters for all MM species in the system. The parameter A is the magnitude (in hartree), the parameter zeta=ζ is the inverse-width (in bohr^{-1}). For example for water, parameterised as in Ref. [Dziedzic2018]:

```
%block mm_rep_params
H   35 2.400
O  550 1.580
%endblock mm_rep_params
```

10.1.10 Questions?

Questions should be directed to Jacek Dziedzic, `J.Dziedzic[-at-]soton.ac.uk.`

[Stone1998] A.J. Stone, GDMA: distributed multipoles from Gaussian98 wavefunctions (technical report), University of Cambridge (1998).

[Stone2005] A.J. Stone, Journal of Chemical Theory and Computation **6** 1128-1132 (2005).

[Dziedzic2016] J. Dziedzic, Y. Mao, Y. Shao, J. Ponder, T. Head-Gordon, M. Head-Gordon and C.-K. Skylaris, J. Chem. Phys. **145** 12 124106 (2016).

[Dziedzic2018] J. Dziedzic, T. Head-Gordon, M. Head-Gordon and C.-K. Skylaris, (in preparation) (2018).

[Vitale2015] V. Vitale, J. Dziedzic, S.M.-M. Dubois, H. Fangohr and C.-K. Skylaris, Journal of Chemical Theory and Computation **11** 7 3321-3332 (2015).

CHAPTER

ELEVEN

PROPERTIES

11.1 Energy Decomposition Analysis (EDA)

Author
Max Phipps, University of Southampton

Date
July 2016

11.1.1 Introduction

Energy decomposition analysis (EDA) decomposes the interaction energy (ΔE) of an arbitrary number of non-bonded fragments into its chemical components [Phipps2015]. In the case of the ONETEP EDA [Phipps2016], based on the ALMO [Kaliullin2007] and LMO [Su2009] EDA approaches, these components are combined into a frozen density component (ΔE_{FRZ}) term, polarisation (ΔE_{POL}), and charge transfer (ΔE_{CT}) (as per the original ALMO EDA), as,

$$\Delta E = \Delta E_{\mathrm{FRZ}} + \Delta E_{\mathrm{POL}} + \Delta E_{\mathrm{CT}} \quad .$$

The frozen density component, representing the interaction of the frozen densities of the fragments and subsequent antisymmetrization, is further decomposed into its electrostatics (ΔE_{ES}), exchange (ΔE_{EX}), correlation (ΔE_{CORR}), and Pauli repulsion (ΔE_{REP}) terms as,

$$\Delta E_{\mathrm{FRZ}} = \Delta E_{\mathrm{ES}} + \Delta E_{\mathrm{EX}} + \Delta E_{\mathrm{CORR}} + \Delta E_{\mathrm{REP}} \quad .$$

In the ONETEP implementation, the correlation term is further partitioned into its frozen ($\Delta E_{\mathrm{FRZ-CORR}}$) and Pauli repulsion ($\Delta E_{\mathrm{REP-CORR}}$) terms as,

$$\Delta E_{\mathrm{CORR}} = \Delta E_{\mathrm{FRZ-CORR}} + \Delta E_{\mathrm{REP-CORR}} \quad .$$

These terms may be recombined to obtain the full correlation energy term.

The total interaction energy is expressed in its decomposed form as,

$$\begin{aligned} \Delta E = &\Delta E_{\mathrm{ES}} + \Delta E_{\mathrm{EX}} + \Delta E_{\mathrm{CORR}} \\ &+ \Delta E_{\mathrm{REP}} + \Delta E_{\mathrm{POL}} + \Delta E_{\mathrm{CT}} \quad . \end{aligned}$$

11.1.2 Performing EDA calculations in ONETEP

There are two approaches to performing energy decomposition analysis (EDA) calculations in ONETEP:

1. All-In-One (1S) This approach involves creating a single input file with all the necessary information to complete an EDA calculation (`TASK EDA`).
   - Designed for smaller systems.
2. Three-Stage Fragment-to-Supermolecule (3S) This approach involves three stages. Firstly NGWF tightboxes and density kernels for the fragments are optimised and written to disk. Next the fragment states are superposed to the supermolecule to prepare the supermolecule data (`TASK EDA_PREP`). Finally the EDA calculation is ran using this supermolecule-prepared data (`TASK EDA`).
   - Designed for larger protein-ligand type EDA calculations.

Both approaches are equivalent and produce identical results. The choice of whether to use the 1S or 3S approach is usually determined by system size and parallelisation requirements (please see section [sect:ParallelisationReq]). Further details of calculations using these two approaches and a list of the EDA-specific keywords and descriptions are provided below:

All-in-One

The all-in-one (1S) approach is the default EDA calculation configuration, and is ran by simply setting the `TASK` keyword to `EDA`. In addition to the standard input keywords and blocks, the `EDA_IATM` block is necessary to perform ONETEP EDA calculations. This block defines the fragment atoms (referenced to the atom definitions in `POSITIONS_ABS`), and the fragment charges. Atoms of the supermolecule are arranged in the `POSITIONS_ABS` block by concatenation of the fragment atoms, using their nuclear coordinates found in the supermolecule. For example, if a supermolecule system comprising the fragments H_2O (atoms 1 to 3), OH^- (atoms 4 to 5), and Na^+ (atom 6) is defined in the `POSITIONS_ABS` block,

```
%block POSITIONS_ABS
  O      17.16973430    19.79077059    17.26422010
  H      16.00000000    19.22196603    16.00000000
  H      18.57568311    18.64371289    17.26799954
  O      21.16270452    17.46075058    18.04467287
  H      22.06598884    16.00000000    17.45886087
  Na     20.73940810    20.59012052    20.33500884
%endblock POSITIONS_ABS
```

then the fragment data necessary for the EDA calculation is input as,

```
%block EDA_IATM
  !N_at  Charge
   3     +0
   2     -1
   1     +1
%endblock EDA_IATM
```

Three-Stage

The 3S EDA has been developed to work easily with EDA directory structures, with fragment calculations being performed in separate folders. These data directories are specified in the input file using the EDA_FRAG and EDA_SUPER blocks as will be discussed below. In our EDA calculation example of a supermolecule comprising three fragments provided below, the following directory structure is used:

$$\left.\begin{array}{l}./\text{STAGE1/FRAG01/}\\ ./\text{STAGE1/FRAG02/}\\ ./\text{STAGE1/FRAG03/}\end{array}\right\}\text{Fragment calculations}$$
$$./\text{STAGE2}\}\ \text{Supermolecule preparation calculation}$$
$$./\text{STAGE3}\}\ \text{Supermolecule EDA calculation}$$

It is noted that the user is not limited to three fragments , and that the folder names given above are for example purposes only and any folder and input file names may be used.

In all three stages, it is necessary to ensure that the system parameters are kept at constant values. For example, it is unwise to modify the NGWF radii or box sizes between the calculation stages. It is also important to ensure atomic orderings in the input file are consistent, *i.e.* that the orderings of atoms within each of the fragments do not change between the different calculation stages.

Stage 1

The first stage simply involves the user writing converged fragment data to disk. For each fragment comprising the supermolecule, a separate input file is constructed with the following parameters set:

```
write_denskern T
write_tightbox_ngwfs T
TASK SINGLEPOINT
```

Running these calculations will result in fragment '.dkn' and '.tightbox_ngwfs' files being written to disk. In our example, the three fragment calculation are ran in the directories STAGE1/FRAG01/, STAGE1/FRAG02/, and STAGE1/FRAG03/ using the input filenames 'frag01.dat', 'frag02.dat', and 'frag03.dat' for the respective fragments 1, 2, and 3.

Stage 2

The input file for the second stage is prepared by including the parameters EDA_READ_FRAGS T and TASK EDA_PREP. Running ONETEP with this input file will result in the converged fragment data being loaded in and combined to produce the supermolecule complex data ('.eda', '.dkn' and '.tightbox_ngwfs' files) necessary for stage three. These fragment files' names are set via the EDA_FRAG block, e.g. for a three fragment system:

```
%block EDA_FRAG
  frag1prefix
  frag2prefix
  frag3prefix
%endblock EDA_FRAG
```

where frag1prefix, frag2prefix and frag3prefix are the filename prefixes that will result in loading of the '.dkn' and '.tightbox_ngwfs' files for the three fragments that were converged in stage one. In our example, we assume the second stage calculation is performed in the directory STAGE2/ with the input filename 'stage2.dat'. In this case the block would appear in the input file as,

```
%block EDA_FRAG
  ../STAGE1/FRAG01/frag01
  ../STAGE1/FRAG02/frag02
  ../STAGE1/FRAG03/frag03
%endblock EDA_FRAG
```

Stage 3

The third stage is performed by including the parameters EDA_READ_SUPER T and TASK EDA in the input file. As described earlier for the 1S approach, it is necessary for the user to define the fragment atoms and charges using the EDA_IATM block. On running ONETEP with this input file, the '.dkn', '.tightbox_ngwfs' and '.eda' files for the supermolecule prepared from stage two will be loaded. These files' names are set via the EDA_SUPER block, e.g.

```
%block EDA_SUPER
  supermoleculeprefix
%endblock EDA_SUPER
```

where supermoleculeprefix is the filename prefix that will result in loading of the 'supermoleculeprefix.dkn', 'supermoleculeprefix.tightbox_ngwfs', and 'supermoleculeprefix.eda' files. In our example, where the third stage calculation is performed in the directory STAGE3/ with the input filename 'stage3.dat', the block would appear as,

```
%block EDA_SUPER
  ../STAGE2/stage2
%endblock EDA_SUPER
```

11.1.3 Continuation

Continuation of supermolecule-stage EDA calculations (stage three of the 3S EDA) is controlled using the EDA_CONTINUATION keyword. Continuation files are written to disk within the current working directory when the value of the EDA_WRITE logical keyword is set to true, and continuation files are read by setting the EDA_CONTINUATION logical keyword to true.

It is recommended that the storage of continuation data should be kept separate to the TASK EDA_PREP task calculation data of stage two. This is achieved by performing the stage three calculation in a separate directory to the stage two calculation. This is because the EDA data produced from the TASK EDA_PREP calculation produces a reference to the frozen state which the NGWFs and density kernel will be reset to in a large proportion of EDA calculation cases.

11.1.4 EDA schematic

An example EDA calculation and directory structure is provided in Fig. 11.1.

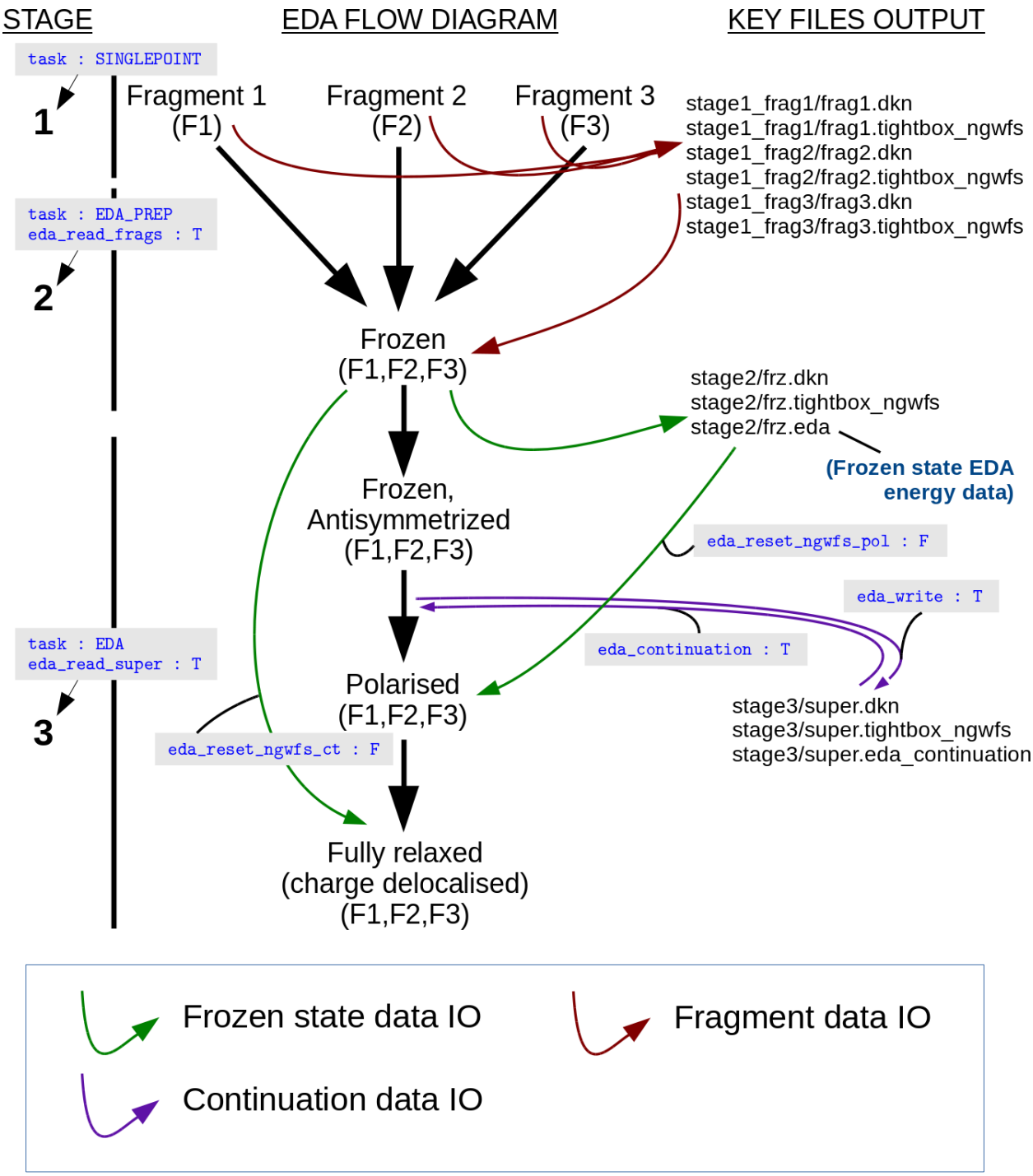

Fig. 11.1: An example 3S EDA calculation (data IO flows are represented by arrows).

11.1.5 Additional functionality

Further details of the EDA functionalities is given below:

Fragment-wise polarisations

Please note that the following functionality is developmental and is subject to change.
It is possible to obtain polarisation energies for each of the fragments by 'freezing' the electronic density during the SCF-MI optimisation. The number of additional polarisation calculations required to perform this is equal to the number of fragments in the system. This functionality is activated using the `EDA_FRAG_ISOL_POL` logical keyword. Also computed is a 'higher order' polarisation contribution that quantifies the difference between the individual fragment polarisation total and the full polarisation energy, i.e.

$$\Delta E_{\text{POL,HO}} = \sum_{A \in \text{X}}^{N_{\text{frag}}} \Delta E_{\text{POL}(A)} - \Delta E_{\text{POL}}$$

where $\Delta E_{\text{POL,HO}}$ is the higher order polarisation contribution, N_{frag} is the number of fragments, $\Delta E_{\text{POL}(A)}$ is the polarisation energy for fragment A that constitutes the supermolecule X, and ΔE_{POL} is the polarisation energy on polarising all fragments simultaneously.

Fragment pair-wise delocalisations

Please note that the following functionality is developmental and is subject to change.
This functionality is activated using the `EDA_FRAG_ISOL_CT` logical keyword.

It is possible to calculate fragment pair delocalisation energies by combining fragments within the SCF-MI optimisation. For example, if we consider a system of interacting $(H_2O)_3$, the delocalisation between any two water molecules is calculated by subtracting the SCF-MI energy of a combined $(H_2O)_2$ 'fragment' interacting with H_2O from the SCF-MI energy of the system with the $(H_2O)_2$ 'fragment' partitioned into its two H_2O constitutents.

Density visualisation

Obtaining electron density (ED) files for visualisation of the EDA frozen, polarisation and fully electronically relaxed states can be done using the `WRITE_DENSITY_PLOT` logical keyword. The output densities are identified by the '_ed_' filename string.

Filename String	Summary
*xxx*_eda_frzidem_ed_density.cube	The ED of the frozen state.
*xxx*_eda_pol_*iii*_ed_density.cube	The ED of fragment *iii* electronically polarised in the field of all other fragments.
*xxx*_eda_pol_ed_density.cube	The ED of the fully electronically polarised state.
*xxx*_eda_relaxed_ed_density.cube	The ED of the fully electronically relaxed state.

Table: The EDA electron density (ED) filename extension descriptions (filename root is denoted by *xxx*).

Visualisation of the density changes during the EDA polarisation and charge transfer processes via electron density difference (EDD) calculations are obtained using the `EDA_DELTADENS` logical keyword. The output densities are identified by the '_edd_' filename string.

Filename String	Summary
*xxx*_pol_*iii*_edd_density.cube	The electronic polarisation EDD of fragment *iii* in the field of all other fragments.
*xxx*_eda_pol_higher_order_edd_density.cube	The higher order electronic polarisation EDD (of the individual fragment-polarised states to the fully polarised state).
*xxx*_eda_pol_edd_density.cube	The fully electronically polarised EDD state.
*xxx*_eda_ct_edd_density.cube	The charge transfer EDD.

Table: The EDA electron density difference filename extension descriptions (filename root is denoted by *xxx*).

Both ED and EDD functionalities are compatible with fragment-specific (see previous section) EDA calculations. Note: electron density differences are currently not computable when the `EDA_CONTINUATION` keyword is set to true. In this case, the edd_cube utility may be used along with the ED cube files produced using the `WRITE_DENSITY_PLOT` logical keyword to calculate EDD files independently of ONETEP (see 'Additional utilities' section).

11.1.6 Parallelisation

Parallelisation requirements

The 1S calculation has a parallelisation strategy with restricted maximum possible number of MPI processes, $N_{\mathrm{proc,max}}$,

$$N_{\mathrm{proc,max}} = \min\left[N_{\mathrm{at}}(\mathrm{A}), N_{\mathrm{at}}(\mathrm{B})\right]$$

where A and B are fragments comprising a supermolecule AB. For example in the case of a water dimer $N_{\mathrm{proc,max}} = 3$.

The 3S EDA allows the user to take full advantage of the parallelisation strategy during the supermolecule stage three, i.e.

$$N_{\mathrm{proc,max}} = \min\left[N_{\mathrm{at}}(\mathrm{A}), N_{\mathrm{at}}(\mathrm{B})\right]|_{\mathrm{S}=1,2}$$
$$N_{\mathrm{proc,max}} = N_{\mathrm{at}}(\mathrm{AB})|_{\mathrm{S}=3}$$

where S is the stage number. For example in the case of a water dimer $N_{\mathrm{proc,max}} = 6$ during the supermolecule stage three (S=3).

ScaLAPACK

The current EDA implementation requires explicit calculation and manipulation of the fragment MO eigenvectors. The ONETEP EDA implementation is compatible with the LAPACK [Anderson1999] and ScaLAPACK [Blackford1997] packages, and has been interfaced to the DSYGVX and PDSYGVX eigensolvers. Compilation with the ScaLAPACK solver is enabled at compilation time using the -DSCALAPACK flag. Use of this package provides significant speed-ups by parallelised computation of the eigenvectors required.

The threshold tolerance to which the eigenvectors are orthogonalised is specified by the `eigensolver_abstol` keyword. It has been observed that the eigensolver may require tighter than the default thresholds. This parameter can be modified in the input, e.g. `eigensolver_abstol = 1.0E-12`.

11.1.7 Additional utilities

utils/edd_cube.F90
This utility calculates the electron density difference between two '.cube' volumetric data files Usage: ./edd_cube cubein1 cubein2 cubeout, where *n*(cubeout) = *n*(cubein1) - *n*(cubein2)

[Phipps2015] Maximillian J. S. Phipps, Thomas Fox, Christofer S. Tautermann, and Chris-Kriton Skylaris. Energy decomposition analysis approaches and their evaluation on prototypical protein-drug interaction patterns. *Chem. Soc. Rev.*, 44:3177–3211, 2015.

[Phipps2016] Maximillian J. S. Phipps, Thomas Fox, Christofer S. Tautermann, and Chris-Kriton Skylaris. Energy decomposition analysis based on absolutely localised molecular orbitals for large-scale density functional theory calculations in drug design. *(Submitted)*, 2016.

[Khaliullin2007] R. Z. Khaliullin, E. A. Cobar, R. C. Lochan, A. T. Bell, and M. Head-Gordon. *J. Phys. Chem. A*, 111:8753–8765, 2007.

[Su2009] Peifeng Su and Hui Li. Energy decomposition analysis of covalent bonds and intermolecular interactions. *J. Chem. Phys.*, 131(1):014102, 2009.

[Anderson1999] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[Blackford1997] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

11.2 Electron Localisation Descriptors

Author
Rebecca J. Clements, University of Southampton

Author
James C. Womack, University of Southampton

Author
Chris-Kriton Skylaris, University of Southampton

Date
July 2019

ONETEP now has the electron localisation descriptors:

- Electron Localisation Function (ELF) [Becke1990]
- Localised Orbital Locator (LOL) [Becke2000]

These descriptors provide a visualisation tool for indicating the location of electron pairs, including bonding and lone pairs, and distinguishing between σ and π bonds. It is a dimensionless quantity in the range of 0 and 1. The plots can be output in $.cube$ or $.dx$ format for visualisation as isosurfaces or volume slices.

The ELF was first introduced by Edgecombe and Becke in 1990 for Hartree-Fock Theory, and later updated for Density Functional Theory by Savin *et al.*. [Becke1990], [Becke2000], [Savin1992] It is based on the Hartree Fock probability of finding two particles of the same spin σ at two different positions of a multielectron system. From this, Edgecombe and Becke obtained the conditional probability of an electron existing in the proximity of a reference electron, which relates to electron localisation. The smaller the probability, the increased likelihood that the reference electron is localised, provided both the electrons are of the same spin. This probability vanishes to zero when the two electrons have the same position, in agreement with the Pauli principle.

This probability is not upper-bounded and so for more convenient graphical interpretation, it is inverted. Localisation is represented at unity. Details are below.

The LOL is similar to the ELF, but with a simpler representation, and produces cleaner plots in some cases. [Schmider2004]

The ELF provides quantum Valence Shell Electron Pair Repulsion (VSEPR) representation of coordination compounds, and the identification of covalent bonding across crystalline solids and surfaces. This provides a useful tool for the main applications of ONETEP; biomolecular simulations, catalysis, and the design of nanostructured materials.

11.2.1 Electron Localisation Function

The starting point to the ELF formula is the exact kinetic energy density for spin σ,

$$\tau_{\sigma}^{exact}(\mathbf{r}) = \sum_{\alpha} \tau(\mathbf{r};\alpha), \tag{11.1}$$

where $\tau(\mathbf{r};\alpha) = (\nabla\psi_{\alpha}(\mathbf{r})) \cdot \left(\nabla \sum_{\beta} K^{\alpha\beta}\psi_{\beta}(\mathbf{r})\right)$

defined in ONETEP [Womack2016] in terms of NGWFs where β are all which overlap with ψ_{α}. Note that the standard $\frac{1}{2}$ coefficient in ONETEP is omitted for the purposes of the ELF, to follow the definition by Becke. The term τ_{σ}^{exact} is extended into a formula to describe electron localisation, a *non-negative* probability density. This will be called the Pauli kinetic energy, D_{σ}. For the individual spin, the Pauli kinetic energy takes the form:

$$D_{\sigma} = \tau_{\sigma}^{exact} - \frac{1}{4}\frac{(\nabla n_{\sigma})^2}{n_{\sigma}}.$$

where $n_{\sigma}(\mathbf{r})$ is the charge density for each spin σ, and $\nabla n_{\sigma}(\mathbf{r})$ is its gradient. D_{σ} is compared with the uniform electron gas as a reference, by taking the ratio:

$$\chi_{\sigma} = \frac{D_{\sigma}}{D_{\sigma}^{0}}.$$

where

$$D_{\sigma}^{0} = \frac{3}{5}\left(6\pi^2\right)^{\frac{2}{3}} n_{\sigma}^{\frac{5}{3}}.$$

The charge density is the local value of $n_{\sigma}\left(\mathbf{r}\right)$ here and the coefficient is the Fermi constant.

Hence, the measure of electron localisation has now become dimensionless. χ_{σ} is then reformulated to avoid the open bounds of the above formula, limiting the ELF to a more desirable finite range of values of 0 to 1 for visual representation:

$$ELF = \frac{1}{1+\chi_{\sigma}^2}$$

11.2.2 Localised Orbital Locator

The LOL is similar to the ELF. Again, the starting point is the exact kinetic energy for spin σ, as in Equation (11.1), with the $\frac{1}{2}$ coefficient omitted.

The uniform electron gas reference, D_{σ}^{0}, is used again here, also known as the local spin density approximation (LSDA):

$$\tau_{\sigma}^{LSDA} = \frac{3}{5}\left(6\pi^2\right)^{\frac{2}{3}} n_{\sigma}^{\frac{5}{3}}.$$

The ratio follows a similar structure to the ELF, except for it is inverted, which again, makes the quantity dimensionless:

$$t_\sigma = \frac{\tau_\sigma^{LSDA}}{\tau_\sigma^{exact}}.$$

The quantity is reformulated to change the infinite range into a 0 to 1 range like before:

$$v_\sigma = \frac{t_\sigma}{1 + t_\sigma}$$

11.2.3 How to use

Keywords

The keywords related to the implementation of the electron localisation descriptors are as follows:

```
Keyword:                Options (default):
eld_calculate           T/F (F)
eld_function            ELF/LOL (ELF)
ke_density_calculate    T/F (F)
do_properties           T/F (F)
cube_format             T/F (T)
dx_format               T/F (F)
```

- Setting $eld\_calculate$ to true turns on the calculation. The calculation will not proceed if this keyword is missing or if it set to false.
- The keyword $eld\_function$ determines which of the ELF or LOL ONETEP is to calculate, by specifying either string. The default here is the ELF, provided the keyword $eld\_calculate$ has been specified.
- As part of this implementation, the kinetic energy density can now also be output, using the logical keyword $ke\_density\_calculate$. This does not automatically output with $eld\_calculate$.
- Electron localisation descriptors and kinetic energy density are available in the formats of $.cube$ or $.dx$ files.
- For spin polarised systems, there will be an ELF output for each spin individually, showing the electron localisation for one of the spins.

In order to use any of the above keywords, ONETEP's properties calculation must be enabled, using $do\_properties$ or setting the task to $properties$, if reading in density results of an energy minimisation calculation. To produce the density plot during the original energy calculation, the input should include:

```
task                singlepoint
write_density_plot  T
```

Example input file

Below is an example input for using the ELF, for the water molecule:

```
task            singlepoint
cutoff_energy   900.0 eV
maxit_ngwf_cg   100
output_detail   verbose
do_properties   T
cube_format     T
```

(continues on next page)

(continued from previous page)

```
dx_format       F
grd_format      F
eld_calculate   T
eld_function    ELF

%block lattice_cart
40.000000000000    0.000000000000    0.000000000000
 0.000000000000   40.000000000000    0.000000000000
 0.000000000000    0.000000000000   40.000000000000
%endblock lattice_cart

%block positions_abs
O 20.000000000  20.000000000  20.000000000
H 18.565580829  18.889354011  20.000000000
H 21.434419171  18.889354011  20.000000000
%endblock positions_abs

%block species
O O 8 4 8.0
H H 1 1 8.0
%endblock species

%block species_pot
O <path to oxygen.recpot>
H <path to hydrogen.recpot>
%endblock species_pot
```

[Becke1990] A. D. Becke and K. E. Edgecombe. A simple measure of electron localization in atomic and molecular systems. *J. Chem. Phys.*, 92(9):5397-5403, 1990.

[Becke2000] A. D. Becke and H. L. Schmider. Chemical content of the kinetic energy density. *Journal of Molecular Structure (Theochem)*, 527:51–61, 2000.

[Savin1992] A. Savin, O. Jepsen, J. Flad, O. K. Andersen, H. Preuss, and H. G. von Schnering. Electron localization in solid-state structures of the elements: the diamond structure. *Angewandte Chemie International Edition in English*, 31(2):187–188, 1992.

[Schmider2000] H. Schmider and A. Becke. Chemical content of the kinetic energy density. *Journal of Molecular Structure (Theochem)*, 527(1):51 – 61, 2000.

[Womack2016] J. C. Womack, N. Mardirossian, M. Head-Gordon, and C.-K. Skylaris. Self-consistent implementation of meta-gga functionals for the ONETEP linear-scaling electronic structure package. *The Journal of Chemical Physics*, 145(20):204114, 2016.

CHAPTER

TWELVE

RELAXATION

12.1 Geometry Relaxation

Author
Loukas Kollias, University of Southampton, United Kingdom

Author
Chris-Kriton Skylaris, University of Southampton, United Kingdom

Date
November 2022

Table of Contents

12.1.1 Introduction to geometry relaxation

Geometry relaxation is a process that moves the positions of atomic nuclei to their equilibrium values in a system. This system can be a chemical structure of higher organisation, e.g., a molecule, a cluster of molecules, or a crystal lattice. It is often called geometry optimisation as it is an optimisation process, where the objective function is the total energy of the system. More elaborately, it is a process that searches for a solution to a minimisation problem, i.e., minimise the total energy that is a function of the atomic coordinates. In place of atomic coordinates, we usually consider the positions of the nuclei of atoms in the system. This is performed in an iterative fashion and a candidate solution is provided at every iteration that is called a "step". At every step, the optimisation algorithm dislocates the positions of the nuclei then it optimizes the electronic structure, and finally it calculates the total energy of the chemical structure. Consequently, a geometry optimisation is a series of single point energy calculations that completes when certain convergence criteria are met. In ONETEP, geometry relaxation (optimisation) is invoked by setting the TASK variable in the input file as,

task : GeometryOptimization

Potential Energy Surface

The potential energies, E, of all possible configurations $(\mathbf{R}_A, \mathbf{R}_B, \ldots, \mathbf{R}_N)$ of the atoms in the system lie on a surface. This is called the potential energy surface (PES), and it is characterised by the equation below,

$$E = E(\mathbf{R}_A, \mathbf{R}_B, \ldots, \mathbf{R}_N) \tag{12.1}$$

The PES is the search domain of the optimisation algorithm, even though most geometry relaxation methods search in a region close to the initial guess structure provided by the user. Each point on the PES is associated with a value of the potential energy and a specific configuration of atoms. The initial guess geometry can be set in various ways, e.g., from experiment or based on a crystal lattice. This structure is now a point on the PES and has an associated energy We need to navigate the PES so that we find points of lower energy, hence more stable configurations compared to the initial one.

Local Optimisation

At the first step, updated atomic positions are calculated using an optimisation method (these are discussed in section 4). The first and second derivatives of the energy, E, with respect to atomic (nuclear) positions, $\mathbf{R}$, are often calculated to estimate the next point. We remind the reader that the first derivative is related to the forces on each atom as a result of its interaction with other atoms in the system. This is a $3N$ matrix, where N is the number of atoms in the system it has the form [Scherlis2006],

$$\mathbf{F}_A = -\frac{dE}{d\mathbf{R}_A} \tag{12.2}$$

where the nuclear coordinates of an atom A that belongs to the system in Cartesian space are: $\mathbf{R}_A = (x_A, y_A, z_A)$. If the net forces on the atoms are zero, then we have an extremum. Then we need to calculate the second derivative to investigate if this is a minimum (stable structure) or a maximum (transition state). The Hessian matrix possesses information about the second derivatives, hence it is frequently called the force constant matrix. This is a $3N \times 3N$ matrix, and it can be written in the following form for any atoms A, B in the system,

$$\mathbf{H}_{AB} = \frac{\partial^2 E}{\partial \mathbf{R}_A \partial \mathbf{R}_B} \tag{12.3}$$

This calculation is important to examine if this point is a minimum. To do this, vibrational frequencies are calculated from the Hessian [Schlegel2011]. A minimum is characterised by a net zero force and positive vibrational frequencies. In simple terms, the minimum is located where the first derivative is zero and the second derivative (that is relevant to the local curvature) is positive. Consequently, one needs to calculate the energy and its first and second derivatives to investigate if a point on the PES is a minimum.

By default, the positions of all nuclei are modified during optimisation. Nevertheless, we can exclude a subset of the nuclei of the system from the optimisation. This way we can keep the positions of certain nuclei constant, if needed, even when positions of other nuclei are changed.

12.1.2 Convergence Criteria

There are a few criteria that can be examined to assess whether the geometry relaxation has converged to a local minimum structure. These include criteria concerning the energy and the net forces on atoms in the system. Additionally, the geometry relaxation can be stopped if a certain number of steps has been made regardless of the energy and forces in the final structure. This is not really a convergence criterion but it is important to be able to stop the process for a number of reasons which are discussed later in this section.

Force tolerance

Atomic positions need to be relaxed as the net force on each atom should be zero. Practically, it is difficult, if at all possible, to have a net zero force on every atom. Consequently, this process minimizes the magnitude of the net force on each atom so that it approaches zero as much as possible, and within a certain tolerance. The default force tolerance is 0.002 $E_H\ a_0^{-1}$. This criterion can be modified by the user through setting the following parameter in the input file,

geom_force_tol : [VALUE] [UNITS]
e.g.,
geom_force_tol : 0.051422 eV/ang
or
geom_force_tol : 0.001 Ha/bohr
or
geom_force_tol : 0.001

if units are not specified, they are thought to be $E_H a_0^{-1}$.

Maximum number of steps

Another criterion that can be assessed to stop the relaxation process is the maximum number of steps to be made. The user can have both criteria, so if any of those is satisfied after a certain step, then the process is terminated. The maximum number of steps does not guarantee a minimum energy structure, but it is a good check to stop the optimisation if it does not descend the PES in a satisfactory way or if the time of the calculation is limited. In any case, it will provide us with a candidate structure after a certain number of steps given an initial guess and a specific optimisation algorithm. This criterion can be controlled by the user through setting the following parameter in the input file,

geom_max_iter : [VALUE]
e.g.,
geom_max_iter : 100

if this is not specified by the user in the input file, then its default value is 50. This means that 50 geometry relaxation steps will be performed at most. Fewer steps will be performed if other criteria, such as force, energy tolerance, are satisfied before that.

Convergence window

The convergence window is the number of steps where all the criteria set for the geometry relaxation must be satisfied to declare convergence. The default value is 2 steps. This is recommended to achieve a balance between accuracy and efficiency. This criterion can be controlled by the user through setting the following parameter in the input file,

geom_convergence_win : [VALUE]
e.g.,
geom_convergence_win : 3

Energy tolerance

Another convergence criterion that can be specified by the user is the energy tolerance. This tells the geometry relaxation to stop if the difference between the maximum and minimum energies per atom over geom_convergence_win steps is less than this value. The default value is $10^{-6} E_H$. This criterion can be controlled by the user through setting the following parameter in the input file,

geom_energy_tol : [VALUE] [UNITS]
e.g.,
geom_energy_tol : 1e − 7 Ha
or
geom_energy_tol : 2.72e − 6 eV

Displacement tolerance

Another way to check if the geometry relaxation has converged is to assess the maximum displacement of the positions of the nuclei. In practice, we calculate the set of distances of nuclei between two consecutive steps. If the largest value in this set is equal to or less than the tolerance, then the maximum distance that an atom has been displaced is considered small enough to tell the optimisation to stop. In other words, atoms have not moved considerably between two consecutive steps, hence the relaxation has converged. The default value is $0.005\ a_0$. If units are not defined, then these are thought to be a_0. Units "ang" correspond to Å. This criterion can be controlled by the user through setting the following parameter in the input file,

geom_disp_tol : [VALUE] [UNITS]
e.g.,
geom_disp_tol : 0.002 bohr
or
geom_disp_tol : 0.001 ang

12.1.3 Options

Output detail

The output detail of the geometry relaxation process can be controlled by the user. The default behaviour is to follow the output_detail variable that controls the overall output detail of the ONETEP calculation. The available settings for this variable are BRIEF, NORMAL, VERBOSE, PROLIX, and MAXIMUM. The level of the output detail relevant to geometry relaxation can be controlled by setting the following parameter in the input file,

geom_output_detail : [VALUE]
e.g.,
geom_output_detail : VERBOSE

Continuation

The user can select whether to continue from a previous geometry relaxation or start from scratch. The default value of this logical variable is FALSE that means geometry relaxations start from scratch. This option can be controlled by the user through setting the following parameter in the input file,

geom_continuation : [VALUE]
e.g.,
geom_continuation : FALSE

Steps between backups

The user can control the number of geometry optimisation steps between backups of all data for continuation. In other words, this tells the program to write a backup for continuation every geom_backup_iter steps. The default value of this variable is 1, so ONETEP writes a backup after every step of the geometry relaxation. This value can be increased to save time on writing continuation data. This option can be controlled by the user through setting the following parameter in the input file,

geom_backup_iter : [VALUE]
e.g.,
geom_backup_iter : 2

Density Kernel and NGWFs

Steps between reset of density kernel and NGWFs

The user can control the stride with which the density kernel and the Nonorthogonal Generalized Wannier Functions (NGWFs) are being reset. In other words, this tells the program to reset the kernel and NGWFs every geom_reset_dk_ngwfs_iter steps. The default value of this variable is 6, so ONETEP resets these quantities every 6 geometry relaxation steps. Resetting the density kernel and NGWFs every once in a while can prevent problems in converging the energy of the NGWFs during optimisation. This option can be controlled by the user through setting the following parameter in the input file,

geom_reset_dk_ngwfs_iter : [VALUE]
e.g.,
geom_reset_dk_ngwfs_iter : 10

Reuse density kernel and NGWFs

The user can control whether to re-use an existing density kernel and set of NGWFs. The default value of this logical variable is TRUE except from the case of a Density-Functional based Tight-Binding (DFTB) calculation, where it is FALSE by default. This option can be controlled by the user through setting the following parameter in the input file,

geom_reuse_dk_ngwfs : [VALUE]
e.g.,
geom_reuse_dk_ngwfs : FALSE

12.1.4 Geometry Relaxation methods

There is a variety of optimisation algorithms implemented in ONETEP. The user can select which algorithm to use by setting the following parameter in the input file,

geom_method : [VALUE]

e.g.,

geom_method : TPSD

These methods are being discussed in this section.

BFGS

The Broyden-Fletcher-Goldfard-Shanno (BFGS) algorithm is a very popular quasi-Newton optimisation method that is implemented in various computational chemistry codes, including ONETEP. In ONETEP, the BFGS algorithm follows the implementation by Pfrommer et al. [Pfrommer1997]. This method uses an initial Hessian matrix that is updated in an iterative fashion. The domain of this Hessian includes both internal and cell degrees of freedom, hence both nuclear coordinates, $\mathbf{R}$, and lattice vectors, $\mathbf{h}$, are relaxed. The following notation is used for the lattice vectors,

$$\mathbf{h} = [UVW] \tag{12.4}$$

in more detail, if the unit vectors along the x, y, and z directions in Cartesian space are $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$, respectively, then

$$\mathbf{h} = U\hat{\mathbf{i}} + V\hat{\mathbf{j}} + W\hat{\mathbf{k}} \tag{12.5}$$

The BFGS algorithm uses information from both forces and the Hessian matrix to investigate a minimum. The latter is computationally expensive as it is $O(N^2)$ [Aarons]. The Hessian should be an operator that transforms between the changes in nuclear positions and forces,

$$\mathbf{H}_{n+1}\Delta\mathbf{R}_n = \Delta\mathbf{F}_n \tag{12.6}$$

where $\Delta\mathbf{R}_\mathrm{n}$ and $\Delta\mathbf{F}_\mathrm{n}$ are the changes in positions and forces between steps $n-1$ and n, respectively.

Then, the algorithm updates its guess solution based on,

$$\mathbf{H}_{n+1}^{-1} = (\mathbf{I} - \mathbf{A})\mathbf{H}_n^{-1}(\mathbf{I} - \mathbf{A}) + \mathbf{A} \tag{12.7}$$

where,

$$\mathbf{A} = \rho_n \Delta\mathbf{F}_n (\Delta\mathbf{R}_n)^T \tag{12.8}$$

$$\rho_n = \frac{1}{(\Delta\mathbf{F}_n)^T \Delta\mathbf{R}_n} \tag{12.9}$$

Therefore, the BFGS algorithm updates the inverse Hessian matrix at every step, as it is more computationally efficient than updating the Hessian and then inverting it at every step. At last, the BFGS algorithm can be selected by setting the geom_method variable to BFGS in the ONETEP input file.

Initial Inverse Hessian Matrix

In ONETEP, the initial inverse Hessian matrix is set up as,

$$H_0^{-1} = \begin{bmatrix} (3\Omega B_0)^{-1} & 0 & \dots & & & 0 \\ 0 & \ddots & 0 & \dots & & \vdots \\ \vdots & 0 & (3\Omega B_0)^{-1} & 0 & \dots & \\ & \vdots & 0 & g_0^{-1}\langle m_{ionic}^{-1}\rangle\omega_0^{-2} & 0 & \\ & & \vdots & 0 & \ddots & 0 \\ 0 & \dots & & & 0 & g_0^{-1}\langle m_{ionic}^{-1}\rangle\omega_0^{-2} \end{bmatrix} \tag{12.10}$$

$$\Omega = det(\mathbf{h}) \tag{12.11}$$

$$\mathbf{h} = (1+\epsilon)\mathbf{h_0} \tag{12.12}$$

$$\mathbf{g_0} = \mathbf{h_0}^T\mathbf{h_0} \tag{12.13}$$

where B_0 is the bulk modulus, Ω is the cell volume, $\mathbf{g_0}$ is the $3N \times 3N$ metric tensor of the initial configuration, $\langle \mathrm{m_{ionic}} \rangle$ is the average ionic mass, ω_0^{-2} is the average phonon frequency at the Γ point, and ϵ is the finite strain tensor. This is a block-diagonal matrix. The upper left (9×9) diagonal part describes cell-cell interactions, while the bottom right $(3N \times 3N)$ diagonal part describes ion-ion interactions. This way, we can calculate elastic properties and phonon frequencies at the Γ-Point [Pfrommer1997] [Aarons].

Line Search

So far we have learnt how to calculate the energy, its gradient and its Hessian with respect to nuclear positions. Now we are interested in how to decide where the next point for which these quantities are calculated should be on the PES. In simple terms, we need to find out where we should look after calculating the energy of a point and the local curvature of its region. We identify the search direction, p_n, after every geometry relaxation step using the following expression,

$$\mathbf{p}_n = -\mathbf{H}_n^{-1}\nabla E(\mathbf{R}_n) \tag{12.14}$$

hence the next point is calculated using the search direction and an arbitrarily chosen step length, λ_n, as follows,

$$\mathbf{R}_{n+1} = \mathbf{R}_n + \lambda_n \mathbf{p}_n, \qquad \lambda_n > 0 \tag{12.15}$$

Conditions for step length

The step length, λ_n, should be determined following Wolfe's conditions [Wolfe1969] [Wolfe1971]. In ONETEP, weak Wolfe-Powell conditions are used. These are described in detail by Gilbert [Gilbert1997] and Yuan et al. [Yuan2017]. The following conditions should be satisfied for the next step, $\mathbf{R}_{n+1}$,

$$E(\mathbf{R}_{n+1}) \leq E(\mathbf{R}_n) + \omega_1 \lambda_n \nabla E(\mathbf{R}_n)^T \mathbf{p}_n \tag{12.16}$$

$$\nabla E(\mathbf{R}_{n+1})^T \mathbf{p}_n \geq \omega_2 \nabla E(\mathbf{R}_n)^T \mathbf{p}_n \tag{12.17}$$

$$0 < \omega_1 < 0.5 \tag{12.18}$$

$$\omega_1 < \omega_2 < 1 \tag{12.19}$$

where ω_1 and ω_2 are constants independent of the current step. These conditions lead to a descending search direction on the PES [Gilbert1997] [Yuan2017]. They also help with the convergence of the optimisation algorithm [Gilbert1997] [Yuan2017].

Limited-Memory BFGS (L-BFGS)

The limited-memory BFGS (L-BFGS) method can be useful when we have large systems, hence the computational cost of calculating the Hessian matrix is very high. In L-BFGS, the Hessian, that is a $(3\mathrm{N}+9) \times (3\mathrm{N}+9)$ matrix, where N is the number of atoms in the system, is not stored in full. A set of vectors of length N is stored instead, hence the optimisation converges at a linear rate. An approximate Hessian matrix is constructed based on the matrix that was used in the last few steps. This matrix should be sparse, symmetric, and positive-definite [Liu1989]. As in the standard BFGS algorithm, the inverse Hessian matrix is updated at each optimisation step. In L-BFGS, we do not need to store the Hessian but we need to access information about the positions and gradients at previous steps [Packwood2016]. We choose to store a certain number of sets of positions and gradients, namely m, and every time a new optimisation step is made, the oldest set is removed from storage to make space for the latest set to be stored. The number of sets, m

is specified by the user. By default, ONETEP uses m equal to 30. We declare the difference in positions and gradients between steps $n-1$ and n as $\mathrm{s_n}$ and $\mathrm{y_n}$. Then, we have the following expressions [Aarons],

$$\mathbf{s}_n = \mathbf{R}_n - \mathbf{R}_{n-1} \tag{12.20}$$

$$\mathbf{y}_n = \nabla E(\mathbf{R}_n) - \nabla E(\mathbf{R}_{n-1}) \tag{12.21}$$

$$\rho_n = \frac{1}{\mathbf{y}_n^T \mathbf{s}_n} \tag{12.22}$$

$$\mathbf{S}_n = \left[\mathbf{s}_0, ..., \mathbf{s}_{n-1}\right] \tag{12.23}$$

$$\mathbf{Y}_n = \left[\mathbf{y}_0, ..., \mathbf{y}_{n-1}\right] \tag{12.24}$$

An initial approximation of the inverse Hessian matrix, H_0^{-1}, is used at the first iteration of the algorithm. In contrast with the standard BFGS, the initial matrix approximation can be different at each step [Liu1989]. The inverse Hessian can be updated using [Byrd1994],

$$\mathbf{H}_n^{-1} = \mathbf{H}_0^{-1} + \begin{bmatrix}\mathbf{S}_n & \mathbf{H}_0^{-1}\mathbf{Y}_n\end{bmatrix} \mathbf{W}_n \begin{bmatrix}\mathbf{S}_n^T \\ \mathbf{Y}_n^T\mathbf{H}_0^{-1}\end{bmatrix} \tag{12.25}$$

$$\mathbf{W}_n = \begin{bmatrix}\mathbf{R}_n^{-T}(\mathbf{D}_n + \mathbf{Y}_n^T\mathbf{H}_0^{-1}\mathbf{Y}_n)\mathbf{R}_n^{-T} & -\mathbf{R}_n^{-T} \\ -\mathbf{R}_n^{-T} & 0\end{bmatrix} \tag{12.26}$$

$$\mathbf{D}_n = \begin{bmatrix}\mathbf{s}_0^T\mathbf{y}_0 & 0 & \dots & 0 \\ 0 & \mathbf{s}_1^T\mathbf{y}_1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{s}_n^T\mathbf{y}_n\end{bmatrix} \tag{12.27}$$

In BFGS, $\mathbf{H}_n^{-1}$ is multiplied by vector $\mathbf{v}$ and gives a product vector, $\mathbf{p}$.

$$\mathbf{p} \rightarrow \mathbf{H}_n^{-1}\mathbf{v} \tag{12.28}$$

but in L-BFGS we do not compute $\mathbf{H}_n^{-1}$. Consequently, we need to find a way to obtain vector $\mathbf{p}$. In ONETEP, this is achieved by using the Basic Linear Algebra Subprogram (BLAS) libraries to perform matrix operations in the following fashion as explained in detail in [Aarons],

$$\mathbf{p} = \mathbf{H}_0^{-1}\mathbf{v} \tag{12.29}$$

$$\mathbf{w}_{1:m} = \mathbf{Y}_n^T\mathbf{p} \tag{12.30}$$

$$\mathbf{w}_{m+1:2m} = \mathbf{S}_n^T\mathbf{v} \tag{12.31}$$

$$\mathbf{w}_{1:m} \rightarrow \mathbf{R}_n^{-1}\mathbf{Y}_n^T\mathbf{H}_0^{-1}\mathbf{v} \tag{12.32}$$

$$\mathbf{w}_{m+1:2m} \rightarrow \mathbf{R}_n^{-1}\mathbf{S}_n^T\mathbf{v} \tag{12.33}$$

$$\mathbf{\Xi} = \mathbf{T}_n \tag{12.34}$$

$$\mathbf{\Xi} \rightarrow \mathbf{R}_n^{-1}\mathbf{\Xi} \tag{12.35}$$

$$\mathbf{w}_{1:m} \rightarrow \mathbf{w}_{1:m} - \mathbf{\Xi}\mathbf{w}_{m+1:2m} \tag{12.36}$$

$$\mathbf{p} \rightarrow \mathbf{p} + \mathbf{S}_n^T\mathbf{w}_{1:m} \tag{12.37}$$

$$\mathbf{t} = -\mathbf{Y}_n^T\mathbf{w}_{m+1:2m} \tag{12.38}$$

$$\mathbf{p} \rightarrow \mathbf{p} + \mathbf{H}_0^{-1}\mathbf{t} \tag{12.39}$$

where, $\mathbf{w}$, $\mathbf{t}$ are vectors of lengths $2\mathrm{m}$ and n, respectively. $\mathbf{\Xi}$ is a $\mathrm{m} \times \mathrm{m}$ matrix. The L-BFGS algorithm uses this procedure until convergence.

Options

The following variables can be set when the L-BFGS algorithm is selected. The user is advised to follow the default behaviour here. Nevertheless, there is an option to modify these settings as explained in the following text.

- L-BFGS maximum updates
  This variable determines the number of vectors updated in L-BFGS. The default value is 30. This means that 30 vectors will be updated at each iteration. This variable can be controlled by the user through setting the following parameter in the input file,

  lbfgs_max_updates : [VALUE]
  e.g.,
  lbfgs_max_updates : 30

- L-BFGS block length
  This variable determines the number of updates that are stored in an unbounded L-BFGS calculation before reallocation. The default value is 30. This means that 30 updates will be stored. This variable can be controlled by the user through setting the following parameter in the input file,

  lbfgs_block_length : [VALUE]
  e.g.,
  lbfgs_block_length : 30

- Estimated bulk modulus
  The user can provide an estimate of the bulk modulus. The default value of this variable is 0.017 $E_H\ a_0^{-3}$. This is one of the parameters used to initialize the inverse Hessian matrix. This option can be controlled by the user through setting the following parameter in the input file,

  geom_modulus_est : [VALUE] [UNITS]
  e.g.,
  geom_modulus_est : 0.02 Ha / bohr ∗ ∗ 3

- Estimated average phonon frequency
  The user can provide an estimate of the average phonon frequency at the Γ-point. The default value of this variable is 0.0076 E_H. This is one of the parameters used to initialize the inverse Hessian matrix. This option can be controlled by the user through setting the following parameter in the input file,

  geom_frequency_est : [VALUE] [UNITS]
  e.g.,
  geom_frequency_est : 0.008 Ha
  or
  geom_frequency_est : 0.218 eV

- Print inverse Hessian matrix

The user can choose whether to print the inverse Hessian matrix. The default value of this variable is FALSE, so this matrix is not printed unless the user specifies otherwise. This option can be controlled by the user through setting the following parameter in the input file,

geom_print_inv_hessian : [VALUE]
e.g.,
geom_print_inv_hessian : FALSE

Also, certain preconditioners can be used in the L-BFGS algorithm. These are the exponential (EXP) [Packwood2016] and forcefield (FF)-based [Mones2018] preconditioners. It should be noted that no preconditioner is used by default. Preconditioner options are preset to default values, but the user has the option to change these, if needed. These are r_{cut}, r_{NN}, A, μ, and they are used in the following expressions to define the preconditioner P options. Either can be selected by setting the following variable in the input file,

geom_precond_type : [VALUE]
e.g.,
geom_precond_type : EXP

The preconditioners can be modified (if needed) by setting the appropriate variables,

geom_precond_exp_c_stab : [VALUE]
e.g.,
geom_precond_exp_c_stab : 0.1

geom_precond_exp_a : [VALUE]
e.g.,
geom_precond_exp_a_stab : 3.0

geom_precond_exp_r_nn : [VALUE] [UNITS]
e.g.,
geom_precond_exp_r_nn : 0.1 bohr

geom_precond_exp_r_cut : [VALUE][UNITS]
e.g.,
geom_precond_exp_r_cut : 0.1 bohr

geom_precond_exp_mu : [VALUE][UNITS]
e.g.,
geom_precond_exp_mu : 0.1 Ha / bohr ∗ ∗ 2

geom_precond_ff_c_stab : [VALUE][UNITS]
e.g.,
geom_precond_ff_c_stab : 0.1 Ha / bohr ∗ ∗ 2

geom_precond_ff_r_cut : [VALUE][UNITS]
e.g.,
geom_precond_ff_r_cut : 3.8 bohr

At last, the L-BFGS algorithm can be selected by setting the geom_method variable to LBFGS in the ONETEP input file.

Coordinate system

The user can choose if Cartesian or internal (delocalised) coordinates will be used during geometry relaxation. The latter are generated based on the implementation by Andzelm et al. [Andzelm2001]. Cartesian coordinates are used in all geometry relaxation methods by default. The user can select which system of coordinates to use by setting the geom_method variable to either CARTESIAN or DELOCALIZED. For the latter, only the BFGS method is used. The use of delocalised coordinates can improve performance of the geometry relaxation. When Cartesian coordinates are selected by setting the geom_method variable in the input file, the user can choose whether to do a BFGS or L − BFGS optimisation by setting the parameter to FALSE or TRUE, respectively. This is shown below,

geom_lbfgs : [VALUE]
e.g.,
geom_lbfgs : TRUE

Two-point Steepest Descents (TPSD)

The two-point steepest descents (TPSD) optimisation algorithm is implemented in ONETEP. This is based on the work of Barzilai and Borwein [Barzilai1988]. The update equation can be obtained in the same way as in the BFGS algorithm. The search direction, p_n, can be calculated as,

$$p_n = -\nabla E(\mathbf{R}_n) \tag{12.40}$$

In this method, we need to calculate the change in atomic positions, $\Delta\mathbf{R}$, and the gradient of the energy with respect to atomic position, $\Delta(\nabla E(\mathbf{R}))$, between the current iteration, i, and the previous one, $i-1$. Please note that in what follows in this section, we will just rename the quantity $\nabla E(\mathbf{R}_i)$ to $\mathbf{g}_i$ for simplicity.

$$\Delta\mathbf{R} = \mathbf{R}_i - \mathbf{R}_{i-1} \tag{12.41}$$

$$\Delta\mathbf{g} = \mathbf{g}_i - \mathbf{g}_{i-1} \tag{12.42}$$

consequently, the step length, λ_n, is calculated either as,

$$\lambda_n = \frac{\Delta\mathbf{R}\cdot\Delta\mathbf{g}}{\Delta\mathbf{g}\cdot\Delta\mathbf{g}} \tag{12.43}$$

this way we minimise the quantity $\|\Delta\mathbf{R} - \lambda\Delta\mathbf{g}\|^2$ with respect to λ. In ONETEP, this equation is used to calculate the step length, λ_n, by default. Symmetrically, we can use,

$$\lambda_n = \frac{\Delta\mathbf{R}\cdot\Delta\mathbf{R}}{\Delta\mathbf{R}\cdot\Delta\mathbf{g}} \tag{12.44}$$

to minimise the quantity $\|\lambda\Delta\mathbf{R} - \Delta\mathbf{g}\|^2$ with respect to λ.

At last, the TPSD algorithm can be selected by setting the geom_method variable to TPSD in the ONETEP input file.

ASE Optimizers

In addition to the optimisation algorithms implemented in ONETEP, all optimisation algorithms available in the Atomic Simulation Environment (ASE) [ase_optimizers_website] can be used as well. This can be done through the interface between ONETEP and ASE [onetep_ase_interface]. This way, ONETEP works as the calculator that ASE uses to compute the energy and forces of a given system at each step of the relaxation process. Then, the propagation of the geometry relaxation process, i.e., the search direction, step length, and propagation formula, is handled by the optimisation algorithm selected in ASE.

12.1.5 Using the interface between ONETEP and the Atomic Simulation Environment (ASE)

Requirements

Before using the ONETEP / ASE interface, we need to make sure that we have installed python (preferably version 3 as the following code is written using python3 syntax). Then, we should have installed the following python packages: ASE (Atomic Simulation Environment), and OS (Operating System). The OS package lets us execute ONETEP as well as read and write ONETEP files. A couple of important packages that is highly recommended to be installed are NumPy (Numerical Python) and SciPy (Scientific Python). These handle advanced numerical (e.g., linear algebra, powers, etc.) and scientific calculations (e.g., integration, interpolation, etc.), respectively. The ASE package allows for the manipulation of atomic/molecular systems. An interface between ONETEP and ASE is already in place and it lets us call ONETEP from ASE, make calculations in ONETEP, and communicate back with ASE. The advantages of using ASE with ONETEP are numerous. A number of these advantages is given in the following list,

- Optimisers built in ASE can be used to drive geometry relaxation in addition to the ones available in ONETEP.
- Molecular systems can be manipulated, e.g., translated and rotated inside the unit cell.
- Various python packages focused on molecular systems can be used in conjunction with the ASE/ONETEP interface. For example, packages for global optimisation can be used to find the global minimum on the PES.
- We can easily convert between popular molecular input/output files. For example, we can convert a .cif file that contains crystallographic data to an .xyz atomic coordinate file that can be used in ONETEP.
- ASE can be used as a standalone visualisation package. For example, it can be used to visualise a trajectory of atomic positions.

The aforementioned packages can be installed using

python3 -m pip install *package_name*

where one should replace *package_name* with the name of the *package* to be installed. For example,

python3 -m pip install ase

In our case the names of the packages are,

- os
- ase

and optionally,

- numpy
- scipy

ONETEP / ASE interface using intrinsic optimisation algorithms

In this example, we see how to use the ONETEP / ASE interface while opting for an optimisation algorithm that is already implemented in ONETEP. The following code can be written (saved) to a file called input_intrinsic_algorithm.py. Then, we could run our ONETEP calculation by just running this python script as follows,

python3 input_intrinsic_algorithm.py > output_intrinsic_algorithm.log

where we redirect the output to a file called output_intrinsic_algorithm.log.

This way, ASE creates an input file for ONETEP, then ONETEP performs the geometry relaxation, and it communicates the energy and forces back to ASE for post-processing and visualisation.

```
from os import environ, path
from ase.build import molecule
from ase.calculators.onetep import Onetep
from ase.io import read,iread,write

# set ONETEP run command by providing the full
# path to the ONETEP executable. Here we also
# set the number of MPI ranks and threads per rank
# based on what resources we have asked for.
environ["ASE_ONETEP_COMMAND"]="export OMP_NUM_THREADS=$SLURM_THREADS_PER_CORE; srun -n
↪$SLURM_NPROCS /path/to/onetep/bin/onetep PREFIX.dat >> PREFIX.out 2> PREFIX.err"

# read atomic positions from an .xyz file
# that is in the same directory.
mol = read('./mymolecule.xyz')

# set lattice vectors (in Angstrom)
# replace xx, xy, xz, yx, yy, yz,
# zx, zy, zz with desired values
# for the lattice vectors
mol.set_cell([[xx,xy,xz],
              [yx,yy,yz],
              [zx,zy,zz]])

# set calculator as Onetep and a
# label for input and output files.
calc_label = 'mylabel'
calc = Onetep(label=calc_label)

# set Onetep parameters (left-hand side) and
# corresponding values (right-hand side).
# change atom name "A" and any other
# value you want in the code below.
calc.set(pseudo_path='/path/to/pseudo/',
         pseudo_suffix='.recpot',
         task='GeometryOptimization',
         geom_method='TPSD',
         geom_force_tol='0.05 "ev/ang"',
         write_forces=True,
         write_xyz=True,
         xc='PBE',
         cutoff_energy='1200 eV',
         species_ngwf_number={"A":10},
         species_ngwf_radius={"A":12.0}, # [bohr]
         write_hamiltonian=True,
         write_tightbox_ngwfs=True,
         write_denskern=True,
         output_detail='verbose',
         )

# perform the calculation
mol.calc = calc
```

(continues on next page)

(continued from previous page)

```
# get atomic positions, energy, and forces from the
# calculator. In our case, the calculator is ONETEP.
mol.get_positions()
mol.get_forces()
mol.get_potential_energy()
```

ONETEP / ASE interface using ASE optimisation algorithms

In this example, we see how to use the ONETEP / ASE interface while opting for an optimisation algorithm that is implemented in ASE. The following code can be written (saved) to a file called input_ase_algorithm.py. Then we could run our ONETEP calculation by just running this python script as follows,

python3 input_ase_algorithm.py > output_ase_algorithm.log

where we redirect any output to a file called output_ase_algorithm.log.

This way, ASE creates an input file for ONETEP, then ONETEP calculates the energy and forces, and it communicates the energy and forces back to ASE, then ASE updates the structure until the desired convergence threshold has been reached. Final structures, energy, and forces are available in ASE for post-processing and visualisation.

```
from os import environ, path
from ase.build import molecule
from ase.calculators.onetep import Onetep
from ase.optimize.sciopt import SciPyFminBFGS
from ase.io import read,iread,write

# set ONETEP run command by providing the full
# path to the ONETEP executable. Here we also
# set the number of MPI ranks and threads per rank.
# based on what resources we have asked for.
environ["ASE_ONETEP_COMMAND"]="export OMP_NUM_THREADS=$SLURM_THREADS_PER_CORE; srun -n
↪$SLURM_NPROCS /path/to/onetep/bin/onetep PREFIX.dat >> PREFIX.out 2> PREFIX.err"

# read atomic positions from an .xyz file
# that is in the same directory.
mol = read('./mymolecule.xyz')

# set lattice vectors (in Angstrom)
# replace xx, xy, xz, yx, yy, yz,
# zx, zy, zz with desired values
# for the lattice vectors.
# For orthogonal cells, you can set the cell as
# mol.set_cell([xx,yy,zz])
mol.set_cell([[xx,xy,xz],
              [yx,yy,yz],
              [zx,zy,zz]])

# set calculator as Onetep and a
# label for input and output files.
calc_label = 'mylabel'
calc = Onetep(label=calc_label)
```

(continues on next page)

(continued from previous page)

```
# set ONETEP parameters (left-hand side) and
# corresponding values (right-hand side).
# change atom name "A" and any other
# value you want in the code below.
calc.set(pseudo_path='/path/to/pseudo/potential/',
         pseudo_suffix='.recpot',
         task='SinglePoint',
         xc='PBE',
         cutoff_energy='1200 eV',
         species_ngwf_number={"A": 10},
         species_ngwf_radius={"A": 12.0}, # [bohr]
         write_hamiltonian=True,
         write_tightbox_ngwfs=True,
         write_denskern=True,
         output_detail='verbose',
         )

# these files need to be read to
# update the relaxation process.
if path.isfile(calc_label + '.dkn') == True:
   calc.set(read_denskern=True)
if path.isfile(calc_label + '.ham') == True:
   calc.set(read_hamiltonian=True)
if path.isfile(calc_label + '.tightbox_ngwfs') == True:
   calc.set(read_tightbox_ngwfs=True)

# perform the calculation
mol.calc = calc

# get energy and forces from the calculator
# in our case, the calculator is ONETEP.
mol.get_forces()
mol.get_potential_energy()

# select optimisation algorithm and relevant options.
# Here, we select the SciPyFminBFGS algorithm and
# we write the trajectory to a .traj file.
opt = SciPyFminBFGS(mol, trajectory='opt.traj')

# optimise structure until the maximum force on any
# atom is equal or less than 0.05 (units are eV/A).
opt.run(fmax=0.05)

# convert ASE trajectory (.traj) to .xyz format
trj = iread("opt.traj")
trj_xyz = write("opt_traj.xyz", trj, format="xyz")
```

12.1.6 Tutorials

In this section, a series of tutorials is demonstrated. Please create a separate folder (directory in Unix-based systems, such as Linux distributions) for each one of the tutorials, and put all files of each tutorial in the corresponding folder. We will learn how to create a guess geometry for a certain system, e.g., a set of atoms, a molecule, a molecular cluster, and then relax it using ONETEP. These examples include an ethanol molecule and a platinum nanoparticle. We note that the focus of these tutorials is to learn how to use optimisation algorithms in ONETEP, and these can be applied to much more complicated systems following the same rationale as in these tutorials.

Ethanol

At first, we are going to relax the geometry of a simple organic molecule that is ethanol (C_2H_5OH). A guess geometry can be provided either directly in the ONETEP input file or through an XYZ file that can be loaded in ONETEP. Please create a folder named however you prefer, e.g., ethanol_example and put all files in that folder. In this example, we are going to use the Avogadro software [Avogadro] [Hanwell2012] to create an ethanol molecule. In Avogadro, select the pencil icon from the toolbar at the top. Then, select Carbon from the "Draw" toolbar by clicking on the box right next to "Element" on the left hand side. Click and drag somewhere on the black canvas so that a C-C single bond is created and Avogadro will automatically adjust hydrogen atoms to make it a CH_3CH_3 molecule. Select Oxygen in the same way as carbon was previously selected. Click on a carbon atom on the black canvas and drag to make a C-O bond (again, hydrogen atoms will be adjusted automatically). Use the auto-optimize tool that is on the same toolbar as the pencil icon (it has an E sign and a down-pointing arrow below E). This will do a rough relaxation to provide a reasonable initial guess geometry. At last, click on "File" that is the left-most option on the top toolbar and click either on "save as" or hover the mouse over export and then click on "molecule". A new window will appear, where the coordinates can be saved in an XYZ file. Click on "All files" at the bottom right and select "XYZ", then select a name for the file while including the .xyz extension (for example, ethanol.xyz). Exit Avogadro and move this XYZ file to the directory where the ONETEP simulation will be performed. The XYZ format is a readable text file which can be viewed and modified with any text editor. This contains the Cartesian coordinates of all the atoms in the system. An example ethanol.xyz is provided below,

```
9
XYZ file generated by Avogadro.
C     -4.63004     0.41911     0.07151
C     -3.76666     1.67624     0.06260
H     -4.34884    -0.23171     0.89807
O     -2.40610     1.40089    -0.17922
H     -2.09824     0.76731     0.47816
H     -5.67688     0.68678     0.18150
H     -4.50180    -0.12684    -0.85976
H     -4.07361     2.34155    -0.74747
H     -3.88245     2.21413     1.01544
```

Now that we have our molecule, we will relax its geometry in ONETEP.

ONETEP built-in optimizer

First, we are going to use optimisation algorithms which are built-in ONETEP. In this example, we choose the two-point steepest descent, TPSD, algorithm. An ONETEP input file should have at least the following sections,

- general parameters
- lattice vectors block
- species block
- potential block
- positions block
- etc.

An example input script, ethanol_geo_opt.dat, is provided below,

```
# ethanol_geo_opt.dat
# general parameters
# kinetic energy cutoff
cutoff_energy : 1000 eV
# density kernel cutoff
kernel_cutoff : 1000 bohr
# level of detail of the output
output_detail : verbose
# task to be performed
task : GeometryOptimization
# optimization algorithm
geom_method: TPSD
# tolerance for the forces to
# evaluate convergence
geom_force_tol: 0.05 "ev/ang"
# print relevant information
write_denskern : True
write_forces : True
write_hamiltonian : True
write_tightbox_ngwfs : True
write_xyz: True
# Exchange-Correlation functional
xc_functional : PBE

# lattice vectors block
# values are in angstrom (using ang keyword)
%BLOCK LATTICE_CART
ang
    15.000000000 0.0000000000 0.0000000000
    0.0000000000 15.000000000 0.0000000000
    0.0000000000 0.0000000000 15.000000000
%ENDBLOCK LATTICE_CART

# species block
# species, name, atomic number,
# number of NGWFs, NGWF radii (in bohr)
%BLOCK SPECIES
```

(continues on next page)

(continued from previous page)

```
    C C 6 4 12.000000
    O O 8 4 12.000000
    H H 1 1 12.000000
%ENDBLOCK SPECIES

# potentials block
# species, file where the potential can be found.
# It is recommended to copy pseudopotential files to
# the directory, where we perform the calculation.
%BLOCK SPECIES_POT
    C "C.recpot"
    H "H.recpot"
    O "O.recpot"
%ENDBLOCK SPECIES_POT

# positions in Angstrom
%BLOCK POSITIONS_ABS
ang
C -4.63004  0.41911  0.07151
C -3.76666  1.67624  0.06260
H -4.34884 -0.23171  0.89807
O -2.40610  1.40089 -0.17922
H -2.09824  0.76731  0.47816
H -5.67688  0.68678  0.18150
H -4.50180 -0.12684 -0.85976
H -4.07361  2.34155 -0.74747
H -3.88245  2.21413  1.01544
%ENDBLOCK POSITIONS_ABS
```

Lines starting with # are comments so they are not read during runtime and they are helpful to explain code functionality. Before proceeding, let's copy the required pseudopotential files to the current directory by providing the /full/path/to/onetep while using,

cp /full/path/to/onetep/pseudo/carbon.recpot C.recpot
cp /full/path/to/onetep/pseudo/hydrogen.recpot H.recpot
cp /full/path/to/onetep/pseudo/oxygen.recpot O.recpot

Please change /full/path/to/onetep/ to the directory where you installed ONETEP. Inside this directory, the ONETEP source code (src), documentation (doc), binaries (bin), etc. are located. It will usually be something like /home/user/software/onetep, where user is your username. We can use an example submission script to perform this optimisation on a high performance computing (HPC) cluster. An important line that needs to be modified in this script is the full path to the ONETEP top directory as before.

```
#!/bin/bash -l
#SBATCH  -J ethanol   # Job name on the scheduler.
#SBATCH  -p batch     # Queue (partition) type.
#SBATCH  -N 1         # Number of nodes.
#SBATCH  -n 4         # Total number of MPI ranks.
#SBATCH --mem=16G     # Max memory per node.
#SBATCH  -t 01:10:00  # Wallclock time in [hh:mm:ss].

# load modules
```

(continues on next page)

(continued from previous page)

```
# Change the line below the comments to match your
# preferred compiler, mpi, mkl versions.
module load intel-compilers intel-mpi intel-mkl

# on Iridis5 you can use the following modules,
# module load intel-compilers/2021.2.0 intel-mkl/2021.2.0 \
#              intel-mpi/2021.2.0 python/3.9.7

# on ARCHER2 you can use the following modules,
# module load PrgEnv-cray/8.1.0 cce/12.0.3 cray-fftw/3.3.8.11 \
              cray-mpich/8.1.4 cray-python/3.9.7.1

# sim ID.
# Please put a name of your preference to distinguish
# this simulation. This should be the same as the
# ONETEP input file without the .dat extension.
rootname='ethanol_geo_opt'

# Number of threads per MPI rank.
# This value can be modified, even though ONETEP
# works well with 4 threads per rank for most systems.
export OMP_NUM_THREADS=4

# ONETEP top directory full path,
# e.g., "/home/user/software/onetep".
# Need to modify this.
onetep_top="/full/path/to/onetep/"

# Relative location of the ONETEP executable file.
# e.g., ${onetep_top}/bin/onetep.x86_64_gfortran
# Need to modify the name of the executable.
onetep_exe="${onetep_top}/bin/onetep.executable"

# ONETEP launcher. No need to change this.
onetep_launcher="${onetep_top}/utils/onetep_launcher"

# Show shared libraries. No need to change this.
ldd $onetep_exe >\$ldd

# If running on Iridis5, set I_MPI_PMI_LIBRARY
# environmental variable by removing the comment
# sign '#' at the start of the next couple of lines.
#export I_MPI_PMI_LIBRARY\
#=/local/software/slurm/default/lib/libpmi.so

# If running on ARCHER2, export the python-related
# paths. The PYTHONUSERBASE path is an example, hence
# it should be modified to your needs as explained in
# the python section of the ARCHER2 documentation.
# You should copy the correct project code, e.g., t01,
# your username on ARCHER2, and change python3.9 to
# your version of python. You can export these paths
```

(continues on next page)

(continued from previous page)

```
# by removing the comment sign '#' at the start of
# the next four lines.
#export PYTHONUSERBASE=/work/project_code/project_code/username/.local
#export PATH=$PYTHONUSERBASE/bin:$PATH
#export PYTHONPATH\
#=$PYTHONUSERBASE/lib/python3.9/site-packages:$PYTHONPATH

# Command to run ONETEP. No need to change this.
srun \
     -N $SLURM_JOB_NUM_NODES \
     -n $SLURM_NPROCS \
     -e $onetep_exe \
     -t $OMP_NUM_THREADS \
     $onetep_launcher \
     ${rootname}.dat \
     >${rootname}.out \
     2>${rootname}.err
```

Note that if your run this on a local computer (not on a computing cluster), e.g., your personal desktop or laptop, then you'll need to remove the lines starting with #SBATCH and replace srun with mpirun, if you have installed the Intel(R) MPI Library on your computer, so your script will become,

```
#!/bin/bash -l

# sim ID.
# Please put a name of your preference to distinguish
# this simulation. This should be the same as the
# ONETEP input file without the .dat extension.
rootname='ethanol_geo_opt'

# Number of threads per MPI rank.
# This value can be modified, even though ONETEP
# works well with 4 threads per rank for most systems.
export OMP_NUM_THREADS=4

# ONETEP top directory full path.
# e.g., /home/user/software/onetep
# Need to modify this.
onetep_top="/full/path/to/onetep/top/directory"

# Relative location of the ONETEP executable file.
# e.g., ${onetep_top}/bin/onetep.x86_64_gfortran
# Need to modify the name of the executable.
onetep_exe="${onetep_top}/bin/onetep.executable"

# ONETEP launcher. No need to change this.
onetep_launcher="${onetep_top}/utils/onetep_launcher"

# Show shared libraries. No need to change this.
ldd $onetep_exe >\$ldd

# If running on Iridis5, set I_MPI_PMI_LIBRARY
```

(continues on next page)

(continued from previous page)

```
# environmental variable by removing the comment
# sign '#' at the start of the next couple of lines.
#export I_MPI_PMI_LIBRARY\
#=/local/software/slurm/default/lib/libpmi.so

# Command to run ONETEP. No need to change this.
mpirun \
     -np 4 \
     -e $onetep_exe \
     -t $OMP_NUM_THREADS \
     $onetep_launcher \
     ${rootname}.dat \
     >${rootname}.out \
     2>${rootname}.err
```

In the example submission script, submission_script.sb, we request 1 node, 4 MPI ranks in total, 4 OpenMP threads per MPI rank, 16 GB of memory, and 1 hour and 10 minutes to run our simulation using the batch queue, and we can distinguish our simulation on the scheduler as it is named as ethanol. Please note that the number of MPI ranks should be equal or less than the number of atoms in the system. Consequently, for the ethanol molecule we can ask for up to 9 MPI ranks.

$$\text{Number of MPI ranks} \leq \text{Number of atoms in the system} \tag{12.45}$$

Here we chose, slightly less than half that number. The total number of cores, N_{cores}, we asked for can be calculated as follows,

$$N_{cores} = N_{ranks} \times N_{threads} \tag{12.46}$$

$$N_{ranks} = N_{nodes} \times N_{ranks\_per\_node} \tag{12.47}$$

where N_{nodes}, N_{ranks}, $N_{ranks\_per\_node}$, $N_{threads}$, are the number of nodes, the total number of MPI ranks, the number of MPI ranks per node, and threads per MPI rank, respectively. Consequently, in this example we asked for 16 cores (4 MPI ranks in total $\times$ 4 threads per rank). We submit our script to the scheduler as a batch job using,

$$\text{sbatch submission\_script.sb}$$

or we run it on a local computer using a couple of shell commands, the first one gives us permission to execute the script, and the second one actually runs the script

$$\begin{aligned} \text{chmod } +\text{x submission\_script.sh} \\ \text{sh submission\_script.sh} \end{aligned}$$

We can check if our script is running by looking at the queue. In SLURM, we can use the following command,

$$\text{squeue } -\text{u \$USER}$$

An example output of the command is

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST
1915382	batch	ethanol	user	R	10 : 25	1	red465

This tells us that the job has been assigned an identification number that is 1915382, it uses the batch partition, its name is ethanol, it is run by a user named: user, its current status is: R : Running, 10 minutes and 25 seconds have elapsed since starting running the script, it is running on 1 node, and the name of this node is red465. When the script completes its run, nothing will appear under the header above, as no script will be waiting/running in the queue. Please note that if the status is PD, this means that the start of this script is pending, hence the script is waiting in the queue. ONETEP generates certain files at runtime based on the type of calculation. In our case, the following files were generated,

- ethanol_geo_opt.out - The calculation output.
- ethanol_geo_opt.err - Any errors occured during the calculation. If this is empty then no errors occured. Otherwise, another file called ethanol_geo_opt.error_message will be generated, where more information about the error(s) can be found.
- ethanol_geo_opt.tightbox_ngwfs - Information about the NGWFs generated during the calculation.
- ethanol_geo_opt.continuation - Information about restarting a geometry relaxation.
- ethanol_geo_opt.dkn - Information about the density kernel generated during the calculation.
- ethanol_geo_opt.ham - Information about the Hamiltonian used during the calculation.
- ethanol_geo_opt.geom - Trajectory of geometries, energies, and forces during relaxation.
- ethanol_geo_opt.xyz - Trajectory of geometries during relaxation in the XYZ format. This can be used/visualised with many computational chemistry codes, and molecular visualising software.
- ethanol_geo_opt.bib - References that need to be cited for this calculation written using BIB formatting.

We open the output file and we should find a line that reads,

$$\text{Finished geometry optimization after 5 iteration(s).}$$
$$\text{Total Energy } -3.110106446448\text{E}+001 \text{ Eh}$$

Consequently, the geometry relaxation made 5 steps and the final total energy of the system is -31.10106446448 Eh. This number may be slightly different when you run this example on your computer or a computing cluster due to differences in machine precision and number rounding patterns. The outcome of a geometry relaxation is a more stable configuration of the atoms of the system. Consequently, we should visualize the relaxed geometry. We can do that by opening the ethanol_geo_opt.xyz file in a molecular visualisation software. We can use Avogadro, or any other software that is able to visualise molecules, to do this. There we can see that distances and angles between the atoms have changed as they have moved to equilibrium positions in which this molecular configuration is more stable. The optimised structure is shown in Fig. 12.1.

ONETEP built-in optimizer in ASE

The same calculation can be prepared using ASE. Please create a folder named however you prefer, e.g., ethanol_example_ase and put all files in that folder. We will gather information about the system and calculation parameters in one python file and ASE will generate an ONETEP input file for us. Also, you will see how we can translate the atomic coordinates in ASE, center the molecule in the unit cell, and how ASE can automatically set lattice vectors for us. We follow the guidance provided in Section 5.2 to create an input file in ASE. This should look like the following for the ethanol example,

```
# ethanol_geo_opt_ase.py
from os import environ, path
from ase.build import molecule
from ase.calculators.onetep import Onetep
from ase.io import read,iread,write

# set ONETEP run command by providing the full
# path to the ONETEP executable. Here we also
# set the number of MPI ranks and threads per rank.
environ["ASE_ONETEP_COMMAND"]="srun -n $SLURM_NPROCS /path/to/onetep/bin/onetep  PREFIX.
↪dat >> PREFIX.out 2> PREFIX.err"

# read atomic positions from an XYZ file
mol=read('ethanol.xyz')
```

(continues on next page)

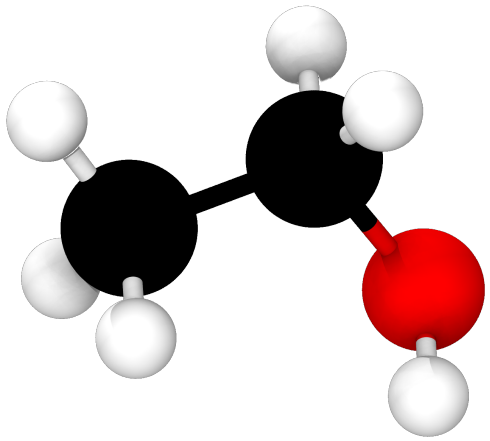

Fig. 12.1: Atomic configuration of the ethanol molecule after relaxing its geometry in ONETEP. Colour code: carbon - black, oxygen - red, hydrogen - white.

(continued from previous page)

```
# center the molecule in unit cell. This
# will automatically set the unit cell,
# hence, the lattice vectors provided to ONETEP.
mol.center(vacuum=7.)

# set calculator as Onetep and a
# label for input and output files.
calc_label='ethanol_geo_opt'
calc = Onetep(label=calc_label)

# set Onetep parameters (left-hand side) and
# corresponding values (right-hand side).
calc.set(pseudo_path='./',
         pseudo_suffix='.recpot',
         task='GeometryOptimization',
         geom_method='TPSD',
         geom_force_tol='0.05 "ev/ang"',
         xc='PBE',
         cutoff_energy='1000 eV',
         kernel_cutoff='1000 bohr',
         species_ngwf_number={
            "C":4,"H":1,"O":4},
         species_ngwf_radius={
            "C":12.0,"H":12.0,"O":12.0}, # bohr
         write_denskern=True,
         write_forces=True,
         write_hamiltonian=True,
         write_tightbox_ngwfs=True,
         write_xyz=True,
         output_detail='verbose',
         )

# perform the calculation
mol.calc = calc

# get atomic positions, energy and forces from the
# calculator. In our case, the calculator is ONETEP.
mol.get_positions()
mol.get_potential_energy()
```

We can use an example submission script to perform this optimisation by running ONETEP through ASE.

```
#!/bin/bash -l
#SBATCH  -J ethanol   # Job name on the scheduler.
#SBATCH  -p batch     # Queue (partition) type.
#SBATCH  -N 1         # Number of nodes.
#SBATCH  -n 4         # Total number of MPI ranks.
#SBATCH --mem=16G     # Max memory per node.
#SBATCH  -t 01:10:00  # Wallclock time in [hh:mm:ss].

# load modules
```

(continues on next page)

(continued from previous page)

```
# Change the line below the comments to match your
# preferred compiler, mpi, mkl, python versions.
module load intel-compilers intel-mpi intel-mkl python

# on Iridis5 you can use the following modules,
# module load intel-compilers/2021.2.0 intel-mkl/2021.2.0 \
#             intel-mpi/2021.2.0 python/3.9.7

# on ARCHER2 you can use the following modules,
# module load PrgEnv-cray/8.1.0 cce/12.0.3 cray-fftw/3.3.8.11 \
              cray-mpich/8.1.4 cray-python/3.9.7.1

# Number of threads per MPI rank.
# This value can be modified, even though ONETEP
# works well with 4 threads per rank for most systems.
export OMP_NUM_THREADS=4

# If running on Iridis5, set I_MPI_PMI_LIBRARY
# environmental variable by removing the comment
# sign '#' at the start of the next couple of lines.
#export I_MPI_PMI_LIBRARY\
#=/local/software/slurm/default/lib/libpmi.so

# If running on ARCHER2, export the python-related
# paths. The PYTHONUSERBASE path is an example, hence
# it should be modified to your needs as explained in
# the python section of the ARCHER2 documentation.
# Please remember to change the project_code, e.g., t01,
# and the username on ARCHER2. Also, please update
# the python version below to the one you are using.
# You can export these paths by removing the comment
# sign '#' at the start of the next four lines.
#export PYTHONUSERBASE=/work/project_code/project_code/username/.local
#export PATH=$PYTHONUSERBASE/bin:$PATH
#export PYTHONPATH\
#=$PYTHONUSERBASE/lib/python3.9/site-packages:$PYTHONPATH

# run ONETEP through python-ASE
python3 ethanol_geo_opt_ase.py > ethanol_geo_opt_ase_out.log
```

If running the script on a local computer, remove the lines starting with #SBATCH, and follow the same procedure as in the previous example to run the script on your local computer. After running this example, an additional output file is generated. This file is named as ethanol_geo_opt_ase_out.log and it contains the output from the python interface between ASE and ONETEP. This is an empty file. This is due to the fact that calculations were performed only in ONETEP and not in ASE. Calculation output can be found in ethanol_geo_opt.out once again. As in the previous example, we can visualise the atomic configuration by opening the XYZ file with a software that is able to visualise molecules, such as Avogadro.

Platinum nanoparticle

ASE built-in optimizer

Now we move to a more advanced example where we can relax the geometry of a platinum nanoparticle, Pt_{13}. Please create a folder named however you prefer, e.g., platinum_example and put all files in that folder. In this example, we will generate the nanoparticle consisting of 13 Pt atoms in ASE and then use an ASE built-in optimisation algorithm to relax its geometry. More information about how to generate nanoparticles in ASE can be found in the ASE manual [ASE_nanoparticle]. In this example, we will use a pseudopotential that was generated on the fly using the CASTEP code [CASTEP] [Pickard2006].

```
# generate_pt13_ase.py
from ase.cluster import Icosahedron
from ase.io import write

# create a Pt nanoparticle with 2 shells
mol = Icosahedron('Pt', 2)
# save coordinates to an XYZ file
write('pt13.xyz',mol)
```

Now we will load our geometry saved in an XYZ file in the script that is designed to relax the geometry. We shall place the molecule at the center of the unit cell and add vacuum in three dimensions so that the atoms and corresponding NGWFs are all inside the unit cell.

```
# pt13_geo_opt_ase.py
from os import environ, path
from ase.build import molecule
from ase.calculators.onetep import Onetep
from ase.io import read,iread,write
from ase.optimize.sciopt import SciPyFminBFGS

# set ONETEP run command by providing the full
# path to the ONETEP executable. Here we also
# set the number of MPI ranks and threads per rank.
environ["ASE_ONETEP_COMMAND"]="srun -n $SLURM_NPROCS /path/to/onetep/bin/onetep PREFIX.
↪dat >> PREFIX.out 2> PREFIX.err"

# load the molecule from an XYZ file
mol = read('pt13.xyz')

# Build unit cell by placing the molecule
# at the centre and add vacuum in three
# dimensions (value in Angstrom).
mol.center(vacuum=7.)

# set calculator as Onetep and a
# label for input and output files.
calc_label = 'pt13_geo_opt'
calc = Onetep(label=calc_label)

# set Onetep parameters (left-hand side) and
# corresponding values (right-hand side).
calc.set(pseudo_path='./',
```

(continues on next page)

(continued from previous page)

```
          pseudo_suffix='_NCP19_PBE_OTF.usp',
          task='SinglePoint',
          xc='PBE',
          cutoff_energy='1000 eV',
          kernel_cutoff='1000 bohr',
          species_ngwf_number={
             "Pt":13},
          species_ngwf_radius={
             "Pt":12.0}, # bohr
          edft=True,
          edft_update_scheme='pulay_mix',
          edft_maxit=10,
          edft_smearing_width='1000 K',
          write_denskern=True,
          write_forces=True,
          write_hamiltonian=True,
          write_tightbox_ngwfs=True,
          write_xyz=True,
          output_detail='verbose',
          )

if path.isfile(calc_label + '.dkn') == True:
   calc.set(read_denskern=True)
if path.isfile(calc_label + '.ham') == True:
   calc.set(read_hamiltonian=True)
if path.isfile(calc_label + '.tightbox_ngwfs') == True:
   calc.set(read_tightbox_ngwfs=True)

# perform the calculation
mol.calc = calc

# get atomic positions, energy and forces from the
# calculator. In our case, the calculator is ONETEP.
mol.get_positions()
mol.get_forces()
mol.get_potential_energy()

# set the optimisation algorithm to ASE built-in
# SciPyFminBFGS and save trajectory in a file.
opt = SciPyFminBFGS(mol, trajectory='opt.traj')
# perform the local optimisation until the maximum
# magnitude of the net force on any atom is 0.05 eV A^(-1).
opt.run(fmax=0.05)

# convert trajectory to the XYZ format
trj = iread("opt.traj")
trj_xyz = write("opt_traj.xyz", trj, format="xyz")
```

We can slightly modify submission_script_ase.sb to run this example. The modified submission script follows,

```
#!/bin/bash -l
#SBATCH  -J pt13      # Job name on the scheduler.
```

(continues on next page)

(continued from previous page)

```
#SBATCH  -p batch     # Queue (partition) type.
#SBATCH  -N 1         # Number of nodes.
#SBATCH  -n 4         # Total number of MPI ranks.
#SBATCH --mem=16G     # Max memory per node.
#SBATCH  -t 01:10:00  # Wallclock time in [hh:mm:ss].

# load modules
# Change the line below the comments to match your
# preferred compiler, mpi, mkl, python versions.
module load intel-compilers intel-mpi intel-mkl python

# on Iridis5 you can use the following modules,
# module load intel-compilers/2021.2.0 intel-mkl/2021.2.0 \
#             intel-mpi/2021.2.0 python/3.9.7

# on ARCHER2 you can use the following modules,
# module load PrgEnv-cray/8.1.0 cce/12.0.3 cray-fftw/3.3.8.11 \
#             cray-mpich/8.1.4 cray-python/3.9.7.1

# Number of threads per MPI rank.
# This value can be modified, even though ONETEP
# works well with 4 threads per rank for most systems.
export OMP_NUM_THREADS=4

# If running on Iridis5, set I_MPI_PMI_LIBRARY
# environmental variable by removing the comment
# sign '#' at the start of the next couple of lines.
#export I_MPI_PMI_LIBRARY\
#=/local/software/slurm/default/lib/libpmi.so

# If running on ARCHER2, export the python-related
# paths. The PYTHONUSERBASE path is an example, hence
# it should be modified to your needs as explained in
# the python section of the ARCHER2 documentation.
# You should copy the correct project code, e.g., t01,
# your username on ARCHER2, and change python3.9 to
# your version of python. You can export these paths
# by removing the comment sign '#' at the start of
# the next four lines.
#export PYTHONUSERBASE=/work/project_code/project_code/username/.local
#export PATH=$PYTHONUSERBASE/bin:$PATH
#export PYTHONPATH\
#=$PYTHONUSERBASE/lib/python3.9/site-packages:$PYTHONPATH

# run ONETEP through python-ASE
python3 pt13_geo_opt_ase.py > pt13_geo_opt_ase_out.log
```

In this calculation, we used Ensemble-DFT as this is a metallic system. Theory and information on how to set Ensemble-DFT parameters can be found in the relevant section of the ONETEP manual. We remind the reader that in simple terms, geometry relaxation is a series of single point energy calculations. The structure for which we calculate the energy at every step is generated based on an optimisation algorithm. In this example, we performed geometry relaxation as a series of single point energy calculations and this is why the task is set to SinglePoint in the ASE input. Then, the search direction, step size, and ultimately, the next step that corresponds to an atomic configuration are calculated

through ASE's built-in optimisation algorithm of choice. In this example, we used the ScipyFminBFGS optimisation algorithm available in ASE [ase_optimizers_website]. In contrast with the previous example, the python output file now contains useful information about the optimisation process. This is of the form of,

```
              Step  Time          Energy         fmax
SciPyFminBFGS: 0  10:01:57  -46453.524595  2.0483
SciPyFminBFGS: 1  10:30:45  -46454.197133  1.7570
SciPyFminBFGS: 2  11:13:25  -46455.317379  0.9590
SciPyFminBFGS: 3  11:52:19  -46455.610507  0.4268
SciPyFminBFGS: 4  12:23:07  -46455.622235  0.0579
SciPyFminBFGS: 5  12:54:25  -46455.624640  0.0093
```

where, the optimisation algorithm is SciPyFminBFGS, the actual time, Time, that the calculation of the step has finished (this uses the time on the computer) is recorded, the Energy is shown in units defined in ASE (these are eV by default), and the maximum force magnitude on any atom in the system, fmax, is provided in units defined in ASE (these are eV$\cdot$ Å $^{-1}$ by default). The trajectory of atomic positions is also printed in files opt.traj and opt_traj.xyz. The latter can be visualised with popular visualisation software. The relaxed geometry of this system is shown in Fig. 12.2.

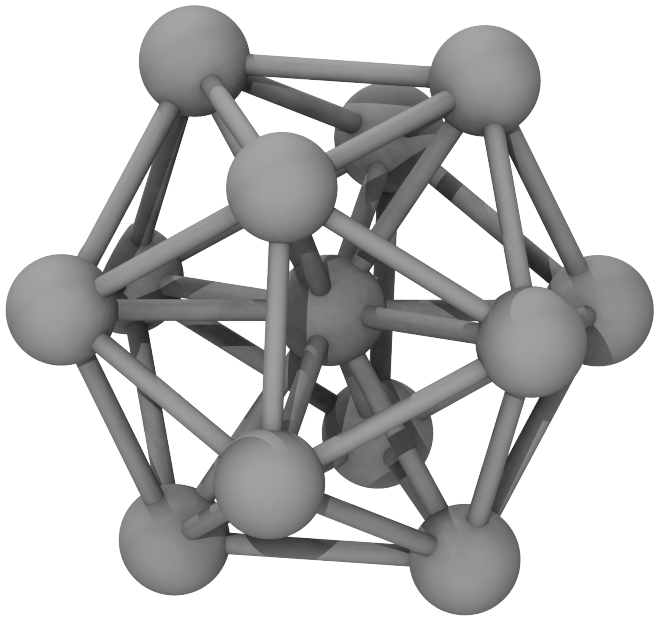

Fig. 12.2: Atomic configuration of the platinum nanoparticle after relaxing its geometry in ONETEP.

12.1.7 Summary

In this guide, we explained how to perform geometry relaxation in ONETEP. This can be achieved using either ONETEP or ASE built-in optimisation algorithms. This way we can obtain more stable structures than our initial guess for a given system. This is fundamental in chemistry as it can lead to many conclusions. First, we can obtain more accurate reaction energies by optimising the reactant and product geometries, and then calculate their energies. Additionally, we understand how different atomic arrangements affect the energy of the system and this is significant to a number of fields including and not limited to separations, catalysis, and biological phenomena.

12.1.8 Acknowledgement

We would like to thank Drs. Manuel Dos Santos Dias and Jacek Dziedzic for their suggestions after reading this manual. Also, we would like to thank Mr. Julian O. Holland for testing the ONETEP/ASE interface and for providing us with useful feedback.

12.2 Simulation Cell Relaxation

These notes describe the usage and functionality implemented in ONETEP for the calculation of the stress tensor and related quantities.

12.2.1 User guide

Table 12.1: Keywords table

Keyword	Type	Default	Description
STRESS	Task	—	Enable stress functionality.
stress_tensor	Logical	F	Enable the calculation of the stress tensor.
stress_elasticity	Logical	F	Enable the calculation of elastic constants
stress_relax	Logical	F	Use the stress tensor to optimise the cell parameters.
stress_assumed_symmetry	String	nosymm	Use assumed symmetry to minimise calculations. Values are nosymm; 3D: cubic ortho, tetra1, tetra2 hexa3d, rhomb1, rhomb2; 2D: recta, squar1, squar2, hexa2d NOTE: Symmetry is assumed, the code will not check if it's correct!
stress_rescale_volume	Real	1.0	Rescaling for cell volume. Might be useful for 1D or 2D systems.
stress_components	Logical	T T T	Which rows/columns of the stress tensor to compute. The flags match X Y Z.
stress_deformation_step	Real	2.0e-3	Unitless strain parameter in finit difference It controls how different the deformation matrix is from the identity
stress_maxit_ngwf_cg	Integer	10	

The *keywords table* lists the input parameters that are available in connection with the stress implementation. The main functionality can be enabled by specifying the corresponding keyword (`stress_tensor`, `stress_elasticity` or `stress_relax`) without providing additional options. If the default values of the unspecified options are found to be unsuitable for the target system they can be overridden with the corresponding keyword in the input file.

The STRESS task gives access to all the features of the stress implementation. It first performs a standard self-consistent calculation for a reference unit cell (as given in the input file), and then performs subsequent calculations according to the requested stress-related quantities. This can be sped up quite considerably by writing out the density kernel (`write_denskern T`) or Hamiltonian (`write_hamiltonian T`) and NGWFs (`write_tightbox_ngwfs T`) files at the end of the self-consistent calculation for the reference unit cell. These should then be read for the calculations of the distorted structures used in constructing the stress tensor (`read_denskern T`, `read_tightbox_ngwfs T` and `read_hamiltonian T`). The code makes sure that the distorted cell calculations do not overwrite these files; if one is performing a cell relaxation calculation then these files can be updated by the self-consistent calculation for each new trial unit cell. *Empirical observation:* performing cell relaxation using only `write_tightbox_ngwfs T` shows the most stable and robust behaviour for the test calculations that I performed so far.

Calculation of the stress tensor

The definition for the stress tensor is

$$\sigma_{\alpha\beta} = -\frac{1}{V}\frac{\partial E}{\partial \epsilon_{\alpha\beta}} \ .$$

Here V is the volume of the unit cell, E is the total energy (of the unit cell), σ and ϵ are the stress and strain tensors, respectively, and $\alpha, \beta = x, y, z$. To convert to experimental units: energy divided by volume has units of pressure. The strain tensor describes a deformation of space away from a reference configuration: $r'_\alpha = (\delta_{\alpha\beta} + \epsilon_{\alpha\beta})\, r_\beta$ (Einstein summation convention) or $\mathbf{r}' = (1+\epsilon)\,\mathbf{r}$. For more details see Section *Background — Definition of stress for a crystal*.

The stress tensor is calculated starting from a self-consistent calculation for a reference unit cell, applying distortions to those reference cell vectors, and using the corresponding total energy differences to approximate the derivatives of the total energy with respect to the distortion parameters. This is available through the STRESS task by setting `stress_tensor` to `T`.

Here are two examples of distortion matrices:

$$\epsilon^-_{xx} = \begin{pmatrix} 1+h & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \ , \quad \epsilon^+_{xy} = \begin{pmatrix} 1 & +h & 0 \\ +h & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \ .$$

These act on the matrix of cell vectors as e.g.

$$\begin{pmatrix} a'_{1x} & a'_{2x} & a'_{3x} \\ a'_{1y} & a'_{2y} & a'_{3y} \\ a'_{1z} & a'_{2z} & a'_{3z} \end{pmatrix} = \begin{pmatrix} 1 & +h & 0 \\ +h & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{1x} & a_{2x} & a_{3x} \\ a_{1y} & a_{2y} & a_{3y} \\ a_{1z} & a_{2z} & a_{3z} \end{pmatrix} \ .$$

The parameter h is controlled by the keyword `stress_deformation_step`. As $|h| \ll 1$, if the calculation for the distorted structure is started from the converged ground state obtained for the reference structure it should converge in a handful of iterations; `stress_maxit_ngwf_cg` can be used to control this and avoid spending too many iterations on a calculation that may have incorrect input and so takes too long to converge.

The stress tensor is a 3×3 symmetric matrix, so it has 6 independent components. Finding all these components then requires 12 calculations for distorted cells ($+h$ and $-h$), to form the centred differences that approximate the total energy derivatives. This can be alleviated if the unit cell is expected to have a definite symmetry. For example, for a cubic crystal $\sigma_{xx} = \sigma_{yy} = \sigma_{zz} \neq 0$ and $\sigma_{xy} = \sigma_{yz} = \sigma_{zx} = 0$, so only two distortions are needed (ϵ^+_{xx} and ϵ^-_{xx}). The keyword `stress_assumed_symmetry` controls the available options; see Table for a quick conversion and Table I of Ref. for the notation and further details. NOTE: Symmetry is assumed, the code will not check if it's correct!

The stress tensor components only make sense if the corresponding spatial directions have periodic boundary conditions, which is the case of a bulk crystal. The keyword `stress_components` gives control over this, and if any of its flags is `F` it will override `stress_assumed_symmetry` to use `nosymm`. For example, for a 2D material with periodic boundary conditions in x and y and open boundary conditions (or a thick vacuum region) in the z direction, respectively, one would specify `T T F` to skip calculation (and usage) of information that involves the z direction. As another example, consider a nanotube which is periodic in the z direction; this would correspond to the flags `F F T`. Lastly, the stress tensor should not be computed for something which is surrounded entirely by vacuum, such as a molecule.

cubic	ortho	tetra1	tetra2	hexa3d	rhomb1	rhomb2
$m\bar{3}, m\bar{3}m$	mmm	$4/mmm$	$4/m$	$6/mmm, 6/m$	$\bar{3}m$	$\bar{3}$
recta	squar1	squar2	hexa2d			
$2mm$	$4mm$	4	$6, 6mm$			

Calculation of the elastic properties

This part is still being developed, it should not be used yet.

Optimisation of the cell parameters

The stress tensor can be used to optimise the cell parameters by minimising the total energy with the `STRESS` task. This feature is enabled by `stress_relax T`. It could also be integrated with the `GEOMETRYOPTIMIZATION` task (not implemented so far).

Let us consider for simplicity that the atomic positions are fixed (relative to the unit cell vectors). We start from a unit cell specified by the vectors $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and want to calculate the optimised vectors $\{\mathbf{a}_1^*, \mathbf{a}_2^*, \mathbf{a}_3^*\}$ that minimise the energy of the system, and so for which the stress tensor vanishes, $\sigma_{\alpha\beta}(\{\mathbf{a}_i^*\}) = 0$. We can relate the known input cell parameters to the unknown optimised ones by a strain matrix,

$$\begin{pmatrix} a_{1x}^* & a_{2x}^* & a_{3x}^* \\ a_{1y}^* & a_{2y}^* & a_{3y}^* \\ a_{1z}^* & a_{2z}^* & a_{3z}^* \end{pmatrix} = \begin{pmatrix} 1+\epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{xy} & 1+\epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{xz} & \epsilon_{yz} & 1+\epsilon_{zz} \end{pmatrix} \begin{pmatrix} a_{1x} & a_{2x} & a_{3x} \\ a_{1y} & a_{2y} & a_{3y} \\ a_{1z} & a_{2z} & a_{3z} \end{pmatrix} .$$

We then Taylor expand the total energy in the strain parameters,

$$E(\epsilon) \approx E(0) + \sum_{(\alpha\beta)} \epsilon_{\alpha\beta} \left.\frac{\partial E}{\partial \epsilon_{\alpha\beta}}\right|_{\epsilon=0} + \frac{1}{2} \sum_{(\alpha\beta),(\alpha'\beta')} \epsilon_{\alpha\beta}\, \epsilon_{\alpha'\beta'} \left.\frac{\partial^2 E}{\partial \epsilon_{\alpha\beta} \partial \epsilon_{\alpha'\beta'}}\right|_{\epsilon=0} .$$

For the stress to vanish we evaluate the first derivative w.r.t. the strain parameters and set it to zero:

$$0 = \left.\frac{\partial E}{\partial \epsilon_{\alpha\beta}}\right|_{\epsilon\neq 0} = \left.\frac{\partial E}{\partial \epsilon_{\alpha\beta}}\right|_{\epsilon=0} + \sum_{(\alpha'\beta')} \epsilon_{\alpha'\beta'} \left.\frac{\partial^2 E}{\partial \epsilon_{\alpha\beta} \partial \epsilon_{\alpha'\beta'}}\right|_{\epsilon=0} .$$

The first derivative of the total energy is approximated by centred differences, but this also provides enough information to approximate the diagonal part of the matrix of second derivatives ($\alpha'\beta' = \alpha\beta$). This leads to an approximate Newton-like step to solve for the strain parameters $\epsilon_{\alpha\beta}$:

$$\epsilon_{\alpha\beta} = - \left.\frac{\partial E}{\partial \epsilon_{\alpha\beta}}\right|_{\epsilon=0} \left(\left.\frac{\partial^2 E}{\partial \epsilon_{\alpha\beta}^2}\right|_{\epsilon=0} \right)^{-1} ,$$

which is implemented as the cell relaxation method for fixed atomic positions.

In practice the optimisation of the cell parameters will not converge in one iteration. The maximum number of iterations or trial unit cells is controlled by `stress_relax_max_iter`. The magnitude of the deformation is controlled by

stress_relax_max_step, which ensures that $|\epsilon_{\alpha\beta}|$ does not exceed the specified value. There are three convergence criteria that must be satisfied simultaneously. stress_relax_energy_tol ensures that the change in total energy per atom between consecutive trial unit cells is below a given value, stress_relax_pressure_tol checks that the pressure of the current unit cell is below a target value, and stress_relax_acell_tol monitors the relative change in cell parameters, $\frac{\max|\Delta a_{i\alpha}|}{\max|a_{i\alpha}|}$.

The atomic positions can also be optimised in tandem with the cell parameters. This is enabled by the flag stress_relax_atoms. The calculation will start by first optimising the atomic positions with fixed cell parameters, and then it will create a guess at the cell parameters to try next. If all the convergence thresholds for the cell optimisation are met the calculation ends, otherwise it keeps iterating by reoptimising the atomic positions and then making a new guess for the cell parameters. The optimisation of the atomic positions proceeds in the same way as with the GEOMETRYOPTIMIZATION task, and the corresponding keywords/flags can be used to control the same aspects of that process.

External pressure can included during cell relaxation using stress_relax_pressure. This will be added to the diagonal elements of the stress tensor which are not set to zero by the assumed symmetry in the calculation. Positive values of pressure will lead to a compression of the unit cell volume.

12.2.2 Background — Definition of stress for a crystal

These notes follow the original papers by Nielsen and Martin [Nielsen1983] [Nielsen1985] [Nielsen1985b] and the discussion in Chapter 3 of the book of Martin [Martin2008]. The definition for the stress tensor is

$$\sigma_{\alpha\beta} = -\frac{1}{V}\frac{\partial E}{\partial \epsilon_{\alpha\beta}} \ .$$

Here V is the volume of the unit cell, E is the total energy (of the unit cell), σ and ϵ are the stress and strain tensors, respectively, and $\alpha, \beta = x, y, z$. To convert to experimental units: energy divided by volume has units of pressure. 1GPa = 10kbar = $10^9 j/m^3$, so it's enough to convert the energy to Joule and the volume to cubic meters.

The strain tensor describes a deformation of space away from a reference configuration: $r'_\alpha = (\delta_{\alpha\beta} + \epsilon_{\alpha\beta})\, r_\beta$ (Einstein summation convention) or $\mathbf{r}' = (1+\epsilon)\,\mathbf{r}$. The way this works is easiest to understand in one dimension and for a one-particle wave function $\Psi(x)$, defined in the interval $[0, L]$ which is the unit cell for this example. Suppose that the unit cell is stretched to $[0, L']$ with $L' = (1+\epsilon)\,L$, and so the wave function will also be stretched to $\widetilde{\Psi}(x')$ with the coordinate in the stretched unit cell being related to the starting one by $x' = (1+\epsilon)\,x$. This is illustrated in Fig. *1* . The wave function at the stretched coordinate is almost the same as the wave function at the unstretched coordinate,

$$\widetilde{\Psi}(x') = C\,\Psi((1+\epsilon)^{-1}x') = C\,\Psi(x) \ ,$$

where C is a constant to be determined. [This is usually horribly confusing; a function returns a specified output for a given input, so if we want to know what is the value of the wave function $\widetilde{\Psi}$ at x' we need to first transform that to the input for the wave function Ψ that will generate the correct output.] The remaining difference is that the wave function has to be normalised to the unit cell,

$$\int_0^L \mathrm{d}x\ |\Psi(x)|^2 = 1 \implies \int_0^{L'} \mathrm{d}x'\ |\widetilde{\Psi}(x')|^2 = (1+\epsilon)\,C^2 \int_0^L \mathrm{d}x\ |\Psi(x)|^2 \ ,$$

and so the complete relation is $\widetilde{\Psi}(x') = (1+\epsilon)^{-1/2}\,\Psi((1+\epsilon)^{-1}x')$. Intuitively, the volume element is changed as $\mathrm{d}x' = (1+\epsilon)\,\mathrm{d}x$, so the normalisation is adjusted to compensate.

Moving now to the usual three-dimensional problem and to a crystalline setting, we first express the position vector in terms of the vectors defining the reference unit cell:

$$\mathbf{r} = r_1\,\mathbf{a}_1 + r_2\,\mathbf{a}_2 + r_3\,\mathbf{a}_3 \ .$$

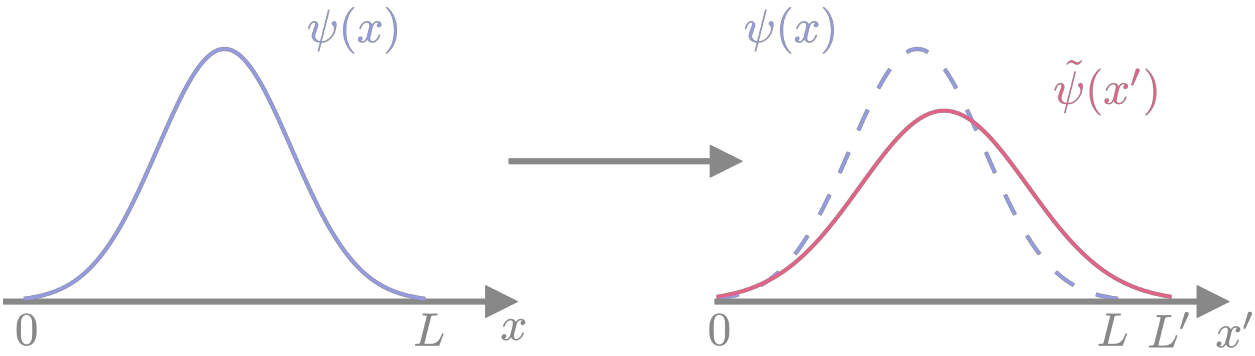

Fig. 12.3: Fig.1: One-dimensional example of coordinate stretching. Comparing the wave function in the stretched coordinate system to the one in the original coordinates shows that its centre (nuclear position) and its amplitude (normalisation) are both changed by the scaling

In Cartesian coordinates this looks like

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{1x} & a_{2x} & a_{3x} \\ a_{1y} & a_{2y} & a_{3y} \\ a_{1z} & a_{2z} & a_{3z} \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \ .$$

The transformation defining the action of the strain tensor ϵ on the crystal can then be converted into a deformation of the unit cell:

The strain tensor is assumed symmetric; a skew-symmetric part would represent a homogeneous rotation of the whole crystal which should leave the energy invariant. The elements of the strain tensor have a simple interpretation when the unit cell vectors are proportional to the unit vectors defining the cartesian axes (e.g., orthorhombic cell): the diagonal elements $\epsilon_{\alpha\alpha}$ ($\alpha = x, y, z$) represent a stretching or compression along the respective unit cell vector or cartesian axis (angles between the unit cell vectors are preserved), while the off-diagonal elements represent shear (the angle between the involved pair of unit cell vectors changes).

It is also common to map the subscripts to integers (Voigt notation),

$$(\epsilon_{xx}, \epsilon_{yy}, \epsilon_{zz}, 2\epsilon_{yz}, 2\epsilon_{xz}, 2\epsilon_{xy}) \rightarrow (\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5, \epsilon_6) \ .$$

To understand how that factor of 2 propagates see the chapter on elasticity of Kittel's book [Kittel] .

The strain can also be represented in a form which uses directly the unit cell vectors. Defining

$$\mathbf{a}_1^* = \frac{\mathbf{a}_2 \times \mathbf{a}_3}{V} \ , \quad \mathbf{a}_2^* = \frac{\mathbf{a}_3 \times \mathbf{a}_1}{V} \ , \quad \mathbf{a}_3^* = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{V} \ , \quad \mathbf{a}_i \cdot \mathbf{a}_j^* = \delta_{ij} \ , \quad V = \mathbf{a}_1 \cdot (\mathbf{a}_2 \times \mathbf{a}_3) \ ,$$

we can write

$$\epsilon = \sum_{i,j} \epsilon_{ij} \, \mathbf{a}_i \otimes \mathbf{a}_j^* \ .$$

This is in general not a symmetric tensor *in Cartesian components*, but we enforce $\epsilon_{ij} = \epsilon_{ji}$. It also only makes life easier if it acts on the matrix of cell vectors from the left,

$$\begin{aligned} \epsilon \, \mathbf{a}_1 &= \epsilon_{11} \, \mathbf{a}_1 + \epsilon_{12} \, \mathbf{a}_2 + \epsilon_{13} \, \mathbf{a}_3 \ , \\ \epsilon \, \mathbf{a}_2 &= \epsilon_{22} \, \mathbf{a}_2 + \epsilon_{12} \, \mathbf{a}_1 + \epsilon_{23} \, \mathbf{a}_3 \ , \\ \epsilon \, \mathbf{a}_3 &= \epsilon_{33} \, \mathbf{a}_3 + \epsilon_{13} \, \mathbf{a}_1 + \epsilon_{23} \, \mathbf{a}_2 \ . \end{aligned}$$

The different elements can then be interpreted as effecting a stretch/compression of the respective unit cell vector (ϵ_{ii}) or shears (ϵ_{ij} with $i \neq j$), which bring cell vectors i and j towards/away from each other.

To be investigated:

$$\sigma = \sum_{i,j} \sigma_{ij} \, \mathbf{a}_i \otimes \mathbf{a}_j^* \ , \quad \sigma_{ij} = -\frac{1}{V} \frac{\partial E}{\partial \epsilon_{ij}} \ ?$$

12.2.3 Computing the stress tensor

Ref. [Knuth2015] has nice derivations and a discussion of finite-difference tests in Section 4. For Projector Augmented Wave (PAW) specifics I looked at Ref. [Kresse1999] , but sadly they wrote "[...] it is also easy to evaluate the stress tensor. We will neither give the full derivation nor the final results here, as the expressions are rather cumbersome and difficult to write in a compact form."

The straightforward numerical calculation by finite differences proceeds in the obvious way (here using the central difference formula):

$$\frac{\partial E_{\mathrm{tot}}}{\partial \epsilon_{\alpha\beta}} \approx \frac{E_{\mathrm{tot}}(\epsilon_{\alpha\beta} = +h) - E_{\mathrm{tot}}(\epsilon_{\alpha\beta} = -h)}{2h} \equiv \frac{\Delta E_{\mathrm{tot}}}{\Delta h} \ ,$$

where the total energies are obtained from self-consistent calculations for the deformed unit cell with only one finite element in the strain tensor. The atomic positions should either be given in internal coordinates or otherwise their Cartesian coordinates have to be scaled. The unitless h has to be tuned via numerical tests, but values $h < 0.01$ are mentioned as reasonable in Ref. [Knuth2015] ; for example Nielsen and Martin used $h = 0.004$ [Nielsen1985b]. *To fully populate the stress tensor using the central difference scheme for the general case one then requires 12 self-consistent calculations.*

References

CHAPTER

THIRTEEN

DENSITY FUNCTIONAL TIGHT BINDING (DFTB)

13.1 Density Functional Tight Binding in ONETEP

Author
Arihant Bhandari, University of Southampton, United Kingdom

Author
Jacek Dziedzic, University of Southampton, United Kingdom

Author
Chris-Kriton Skylaris, University of Southampton, United Kingdom

Date
December 2022

Table of Contents

13.1.1 Introduction to Density Functional Tight Binding (DFTB)

Density Functional Tight binding models have been derived by a Taylor expansion of the DFT energy functional in terms of the electron density and truncation up to a certain order in the expansion [Foulkes1989]. Within DFTB, the following eigenvalue equations are solved via diagonalization:

$$H_{\alpha\beta}M_i^\beta = S_{\alpha\beta}M_i^\beta\epsilon_i, \tag{13.1}$$

where $H_{\alpha\beta}$ is the hamiltonian matrix, $S_{\alpha\beta}$ is the overlap matrix, M_i^β are the orbital coefficients, and ϵ_i are energy eigenvalues. The Hamiltonian is built from parameters. [Elstner1998] proposed a self consistent charge (SCC) extension to the traditional DFTB approach, which optmizes the atomic charges self consistently. Henceforth, a series of SCC DFTB methods have been developed [Gaus2014]. Recently, non-SCC methods have undergone a revival because of their speed and applicability to large systems [Bannwarth2020]. E.g. GFN0 is one such method where atomic charges are found using a charge equilibriation scheme [Pracht2019]. The total energy in GFN0 also includes zeroth order terms such as dispersion, repulsion, electrostatic interactions and short range basis correction.

We have implemented the GFN0 method taking the advantage of linear-scaling capabilities of ONETEP. Here we include D2 dispersion correction [Grimme2006] instead of D4 [Caldeweyher2019]. The standard ensemble-DFT sub-

routines are used for diagonalization and calculation of electronic energies and forces. The parameter files are available in [utils-devel].

13.1.2 Keywords

- `dftb: T/F`

  [Boolean, default `dftb: F`].
  If true, it enables DFTB calculations.

- `dftb_method: GFN0`

  [Text, default `dftb_method: GFN0`].
  Variant of the DFTB method, only GFN0 has been implemented at the moment.

- `dftb_method_param_file: file address`

  [Text, default `dftb_method_param_file: param_gfn0-xtb.txt`].
  Path to the parameter file. A specimen file is supplied in utils-devel/dftb folder.

- `dftb_common_param_file: file address`

  [Text, default `dftb_common_param_file: param_gfn_common.txt`].
  Path to the file for common GFN parameters. A specimen file is supplied in utils-devel/dftb folder.

- `dftb_bc: O O O / P P P`

  [Boolean, default `dftb_bc: P P P`].
  Boundary conditions. Only fully open (O O O) or full periodic (P P P) are implemented.

- `dftb_coord_cutoff: 50.0 ang`

  [Real Physical, default `dftb_coord_cutoff: 40.0 bohr`].
  Cutoff distance for truncating interactions for calculating coordination numbers. Refer to eq. (10) of [Pracht2019].

- `dftb_rep_cutoff: 50.0 ang`

  [Real Physical, default `dftb_rep_cutoff: 40.0 bohr`].
  Cutoff distance for truncating interactions for calculating repulsion energy. Refer to eq. (3) of [Pracht2019].

- `dftb_srb_cutoff: 20.0 ang`

  [Real Physical, default `dftb_srb_cutoff: 14.14 bohr`].
  Cutoff distance for truncating interactions for calculating short-range basis correction energy. Refer to eq. (4) of [Pracht2019].

- `dftb_ewald_parameter: 5.0 ang-1`

  [Real Physical, default `dftb_ewald_parameter: -1.0 bohr-1`].
  If positive, this value is used as the parameter for Ewald summation for periodic electrostatic interactions. Otherwise, the optimum parameter for ewald summation is calculated on the fly.

- `dftb_cartesian_ngwfs: T/F`

  [Boolean, default `dftb_cartesian_ngwfs: F`].
  If true, the program uses Cartesian Gaussian orbitals, otherwise the program uses spherical orbitals as basis. Note that the currently implemented GFN0 method has been developed for a basis of spherical orbitals and may give incorrect results with Cartesian orbitals.

- `dftb_overlap_analytical: T/F`

[Boolean, default `dftb_overlap_analytical: T`].
If false, elements of the overlap are calculated via integrals on grid, otherwise analytically. Note that the gradients of overlap matrix on grid are not yet implemented.

13.1.3 Acknowledgement

We would like to thank Loukas Kollias, Denis Kramer and John R. Owen for useful discussions.

CHAPTER

FOURTEEN

PERFORMANCE CONSIDERATIONS

Author
Jacek Dziedzic, University of Southampton

How quickly ONETEP runs your calculations depends on a number of factors. Choosing the number of nodes, processes and threads has already been described in *Running ONETEP in parallel environments*.

Here we mention techniques that can be used in addition to an efficient parallel decomposition to improve performance.

14.1 Fast sparse-to-dense and dense-to-sparse conversions

In many scenarios, ensemble DFT calculations being a notable example, ONETEP needs to convert between two matrix representations – a distributed sparse one ("SPAM3"), and a distributed dense one ("BLACS"). These conversions happen multiple (hundreds) times in a run, and involve substantial data communication. There are two ways in which ONETEP can do these – conveniently termed "slow" and "fast".

The slow approach is the only approach available until ONETEP 6.1.21. Starting from ONETEP 6.1.22, a fast approach is also available, *and is the default*. This means that no action is required on your part to use the fast approach. The fast approach is up to several times faster (for this part of the calculation).

You can check which approach you are using by examining the timings printed out at the end of your calculation (provided you are not using `timings_level 0`). If your timings include `sparse_spam3toblacs_real_fast` and `sparse_blacstospam3_real_fast` – you are using the fast approach. If instead you have `sparse_spam3toblacs_real_slow` and `sparse_blacstospam3_real_slow` – you are using the slow approach.

To switch between the approaches use:

- `fast_dense_to_sparse T` and `fast_sparse_to_dense T` – for the fast approach,
- `fast_dense_to_sparse F` and `fast_sparse_to_dense F` – for the slow approach.

The fast approach is currently incompatible with the image comms subsystem needed by NEB. This means you cannot use the fast approach with NEB. Please add `fast_dense_to_sparse F` and `fast_sparse_to_dense F` to your input file when using NEB.

14.2 Fast density calculation (for users)

This is a user-level explanation – for developer-oriented material, see *Fast density calculation (for developers)*.

Calculating the electronic charge density is one of the more time-consuming operations in ONETEP. In a typical calculation it has to be performed hundreds of times. There are two ways in which ONETEP can calculate the density – conveniently termed "slow" and "fast".

The slow approach is the only approach available until ONETEP 7.1.7. Starting from ONETEP 7.1.8, a fast approach is also available, *but it is not the default*. This means that action is required on your part to use the fast approach. The fast approach is up to several times faster (for this part of the calculation).

The fast approach works best for "serious" systems, it's not meant to address scenarios with KE cutoffs below 700-800 eV or NGWFs smaller than 8.0 a0.

To switch between the approaches use:

- `fast_density T` – for the fast approach,
- `fast_density F` – for the slow approach.

Be aware that the fast approach *is an approximation*. The approximation is well-controllable, meaning you can get as close with the accuracy to the slow (exact) approach as you like, albeit sacrificing performance as you do so. Conversely, you can make the fast approach as fast as you like, but you will be sacrificing accuracy as you do so, up to a point of making your results worthless. This means care must be taken when changing the parameters of this approach – non-expert users are advised to use the defaults, or even "safe settings" described below.

You can check which approach you are using by examining the timings printed out at the end of your calculation (provided you are not using `timings_level 0`). If your timings include `density_on_dbl_grid_fast` and `density_fast_new_ngwfs` – you are using the fast approach. If instead you have `density_on_dbl_grid` and `density_batch_interp_deposit` – you are using the slow approach.

The main idea behind the fast density approach is *trimming* interpolated NGWFs, that is, ignoring the points where their values are below a prescribed threshold. The value of this threshold, set by `fast_density_trim_threshold` is the main parameter controlling the balance between accuracy and efficiency. It is also independent of NGWF radii. The default is `2E-6`. The parameter already includes grid weights, so you *do not* need to adjust it when changing `psinc_spacing` or `cutoff_energy`. To make your calculation more accurate, decrease the threshold – probably not below `1E-7`. To make your calculation faster, increase the threshold – probably not above `1E-5`.

If you use `fast_density_output_detail VERBOSE` (or higher), ONETEP will print out an estimate of the accuracy of the approximation every time NGWFs change. It will look like something like Fig. 14.1, see the *accuracy of approximation* line. This tells you to how many digits the approximated NGWF charge is equal to the exact (double FFT-box) NGWF charge, in the root-mean-square sense over all NGWFs in the system. In this example our approximated charge is no further from 1.0 (a correctly normalized NGWF) than by 1E-7 (and is slightly closer, because we got 7.16, not 7.0).

As your calculation progresses, this value will fluctuate, and is likely go down slightly, as the NGWFs become more diffuse. As a rule of thumb, if it gets below 5.0-6.0, you will have difficulty converging NGWFs to the default threshold. If it is above 9.0, you are probably using too much accuracy, losing efficiency as you do that.

Another notable quantity in Fig. 14.1 is the *estimated high-memory watermark per MPI rank* (shown in yellow). This is a reminder that the fast density approach uses significantly more memory than the slow approach. The value in the printout is the expected *maximum* memory that fast density uses *per MPI rank*. If your printout is truncated before you reached this line, you most likely already ran out of memory. At this stage, we use an all-or-nothing approach – there is no way to give the algorithm a memory allowance and tell it that it should not consume more. Work on this is in progress. The best way to reduce memory load is to use fewer processes per node and more threads. If this is not sufficient, you can reduce the memory load by using more nodes, but this is not a linear dependence – i.e. you will *not* reduce the load by a factor of two if you add twice as many nodes. Finally, note that what is printed out is the amount of memory consumed by the fast density approach, not by all of ONETEP.

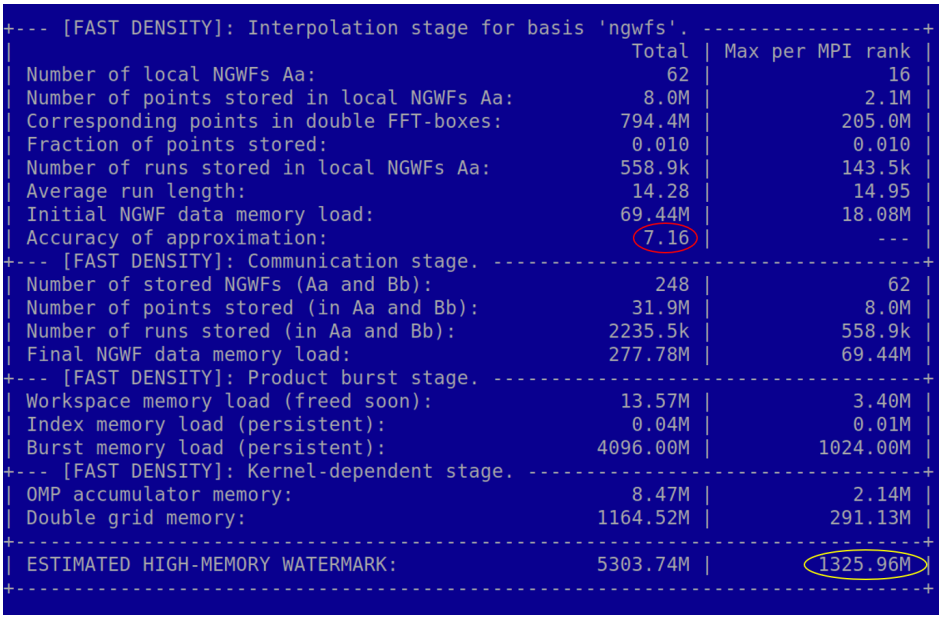

Fig. 14.1: The summary printed by fast density every time the NGWFs change. Of main interest are: *accuracy of approximation* (shown in red) and *estimated high-memory watermark per MPI rank* (shown in yellow).

14.2.1 When is fast density used?

Fast density is only used for energy evaluations done from `hamiltonian_mod` – via `hamiltonian_lhxc_calculate()` and `hamiltonian_energy_components()`. These are the costly density calculations, because they are done hundreds of times in the course of a calculation. All other density calculations (done in forces, properties, eigenstates, linear response, lr_tddft, population, dma, dmft, EDA, implicit solvent restarts) are always done using the exact (slow) method. The rationale is that these are done much less often and possibly require more accuracy.

If you want to know when the fast and slow routines are called, specify `fast_density_output_detail PROLIX` or higher.

14.2.2 More accuracy

The default settings should give you sufficient accuracy to converge NGWFs to the default threshold and to get energies and forces that are negligibly different from those obtained with the slow approach. However, for more difficult systems, particularly if using low kinetic energy cutoffs (say, below 700 eV – like would probably be used with PAW), you might need to adjust the parameters to get desired accuracy.

In addition to adjusting `fast_density_trim_threshold` down (to perhaps 1E-6 or 5E-7), you may want to use `fast_density_off_for_last T` (the default is `F`). This will tell ONETEP to use the slow (but exact) approach for the final energy evaluation. You will know this happened by examining the output file and looking for:

```
! Looks like the last energy evaluation.
! The fast density calculation will now be disabled in the interest of accuracy.
```

Note that this will not be printed if `fast_density_output_detail` is `BRIEF` or if fast density would already have been switched off by `fast_density_elec_energy_tol` (see below). This setting resets any time you start a new NGWF convergence loop – that means that in auto solvation, geometry optimisation, MD, etc. each optimisation will start with fast density turned on.

Also note that this switching is done in the NGWF convergence loop. If you are working with fixed NGWFs (`maxit_ngwf_cg 0` (or negative)), this switching will not take place.

Furthermore, particularly if your calculation struggles to converge to the default NGWF threshold, you can set `fast_density_elec_energy_tol`. This is the energy change per atom between NGWF steps below which ONETEP will switch to the slow (but exact) approach. It's the same quantity that is used as the energy convergence criterion in `elec_energy_tol`. The default is `1E-50`, effectively turning this off. Setting it to `1E-7` will typically have ONETEP switch to the slow approach for the last few NGWF iterations. The higher you set this, the sooner ONETEP will switch to the slow approach. This, of course, eats into your efficiency gain. You will know if and when this happened by examining the output file and looking for:

```
! Energy change per atom: 0.30287E-07 Eh < 0.10000E-06.
! The fast density calculation will now be disabled in the interest of accuracy.
```

Note that this will not be printed if `fast_density_output_detail` is `BRIEF`. This setting resets any time you start a new NGWF convergence loop – that means that in auto solvation, geometry optimisation, MD, etc. each optimisation will start with fast density turned on.

Note that you need at least two NGWF iterations to have a meaningful energy change to examine, so this setting has no effect if you take fewer than two NGWF iterations.

14.2.3 Remaining options

The default output detail of fast density is the same as specified for `output_detail`. You can set it separately by specifying `fast_density_output_detail`. The available options are the same as for all ONETEP output details: `BRIEF`, `NORMAL`, `VERBOSE`, `PROLIX` and `MAXIMUM`.

If, in the future, other methods of trimming NGWFs than by using a fixed threshold become available, you will be able to use `fast_density_trim_by` to control these. Currently the only supported option is `VALUE`.

14.2.4 Example settings

For a quick-and-dirty calculation use:

- `fast_density T`
- `fast_density_trim_threshold 2E-5`.

For a typical calculation just use:

- `fast_density T` (which will use the default of `fast_density_trim_threshold 2E-6`).

For an accurate, but slower calculation use:

- `fast_density T`
- `fast_density_trim_threshold 1E-6`
- `fast_density_off_for_last T`
- `fast_density_elec_energy_tol 1E-7`.

For very safe settings that should provide a modest gain in efficiency, try:

- `fast_density T`
- `fast_density_trim_threshold 5E-7`
- `fast_density_off_for_last T`
- `fast_density_elec_energy_tol 3E-7`.

14.2.5 Compatilibity

Fast density is known to work (to the best of our knowledge) with the following additional functionalities:

- extended NGWFs,
- PBCs and OBCs,
- implicit solvation,
- hybrid functionals and Hartree-Fock exchange,
- `fine_grid_scale` larger than `2.0`,
- PAW,
- DFT+U,
- conduction,
- MD,
- geometry optimisation,
- TS search,
- NEB,
- EDFT and LNV.

Fast density is known *not* to work (this we know with certainty) with the following additional functionalities:

- complex NGWFs,
- TD-DFT (mixed bases are not supported at this point).
- EMFT (regions).

ONETEP will stop with an error if either of these is used with `fast_density T`.

CHAPTER

FIFTEEN

DEVELOPER AREA

Author
Jacek Dziedzic, University of Southampton

Author
James C. Womack, University of Southampton

Author
Nicholas Hine, University of Warwick

15.1 Advice for Contributors

15.1.1 Pre-pull-request checks

The ONETEP source code is distributed with a number of useful scripts for checking some aspects of the correctness of source code. Before creating a pull request, you should ensure that these tests all pass for the branch you want to merge in your private fork:

1. **Check module dependencies** and a few other common issues. Run

   ```
   make checkdeps
   ```

   or, equivalently,

   ```
   ./utils/check_dependencies
   ```

   in the root directory of the repository. If any errors are found, fix them.

2. **Check whitespace**. Run

   ```
   ./utils/check_whitespace
   ```

   in the root directory of the repository. This script can automatically fix whitespace issues – see the output of `./utils/check_whitespace -h` for details.

3. **Check use clauses**. Run

   ```
   ./utils/check_use_clauses src/yourmodule_mod.F90
   ```

   in the root directory of the repository, replacing `yourmodule` with the module you worked on. If there was more than one, do this for every module separately. This script checks for unnecessary use clauses and will help point them out. You are only responsible for the use clauses in the code you touched.

4. You must also run the full QC test suite before creating a pull request. Instructions for that can be found here: *Compiling, testing and running ONETEP*.

Key points about running the QC test suite before creating a pull request:

- Never run the QC tests in the actual repository. Create a copy of your ONETEP installation directory (e.g. `cp -a onetep_jd onetep_jd_qc`) and run the tests in the directory of the copy. Much of the stuff in the install is not needed to run the tests (e.g. all of `src/`), but it's just less hassle to copy the entire thing than pick out the necessary subdirectories.

  Why not run the QC tests in the actual repository? Because they produce outputs, which can become messy to clean up back to pristine state, polluting your repository with spurious changes. This becomes more problematic because symbolic links are used in the QC test suite. Trying to update your repository via `git pull` or `git fetch` may be tricky when you have unfinished or improperly cleaned up QC tests. Just don't run them straight in the repository.

- You should run the test suite with ONETEP compiled with a recent version of a widely used Fortran compiler, e.g. GNU Fortran, Intel Fortran Classic, Cray Fortran.
- The tests should be run in a parallel environment (ONETEP must be compiled with MPI support), ideally also making use of OpenMP threading (set the environment variable `OMP_NUM_THREADS` to a value >1, or use the `-t` argument to `onetep_launcher`).
- No tests should fail, though warnings are acceptable.

If tests fail, please investigate this further before creating your pull request:

- If you updated your local copy (`git merge` or `git pull`) rather than cloning it from scratch (`git clone`), did you remember to `make cleanall` before compiling?
- Check out the master branch from the official repository (ensuring that it is fully updated by fetching from it and merging) and run the failing tests on this.
- If the relevant tests do not fail upstream (i.e. for the master branch on the official repository), you should fix the code in your fork before creating your pull request.
- If the relevant tests also fail upstream, please check the list of issues in the official repository (https://bitbucket.org/onetep/onetep/issues, soon to move to https://github.com/onetep-devel/onetep/issues): there should be an open issue with relevant information. If such an issue has not been reported, please create a new issue explaining which tests are failing and any details about how the failures occur.

If the tests are not failing upstream, the problem is likely to be part of the changes you have implemented and you will be the best person to find it and fix it. If you need help, and want to discuss the issue with other developers, you may create an issue in your fork (not in the official repository) and draw their attention to it via email or on the ONETEP development Slack at https://onetep-devel.slack.com/.

If your tests fail in a way in which the official master branch is known to fail, your pull request may be acceptable – you can go ahead creating the pull request, but make sure that the text of the pull request includes a list of the tests that fail and why you think that this is not a problem.

Ensure that the ONETEP version number in your `src/onetep.F90` file is updated appropriately (see *Version numbering* for a description of ONETEP's version numbering system). If multiple pull requests are open, or it takes time to merge your pull request, it may be necessary to update the version number again during the pull request code review. This is to ensure that the version number increments linearly in the official repository's `master` branch.

Read the next subsection on style conventions to ensure your code adheres to them.

15.1.2 Style conventions

In order to keep the ONETEP codebase relatively consistent and readable, we have adopted the following conventions:

- Each line of code should be 80 columns or fewer in length, as this makes comparison of two versions of code side-by-side easier (some older code does not adhere to this, but all new code should).
- If a line exceeds 80 columns in length, use continuation characters (&) to break it into chunks of <=80 characters (remember to include an extra & at the beginning of the following line when breaking in the middle of a string literal). Do not add & in the next line otherwise, it's unnecessary.
- Always use spaces for indentation, never tabs.
- The blocks in the `do` loop, `select case` (also `select type`) and `if` constructs should be indented by **3 spaces**.
- The contents of subroutines and functions should be indented by **2 spaces**. Use further 2 spaces for internals.
- Line continuations should be indented by **5 spaces** for continued lines.
- Ensure there is no trailing whitespace before you commit (you can use the `./utils/check_whitespace` script described in *Pre-pull-request checks* to check this).
- In general, adding or removing blank lines should be avoided in core modules, as these changes will appear in the commit history.

15.1.3 New functionality

If you add new procedures or significantly change existing procedures, **you must create or update the documentation in the source code**. Examples of how to document procedures can be found in the template module `template_mod.F90` in the `doc` directory. The key components of the documentation of procedures are:

- A human-readable description of the new functionality.
- A list of the arguments and a description of their meaning.
- The author(s) and a changelog describing significant modifications.

When adding new functionality which does not fit into other modules, it may be necessary to create a new source file containing a new module. Note that procedures and variables should always be encapsulated in modules, not 'bare' in a source file.

Before creating a new module, you should consider carefully whether your new functionality fits within the framework of an existing module, or is generic enough to be part of a multi-purpose module, such as *utils* or `services`. If a new module is needed to encapsulate some new functionality, then you should follow the following guidelines:

- Give your module a name which indicates the functionality it contains. If unsure, consult a more experienced developer to discuss an appropriate name.
- The filename for the module should have the form `<module_name>_mod.F90`, where `<module_name>` is the name you have given the module.
- By default, variables and procedures in your module should be private (i.e. they should have the `private` attribute).
- Global module-wide variables (private or public variables declared at the level of the module itself rather than within its routines) constitute "hidden state", which tends to make the behaviour of a routine undesirably dependent on more than just the arguments it is called with. Sometimes these are unavoidable, and there are instances of them in the code. However, they should be minimised as much as possible. Think carefully before declaring any module-level global variables. More experienced developers may be able to suggest ways to encapsulate data inside arguments to routines such that they do not constitute "hidden state".

- Variables and procedures which do have to be public (accessible outside the module) should be explicitly specified (i.e. they should have the `public` a ttribute).
- In general, public variable and procedure names should be prepended by a standard prefix (typically the module name, or a shortened version of the name).

It is recommended that you make a copy of `./doc/template_mod.F90` and use this as a starting point for your new module, as this will make following the above guidelines easier.

15.1.4 Version numbering

There are three parts to the version number, both for development versions and release versions. The first version number is only very rarely incremented by a collective decision of the main authors of the code (ODG). New major versions are released around every 6-12 months and are indicated by incrementing the second number in the full version number (e.g. "2" in "4.2").

The major version number (second number in full version number) indicates whether the associated source code is a release version or a development version:

- Release versions (which are distributed to users) have an *even* second number.
- Development versions (which are under active development) have an *odd* second number.

Within a series with the same second version number, successive versions (indicated by the third number in the full version number) should be compilable and complete with respect to a given new feature.

For minor changes in development versions (e.g. a bugfix or minor change to existing code), we no longer increment the fourth number (which is now absent altogether) to avoid merge conflicts when this is done by multiple people. The script `utils/embed_version_info_in_banner` ensures that pertinent details of the local repository (branch, remote, last commit ID, list of locally modified files) are included in the ONETEP banner during compilation, but they do not go into the repository. Major changes (e.g. a new module or overhaul of existing functionality) should increment the third number.

Bugfixes to a release version (merged into the corresponding release branch on the official repository, e.g. *academic_release_v5.0*) should increment the last (third) number in the full release version number.

At any given time, there is a development version and a release version differing in their second version number by 1.

- 4.0.0 <– first release of v4
- 4.0.1 <– bugfix to v4 in git branch for release
- 4.1.0 <– first development version of v4 (initially same as 4.0.0)
- 4.1.0 <– minor development work (changes will be summarised in banner)
- 4.1.1 <– significant development work
- 4.2 RC3 <– release candidate 3 for v4.2
- 4.2.0 <– next release version
- 4.3.0 <– next development version (initially same as 4.2.0).
- 5.0.0 <– first release of v5

15.2 Preventing accidental pushes to the official repository

- GitHub users in the *Owner* role of the ONETEP repository have write access to the official repository.
- *Owners* may want to take steps to avoid accidentally pushing work to the official repository if they have added this as a remote to their private fork.
- This can be achieved by setting the push address for the remote to an unresolvable URL, e.g.:

```
git remote set-url --push github_official DISABLE
```

15.3 Fast density calculation (for developers)

This section describes the "fast density" approach introduced in ONETEP 7.1.8 in January 2024. This is developer-oriented material – for a user manual, see *Fast density calculation (for users)*.

We focus on the calculation on the double grid. If `fine_grid_scale` is different from 2.0, the density gets interpolated from the double to the fine grid, regardless of the approach for calculating the density on the double grid.

A denotes atoms local to a process. *B* denotes atoms that S-overlap with atoms *A*, they are, in general, not local. NGWFs on *A* are indexed with *a*, NGWFs on *B* are indexed with *b*, and so *Aa* and *Bb* can be used to index NGWFs globally. NGWFs *Aa* are local, NGWFs *Bb* are, in general, not. Depending on who you ask, *Aa* NGWFs are sometimes termed "col", "ket" or "right"; *Bb* NGWFs are sometimes termed "row", "bra" or "left".

The usual ("slow") method for calculating electronic density in ONETEP proceeds as follows:

```
// slow density
for all local atoms A {
  dens_A = 0
  for all NGWFs a on A {
    (1) Transfer \phi_Aa to FFT-box.
    (2) Calculate \rowsum_Aa = sum_\Bb K^Aa,Bb \phi_Bb in FFT-box.
    (3) FFT-interpolate \phi_Aa and \rowsum_Aa to a double FFT-box.
    (4) Multiply dens_Aa = \phi_Aa * \rowsum_Aa in double FFT-box.
    (5) dens_A += dens_Aa.
  }
  (6) Deposit box containing dens_A to double grid.
}
```

All `\phi_Aa` are local, so are `K^Aa,Bb`. Stage (2) involves comms of `\phi_Bb` in PPDs via `function_ops`. Stage (6) involves comms of atom-densities `dens_A` via `cell_grid_deposit_box()`. The algorithm proceeds in batches (to conserve memory), and is OMP-parallelised. Comms must those be perofmed carefully, are done from `$OMP MASTER` regions in `function_ops_sum_fftbox_batch()` and `$OMP CRITICAL` in `density_batch_interp_deposit()`.

The number of FFTs done is $2N_{\text{NGWF}}N_{\text{outer}}N_{\text{inner}}$, where N_{NGWF} is the number of NGWFs in the system, N_{outer} is the number of outer (NGWF) loop iterations, N_{inner} is the number of inner (LNV, EDFT) iterations. We ignore line searches in this estimate for simplicity. In practice real FFTs are done in pairs throught a complex FFT, but we ignore this for simplicity.

The main drawbacks of this approach are:

1. Having to repeat FFTs on `\phi_Aa` in the inner loop, even though they do not change.
2. Having to repeat FFTs on `\rowsum_Aa` in the inner loop, because `K^Aa,Bb` changes.
3. Interspersing comms with calculation in `function_ops_sum_fftbox_batch()`, which makes GPU-porting difficult, and comms tricky.

4. Having to calculate products of \phi_Aa and \phi_Bb in double FFT-boxes, even though "what matters" is almost exclusively contained in the double tight-box of \phi_Aa. We need to do the whole double FFT-boxes because of Fourier ringing from the interpolation.

5. Multiple depositions (and comms) to the same points in (6), because boxes of nearby A overlap.

(1) cannot be addressed directly by storing the interpolated \phi_Aa for the duration of the inner loop, because we cannot afford to store full double FFT-boxes.

The fast density approach hinges on the fact that we can get sufficient accuracy even if we restrict ourselves to a subset of the interpolated points in the double FFT-boxes. It turns out that only ~1-5% of the points in the double FFT-box are needed to recover 99.9999-99.999999% of the charge of the NGWF (which is one electron). This approach is, thus, an approximation, but it is well-controllable.

By using only a fraction of the points in the double FFT-boxes, we are able to store the interpolated NGWFs. This lets us address (1) directly – we only interpolate \phi_Aa at the beginning of the inner loop, and can now afford to store the interpolated versions.

We address (2) and (3) by first interpolating only \phi_Aa, and then communicating them to where they are needed (and where they become \phi_Bb). We use remote_mod for that, which separates the comms from the FFTs. Of course we only communicate the relevant points, not the entire double FFT-boxes. Thus, we longer have to interpolate sum_\Bb K^Aa,Bb \phi_Bb, and we avoid doing FFTs in the inner loop entirely. The number of FFTs is now $N_{\mathrm{NGWF}}N_{\mathrm{outer}}$ (we interpolate all NGWFs every time they change), which saves 1-2 orders of magnitude in the number of FFTs. There is a price to pay, though: we need memory to store the interpolated NGWFs, we have to communicate interpolated NGWFs rather than coarse-grid PPDs, and we need to do sum_Bb K^Aa,B phi_Bb on the new representation somehow. If the latter can be done efficiently, we are addressing (4) above, too.

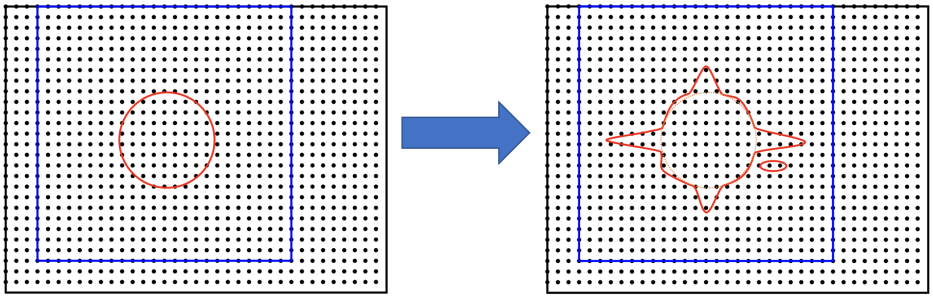

Fig. 15.1: "Spilling" of an NGWF due to Fourier interpolation. Left panel – NGWF (red) in an FFT-box (blue) on the coarse grid in the simulation cell (black). Right panel – NGWF (red) in a double FFT-box (blue) on the double grid in the simulation cell (black). **The grid on the right is twice as fine** (not shown). The orange dashed line shows the original shape of the NGWF.

The main question is how to store and manipulate interpolated NGWFs accurately and efficiently, that is, how to store and manipulate the points encompassed by a chosen isoline (red) in Fig. 15.1, right panel. The region of interest does not have to be contiguous.

The naive approach of storing just the points in the "double tight-box" of the NGWF (as shown Fig. 15.2) turns out to be almost sufficiently accurate, but not quite, to get convergence to default RMS thresholds. It's also not controllable.

An alternative is to choose a threshold for the values of the NGWF and to construct the smallest cuboid (or actually a parallelepiped) that encompasses all those points, as shown Fig. 15.3. This is controllable (via the threshold), but not very economical, because we store many more points than are necessary. This is because the ringing extends almost exclusively along the axes of the cell, practically to the faces of the double FFT-box. We'd like to keep this ringing, but not the points in the bulk of the double FFT-box.

We can find which points we need (all points whose value is below a chosen threshold), and then remember their positions and values, like shown in Fig. 15.4. For positions it is convenient to use a linear index on the double grid – this makes them absolute (rather than relative to the double FFT-box), and permits handling PBCs at this stage.

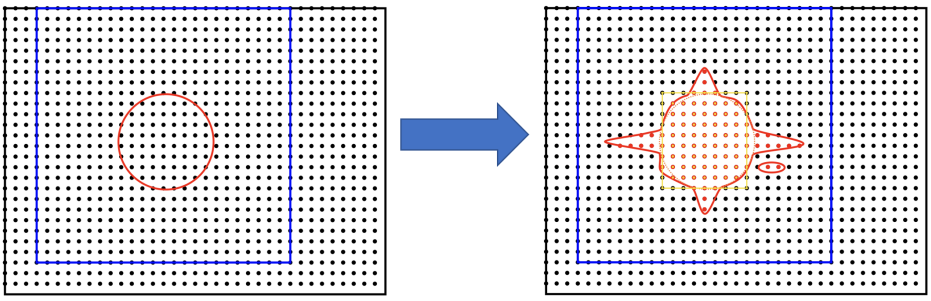

Fig. 15.2: Storing only the points in the double tight-box (shown in orange) is not sufficiently accurate. Some points that matter are contained in the Fourier ringing that is outside the tight-box.

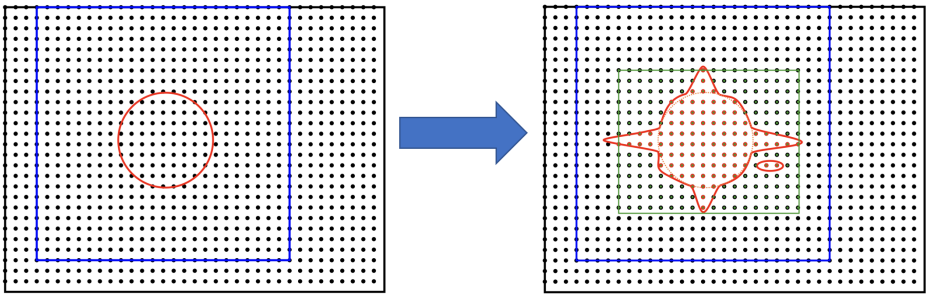

Fig. 15.3: Storing all the points in a box that covers an isoline is impractical, because the Fourier ringing extends far along the box axes, and we wind up storing points that are irrelevant (shown in green and not in red).

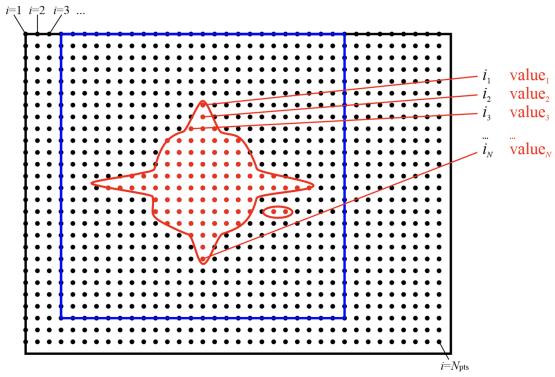

Fig. 15.4: Storing "just the points we need", in a naive fashion, by storing *where* (linear indices on the double grid), and *what* (values). This automatically takes care of the non-contiguity of the data we want to store.

Of course, such indexed approach is inefficient. Manipulating this representation is slow, and it takes more space than needed – we need to store an integer index in addition to every double precision value. However, we can exploit the fact that much of the region of interest is contiguous. This lets us proceed with a run-length encoding, as shown in Fig. 15.5 – we can store *starting positions*, *run lengths* (counts), and *values*.

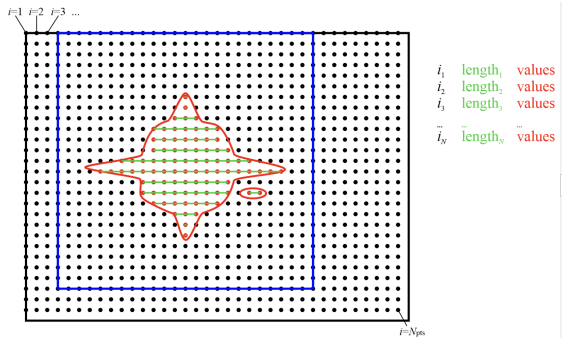

Fig. 15.5: Storing "just the points we need", using RLE compression, by storing *where* (linear indices on the double grid), *how many* (run lengths) and *what* (values). This automatically takes care of the non-contiguity of the data we want to store, but is efficient. Note that there are, in general, multiple values (red) for every *run* (shown in green).

For typical thresholds, the average run length is about 30. That means we only have to store two integers for every 30 double precision real values, so the overhead is minimal. Recall that we only keep 1-5% of the points in the double FFT-box, which means we can afford storing and communicating such *compressed trimmed NGWFs*. As will be shown later, manipulating them can also be done efficiently.

Recall that we are interested in calculating *products* of such NGWFs, and this has to be done in the inner loop, because even if we can afford to store interpolated NGWFs, we cannot afford to store entire products, there's just too many of these. However, we can store information on which parts of the NGWFs overlap, which boils down to determining which runs in `\phi_Aa` overlap where with which runs in `\phi_Bb` and by how much. These overlaps are the *bursts* in which we will calculate sums and products later.

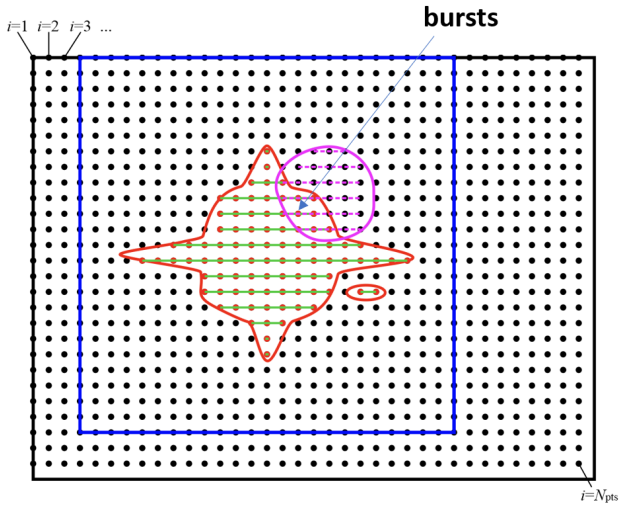

Fig. 15.6: To calculate the product between two compressed trimmed NGWFs, we first determine *bursts*, i.e. the overlaps between runs of NGWF *Aa* (green) and runs of NGWF *Bb* (magenta). Bursts are shown as green-magenta dashes.

The bursts can be determined outside of the inner loop and stored. Bursts do not store any information on the values, only start and end indices to the values in `\phi_Aa` and `\phi_Bb`. Using this information we can calculate `\sum_\Bb K^Aa,B \phi_Bb` more efficiently in the inner loop.

The fast density approach thus proceeds in two stages – one that is performed every time NGWFs change, and one that is performed in the inner loop. In pseudocode they look as follows:

```
// density_fast_new_ngwfs():
OMP for all local NGWFs Aa {
  (1) FFT-interpolate \phi_Aa to a double FFT-box.   } Happens in
  (2) Trim the data in double FFT-box, RLE-compress. } density_fast_interp_ngwfs().
  (3) Pack the result in an array of doubles.        }
}
(4) Communicate the necessary trimmed Bb NGWFs from remote ranks
    via remote_comm_trimmed_ngwfs_of_neighbours().
(5) Determine product bursts via trimmed_ngwfs_prepare_bursts():
    for all local NGWFs Aa {
      OMP for all atoms B that overlap with A {
         for all NGWFs b on B {
           Determine bursts between Aa and Bb.
         }
      }
    }
```

Only stage (4) involves MPI comms.

The density-kernel-dependent stage that needs to be done inside the inner loop looks like this:

```
// density_on_dbl_grid_fast():
for all local NGWFs Aa {
  accum_Aa = 0
  (1) OMP for all Bb that overlap Aa {
        (2) Accumulate rowsum_A = K^AaBb \phi_Bb using bursts.
      }
  (3) Deposit \phi_Aa * rowsum_A to a proc-local double grid.
}
(4) Flatten proc-local double grids to usual distributed representation.
```

Only stage (4) involves MPI comms.

Typical speed-ups obtained using this approach range from 2x to 6x for the total time spent calculating the density, and between 10% and 50% can be shaved off the total calculation walltime.

The main drawback of this approach is increased memory consumption. There are two main components:

(A) The trimmed NGWF data itself, which is, to a large extent, replicated. A single trimmed NGWF can be needed on many processes, because it could be a \phi_Bb to many NGWFs Aa. Moreover, this memory requirement does not scale inverse-linearly with the number of processes. That is, increasing the node count by a factor of two doesn't reduce the memory requirement by a factor of two, because there is more replication.

(B) The burst data.

Both (A) and (B) depend on the trimming threshold, and the shape of the NGWFs. Both tend to increase during the NGWF optimisation as the NGWFs delocalise somewhat.

A typical plot of the memory used by both approaches is shown in Fig. 15.7. For this calculation the slow approach took 1893s (33.3% of the total walltime), and the fast approach took 727s (14.8% of total walltime), for a speed-up of 2.6x. One-eighth of the total walltime was shaved off. Of the 727s only 66s were spent doing FFTs.

The accuracy of the fast density approach can be demonstrated on an example – we consider the binding energy of mannitol to 300 water molecules. The system is shown in Fig. 15.8.

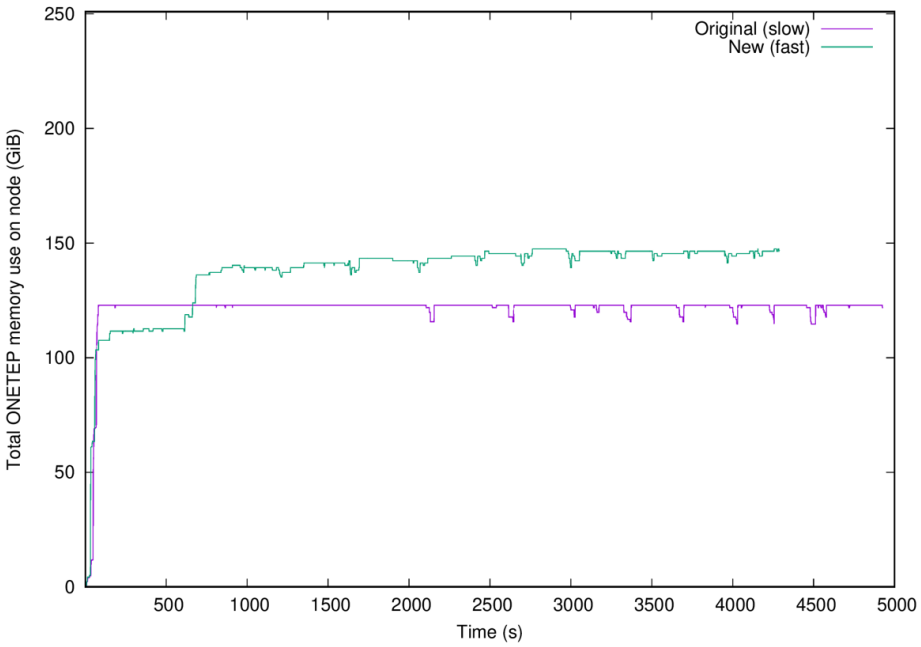

Fig. 15.7: Memory used by the slow approach (magenta) and the new approach (cyan) for a calculation on ethylene carbonate (4000 atoms) on 32 nodes of Archer2. 9 a0 NGWFs, 829 eV, EDFT. 117x117x117 FFT-box, 140x140x140 cell. 16 OMP threads were used.

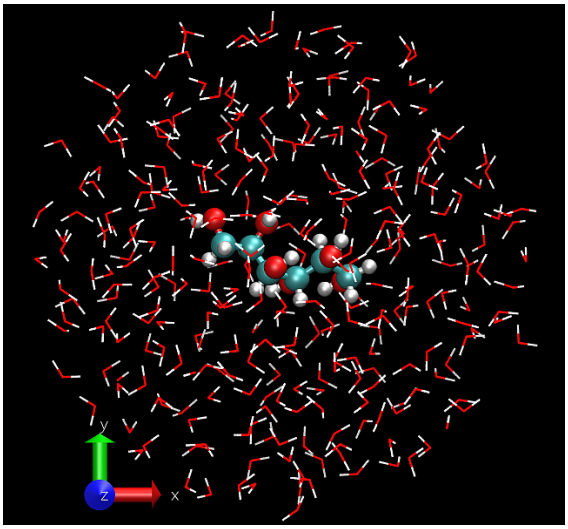

Fig. 15.8: Our testcase: mannitol bound to 300 water molecules (926 atoms). 811 eV KE cutoff, 9 a0 NGWFs, 8 LNV iterations.

I ran this testcase on 16 nodes of HSUper (8 processes on a node, with 9 threads each, saturating all 72 CPU cores on a node) – both the complex and the water shell. The mannitol itself was ran on 1 node, because it's small. I used the default settings, only specifying *fast_density T*.

Quantity	Slow density	Fast density
Binding energy (kcal/mol)	-79.9027	-79.9014
Total walltime	1016s	718s (-29%)
Time to calculate density	520s	239s
Time spent doing FFTs	160s	23s
High-mem watermark on a node	55.6 GiB	69.7 GiB

As seen from the above table, we are accurate to ~0.001 kcal/mol, while shaving off 29% from the total walltime. The density calculation itself was faster by a factor of 2.2.

I also performed scaling tests, using a 701-atom protein scoop at 827 eV, 9a0 NGWFs, 8 LNV iterations. I only ran it for 4 NGWF iterations because it's not properly terminated, which leads to a zero band-gap, meaning it cannot be reliably converged to default thresholds (with either density method). These were run on different numbers of nodes of HSUper, with 8 processes and 9 OMP threads per node.

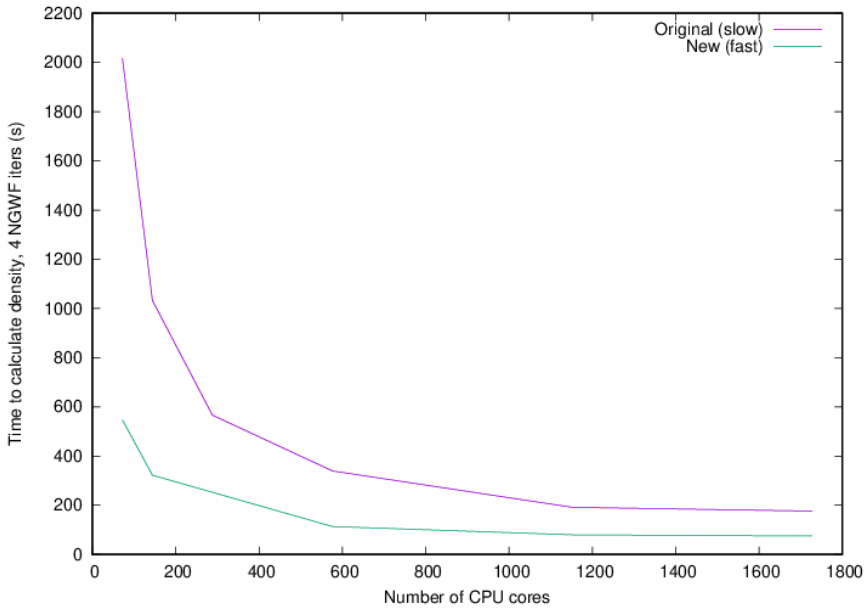

Fig. 15.9: Time to calculate density in 4 NGWF iterations of a 701-atom protein scoop.

Fig. 15.9 shows the walltime of the density calculation for the slow and fast approaches, while Fig. 15.10 shows the strong scaling with respect to one node. It is clear that the fast approach is quite a bit faster than the original approach, although it does not scale that well to high core counts. Keep in mind that we pushed this system quite far by running 701 atoms on over 1600 CPU cores.

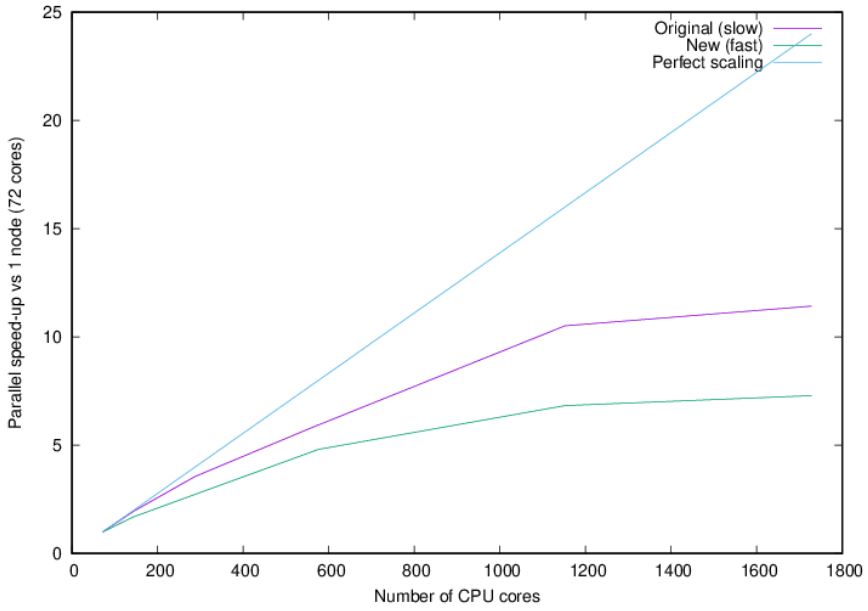

Fig. 15.10: Strong parallel scaling of the two methods of calculating the density in for a 701-atom protein scoop.

15.3.1 Directions for improvement

This approach could be improved in a number of ways:

1. Setting up an allowance for used RAM, similarly to what is done in HFx. This would enable graceful performance degradation in low-memory scenarios. This could be achieved by not storing all the bursts, only those that fit within the allowance. The remaining bursts would have to be recalculated on the fly in the inner loop.
2. Making it compatible with EMFT, complex NGWFs and mixed bases.
3. Adding an `$OMP SCHEDULE` toggle between `STATIC` and `DYNAMIC` in `density_on_dbl_grid_fast()` for more control over determinism vs efficiency. Currently we use `SCHEDULE(STATIC)` to get more deterministic results, but `SCHEDULE(DYNAMIC)` offers better efficiency. Toggling this at runtime is not trivial (`omp_set_schedule()`).
4. Having a dynamic *fast_density_trim_threshold* – we could probably start the NGWF optimisation with a cruder approximation, tightening it as we go along.
5. Dynamically selecting `MAX_TNGWF_SIZE`. It's currently a constant, and `persistent_packed_tngwf` is not an allocatable.
6. A smarter way to flatten the computed density. Currently each process has their own density that spans the entire cell and only contains contributions from the *Aa* NGWF it owns. We flatten it by a series of reduce operations over all nodes. This is the main killer of parallel performance.
7. We should also consider a more FFT-heavy approach in which we only FFT-interpolate local (*Aa*) NGWFs, but communicate *Bb* on the coarse grid, in PPDs. Then we'd use the latter to build `rowsum_Aa` in FFT-boxes, just like in the old scheme (except with the comms done in advance) in the kernel-dependent stage. These would be then FFT-interpolated like in the old scheme, but then trimmed and the rest would proceed like in the fast scheme. This approach would conserve memory (no storing of individual trimmed Bb's, only `rowsum_Aa`'s, ditto for bursts), and do fewer comms (communicating NGWFs in PPDs via `remote_comm_ngwfs_of_neighbours()`). There would fewer burst ops, because we'd be dealing with entire rowsums instead of individual *Bb*. But we would be repeating FFTs in the inner loop.

15.4 History

The earliest versions of the code, dating back to before 2005, were committed to a revision control system based on CVS. These files are still available on the TCM filesystem at /u/fs1/onestep/CVS_REPOSITORY. In around 2009 we moved to Subversion, using a repository still hosted on the TCM filesystem in Cambridge. The SVN history was migrated to Bitbucket, and subsequently to GitHub (2023) and can still be browsed within the current Git repository (though attribution to authors is often not correctly recorded).

In June 2018, the ONETEP project was migrated from a Subversion repository to a Git repository. The hosting of source code was simultaneously moved from cvs.tcm.phy.cam.ac.uk to Bitbucket.

In July 2023, the ONETEP project was migrated from Bitbucket to GitHub.

Note: A pdf version of the documentation can be downloaded `here`

BIBLIOGRAPHY

[Scherlis2006] D. A. Scherlis, J.-L. Fattebert, F. Gygi, M. Cococcioni, N. Marzari, "A unified electrostatic and cavitation model for first-principles molecular dynamics in solution", *J. Chem. Phys.* **2006**, 124, 074103.

[Schlegel2011] H. B. Schlegel, "Geometry optimization", *WIREs Comput. Mol. Sci.* **2011**, 1, 790.

[Liu1989] D. C. Liu and J. Nocedal, "On the Limited Memory BFGS method for large scale optimization", *Math. Program.* **1989**, 45, 503.

[Pfrommer1997] B. G. Pfrommer, M. Cote, S. G. Louie, and M. L. Cohen, "Relaxation of Crystals with the Quasi-Newton Method", *J. Comput. Phys.* **1997**, 131, 233.

[Andzelm2001] J. Andzelm, R. D. King-Smith, G. Fitzgerald, "Geometry optimization of solids using delocalized internal coordinates", *Chem. Phys. Lett.* **2001**, 335, 321.

[Barzilai1988] J. Barzilai and J. M. Borwein, "Two-Point Step Size Gradient Methods", *IMA J. Numer. Anal.* **1988**, 8, 141 (1988).

[Aarons] J. Aarons, "A new CASTEP and ONETEP Geometry Optimiser" http://www.hector.ac.uk/cse/distributedcse/reports/castep-geom/castep-geom/HTML/dCSE_project.html.

[onetep_ase_interface] https://wiki.fysik.dtu.dk/ase/ase/calculators/onetep.html. Copyright 2022, ASE-developers.

[ase_optimizers_website] https://wiki.fysik.dtu.dk/ase/ase/optimize.html. Copyright 2022, ASE-developers.

[Wolfe1969] P. Wolfe, "Convergence conditions for ascent methods", *SIAM Rev.* **1969**, 11, 226.

[Wolfe1971] P. Wolfe, "Convergence conditions for ascent methods. II: Some corrections", *SIAM Rev.* **1969**, 13, 185

[Gilbert1997] J. C. Gilbert, "On the realization of the Wolfe conditions in reduced quasi-Newton methods for equality constrained optimization", *SIAM J. Optim.* **1997**, 7, 780.

[Yuan2017] G. Yuan, Z. Wei, X. Lu, "Global convergence of BFGS and PRP methods under a modified weak Wolfe-Powell line search", *Appl. Math. Model.* **2017**, 47, 811.

[Avogadro] Avogadro: an open-source molecular builder and visualisation tool. Version 1.95.1. http://avogadro.cc/

[Hanwell2012] Avogadro: an open-source molecular builder and visualisation tool. Version 1.XX. http://avogadro.cc/ Marcus D Hanwell, Donald E Curtis, David C Lonie, Tim Vandermeersch, Eva Zurek and Geoffrey R Hutchison; "Avogadro: An advanced semantic chemical editor, visualisation, and analysis platform" Journal of Cheminformatics 2012, 4:17.

[ASE_nanoparticle] https://wiki.fysik.dtu.dk/ase/ase/cluster/cluster.html. Copyright 2022, ASE-developers.

[Packwood2016] D. Packwood, J. Kermode, L. Mones, N. Bernstein, J. Woolley, N. Gould, C. Ortner, and G. Csányi, "A universal preconditioner for simulating condensed phase materials", *J. Chem. Phys.* **2016**, 144, 164109.

[Mones2018] L. Mones, G. Csányi, and C. Ortner, "Preconditioners for the geometry optimisation and saddle point search of molecular systems", *ArXiv* **2018**, 1804.01590.

[Byrd1994] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited memory methods", *Math. Program* **1994**, 63, 129.

[CASTEP] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson, M. C. Payne, "First principles methods using CASTEP", *Z. Kristallogr.* **2005**, 220, 567.

[Pickard2006] C. J. Pickard, "On-the-fly pseudopotential generation in CASTEP", **2006** https://www.tcm.phy.cam.ac.uk/castep/otfg.pdf

[Nielsen1983] O. H. Nielsen and R. M. Martin, First-principles calculation of stress, Phys. Rev. Lett. 50, 697 (1983).

[Nielsen1985] O. H. Nielsen and R. M. Martin, Quantum-mechanical theory of stress and force, Phys. Rev.B 32, 3780 (1985).

[Nielsen1985b] O. H. Nielsen and R. M. Martin, Stresses in semiconductors: Ab initio calculations on Si, Ge, and GaAs, Phys. Rev. B 32, 3792 (1985)

[Martin2008] R. M. Martin, Electronic structure, 1st ed. (Cambridge Univ. Press, Cambridge, 2008)

[Kittel2004] C. Kittel, Introduction to Solid State Physics, 8th ed. (Wiley, 2004).

[Knuth2015] F. Knuth, C. Carbogno, V. Atalla, V. Blum, and M. Scheffler, All-electron formalism for total energy strain derivatives and stress tensor components for numeric atom-centered orbitals, Comput. Phys. Commun. 190, 33 (2015)

[Kresse1999] G. Kresse and D. Joubert, From ultrasoft pseudopotentials to the projector augmented-wave method, Phys. Rev. B 59, 1758 (1999).

[Foulkes1989] W. Matthew C. Foulkes, Roger Haydock, *Phys. Rev. B* **1989**, 39, 12520, https://doi.org/10.1103/PhysRevB.39.12520

[Elstner1998] Marcus Elstner et. al., *Phys. Rev. B* **1998**, 58, 7260, https://doi.org/10.1103/PhysRevB.58.7260

[Gaus2014] Michael Gaus, Qiang Cui, Marcus Elstner, "Density functional tight binding: application to organic biological molecules", *WIREs Comput. Mol. Sci.* **2014**, 4, 49, https://doi.org/10.1002/wcms.1156

[Bannwarth2020] Christoph Bannwarth et. al., "Extended tight-binding quantum chemistry methods", *WIREs Comput. Mol. Sci.* **2021**, 11, 1, https://doi.org/10.1002/wcms.1493

[Pracht2019] Philipp Pracht, Eike Caldeweyher, Sebastian Ehlert, Stefan Grimme, "A robust non-self-consistent tight-binding quantum chemistry method for large molecules", *ChemRxiv* **2019**, https://doi.org/10.26434/chemrxiv.8326202.v1

[Grimme2006] Stefan Grimme, "Semi-empirical GGA-type density functional constructed with a long-range dispersion correction", *J. Comput. Chem.* **2006**, 27, 1787, https://doi.org/10.1002/jcc.20495

[Caldeweyher2019] Eike Caldeweyher et. al., "A generally applicable atomic-charge dependent London dispersion correction", *J. Chem. Phys.* **2019**, 150, 154122, https://doi.org/10.1063/1.5090222

[utils-devel] https://github.com/onetep-devel/utils-devel