BigDFT User Manual

http://inac.cea.fr/L_Sim/BigDFT

- Stefan.Goedecker@unibas.ch, http://www.unibas.ch/comphys/comphys
- Luigi.Genovese@esrf.fr
- Damien.Caliste@cea.fr
- Alessandro.Mirone@esrf.fr
- Thierry.Deutsch@cea.fr

Contents

Chapter 1

Installing BigDFT

The compilation and installation of BigDFT rely on the GNU standard building chain: 'configure', 'make', 'make install'. BigDFT can be used as an independent program (as described in this manual), or as a library, to be embedded in other softwares, like inside ABINIT.

1.1 Building the executables

1.1.1 Configure

The BigDFT build system is based on standard GNU autotools. The end user does not need to have the autotools package installed on his computer, the `configure` script provided in the BigDFT package will create the appropriate `Makefile` and set all the compilation options, like: the optimization level, the associated libraries to link with, and so on.

After the package has been untarred, the sources should be configured to the local architecture of the system. Thanks to the autotools, it is possible to generate several builds from the same source tree. It is advised to create a compilation directory, either inside or outside the source tree. Lets call this directory `compile-gFortran` for instance. One starts the configure from there `'source tree path'/configure`.

One can tune the compilation environment using the following options:

- `FC`: Specify the compiler (including MPI aware wrappers).
- `FCFLAGS`: Specify the flags, like the optimisation flags, to pass to the compiler (default are `-g -O2` for GNU compilers).
- Linear algebra options:
  - `--with-ext-linalg`: Give the name of the libraries replacing BLAS and LAPACK (default = none specified). Use the -l before the name(s).

  - `--with-ext-linalg-path`: Give the path of the other linear algebra libraries (default = `-L/usr/lib`). Use the -L before the path(es).

- Accelarators:
  - `enable-cuda-gpu`: Compile CUDA support for GPU computing.
  - `--with-cuda-path`: Give the path to the NVIDIA CUDA tools (default is `/usr/local/cuda`).
  - `--with-nvcc-flags`: Specify the flags for the NVIDIA CUDA Compiler.
  - `--enable-opencl`: Compile OpenCL support for GPU computing (compatible with `--enable-cuda-gpu`).
  - `--with-ocl-path`: Give the path to the OpenCL installation directory (default is `/usr`).

- Optional libraries:
  - `--with-etsf-io`: Use ETSF file format (binary based on NetCDF) for densities, potentials and wavefunction files.
  - `--with-archives`: Use compression (tar.bz2) for position files during geometry optimisation.

- `--prefix=DIR`: Specify your installation directory (`/usr/local` is default).

An example of compilation using the MKL from Intel instead of basic BLAS and LAPACK installation:

```
../configure --with-ext-linalg="-lmkl_ia32 -lmkl_lapack"
--with-ext-linalg-path="-L/opt/intel/mkl72/lib/32"
--prefix=/home/caliste/usr FC=ifort
```

An other example, compiling CUDA parts:

```
../../sources/bigdft-1.3.0-dev/configure
FC=mpif90 FCFLAGS="-O2 -assume 2underscores"
CC=icc CXX=icc CXXFLAGS="-O2 -I/applications/cuda-2.2/include/"
CFLAGS="-O2 -I/applications/cuda-2.2/include/"
--with-ext-linalg="-L/applications/intel/mkl/lib/em64t
-lmkl_scalapack_lp64 -lmkl_blacs_intelmpi20_lp64
-lmkl_intel_lp64 -lmkl_lapack -lmkl_sequential -lmkl_core"
--enable-cuda-gpu --with-cuda-path=/applications/cuda-2.2
```

The other options available can be browsed via the `--help` option. Some of them are listed here (and they can be of course combined with each other, when it does make sense):

- `--disable-mpi`: Force not to use MPI during build. By default the configure will try to detect if the compiler has some native MPI capabilities. If not MPI will be automatically disabled.

- `--enable-debug`: Creates a slower version of the executable in which any of the array allocated is filled by `NaN` after its boundaries. Useful to detect runtime errors during developments

- `--with-memory-limit=<mem>`: Creates a version of the executable which abort the run if one of the processes allocates more memory than `<mem>` (in Gb). This version is not necessarily slower that traditional copilation.

1.1.2 Make

Make the package and create the 'bigdft' executable, issuing `make`. The GNU option `-j`*n* is working with whatever value of *n* (tested up to 16).

1.1.3 Install

To install the package, issue `make install`. It will copy all files to the specified prefix (see configure).

1.1.4 Clean

Clean the source tree of the 'make' action by `make clean`.

1.2 Building a library

To avoid to create the binary executable, use `--disable--build-binary` option.

The main subroutine of the BigDFT package is the `call_bigdft` routine. For a given set of input coordinates and input parameters it returns the total energy and the forces acting on the nuclei. The `BigDFT.f90` main program calls the `call_bigdft` routine and can also do a geometry optimization by calling the `geopt` routine (which in turn calls again `call_bigdft`). For other standard applications other main programs exist. At present main programs to do a vibrational analysis, saddle point search and global optimization have been developed. Users are encouraged to write their own main programs for specific applications. The BigDFT API will be the object of a forthcoming chapter.

Chapter 2

File formats and conventions in BigDFT

2.1 The basis set

2.1.1 One dimensional functions

The maximally symmetric Daubechies family of degree 16 is used to represent the Kohn-Sham orbitals. The two fundamental functions of this family, the scaling function ϕ and the wavelet ψ are shown in Fig. 2.1.1 for the one-dimensional case. To form a basis set these functions have to be centered on the nodes of a regular grid.

2.1.2 Wavelet basis sets in three dimensions

A 3-dim wavelet basis is made by products of 1-dim functions. 1 scaling function and 7 wavelets can be centered on the nodes (i,j,k) of a regular 3-dim Cartesian grid. They all are products of 1-dim scaling functions and wavelets:

$$\begin{aligned}
\phi_{i,j,k}(x,y,z) &= \phi(x-i)\phi(y-j)\phi(z-k) \\
\psi^1_{i,j,k}(x,y,z) &= \phi(x-i)\phi(y-j)\psi(z-k) \\
\psi^2_{i,j,k}(x,y,z) &= \phi(x-i)\psi(y-j)\phi(z-k) \\
\psi^3_{i,j,k}(x,y,z) &= \phi(x-i)\psi(y-j)\psi(z-k) \\
\psi^4_{i,j,k}(x,y,z) &= \psi(x-i)\phi(y-j)\phi(z-k) \\
\psi^5_{i,j,k}(x,y,z) &= \psi(x-i)\phi(y-j)\psi(z-k) \\
\psi^6_{i,j,k}(x,y,z) &= \psi(x-i)\psi(y-j)\phi(z-k) \\
\psi^7_{i,j,k}(x,y,z) &= \psi(x-i)\psi(y-j)\psi(z-k)
\end{aligned}$$

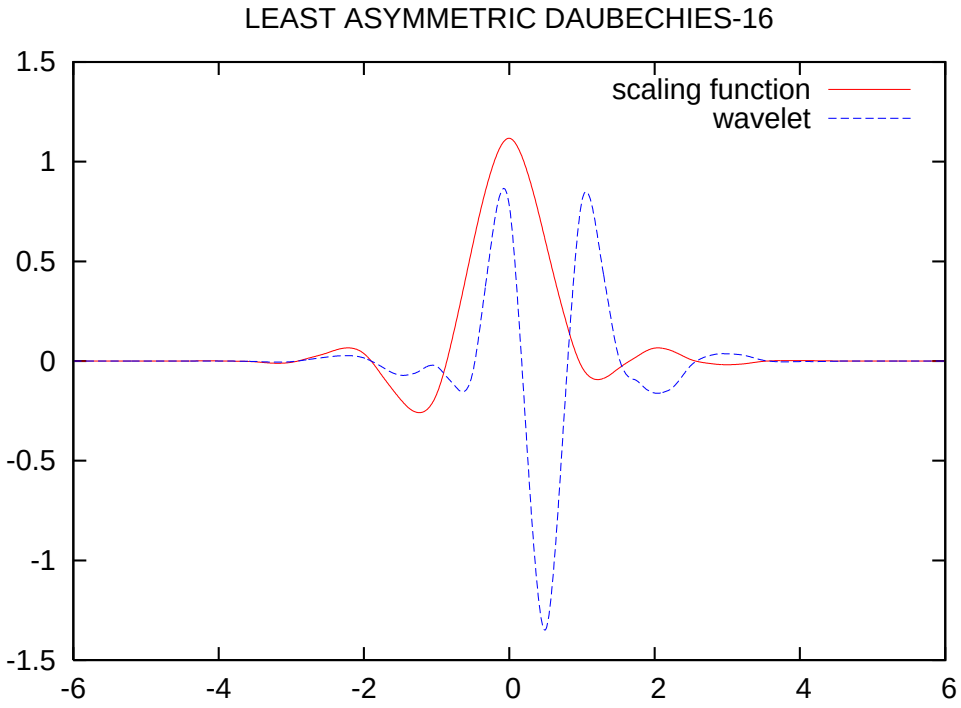
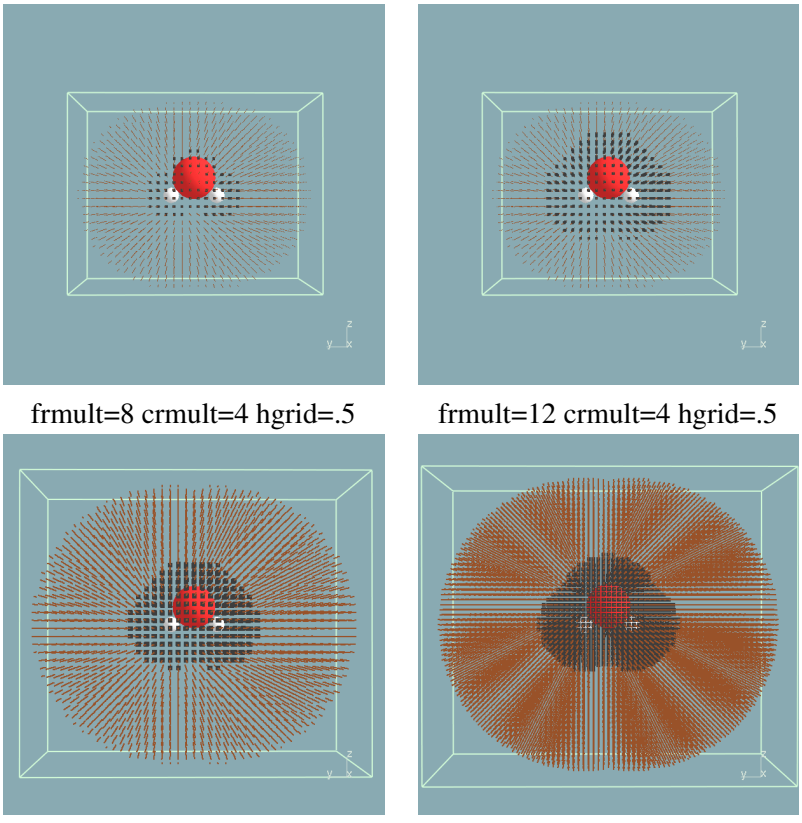


Figure 2.1: Daubechies Scaling Function and Wavelet of order 16

If a grid point carries the 7 wavelets in addition to the scaling function, it belongs to the high resolution region. In the low resolution region a grid point carries only a single scaling function. In the high resolution region, the resolution is doubled in each direction with respect to the low resolution region. The grid spacing is specified by the parameters **hgridx**, **hgridy**, **hgridz**. The low resolution region is constructed in the following way. Around each atom one draws a sphere whose radii are the 'size of the atom' times the adimensional parameter **crmult**. All the grid points being contained in the union of all these spheres form the low resolution region. The high resolution is constructed in a similar way. One draws spheres whose radii are the 'size of the bonding region' times the parameter **frmult** around each atom. All the grid points being contained in the union of all these spheres form the high resolution region. Default values for the 'size of the atom' and the 'size of the bonding region' are contained in the package. The user can however choose different values and these values can be specified by adding an additional line at the end of the pseudopotential parameter files which contain first the alternative values of **crmult** and then **frmult**.

2.1.3 Visualizing the simulation grid with V_Sim package

The grid can be visualized by calling the `memguess` program with the option **y**. Then the output file 'grid.xyz' contains the atomic positions and in addition the grid points which denoted by 'g' if the are in the coarse region and by 'G' if they are in the fine region. Whereas the total number of basis functions is nearly independent of the orientation of the molecule, the size of some work arrays depends on the orientation. When the `memguess` routine is called with the option **o**, it will rotate a molecule that will be calculated with free boundary conditions in such a way that the simulation cell size and hence the size of the work arrays are as small as possible. The rotated output is written in the file posout_000.xyz.



frmult=8 crmult=4 hgrid=.5 frmult=12 crmult=4 hgrid=.5

frmult=12 crmult=6 hgrid=.5 frmult=12 crmult=6 hgrid=.3

Figure 2.2: Illustration of the grid and its parameters

2.2 Format of Input/Output files for atomic coordinates

BigDFT supports atomic files which are of two types. The first one is a particular generalisation of the traditional `.xyz` format and can be obtained from this via

simple modifications. The second one (`.ascii`) is peculiar of BigDFT and V_Sim. Both formats work with the two codes.

2.2.1 The `.xyz` format

By default the atomic input coordinates are in the file `posinp.xyz`. If the same operation (*e.g.* geometry optimization) has to be done for several structures one can also give a list of input files whose names (without the .xyz extension) have to be contained in a file '`list_posinp`'. The first line in the '`list_posinp`' file has to be the number of input files and each consecutive line contains then one filename.

All the input and output files for atomic coordinates are in the `.xyz` format, *i.e.* they can be visualized with any standard visualization package and in particular with V_Sim. The first line contains the number of atoms and then the units, which are specified by 'atomic', 'atomicd0' or 'Bohr' if atomic units are used or by 'angstroem' or 'angstroemd0' if the angström unit is used. 'd0' formats (`1pe24.17` in Fortran) guarantee that not a single bit is lost during write and read of the numbers.

The second line contains the boundary conditions specified by the keywords 'free' for free boundary conditions, 'periodic' for periodic boundary conditions or 'surface' for surface boundary conditions where the x and z direction have periodic boundary conditions and the y direction free boundary conditions. The keywords 'periodic and 'surface' have to be followed by 3 real numbers giving the length of the orthorhombic periodic cell. In the case of surface boundary condition the second of these numbers is ignored.

The following lines contain the name of the chemical element followed by the 3 Cartesian coordinates. Chemical elements are identified by their pseudopotential. If a element is for instance denoted by 'Si' the element will be described by the file 'psppar.Si' which has to be present in the working directory. of the BigDFT.run. A silicon atom could however also be denoted by 'Si_lda' if there is file 'psppar.Si_lda. BigDFT supports the GTH and HGH pseudopotentials in the format which can be downloaded from the ABINIT website (www.abinit.org). The 3 Cartesian coordinates can be followed by optional additional information.

In the case of spin-polarized calculation the polarization in the region around the atom can be given by an integer. In addition the charge in the region around the atom can be specified. Finally it can be specified if the atom is fully or partially fixed during a geometry optimization. '`f`' stands for completely frozen, '`fxz`' if the atom can only move along the y axis and '`fy`' if the atom can only move in the XY plane. Below are some examples:

This hydrogen atom is frozen during the geometry optimization

```
H  1.2  3.4  5.6   f
```

This hydrogen atom has a spin up polarization

```
H  1.2  3.4  5.6   1
```

This chlorine atom has an additional electron and no spin polarization

```
Cl  1.2  3.4  5.6   0  -1
```

Same as above, but also frozen

```
Cl  1.2  3.4  5.6   0  -1   f
```

2.2.2 The `.ascii` format

BigDFT can also use another text file format for input (and thus output) atomic positions. Here is its structure:

- 1st line is arbitrary
- 2nd line must contain dxx dyx dyy values
- 3rd line must contain dzx dzy dzz values other lines may contain:
  - keywords, with the syntax [#!]keyword: followed by a list of keywords, separated by commas or blank spaces.
  - comments beginning with '#' or '!'
  - x y z name [label] for atomic position and optional labels

dxx dyx dyy dzx dzy dzz values define the box that contains the atoms. The format allows non-orthogonal boxes, but BigDFT supports only orthorombic supercells, so dyx, dzx and dzy must be zeros. When the keyword angdeg is used, the six values contains the three lengths of basis vectors in dxx dyx and dyy, and the three angles (bc, ac, ab) in dzx dzy and dzz.

After the three first mandatory lines, all subsequent lines can be comment lines (ignored), *i.e.* empty, containing only blanks, or beginning with ! or with #. A non comment line must contain 'x y z name', giving the 3 coordinates and the name of the atom. The coordinates of the atoms are given in the orthonormal basis not in the box basis. In case the keyword reduced has been specified, the three coordinates x, y and z are given in the box basis.

The keywords can be positionned anywhere after the first three lines. The following list summurises all available keywords:

- `reduced`: All atomic coordinates are given in reduced coordinates with respect to the box definition.
- `angdeg`: The box definition contains three distances and three angles instead of the classical six projection values.
- `bohr` or `bohrd0`: The units for the distance are Bohr.
- `angstroem` or `angstroemd0`: The units for the distance are Angstrms.

- `atomic` or `atomicd0`: The units for the distance are Bohr.
- `periodic`, `surface` or `freeBC`: The periodicity of the system is either 3D, 2D (free direction is y) or 0D.

Additional informations can be provided after the atom names, like for the `.xyz` format.

2.3 The input file '`input.dft`'

This file contains all the parameters required for a single wavefunction calculations:

- **hgridx**, **hgridy**, **hgridz**: The grid spacing of the Cartesian grid, in Bohr. As described above the nodes of this grid serve as the centers for the scaling function/ wavelet basis. Values are in most cases between .3 and .6.
- **crmult**, **frmult**: Coarse Region Multiplier and Fine Region Multiplier which serve to determine the radius for the low/high resolution sphere around the atom. Values are typically of the order of 5 for **crmult** and of the order of 10 for **frmult**.
- **ixc**: integer specifying which exchange correlation functional will be used. The Abinit conventions are used and detailed information can be found on the Abinit Web page (www.abinit.org/documentation/helpfiles/for-v5.8/input_variables/varbas.html#ixc). Here is only a short summary of some widely used functionals:

1	Pade LDA from Abinit XC library;
11	PBE from Abinit XC library;
−020	Pade LDA from libXC;
−101130	PBE from libXC;
−116133	PBEsol from libXC;
−406	PBE0 Hybrid functional (local part from libXC).

  A positive integer refers to a functional from ABINIT library and a negative one from libXC library. In this case, you concatenate the codes for the exchange part and for the correlation part (see the table in appendix A.2).
- **ncharge**, **elecfield**: Total charge of the system and uniform electric field E_x, E_y, E_z in units of Ha/Bohr. A positive **ncharge** means that electrons are taken away. The electric field can not have components in periodic directions, e.g. $E_x = E_z = 0$ in case surface BC is used.
- **nspin**, **mpol**: nspin=1, closed shell system without spin polarization; nspin=2: spin polarized system; nspin=4: non-collinear magnetic system.
- **gnrm_cv** convergence criterion for the wavefunction optimization (norm of the gradient). Reasonable values are in most cases between 1.d-4 and 1.d-5.

- **itermax**, **nrepmax**: Maximum number of gradient evaluations for a single cycle in a wavefunction optimization and maximum number of cycles. At the end of each cycle a subspace diagonalization is done which helps in cases where one has near degeneracy between the HOMO and LUMO orbitals. 50 and 2 are usually sufficient.

- **ncong**, **idsx**: ncong gives the number of iterations in the solution of the preconditioning equation. For free boundary conditions 5 is a good value whereas for other boundary condition a value from 0 to 2 is in general sufficient. Large values of **ncong** lead to a smaller number of iterations in the wavefunction optimization and better forces, but each iteration is more costly. So an optimal compromise value has to be found. **idsx** gives the history length of the DIIS convergence acceleration in the wavefunction optimization. 6 is usually a good value for fast convergence.
  In case of convergence problems it can be advantageous to switch off DIIS by setting **idsx** = 0. The memory requirements grow considerably with large values of **idsx**. If memory is the limiting factor one has to choose **idsx** smaller than the value which gives the fastest convergence. The '`memguess`' tool can be used to predict the memory requirements for different choices of **idsx**.

- **dispersion**: A non-zero values activates an empirical add-on treatment of dispersion effects. The values 1, 2 and 3 specify different switching on functions using the convention of Q. Hill and C.-K. Skylaris. Proc. R. Soc. A 465 (2009) 669

- **InputPsiId** , **output_wf**, **output_denspot** : **InputPsiId** specifies how the input guess wavefunction is generated:

  - **InputPsiId** = -2: Random numbers are used as input guess. This is of course a poor input guess which will need many iterations of the wavefunction optimization and might even lead to divergences.
  - **InputPsiId** = -1: The input wavefunction is imported from the CP2K code which uses Gaussian functions. The basis set should be contained in a file named `gaubasis.dat` whereas the coefficients should appear in the `gaucoeff.dat` file. Both files are the output files of CP2K code. See the H2O-CP2K test for an example.
  - **InputPsiId** = 0: A subspace diagonalization in a minimal atomic basis set is used. This input guess should be used in general if one starts a new calculation.
  - **InputPsiId** = 1: The previously calculated wavefunctions (for instance from the previous geometry optimization step) are used as input guess. Setting **InputPsiId** to this value does only make sense from within a main program where `call_cluster` was called previously. The old wavefunction is passed to the new call via the data structure 'restart'

- **InputPsiId** = 2: The input wavefunction is read from the 'wavefunction.*' files which contain all the scaling function and wavelet coefficients. In case some parameters such as **hgridx** or **crmult** have changed, compared to the previous run, the wavefunctions will be also transformed to the new parameter set. Use also this value to restart from ETSF file format wavefunction-etsf.nc.
- **InputPsiId** = 10: The same as 0 but activates the 'Gaussian help': after convergence the wavefunctions are projected onto the localised basis set used for the input guess, and Mulliken Charge Population Analysis (MCPA) is performed on this basis. The user has the possibility to perform MCPA with separate basis set (functionality to be added)
- **InputPsiId** = 11: Restart with Gaussian approximation contained in the 'restart' data structure.
- **InputPsiId** = 12: The input wavefunction is read from the 'wavefunctions.gau' file, which contains an approximation in a minimal Gaussian basis set of the previously calculated wavefunctions.
- **output_wf** = 0, Do not write the wavefunctions to disk.
- **output_wf** = 1, The output wavefunctions are written at the end of the wavefunction optimization into plain text files. If **InputPsiId** was greater than 10 the wavefunction will be written in the Gaussian approximation into a single 'wavefunctions.gau' file otherwise into 'wavefunction.*' files. Writing a 'wavefunction.*' file for each orbital can take a considerable amount of time and disk space.
- **output_wf** = 2, The output wavefunctions are written at the end of the wavefunction optimization into Fortran binary files. This format is not portable between compilers and machines.
- **output_wf** = 3, The output wavefunctions are written at the end of the wavefunction optimization into ETSF binary files. This format is portable between compilers and machines since based on NetCDF. Parallel IO are taken into account.
- **output_denspot** = 0, No output density is written.
- **output_denspot** = 1, Output electronic density is written in the .pot format of V_Sim into the file 'electronic_density.pot' (Deprecated, use 11 instead).
- **output_denspot** = 2, In addition to the electronic density, the potential ('local_potential.pot') and its components ('ionic_potential.pot' and 'Hartree_potential.pot') are also output in plain text '.pot' files (Deprecated, use 12 instead).
- **output_denspot** = 11, Same as **output_denspot** = 1, but files are writtent in ETSF file format (portable binary format based on NetCDF).

  - **output_denspot** = 12, Same as **output_denspot** = 2, but files are writtent in ETSF file format (portable binary format based on NetCDF).
  - **output_denspot** = 21, Same as **output_denspot** = 1, but files are writtent in '.cube' file format (plain text).
  - **output_denspot** = 22, Same as **output_denspot** = 2, but files are writtent in '.cube' file format (plain text).

- **rbuf**, **ncongt**: Far reaching tails of the wavefunctions decaying into the vacuum are added in a perturbative treatment if the variable **rbuf** is set to a strictly positive value. This allows to do a calculation with some moderate value of **crmult** and then to extrapolate to the limit of large **crmult**. This procedure is not variational and gives too low energies. The true energy is in between the two energies and in general much closer to the extrapolated energy. This procedure can also be used to judge whether the chosen value of **crmult** is large enough for a certain required precision. **rbuf** gives the amount by which the radii for the coarse resolution region are increased in atomic units. **ncongt** gives the number of iterations used in the perturbation calculation. Reasonable values for **ncongt** are around 30.

- **norbv**,**nvirt**, **nplot**: Usually unoccupied orbitals are not calculated since they are not needed for the total energy and other physical properties of the electronic ground state. Putting **norbv** to a non-zero value will result in the calculation of **norbv** virtual orbitals in a postprocessing routine after the occupied orbitals have been calculated, which uses Davidson iterative treatment . **nplot** of these orbitals will be written in the 'virtual.*.pot' files and **nplot** (if that many exist) of the highest occupied orbitals will be written in the 'orbital.*.pot' file The Kohn Sham eigenvalues are written in the ordinary output file. **nvirt** corresponds to the actual number of orbitals the convergence process is taking care of during the Davidson iterations.

- **disable sym**: a logical to set to 'T' to disable the usage of symmetry operations in the calculations. By default symmetries are used to update the density in periodic boundary condition calculations, and surface boundary conditions also.

2.4 The input file '`input.geopt`'

If this file exists, BigDFT will do a geometry optimization and the file contains all the required parameters.

- **geopt_approach**: This character string specifies the method used for the geometry optimization.
  - *VSSD*: Variable Stepsize Steepest Descent method;

- *SDCG*: A combination of Steepest Descent and Conjugate Gradient;
- *LBFGS*: Limited Memory BFGS;
- *BFGS*: Preconditioned steepest descent with energy feedback. A preconditioning matrix is build up according to the BFGS algorithm. The initial Hessian is a diagonal matrix where the diagonal elements are the inverse of the step size. This method is usually the most efficient one.
- *PBFGS*: Same as BFGS except that an initial Hessian is obtained from a force field. Force field parameters are only available for systems consisting of H,C,N,O. For such systems the method is the most efficient one in general
- *AB6MD*: The molecular dynamic routines from ABINIT 6.

- **ncount_cluster_x**: Maximum number of force evaluations to be used for the geometry optimization.

- **frac_fluct**, **forcemax**: Convergence criteria for the geometry optimization. The geometry optimization stops either if the norm of the individual forces acting on any atom in the system is smaller than **forcemax** or if the forces get noisy. Noise is present because of the underlying integration grid. The parameter **frac_fluct** specifies how small the forces should become compared to the noise level to stop the geometry optimization. Values in between 1 and 10 is are reasonable for this parameter. A value of 2 means that the geometry optimization will stop when the largest atomic force is comparable to the 2 times the average noise in the forces themselves. For values of **frac_fluct** smaller than 1 one can under certain circumstances obtain better relaxed geometries but one risks that the geometry optimization will not converge since the forces are too noisy. In such a case one should closely monitor the progress of the geometry optimization by looking at the 'posout.*' files which are written at each step of the geometry optimization and at the 'geopt.mon' file.

- **randdis**: This parameter allows to add random displacements with amplitude **randdis** to the atomic positions in the input file 'posinp'. This can for instance be useful to break degeneracies (which would lead to convergence problems in the wavefunction optimization) in highly symmetric structures.

- **betax**: This is the stepsize for the geometry optimization. This stepsize is system dependent and it has therefore to be determined for each system. If the VSSD method is used one can start with a small stepsize of around 1 and VSSD will suggest than a better value for **betax** in the last line of the 'geopt.mon' file. Whether the stepsize is correct can also be seen from the 'geopt.mon' output of the SDCG method. In this case the average stepsize in terms of **betax** should be around 4 (after a brief initial period where it is around 8). In contrast to the SDCG method, the BFGS method is not

very sensitive to the correct stepsize but nevertheless one should try to find reasonable values also in this case.

- **ionmov**: in case of **geopt_approach = AB6MD**, the **betax** line should be replaced by this one. It contains an integer value as described in the ABINIT manual. Possible values are:
  - 6: simple velocity-Verlet molecular dynamic.
  - 7: quenched molecular dynamic, when the scalar product force / velocity becomes negative, the velocity is set to zero. The force criterion is tested at each step.
  - 8: Nose-Hoover thermostat.
  - 9: Langevin dynamic (adding a friction force and a Gaussian random force on atoms).
  - 12: Isokinetic ensemble molecular dynamics. The equation of motion of the ions in contact with a thermostat are solved with the algorithm proposed by Zhang [J. Chem. Phys. 106, 6102 (1997)], as worked out by Minary et al [J. Chem. Phys. 188, 2510 (2003)]. The conservation of the kinetic energy is obtained within machine precision, at each step.
  - 13: Iosthermal/isenthalpic ensemble. The equation of motion of the ions in contact with a thermostat and a barostat are solved with the algorithm proposed by Martyna, Tuckermann Tobias and Klein [Mol. Phys., 1996, p. 1117].
- **dtion**: the time step for molecular dynamic, in atomic time units. One atomic time unit is 2.418884e-17 seconds, which is the value of Planck's constant in Hartree*sec. The following lines depend on the choosen value for **ionmov**.
- if **ionmov = 6**: one line containing the initial temperature in kelvin. If negative, the initial velocities are all zero. If positive, random speeds are chosen to match the given temperature (NOT IMPLEMENTED YET!).
- if **ionmov** $>$ **7**: one line containing two temperatures in kelvin. When different, the temperature is linearly change at each geometry step to go from the first value to the second.
- if **ionmov = 8**: one additional line containing the thermostat inertia coefficient for Nose-Hoover dynamic.
- if **ionmov = 9**: two additional lines containing first the friction coefficient and then a value in Bohr corresponding at a distance where the atoms can bounce on (see the ABINIT documentation for further details).
- if **ionmov = 13**: several additional lines containing first the number of thermostats in the chain of thermostats. Then a line with the mass of each thermostat in the chain. And finally a line with two values for the barostat mass, depending on optcell value (NOT IMPLEMENTED YET!).

2.5 The input file 'input.kpt'

This file is used to specify a set of k points. If this file does not exist, only the Γ point will be used. The k point generation relies on the ABINIT implementation. This files contains the following information:

- **kptopt**: This character string specifies the method used to generate the k point mesh.
  - *auto*: Automatic generation is used, based on the k point density we wish in Fourier space, taking into account the symmetries of the system.
  - *MPgrid*: A Monkhorst-Pack grid, using only the special k points, taking into account the symmetries of the system.
  - *manual*: A manual set of k points.

  The following lines depend on the choice of **kptopt**.

- if **kptopt = 'auto'**: One additional line containing a real space length **kptrlen**. BigDFT will automatically generate a large set of possible k point grids, and select among this set, the grids that give a length of smallest vector LARGER than **kptrlen**, and among these grids, the one that, reduces to the smallest number of k points.

- if **kptopt = 'MPgrid'**: Several additional lines. The first line should contains three integers describing the mesh of Monkhorst-Pack grid in reciprocal space. The second line contains one integer **nshiftk** that give the number of shift one would like to apply to the MMP grid to obtain the final k point mesh. Then the file contains **nshiftk** lines with three real numbers each, giving the shift to apply in reciprocal space.

- if **kptopt = 'manual'**: Several additional lines. The first contains **nkpt**, an integer giving the number of manually defined k points. Then **nkpt** lines follow, with four real values each, the three first being the coordinates in reciprocal space (in $[0;0.5]$) and the fourth bing the weight. If the sum of all weights does not equal to one, weight values are renormalised.

After the regular k point mesh for the self-consistent loop, one can define in addition a specific path of k points to be used to compute a band diagram. This requires also to define Davidson usage in the 'input.dft' file. The band structure is define if the following lines are present:

- **bands** keyword: a character string containing *bands*.

- the next line contains one integer, **nseg**, the number of different segments for the path.

- the next line contains **nseg** integer values, defining the number different k points to be generated in each segment. It contains also a last integer value which is the granularity, *i.e.* the number of k-points to be treated simultaneously in the Davidson treatment.
- then follow **nseg + 1** lines containing each three floating point values with the coordinates of the vertices in reduced coordinates of the Brillouin zone.

2.6 The input file '`input.mix`'

If this file is present, the SCF loop is run with a diagonalisation scheme instead of the direct minimisation scheme. This file controls how the density or the potential is mixed between each iteration of diagonalisation. It must contain the following lines:

- **iscf**: An integer giving the mixing scheme and the mixing target. It follows the ABINIT convention. For values lower than 10, the potential is mixed, while for values greater than 10, the density is mixed.
- **itrpmax**: Maximum number of diagonalisation iterations.
- **rpnrm_cv**: The stop criterion on the residue of potential or density.
- **norbsempty Tel**: The number of additional bands and the electronic temperature.
- **alphamix alphadiis**: The multiplying factors for the mixing and for the electronic DIIS.

Each diagonalisation step is done with a direct minimisation scheme at a fixed potential. The parameters of this minimisation are the usual ones in the `input.dft` file. The overall convergency of the diagonalisation is not very sensitive to the good convergency of the direct minisation steps. Thus a itermax value of 5 in `input.dft` is often advicable.

2.7 The input file '`input.perf`'

This file is used to specify values in order to optimize the performances of BigDFT. If this file does not exist, default values are used. On the contrary to other BigDFT input file, this file has optional key / value entries. The keywords are not case sensitive. Keywords can specify options as:

- **'debug' T/F**: The debug mode is enable mainly for memory profiling.
- **'fftcache' ncache_size**: Specify the cache size for FFT in kBytes.

- **'accel' NO/CUDAGPU/OCLGPU**: Specify the use of CUDA (resp. OCL) versions of various subroutines. For CUDA, a 'GPU.config' file is needed.
- **'blas' T/F**: Use or not the CUBlas acceleration.
- **'projrad' real**: Radius of the projector as a function of the maxrad.
- **'exctxpar' key**: Exact exchange parallelisation scheme.
- **'ig_diag' T/F**: Input guess: (T:Direct, F:Iterative) diagonalisation of Hamiltonian.
- **'ig_norbp' int**: Input guess: Orbitals per process for iterative diagonalisation.
- **'ig_blocks' int int**: Input guess: Block sizes for orthonormalisation.
- **'ig_tol' real**: Input guess: Tolerance criterion.
- **'methortho' key**: Orthogonalisation (0=Cholesky,1=GS/Chol,2=Loewdin).
- **'rho_commun' key**: Density communication scheme.
- **'verbosity' int**: Determines the amount of output from little (0) to detailed (2 by default)
- **'psp_onfly' T/F**: Switch on the once-and-for-all strategy for calculating the PSP projectors, which is faster but more memory demanding (considered by `memguess`). The default is on-the-fly strategy (T).

Chapter 3

Running BigDFT executables

3.1 Estimating the memory usage: `memguess`

Before running BigDFT, it is recommended to run the 'memguess' program. If it runs correctly all input files are available (this routine needs a posinp.xyz and does not accept a list_posinp file). It then allows to estimate the required memory and to find an optimal number of MPI processes for a parallel run. For good load balancing each MPI process should roughly treat the same number of orbitals.

The 'memguess' program prints out the number of orbitals and how many orbitals are treated by each MPI process. On a parallel machine with a high performance interconnect network one can choose the number of MPI processes **nproc** equal to the number of occupied orbitals, *i.e.* each MPI process treats one orbital. On machines with slower networks each MPI process should have at least 2 to 4 orbitals.

```
memguess <nproc> [option]
```

- **nproc**: Number of MPI processes;
- **o**: The molecule will be rotated such that the size of the workarrays is minimal;
- **y**: Generate a file 'grid.xyz' containing the coarse and the fine grid points;
- **GPUtest ¡nrep¿**: Case of a CUDAGPU calculation, to test the speed of 3d operators ¡nrep¿ is the number of repeats.
- **upgrade**: Ugrades input files older than 1.2 into actual format;
- **convert ¡from.[cube,etsf]¿ ¡to.[cube,etsf]¿**: Converts file "from" to file "to" using the given formats;
- **atwf ¡ng¿**: Calculates the atomic wavefunctions of the first atom in the gatom basis and write their expression in the "gatom-wfn.dat" file, ¡ng¿ is the number of gaussians used for the gatom calculation.

3.2 Single point or geometry optimization: `bigdft`

The MPI version is executed on most machines with the `mpirun` command followed by the name of the executable which is `bigdft` in our case. The treatment of each orbital can be speeded up by using the mixed MPI/OpenMP implementation where each MPI processes uses several OpenMP threads to do the calculations for its orbitals faster. The OpenMP is simply activated by compiling the program with an OpenMP flag and by specifying the number of OpenMP threats by export **OMP_NUM_THREADS=4** if 4 threads are for instance desired.
The BigDFT program monitors during a run the memory utilization and the time spent in various subroutines. Detailed information is written in the files 'malloc.prc' and 'time.prc'. At the end the program checks whether the number of deallocations was equal to the number of allocations and whether the total memory went back to zero. If this should not be the case please send a bug report to the developers of BigDFT.

3.3 Doing a path minimization: `NEB`

A NEB implementation is present in the BigDFT package, thanks to the initial routines provided by Carlos Sbraccia. This implementation is launched with the help of the `NEB` executable. This program is responsible for initialising the path from the two minima and then to run the path minimisation, using BigDFT to compute the forces. Each replica is launched by an instance of `bigdft` and not in the same program `NEB`. This allows to run the NEB algorithm on a super computer with a queue system, each replica going separately in the queue. The process of running the different `bigdft` instances, in different directories and wait for their completions is done by two shell scripts: `NEB_driver.sh` and `NEB_include.sh`. The first one is very generic and should not be touched by users. It is provided in the `src/` directory of the package. The second one must be adapted by users to suit to their running machine. One example is provided in the `tests/NEB/` directory of the package.

The `NEB` executable can be run from everywhere but the scripts `NEB_driver.sh` and `NEB_include.sh` must be in the same directory. It takes arguments from the command line, using a redirection from a file is good practice. See the `tests/NEB` directory of the package:

```
./NEB < input
```

The file `input` contains a list of variables relevant to the NEB algorithm. Some of them are explained here:

- `scratch_dir` is where the `NEB_driver.sh` script will create the directories where to run the `bigdft` instances. It is usually on a local disk.
- `job_name` a name that will be used to named all generated files and directories.

- `climbing` when set to `.TRUE.`, the highest replica follow opposite parallel forces in addition to the usual perpendicular ones.
- `optimization` when set to `.TRUE.`, tries to optimize the geometry of the first and last replicas (if not local minima already).
- `minimization_scheme` specifies the scheme used to minimze the NEB. The possible values are 'steepest_descent', 'fletcher-reeves', 'polak-ribiere', 'quick-min', 'damped-verlet' and 'sim-annealing'.
- `tolerance` is a criterion not to take into account in the list of moving atoms the ones that move less than this value between the first and the last replica.
- `convergence` is the stop criterion on perpendicular forces, in eV/Å.
- `num_of_images` is the number of desired replicas.
- `*_config` the file describing the first and the last replica (in BigDFT 1.3, the file extension .xyz must be omitted).

At each NEB iteration, the driver script is run. It creates the directories for forces calculations if needed, create the input files using the include script, run the jobs, grep the energy and the forces and return. The script file `NEB_include.sh` must contain the following shell functions:

- `init_jobs()`, run once after directory creation. It can be used.
- `finalise_jobs()`, run once after all jobs have finished.
- `wait_jobs()`, run each time the script poll all replicas for completion.
- `make_input()`, run once in each scratch directories to create the file 'pos-inp.xyz' from the NEB restart file and the initial ones.
- `run_job()`, run once in each scratch directories to start the force calculations. In the example, it runs the replicas directly on the host machine, but in this shell function, a submission file may be created instead and submitted to the queue system for later run.
- `check_job()` is used periodically by the driver system to check if a job is finished or not. It must return different values: -1 if the job is still not started (maybe be still in the queue for instance), 0 if the job is running but not finished yet, 1 if it exited with success and 2 if it exited with a failure.
- `grep_forces()` is run once after job termination to get the energy and the forces. Energy must be in Hartree and forces in Hartree per Bohr.

For NEB purposes, users of BigDFT usually have only to modify the `run_job()` functions from the examples to adapt it to their machine.

The output files are:

- `job_name.NEB.dat`: A three column file containing for each replica a reaction coordinate, the energy in eV and the maximum value of remaining perpendicular forces. This file is updated at each iteration.
- `job_name.NEB.log`: A file, giving for each NEB iteration the value of the highest replica in eV and the value of the maximum perpendicular forces on all atoms and replicas.
- `job_name.NEB.restart`: A file with the coordinates of each replicas. This file is generated by the `NEB` executable at each iteration and used by the driver script to generate the 'posinp' files.

3.4 Doing frequencies calculations: `frequencies`

Using the executable `frequencies`, it is possible to calculate the vibrational properties of a system by finite difference. The atomic system is moving for each direction and each atom in a small step in order to calculate the Hessian matrix by a finite difference scheme. A checkpoint restart is implemented. The file 'frequencies.res' contains the previous calculations.

3.4.1 The '`input.freq`' file

There are three parameters:

- `freq_alpha` to determine the step for frequency step (i.e. alpha*hx, alpha*hy, alpha*hz);
- `freq_order` which determines the order of the finite difference scheme:
  - -1 calculates $\frac{f(x_0)-f(x_0-h)}{h}$;
  - 1 calculates $\frac{f(x_0+h)-f(x_0)}{h}$;
  - 2 calculates $\frac{f(x_0+h)-f(x_0-h)}{2h}$ (this is the default);
  - 3 calculates $\frac{f(x_0+2h)+f(x_0+h)-f(x_0-h)-f(x_0-2h)}{6h}$.
- `freq_method` which determines the method (only 1 at the present stage).

Chapter 4

XANES calculations

4.1 Abstract

We have implemented an original procedure for calculating XANES spectra within the BigDFT code. Our approach consists firstly in projecting the photoelectron wave-function onto the pseudopotential functions basis with the help of a reversed PAW projector[1] and, then, propagating this initial state by iterative applications of the Hamiltonian. The Hamiltonian is the one provided by the BigDFT code. The obtained spectra therefore correspond exactly to the electronic structure of the self consistent ground state.The spectra are built by obtaining at each new application of the Hamiltonian a new coefficient of the spectral decomposition in Chebychev polynomials. The method therefore systematically improves the calculated spectra resolution at each step and can be stopped once the required resolution is reached. Moreover, the Chebychev polynomials have a higher nodes' density at the extrema of the spectral range than in the middle. For practical applications this makes convergence even faster because the broadening due to lifetime is narrow near the edge and large elsewhere. The execution times represent a break-through for this kind of calculations and reflect the exceptional features of the BigDFT code. For a cluster contained in a 25Å side cube, the spectra are obtained within half an hour, running on a single processor.

4.2 Introduction

The interaction between matter and a photon described by a wave vector k and polarisation ε, is written, discarding elastic Thomson scattering (the $A^2(r)$ term in the Schrödinger equation which becomes dominant off-resonance), discarding also the spin-magnetic field interaction [2], and using CGS units, as :

$$H_{int} = \left(\frac{2\pi\hbar c^2}{\omega V}\right)^{1/2} (a^{\dagger}_{k,\varepsilon} exp(-i\mathbf{k}\cdot\mathbf{r}) + a_{k,\varepsilon} exp(i\mathbf{k}\cdot\mathbf{r})) \frac{e}{mc}\mathbf{p}\cdot\varepsilon \qquad (4.1)$$

where r is the electron coordinate, p the kinetic moment, $a^{\dagger}{}_{k,\varepsilon}$, $a_{k,\varepsilon}$ are creation annihilation photon operators, $\omega = kc$, and V is the space volume.

The photon absorption is a first order perturbative process whose probability per unit of time $w(\omega)$, for photon with polarization ε state, is obtained from H_{int} matrix elements by the Fermi golden rule. For one photon in the V volume:

$$w(\hbar\omega) = \frac{(2\pi)^2 e^2 \hbar}{\hbar\omega V m^2} \sum_n |< n|exp(i\mathbf{k}\cdot\mathbf{r})\mathbf{p}\cdot\varepsilon|0>|^2 \delta(\hbar\omega - E_n) \tag{4.2}$$

where $< n|$ is a complete set of eigenstates of matter, $< 0|$ being the initial state, E_n is the energy difference between $< n|$ and $< 0|$ and we have retained only the resonating denominator. The absorption cross section is the quantity directly observable in experiment. It is obtained from the above equation dividing by the photon flux c/V:

$$\sigma(\hbar\omega) = \frac{(2\pi)^2 e^2 \hbar}{\hbar\omega m^2 c} \sum_n |< n|exp(i\mathbf{k}\cdot\mathbf{r})\mathbf{p}\cdot\varepsilon|0>|^2 \delta(\hbar\omega - E_n) \tag{4.3}$$

The matrix element can be simplified in terms of $\mathbf{r}$ by using the following identity for $\mathbf{p}$:

$$\begin{aligned} exp(ik\cdot r)p\cdot\varepsilon &\simeq (1 + ik\cdot r + ...)\frac{im}{\hbar}[H, r\cdot\varepsilon] \\ &= \frac{im}{\hbar}([H, r\cdot\varepsilon] + i[H, k\cdot rr\cdot\varepsilon]/2 + ...) \end{aligned} \tag{4.4}$$

Such substitution leads to:

$$\sigma(\hbar\omega) = (2\pi)^2 \alpha_0 \hbar\omega f(\hbar\omega) \tag{4.5}$$

with

$$f(\hbar\omega) = \sum_n |< n|i>|^2 \delta(\hbar\omega - E_n) \tag{4.6}$$

where

$$|i>= \mathbf{r}\cdot\varepsilon + i\frac{(\mathbf{k}\cdot\mathbf{r})(\mathbf{r}\cdot\varepsilon)}{2} + ...|0> \tag{4.7}$$

is the state resulting from the application of the interaction operator on the ground state.

The $f(\hbar\omega)$ is formally determined by its distribution moments which can be computed by iterative applications of the Hamiltonian :

$$\int f(E)E^n dE = |< i|H^n|i>| \tag{4.8}$$

To avoid numerical ill-conditioning coming from the numerical linear dependence of high order powers we proceed as explained in [3]. The H Hamiltonian is rescaled and shifted to a new H' whose spectra is comprised within $]-1,1[$:

$$< n|H'|n > \in]-1,1[$$

The overlap integrals between the spectra and Chebychev polynomials $T_n(x)$ is calculated by recurrence. The Chebychev polynomials $T_n(x)$ are:

$$T_n(x) = cos(n * arccos(x))$$

and satisfy the recurrence relation:

$$T_{m+1}(x) = 2xT_m(x) - T_{m-1}(x)$$

The overlap integral are:

$$\mu_n = \int f'(x)T_n(x) =< i|T_n(H')|i > |$$

and can be obtained by applying the recurrence relation to the left side of $|i>$ state:

$$T_{m+1}(H')|i >= 2H'T_m(H')|i > -T_{m-1}(H')|i >$$

and then obtaining the scalar product. The spectra is finally recovered as:

$$f(b+ax) = \frac{(\mu_0 + 2\sum_{n=1}^{\inf} \mu_n T_n(x))}{a\pi\sqrt{1-x^2}}$$

where b and a are the shift and the scaling factor respectively, involved in the $H' \rightarrow H$ transformation. The spectra can be obtained up to an arbitrary degree of resolution by applying iteratively the recurrence relation for a sufficient number of times. To avoid oscillation due to the truncation of the summation, a convolution is applied multiplying the μ_n components by the Jackson kernel [3].

4.3 Use of the code

The binary executable for absorption spectra calculation is named `abscalc`. Besides the usual files which are needed for a basic BigDFT calculation, `abscalc` needs to find, in the current work directory, at least the file '`input.abscalc`'. This file is specific to absorption calculations and its structure will be explained here. The usual files needed for a basic BigDFT calculation and by `abscalc` are, we recall it here: '`input.dft`', 'posinp.xyz', and the pseudopotential files 'psppar.XX' for all the elements entering in the structure.

Others files may be needed when one selects the option of importing the self consistent potential from a previous SCF calculation, as will be explained later.

Concerning the 'posinp.xyz' file we suggest, for a good result, to set a large box diameter of about 25Å or more, and periodic boundary conditions.

The structure of the file '`input.abscalc`' is the following:

- The input parameter *in%iabscalc_type* which can be equal to 1 or 2. In the first case the spectra is calculated as a series of Chebychev polynomials, while in the second case the Lanczos tridiagonalisation is applied. We suggest the first option because the Lanczos method requires, in order to keep orthogonality, that all the vectors of the tridiagonal base be stored in memory and this limits the number of steps that one can perform.

- The parameter *in%iat_absorber*, an integer number, is the position (starting from one) in the *posinp.xyz* file of the absorber atom.

- *in%L_absorber* : this parameter is the angular moment of the multipole component of the interaction operator that one wants to consider. It is 1 for dipole interaction, 2 for quadrupolar ...

- *in%Gabs_coeffs(i), i=1,2*in%L_absorber +1.*

  This are the components of the interaction operator, in Cartesian coordinates. These correspond to the BigDFT internal representation of tensors, which is based on spherical tensor. For dipolar interaction these will be therefore the three x, y, z components. In this present version the input are real numbers. Entry for the imaginary parts will have to be added in the future versions for magnetic dichroism.

- *in%potshortcut* : This parameter determines how the local potential is obtained. If it is set to 1 the potential is obtained from the superposition of atomic charges. If it is set to 2 the potential is read from a previous SCF calculation. In this case the program searches, in the current work directory, the file *b2B_xanes.cube*. Such file contains the atomic positions and the local potential of a previous SCF run and can be created setting the parameter *in%output_grid* to -2 in the file `input.dft`. The SCF potential is interpolated, using interpolant wavelets, to get the potential on the target grid. The SCF grid of the previous run must have less points than the target grid. This is easily the case, because in XANES calculations one usually sets a large diameter for the box. The fact of choosing a large box limits the spurious oscillations due to bands effects and to interference with the replica of the absorber atom in the other Bravais cells.

- *in%nsteps* : this is the total number of Hamiltonian applications. The larger this number, the better the resolution you get in the calculated spectra.

4.3.1 Note about the absorber pseudopotential

In the actual version of the code (1.4) for the absorber pseudopotential one must use a $Z+1$ approximation. For future versions of the code the development of

on-the fly pseudopotentials, corresponding to the excited electronic structure, is foreseen.

4.4 Description of the output

In case of *in%iabscalc_type* equal to 1, the spectra is written, in the current work directory, with the name : *cheb_spectra_NUM* where *NUM* is the number of Chebychev components. The number of components increases at each Hamiltonian application and ,during the calculation, several partial results, with increasing number of components, are written.

4.5 The reversed PAW method

The BigDFT code, in the actual version as well as in the previous ones, uses a norm-conserving pseudo-potential. From this pseudo-potential we extract the corresponding PAW projector. This is the reverse of the PAW method. In the PAW method the projector determines the pseudopotential. In our case instead, the PAW projector is a function of the pseudo-potential. The implementation of such function is detailed here. We need to construct the projector for the absorber atom only because the core orbitals don't overlap other atoms. For the absorber atom we proceed by calculating the SCF solution in the isolated atom case, once using pseudo-potentials, and another using the all-electrons potential. Then, for these potentials, we calculate an almost-complete set of eigen-functions which will be the basis for the projector. We need to do this only for selected angular moments. For example for the K edge and dipolar interaction only $l = 1$ eigen-functions are calculated.

To calculated the almost-complete set of eigen-functions we solve the radial Schrödinger equation in the interval $[0, R_c]$ for the lowest N_c eigenvalues. The radius R_c must be large enough to contain the core orbitals. We have hard-coded it to the value of $5au$. We have set $N_c = 200$. With this choice of parameters the maximum eigenvalue of the almost-complete basis is of the order of $XXXHartree$. We can compare this value with the maximum eigenvalue representable with wavelets on a grid with spacing $d = 0.3au$, for a typical BigDFT calculation, which is $Emax = XXXX$. Our almost-complete basis is indeed complete, after this comparison, because it covers completely the BigDFT spectra. Naming $\tilde{\psi}_{nlm}$ and ψ_{nlm} the solutions for the pseudo-potential and the all-electrons case respectively, the projector is thus written :

$$P = \sum_{nlm} |\tilde{\psi}_{nlm} >< \psi_{n+\Delta_l,lm}|$$

where Δ_l is the number of core orbitals contained in the pseudopotential. We show in fig xxx the radial parts $\tilde{\psi}_{nl}(r)$ and $\psi_{n+\Delta_l,l}(r)$ for $l = 1$ in the case of Iron, with a HGH pseudopotential with $Zion = 16$, for $n = 1, 2, xx, xxx$. In figure we show the

corresponding pseudo-eigenvalues $\tilde{E}_{n,l=1}$ and the all-electrons ones $E_{n+\Delta_l,l=1}$. We can see that the agreement is within XXX for the first XXx eigenvalues and then increase still remaining better than XXX per cent, while the agreement between the psuedo-wavefunctions and the all-electrons ones remains excellent.

4.6 Test against an exactly solvable case.

In progress

Bibliography

[1] ** This reversed PAW procedure, that we describe and document here, looks similar to the one which seems to be used in Taillefumier et al., Phys. Rev. B 66, 195107 (2002)

[2] M. Blume, in Resonant Anomalous X-Ray Scattering, edited by G. Materlik, J. Sparks and K. Fisher (Elsevier, Amsterdam, 1994), p. 495.

[3] A. Weiss, G. Wellein, A. Alvermann and H. Fehske Rev. Mod. Phys. 78, 275 (2006)

Appendix A

XC functionals codes

A.1 Native ABINIT XC codes

0	No semi-local xc, Hartree potential
1	LDA or LSD, Teter Pade parametrization (S. Goedecker, M. Teter, J. Hütter, Phys. Rev. B54, 1703 (1996)), which reproduces Perdew-Wang (which reproduces Ceperley-Alder!)
2	LDA, Perdew-Zunger-Ceperley-Alder (no spin-polarization)
3	LDA, old Teter rational polynomial parametrization fit to Ceperley-Alder data (no spin-polarization)
4	LDA, Wigner functional (no spin-polarization)
5	LDA, Hedin-Lundqvist functional (no spin-polarization)
6	LDA, "X-alpha" functional (no spin-polarization)
7	LDA or LSD, Perdew-Wang 92 functional
8	LDA or LSD, x-only part of the Perdew-Wang 92 functional
9	LDA or LSD, x- and RPA correlation part of the Perdew-Wang 92 functional
11	GGA, Perdew-Burke-Ernzerhof GGA functional
12	GGA, x-only part of Perdew-Burke-Ernzerhof GGA functional
13	GGA potential of van Leeuwen-Baerends, while for energy, Perdew-Wang 92 functional
14	GGA, revPBE of Y. Zhang and W. Yang, Phys. Rev. Lett. 80, 890 (1998)
15	GGA, RPBE of B. Hammer, L.B. Hansen and J.K. Norskov, Phys. Rev. B 59, 7413 (1999)
16	GGA, HTCH93 of F.A. Hamprecht, A.J. Cohen, D.J. Tozer, N.C. Handy, J. Chem. Phys. 109, 6264 (1998)
17	GGA, HTCH120 of A.D. Boese, N.L. Doltsinis, N.C. Handy, and M. Sprik, J. Chem. Phys 112, 1670 (1998) – The usual HCTH functional
20	Fermi-Amaldi xc (-1/N Hartree energy, where N is the number of electrons per cell G=0 is not taken into account however), for TDDFT tests. No spin-polarisation.
21	same as 20, except that the xc-kernel is the LDA (ixc=1) one,

	for TDDFT tests
22	same as 20, except that the xc-kernel is the Burke-Petersilka-Gross hybrid, for TDDFT tests
23	GGA of Z. Wu and R.E. Cohen, Phys. Rev. 73, 235116 (2006)
26	GGA, HTCH147 of A.D. Boese, N.L. Doltsinis, N.C. Handy, and M. Sprik, J. Chem. Phys 112, 1670 (1998)
27	GGA, HTCH407 of A.D. Boese, and N.C. Handy, J. Chem. Phys 114, 5497 (2001)
100	Hartree-Fock exchange only

A.2 libXC functional codes

In the input file '`input.dft`', you have to specify a code from this table made with a negative sign concatenated with two integers, one for exchange part and another one for correlation part.

XC_LDA_X	1	Exchange
XC_LDA_C_WIGNER	2	Wigner parametrization
XC_LDA_C_RPA	3	Random Phase Approximation
XC_LDA_C_HL	4	Hedin & Lundqvist
XC_LDA_C_GL	5	Gunnarson & Lundqvist
XC_LDA_C_XALPHA	6	Slater Xalpha
XC_LDA_C_VWN	7	Vosko, Wilk, & Nussair
XC_LDA_C_VWN_RPA	8	Vosko, Wilk, & Nussair (RPA)
XC_LDA_C_PZ	9	Perdew & Zunger
XC_LDA_C_PZ_MOD	10	Perdew & Zunger (Modified)
XC_LDA_C_OB_PZ	11	Ortiz & Ballone (PZ)
XC_LDA_C_PW	12	Perdew & Wang
XC_LDA_C_PW_MOD	13	Perdew & Wang (Modified)
XC_LDA_C_OB_PW	14	Ortiz & Ballone (PW)
XC_LDA_C_2D_AMGB	15	Attacalite et al
XC_LDA_C_2D_PRM	16	Pittalis, Rasanen & Marques correlation in 2D
XC_LDA_C_vBH	17	von Barth & Hedin
XC_LDA_C_1D_CSC	18	Casula, Sorella, and Senatore 1D correlation
XC_LDA_X_2D	19	Exchange in 2D
XC_LDA_XC_TETER93	20	Teter 93 parametrization
XC_LDA_X_1D	21	Exchange in 1D
XC_GGA_X_PBE	101	Perdew, Burke & Ernzerhof exchange
XC_GGA_X_PBE_R	102	Perdew, Burke & Ernzerhof exchange (revised)
XC_GGA_X_B86	103	Becke 86 Xalpha,beta,gamma
XC_GGA_X_B86_R	104	Becke 86 Xalpha,beta,gamma (reoptimized)
XC_GGA_X_B86_MGC	105	Becke 86 Xalpha,beta,gamma (with mod. grad. correction)
XC_GGA_X_B88	106	Becke 88
XC_GGA_X_G96	107	Gill 96
XC_GGA_X_PW86	108	Perdew & Wang 86

XC_GGA_X_PW91	109	Perdew & Wang 91
XC_GGA_X_OPTX	110	Handy & Cohen OPTX 01
XC_GGA_X_DK87_R1	111	dePristo & Kress 87 (version R1)
XC_GGA_X_DK87_R2	112	dePristo & Kress 87 (version R2)
XC_GGA_X_LG93	113	Lacks & Gordon 93
XC_GGA_X_FT97_A	114	Filatov & Thiel 97 (version A)
XC_GGA_X_FT97_B	115	Filatov & Thiel 97 (version B)
XC_GGA_X_PBE_SOL	116	Perdew, Burke & Ernzerhof exchange (solids)
XC_GGA_X_RPBE	117	Hammer, Hansen & Norskov (PBE-like)
XC_GGA_X_WC	118	Wu & Cohen
XC_GGA_X_mPW91	119	Modified form of PW91 by Adamo & Barone
XC_GGA_X_AM05	120	Armiento & Mattsson 05 exchange
XC_GGA_X_PBEA	121	Madsen (PBE-like)
XC_GGA_X_MPBE	122	Adamo & Barone modification to PBE
XC_GGA_X_XPBE	123	xPBE reparametrization by Xu & Goddard
XC_GGA_X_2D_B86_MGC	124	Becke 86 MGC for 2D systems
XC_GGA_X_BAYESIAN	125	Bayesian best fit for the enhancement factor
XC_GGA_X_PBE_JSJR	126	JSJR reparametrization by Pedroza, Silva & Capelle
XC_GGA_X_2D_B88	127	Becke 88 in 2D
XC_GGA_X_2D_B86	128	Becke 86 Xalpha,beta,gamma
XC_GGA_X_2D_PBE	129	Perdew, Burke & Ernzerhof exchange in 2D
XC_GGA_C_PBE	130	Perdew, Burke & Ernzerhof correlation
XC_GGA_C_LYP	131	Lee, Yang & Parr
XC_GGA_C_P86	132	Perdew 86
XC_GGA_C_PBE_SOL	133	Perdew, Burke & Ernzerhof correlation SOL
XC_GGA_C_PW91	134	Perdew & Wang 91
XC_GGA_C_AM05	135	Armiento & Mattsson 05 correlation
XC_GGA_C_XPBE	136	xPBE reparametrization by Xu & Goddard
XC_GGA_C_LM	137	Langreth and Mehl correlation
XC_GGA_C_PBE_JRGX	138	JRGX reparametrization by Pedroza, Silva & Capelle
XC_GGA_X_OPTB88_VDW	139	Becke 88 reoptimized to be used with vdW functional of Dion et a
XC_GGA_X_PBEK1_VDW	140	PBE reparametrization for vdW
XC_GGA_X_OPTPBE_VDW	141	PBE reparametrization for vdW
XC_GGA_XC_LB	160	van Leeuwen & Baerends
XC_GGA_XC_HCTH_93	161	HCTH functional fitted to 93 molecules
XC_GGA_XC_HCTH_120	162	HCTH functional fitted to 120 molecules
XC_GGA_XC_HCTH_147	163	HCTH functional fitted to 147 molecules
XC_GGA_XC_HCTH_407	164	HCTH functional fitted to 147 molecules
XC_GGA_XC_EDF1	165	Empirical functionals from Adamson, Gill, and Pople
XC_GGA_XC_XLYP	166	XLYP functional
XC_GGA_XC_B97	167	Becke 97
XC_GGA_XC_B97_1	168	Becke 97-1
XC_GGA_XC_B97_2	169	Becke 97-2
XC_GGA_XC_B97_D	170	Grimme functional to be used with C6 vdW term

XC_GGA_XC_B97_K	171	Boese-Martin for Kinetics
XC_GGA_XC_B97_3	172	Becke 97-3
XC_GGA_XC_PBE1W	173	Functionals fitted for water
XC_GGA_XC_MPWLYP1W	174	Functionals fitted for water
XC_GGA_XC_PBELYP1W	175	Functionals fitted for water
XC_GGA_XC_SB98_1a	176	Schmider-Becke 98 parameterization 1a
XC_GGA_XC_SB98_1b	177	Schmider-Becke 98 parameterization 1b
XC_GGA_XC_SB98_1c	178	Schmider-Becke 98 parameterization 1c
XC_GGA_XC_SB98_2a	179	Schmider-Becke 98 parameterization 2a
XC_GGA_XC_SB98_2b	180	Schmider-Becke 98 parameterization 2b
XC_GGA_XC_SB98_2c	181	Schmider-Becke 98 parameterization 2c
XC_HYB_GGA_XC_B3PW91	401	The original hybrid proposed by Becke
XC_HYB_GGA_XC_B3LYP	402	The (in)famous B3LYP
XC_HYB_GGA_XC_B3P86	403	Perdew 86 hybrid similar to B3PW91
XC_HYB_GGA_XC_O3LYP	404	hybrid using the optx functional
XC_HYB_GGA_XC_mPW1K	405	mixture of mPW91 and PW91 optimized for kinetics
XC_HYB_GGA_XC_PBEH	406	aka PBE0 or PBE1PBE
XC_HYB_GGA_XC_B97	407	Becke 97
XC_HYB_GGA_XC_B97_1	408	Becke 97-1
XC_HYB_GGA_XC_B97_2	410	Becke 97-2
XC_HYB_GGA_XC_X3LYP	411	maybe the best hybrid
XC_HYB_GGA_XC_B1WC	412	Becke 1-parameter mixture of WC and PBE
XC_HYB_GGA_XC_B97_K	413	Boese-Martin for Kinetics
XC_HYB_GGA_XC_B97_3	414	Becke 97-3
XC_HYB_GGA_XC_mPW3PW	415	mixture with the mPW functional
XC_HYB_GGA_XC_B1LYP	416	Becke 1-parameter mixture of B88 and LYP
XC_HYB_GGA_XC_B1PW91	417	Becke 1-parameter mixture of B88 and PW91
XC_HYB_GGA_XC_mPW1PW	418	Becke 1-parameter mixture of mPW91 and PW91
XC_HYB_GGA_XC_mPW3LYP	419	mixture of mPW and LYP
XC_HYB_GGA_XC_SB98_1a	420	Schmider-Becke 98 parameterization 1a
XC_HYB_GGA_XC_SB98_1b	421	Schmider-Becke 98 parameterization 1b
XC_HYB_GGA_XC_SB98_1c	422	Schmider-Becke 98 parameterization 1c
XC_HYB_GGA_XC_SB98_2a	423	Schmider-Becke 98 parameterization 2a
XC_HYB_GGA_XC_SB98_2b	424	Schmider-Becke 98 parameterization 2b
XC_HYB_GGA_XC_SB98_2c	425	Schmider-Becke 98 parameterization 2c
XC_MGGA_X_LTA	201	Local tau approximation of Ernzerhof & Scuseria
XC_MGGA_X_TPSS	202	Perdew, Tao, Staroverov & Scuseria exchange
XC_MGGA_X_M06L	203	Zhao, Truhlar exchange
XC_MGGA_X_GVT4	204	GVT4 from Van Voorhis and Scuseria (exchange part)
XC_MGGA_X_TAU_HCTH	205	tau-HCTH from Boese and Handy
XC_MGGA_X_BR89	206	Becke-Roussel 89
XC_MGGA_X_BJ06	207	Becke & Johnson correction to Becke-Roussel 89
XC_MGGA_X_TB09	208	Tran & Blaha correction to Becke & Johnson
XC_MGGA_X_RPP09	209	Rasanen, Pittalis, and Proetto correction to Becke & Johnson

XC_MGGA_C_TPSS	231	Perdew, Tao, Staroverov & Scuseria correlation
XC_MGGA_C_VSXC	232	VSxc from Van Voorhis and Scuseria (correlation part)
XC_LCA_OMC	301	Orestes, Marcasso & Capelle
XC_LCA_LCH	302	Lee, Colwell & Handy