A user's guide for *atompaw* code

Marc Torrent
Commissariat à l'Energie Atomique et aux Energies Alternatives
DAM, DIF. F-91297 Arpajon - France

Natalie A. W. Holzwarth
Wake Forest University. Winston-Salem, NC 27109 - USA

Contact emails : marc.torrent@cea.fr, natalie@wfu.edu
Source code URL : http://pwpaw.wfu.edu

Revised August 2, 2013

*Compatible with **atompaw v4.0** and later*

METHOD FOR PAW DATASET GENERATION 2

HOW TO USE *ATOMPAW* 3

INPUT FILE FOR *ATOMPAW* 4

DETAILED DESCRIPTION OF KEYWORDS 5

1. **Atomic all-electron computation** 5
2. **Partial-wave basis generation** 8
3. **Test configurations** 11
4. **Output for various DFT codes** 12
   ABINIT — *http://www.abinit.org* 12
   Quantum Expresso — *http://www.pwscf.org* 14

EXAMPLES 15

ADVICE FOR USE 17

Short write-up 17

Detailed write-up 18

APPENDIX 22

Appendix A: use of LibXC library 23

Appendix B: comparison between DFT codes 24

Method for PAW dataset generation

PAW calculations require, for each atomic species, a set of basis (partial-waves) and projectors functions plus some additional atomic data stored in a *PAW dataset*. A PAW dataset has to be generated in order to reproduce atomic behavior as accurately as possible while requiring minimal CPU and memory resources in executing the DFT code for the crystal simulations. These two constraints are conflicting.

The PAW dataset generation is done according the following procedure:

All parameters that should be given in an *atompaw* input file are in **bold**.

1- Choose and define the concerned chemical species: **name** and **atomic number.**

2- Solve the atomic all-electrons problem in a given atomic configuration. The atomic problem is solved within the DFT formalism, using an **exchange-correlation functional** and either a Schrödinger (default) or **scalar-relativistic approximation**. It is a spherical problem and it is solved on a **radial grid**. Other approximations can be given (as, for example, the behavior of the **nuclear potential**). The atomic problem is solved for a given **electronic configuration** that can be an ionized/excited one.

3- Choose a set of electrons that will be considered as frozen around the nucleus (**core electrons**). The others electrons are valence ones and will be used in the PAW basis. The **core density** is then deduced from the core electrons wave functions. A **smooth core density** equal to the core density outside a given r_{core} **matching radius** is computed.

4- Choose the **size of the PAW basis** (number of partial-waves and projectors). Then choose the partial-waves included in the basis. The later can be **atomic eigen-functions** related to valence electrons (bound states) — in fact this is mandatory with *atompaw* — and/or **additional atomic functions**, solution of the wave equation for a given l **quantum number** at arbitrary **reference energies** (unbound states).

5- Generate **pseudo partial-waves** (smooth partial-waves build with a **pseudization scheme** and equal to partial-waves outside a given r_c **matching radius**) and associated **projector functions**. Pseudo partial-waves are solutions of the PAW Hamiltonian deduced from the atomic Hamiltonian by pseudizing the effective potential (a local **pseudopotential** is built and equal to effective potential outside a r_{vloc} **matching radius**). Projectors and partial-waves are then orthogonalized with a chosen **orthogonalization scheme**.

6- Build a *compensation charge density* used later in order to retrieve the total charge of the atom. This compensation charge density is located inside the PAW spheres and based on an analytical **shape function** (which analytic form and **localization radius** r_{shape} can be chosen).

7- Eventually, if desired, test the resulting PAW dataset on several **electronic test configurations**.

How to use *atompaw*

1- Compile *atompaw*:

atompaw uses the standard *Linux* installation procedure (*autoconf*) …

- In atompaw source tree, type: mkdir build ; cd build
  ../configure

  If the *configure* script complains, add additional options:
  FC=… to select a specific Fortran compiler
  --prefix="…" to specify the destination directory
  --with-linalg-libs="…" to select your Blas/Lapack libraries
  --enable-libxc --with-libxc-incs=" " --with-libxc-libs=" " to add LibXC support (see appendix)
  configure --help *for all available options*…
- Then compile and install the code: make ; make install

2- Edit an input file in a text editor (content of input is explained in the following).

3- Run *atompaw*: atompaw < inputfile

Partial-waves, PS partial-waves and projectors are given in wfn.i files.

Logarithmic derivatives from atomic Hamiltonian and PAW Hamiltonian resolutions are given in logderiv.l files.

A summary of the atomic all-electrons computation and PAW dataset properties can be found in the Atom_name file (*Atom_name* is the first parameter of the input file).

Resulting PAW dataset can be output in several different formats:

- Atom_name.atomicdata file (keyword: PWPAWOUT)

  Specific format for ***pwpaw,*** and ***soccoro*** codes
- Atom_name.XCfunc.xml file (keyword: XMLOUT)

  Normalized *xml* file according to specifications from http://wiki.fysik.dtu.dk/stuff//pawxml/pawxml.xhtml; can be used with ***abinit*** code
- Atom_name.XCfunc-paw.abinit file (keyword: ABINITOUT)

  Specific format for ***abinit*** code
- Atom_name.XCfunc-paw.upf file (keyword: PWSCFOUT)

  Specific format for ***pwscf*** code

Additional details can be found in:

- "Notes for revised form of atompaw code"
  http://www.wfu.edu/~natalie/papers/pwpaw/notes/atompaw/atompawEqns.pdf
- "Part I manuscript.pdf"
  http://dx.doi.org/10.1016/S0010-4655(00)00244-7
  (be careful: some obsolete chapters inside).

Input file for *atompaw*

In red, mandatory arguments
In green, optional arguments

Keywords are in normal font
Numbers are in italics

Atomic all-electrons computation

```
Atom_name Z
XC_functional rel_keyword nucleus_keyword grid_keyword logderivrange
n_s^max n_p^max n_d^max n_f^max n_g^max
n l occ_nl
n l occ_nl
...
0 0 0
c or v
c or v
c or v
c or v
c or v
```

One line for each empty or partially occupied (n,l) state

One line for each (n,l) state
l=0 states first
then l=1 states…

Partial-waves basis generation

```
...
l_max
r_paw r_shape r_vloc r_core
y
E_ref
n
y
E_ref
n
y
E_ref
n
...
y
E_ref
n
projector_keyword ps_scheme ortho_scheme shapefunction
l_loc E_loc Vloc_scheme
rc_1
```

Repeated for each additional l=0 partial-wave

Repeated for each additional l=1 partial-wave

Repeated for each additional $l=l_{max}$ partial-

One paragraph for each $0 \le l \le l_{max}$

If `projector_keyword` ≠ "Bloechl"
One line for each partial-wave

Test configurations

```
rc_2
...
rc_basis_size
1
n l occ_nl
n l occ_nl
...
0 0 0
2
```

One line for each empty or partially occupied (n,l) state

As many times as desired

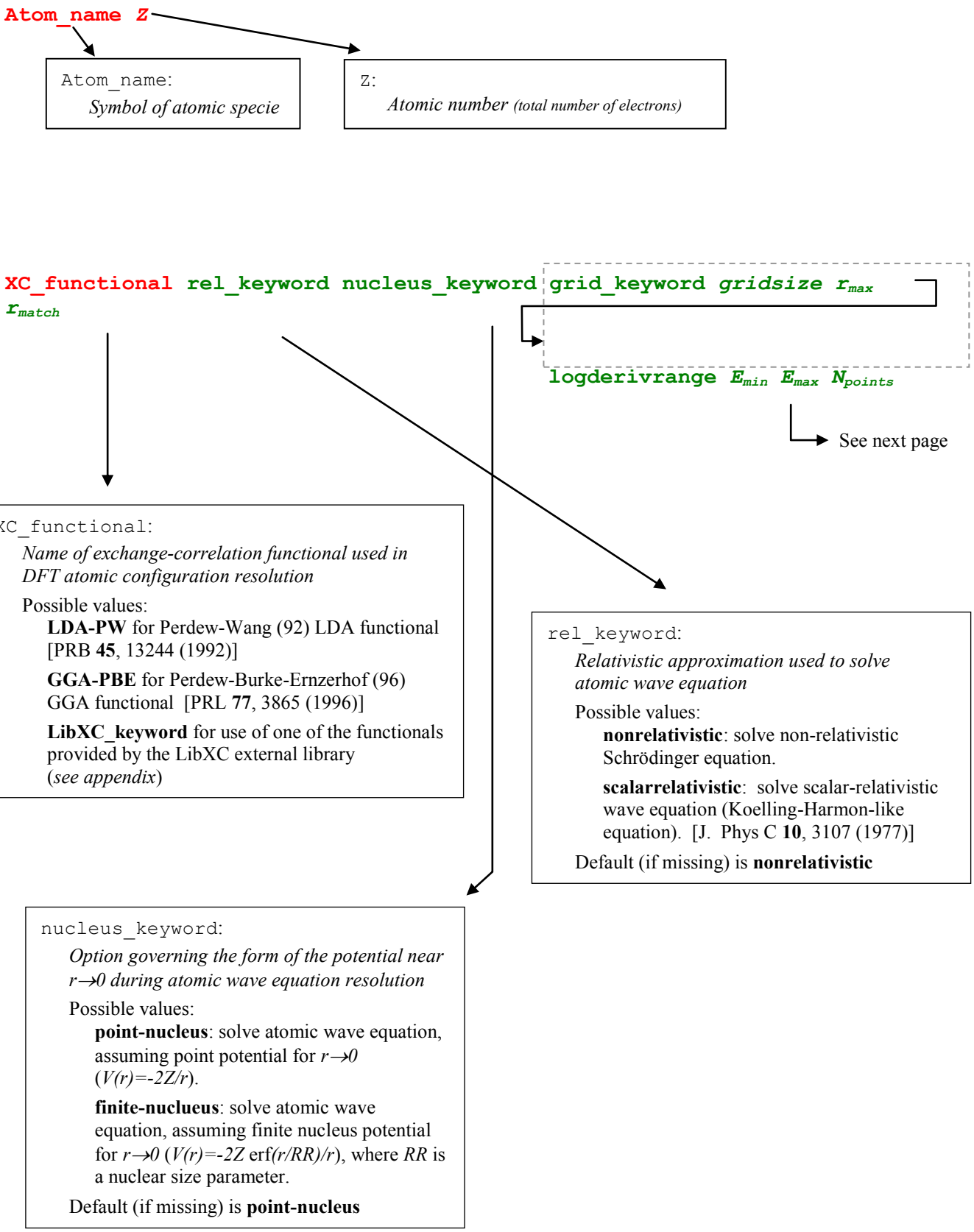
Output for various codes

Output for ABINIT

Output for PWscf

Detailed description of keywords

1. Atomic all-electron computation



These keywords should be on the same line as previous ones…

`grid_keyword` *gridsize* r_{max} r_{match}:

Options governing the analytic form of the radial grid used in atompaw

Analytical form of the grid is determined by `grid_keyword` *Its step and size can be defined by* `gridsize`, r_{max} *and* r_{match}.

Possible values for `grid_keyword`:

lineargrid: use a linear grid : $r_i=h(i-1)$

loggrid: use a logarithmic grid : $r_i=(h/Z).(exp[h(i-1)]-1)$

loggridv4: use a logarithmic grid : $r_i=(r_0/Z).(exp[h(i-1)]-1);$ with $r_0=10^{-5}$

Default (if missing) is **lineargrid**

Additional (optional) arguments:

gridsize:

Define the number of points in the grid.

Default (if missing) is **20001** when grid is linear
2001 when grid is logarithmic

r_{max}: (atomic units)

Optional argument, but if present must follow `gridsize` in the input.
Define the maximum radius of the grid.

Default (if missing) is **50. a.u.** when grid is linear
80 a.u. when grid is logarithmic (**loggrid**)
100 a.u. when grid is logarithmic (**loggridv4**)

r_{match}: (atomic units)

Optional argument , but if present must follow r_{max} in the input.
This changes the usage of `gridsize` so that the value of r_{match} defines an explicit grid point by adjusting the step size (h) so that there are `gridsize` grid points between 0 and r_{match}.
A typical value for r_{match} is the PAW radius, r_{paw} (in order to keep it constant when the grid size changes). The grid is then continued to the first point $r_i>=r_{max}$.

Default (if missing) is $\boldsymbol{r_{max}}$

`logderivrange` E_{min} E_{max} N_{points} :

Options governing the plotting of logarithmic derivative

logderivrange is an optional argument.

Additional (optional) arguments:

E_{min}: (Rydberg)

Optional argument, but if present must follow `logderivrange` in the input.
Define the minimum energy of the range used to plot logarithmic derivatives.

Default (if missing) is **-5.0 Rydberg**

E_{axn}: (Rydberg)

Optional argument, but if present must follow E_{min} in the input.
Define the maximum energy of the range used to plot logarithmic derivatives.

Default (if missing) is **4.95 Rydberg**

N_{points}:

Optional argument, but if present must follow E_{max} in the input.
Define the number of points (energies) used to plot logarithmic derivatives.

Default (if missing) is **200**

n_s^{max} n_p^{max} n_d^{max} n_f^{max} n_g^{max}

n_l^{max}:
Maximum n quantum number for l electrons

Example:
For Nickel ($1s^2\ 2s^2\ 2p^6\ 3s^2\ 3p^6\ 3d^8\ 4s^2$), enter:

```
4 3 3 0 0
```

n l occ_{nl}
n l occ_{nl}
...
0 0 0

One line for each empty or partially occupied (n,l) state

For each electronic shell of the atomic species, enter a line with:
n,l: quantum numbers of the shell
occ_{nl}: electronic occupation of the shell

Actually, only empty or partially occupied shells are needed; full shells can be omitted.

Charged/excited configurations are (of course) accepted.

A "0 0 0" (zero zero zero) line ends the configuration.

Example:
For excited Nickel ($1s^2\ 2s^2\ 2p6\ 3s^2\ 3p^6\ 3d^{8.5}\ 4s^{1.5)}$,
simply enter:

```
3 2 8.5
4 0 1.5
```

c or **v**
c or **v**
c or **v**
c or **v**
c or **v**
...

One line for each (n,l) state
l=0 states first
then l=1 states…

c or v:
Core or valence characteristic of electronic shell

For each electronic shell, enter a **c** or **v** keyword, which can be:

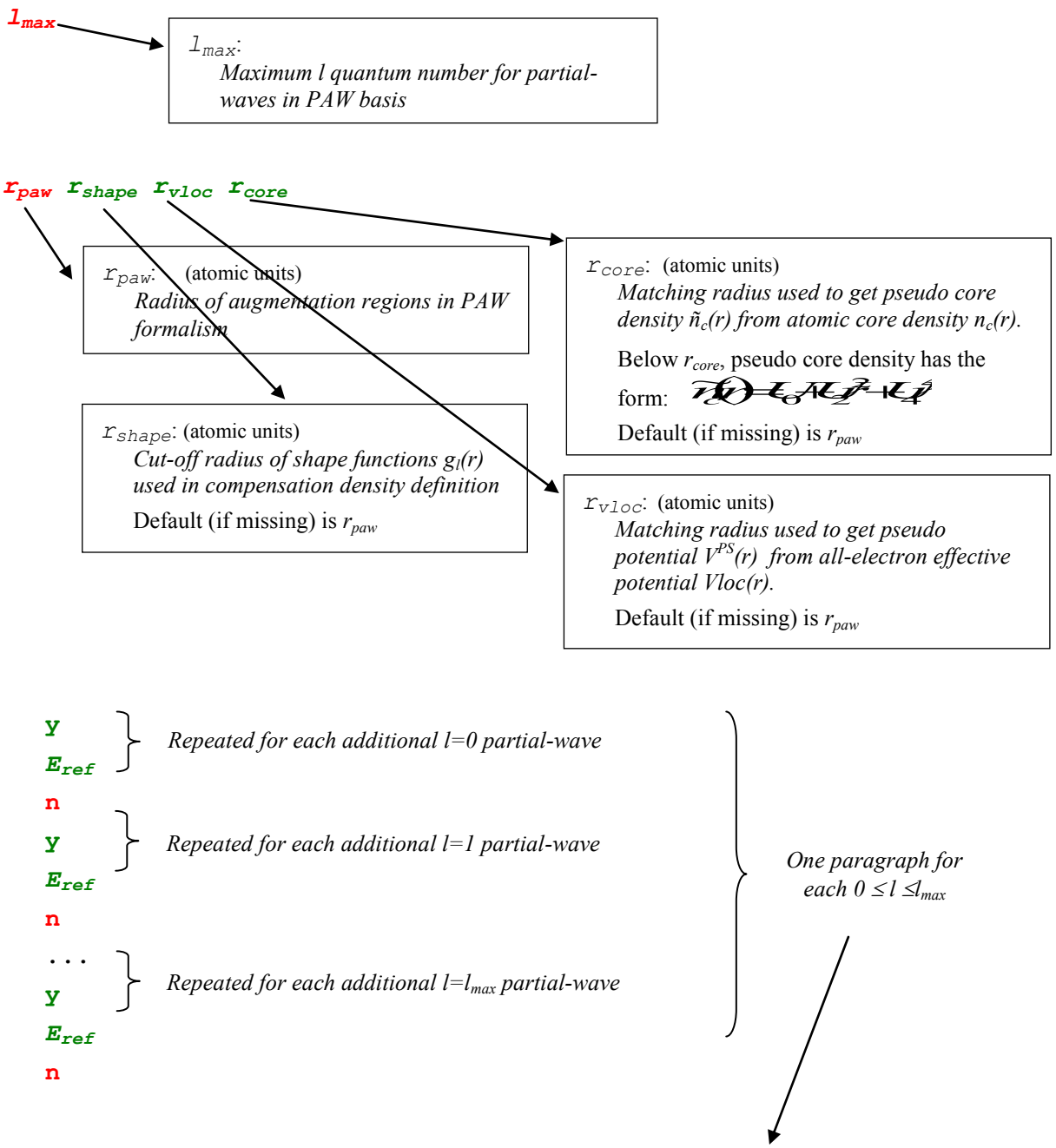
c: the electronic shell is a CORE shell, frozen around the nucleus and included in the core density of the PAW data set.

v: the electronic shell is a VALENCE shell containing valence electrons included in the PAW data set. In addition, note that the partial-wave associated with such a valence state will be NECESSARILY included in the PAW partial-waves basis (see below).

Example:
For Nickel ($1s^2\ 2s^2\ 2p^6\ 3s^2\ 3p^6\ 3d^8\ 4s^2$), with $3p^6\ 3d^8\ 4s^2$ in the valence, enter:

```
c              ! 1s
c              ! 2s
c              ! 3s
v              ! 4s valence
c              ! 2p
v              ! 3p valence
v              ! 3d valence
```

2. Partial-wave basis generation



Definition of partial-waves basis elements:

By construction, the basis already contains each atomic wave function associated with a valence state (each wave function marked as "v" in the atomic all-electron configuration). These are "bound states".

To add additional basis elements ("unbound states"), proceed as follow:

For each l angular momentum (from 0 to l_{max}),

1- Enter **y** to add an additional partial-wave

2- Enter $\boldsymbol{E_{ref}}$ (real number, Rydberg units), reference energy used to build the partial-wave. The later is obtained by inverting the Schrödinger equation at energy E_{ref} and l angular momentum.

Go to point 1- to add another partial-wave associated with l

Or

Enter **n**

projector_keyword ps_scheme ortho_scheme shapefunction *tol*

See next page

`projector_keyword`:

Option governing the scheme used to generate (smooth) PS partial-waves and associated projectors

Possible values:

Bloechl [or VNCT]: use P. Blöchl PS wave functions and projectors generation scheme [PRB **50**, 17953 (1994)]: A cutoff-function $k(r)=[\sin(\pi r/r_{paw})/(\pi r/r_{paw})]^2$ is used (in a Schrödinger-like equation) to deduce PS partial-waves. Projectors are then orthogonalized with a Gram-Schmidt procedure.
In that case, `ps_scheme` and `ortho_scheme` keywords are ignored.

Vanderbilt [or VNCTV]: use a polynomial function to "pseudize" partial-waves and D. Vanderbilt projectors generation scheme [PRB **41**, 7892 (1990)]:The polynomial function used to "pseudize" partial-waves is identical as the one used when `ps_scheme`=**polynom** (see below)
In that case, `ps_scheme` and `ortho_scheme` keywords are ignored.

custom: get PS wave functions according to `ps_scheme` keyword (see below) and projectors according to `ortho_scheme` keyword (see below).

modrrkj: use modified RRKJ form for wave function; can be used with **vanderbiltortho**, **gramschmidtortho**, or **svdortho** values for the ortho_scheme

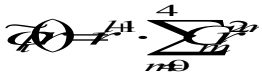
`ps_scheme`:

Option governing the scheme used to generate (smooth) PS partial-waves when `projector_keyword`=**custom**

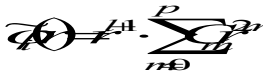
Possible values:

bloechlps: use P. Bloechl PS wave functions and projectors generation scheme [PRB **50**, 17953 (1994)]: a cutoff-function $k(r)=[\sin(\pi r/r_c)/(\pi r/r_c)]^2$ is used (in a Schrödinger-like equation) to deduce PS partial-waves. In that case `ortho_scheme` keyword has to be **gramschmidtortho**.

polynom: use a eighth degree polynomial function to "pseudize" partial-waves.

Below matching radius, PS wave function has the form: 

polynom2 *p* q_{cut}: use a polynomial of degree 2***p*** to "pseudize" partial-waves.

Below matching radius, PS wave function has the form: 

For $m \geq 4$, C_m coefficients are computed so that to minimize Fourier coefficients of PS partial-wave for $q > q_{cut}$ (*Fourier filtering*).

Defaults values of ***p*** and q_{cut} (if missing) are: $p=4$; $q_{cut}=10.0$

rrkj: use RRKJ scheme to get PS wave functions [PRB **41**, 1227 (1990)].
Below matching radius, PS wave function is a sum of 2 Bessel functions:

Default (if missing) is **bloechlps**

`ortho_scheme`:

Option governing the scheme used to generate and orthogonalize projectors when `projector_keyword`=**custom**

Possible values:

gramschmidtortho: use a Gram-Schmidt –like procedure to orthogonalize projectors and PS partial-waves.

vanderbiltortho: use D. Vanderbilt procedure to orthogonalize projectors and PS partial-waves (see [PRB **41**, 7892 (1990)]).

Default (if missing) is **gramschmidtortho**

Note that:

`projector_keyword`=**vanderbilt** is strictly equivalent to "**custom polynom vanderbiltortho**"

`projector_keyword`=**bloechl** is equivalent to "**custom bloechlps gramschmidtortho**"
AND all r_c (defined later) equal to r_{paw}

`shapefunction` *tol*:

Option governing the analytic form of shape functions $g_l(r)$ used in compensation density definition

Can be:

sinc: $g_l(r) = N \cdot r^l \cdot k(r)$ with $k(r) = \left[\sin(\pi r / r_{shape})/(\pi r / r_{shape})\right]^2$

tol parameter is ignored and can be omitted

gaussian *tol*: $g_l(r) = N \cdot r^l \cdot k(r)$ with $k(r) = \exp\left[-(r/d)^2\right]$

d parameter is deduce so that $k(r_{shape})$=***tol***
Default of ***tol*** parameter (if missing) is 10^{-4}

besselshape: $g_l(r) = \alpha_1^l \cdot j_l(q_1^l r) + \alpha_2^l \cdot j_l(q_2^l r)$ (see [PRB **59**, 1758 (1999)])

tol parameter is ignored and can be omitted

Default (if missing) is **sinc**

l_{loc} E_{loc} `Vloc_scheme`

`Vloc_scheme`:

Option governing the scheme used to get $V^{PS}(r)$ (local) pseudopotential from all-electron effective potential $V_{eff}(r)$. Matching radius for pseudization is r_{vloc}.

Can be:

troulliermartins: use a norm-conserving Troullier-Martins scheme. A PS wave function is deduce from atomic one and chosen to have the form $\phi^{PS}(r) = r^{l_{loc}} e^{p(r)}$ for $r<r_{vloc}$ where *p* is an even 12th order polynomial. Then V^{PS} is deduced by inverting the wave equation at $l=\boldsymbol{l_{loc}}$ and $E=\boldsymbol{E_{loc}}$.

ultrasoft: use a pseudization scheme without norm conservation constraint. A PS wave function is deduce from atomic one and chosen to have the form $\phi^{PS}(r) = r^{l_{loc}} \sum_{n=0}^{3} c_n r^{2n}$ for $r<r_{vloc}$. Then V^{PS} is deduced by inverting the wave equation at $l=\boldsymbol{l_{loc}}$ and $E=\boldsymbol{E_{loc}}$.

bessel: V^{PS} is simply derived from V_{eff} by a simple pseudization scheme using a zero-order spherical Bessel function: $V^{PS}(r) = c\,\frac{\sin(qr)}{r}$ for $r<r_{vloc}$.

In that case, $\boldsymbol{l_{loc}}$ and $\boldsymbol{E_{loc}}$ are ignored and can be omitted.

Default (if missing) is **troulliermartins**

l_{loc} E_{loc}:

l quantum number and reference energy (Rydberg units) for use when `Vloc_scheme`=**troulliermartins** *or* `Vloc_scheme`=**ultrasoft**

rc_1
rc_2
...
rc_{basis_size}

If `Projector_keyword` ≠ "Bloechl"
One line for each partial-wave

rc_i: (atomic units)

Matching radius used to get pseudo partial-waves $\tilde{\varphi}_i(r)$ from partial-wave $\varphi_i(r)$.

As many radii as partial-waves have to be entered (one per line).

If `projector_keyword` is **bloechl** or **VNCT**, these radii DO NOT HAVE TO BE GIVEN. In that case, they all are taken as $r_c=r_{paw}$.

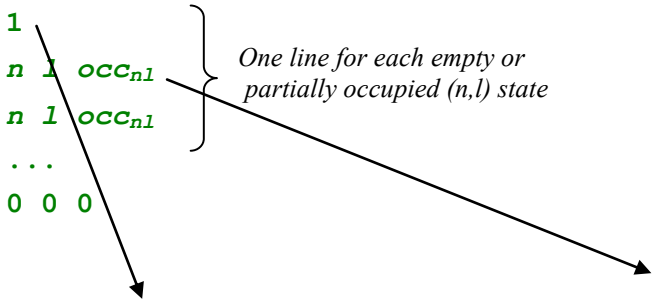
3. Options for further exploring, testing, and outputing datasets

After the basis and projector functions have been calculated, the program enters a keyword driven mode. The keywords can be listed in any order. Typically the user would choose to output the dataset in one or more formats (ABINITOUT, XMLOUT, PWSCFOUT, PWPAWOUT) or to try several different pseudopotential paprameters (EXPLORE) or test the given data set (SCFPAW). The looping structure of the program is stopped with an "END" or "0" keyword.

a. Test configurations

After the arguments used to generate the PAW dataset, the user can give electronic test configurations in order to test the validity of the created PAW dataset. For each electronic configuration the PAW Hamiltonian will be solved and resulting states printed.

Each test configuration has to be given as follow:



Enter **1 or SCFPAW** to begin a new test configuration

For each electronic shell of the atomic specie, enter a line with:
n,l: quantum numbers of the shell
occ_{nl}: electronic occupation of the shell

Actually, only shells whose electronic occupation is different from the all-electron computation one are needed; other shells can be omitted.

Charged/excited configurations are (of course) accepted.

A "0 0 0" (zero zero zero) line ends the configuration.

To end the list of configurations:

Enter a line with 0 (zero) to finish the calculations:

0

Tests configurations are not mandatory and one can directly enter another integer value (0, 2, 3, or 4) or keyword without having given any configuration.

b. Output for various DFT codes

In the rest of the input file, the user can optionally ask ***atompaw*** to write the PAW dataset in a specific format for various DFT codes. In addition, it is possible to apply a specific treatment to data for these codes.

ABINIT — *http://www.abinit.org*

To obtain a file formatted for ***abinit*** code, enter the following lines:

```
2
coreWF_keyword proj_optim_keyword comp_in_XC_keyword reduced_grid_keyword
```

Enter **2** or ABINITOUT to activate output in ***abinit*** format

See next page

`coreWF_keyword`:

*Option for the printing of core wave-function in a file formatted for **abinit**.*

Possible values:

noprtcorewf: no additional printing

prtcorewf: print an additional file, named `Atom_name.XCfunc-corewf.abinit` containing core wave-functions in ***abinit*** format.

Default (if missing) is **noprtcorewf**

`comp_in_XC_keyword`:

Option governing the use of compensation density in eXchange-Correlation potential.

Possible values:

noxcnhat: exchange-correlation potential does not include compensation density (Blöchl's formalism).
This choice is safer as it avoids numerical problems in XC terms calculation.
Compatible with ***abinit*** v6.1+

usexcnhat: exchange-correlation potential includes compensation density (Kresse's formalism).
This choice can produce numerical problems in XC calculation.

For further explanation, see [Comp. Phys. Comm. **181**, 1862 (2010)]

Default (if missing) is **noxcnhat**

`reduced_grid_keyword` *`gridsize logstep`*

Option for the use of a reduced grid.

This option in essentially useful when *atompaw* uses a linear grid, in order to reduce the grid size for the use in ***abinit***.

Possible values:

nospline: no additional printing

logspline: PAW dataset is transferred into a logarithmic grid (except non-local projectors). This gridgrid is defined by: *r(i>1)=a.exp[b.(i-2)]* and *r(1)=0* ; The user has to give the size of the grid `gridsize` and the «logarithmic step» (b in the above formula) `logstep`.

`gridsize`:
Optional argument, but if present must follow `logspline` in the input.
Define the size of the auxiliary logarithmic grid.
Default (if missing) is **350**

`logstep`:
Optional argument, but if present must follow `gridsize` in the input.
Define the logarithmic step of the auxiliary logarithmic grid.
Default (if missing) is **0.035**

Default (if missing) is **nospline**

---

*Output for **ABINIT** – continued…*

`proj_optim_keyword` *`ecut gfact werror`*:

Option for the optimization of projectors

Possible values:

nooptim: no additional printing

rsoptim: optimize projectors using "Real Space Optimization", as in [PRB **44**, 13063 (1991)]. It tries to improve the development of non-local projectors by "smoothing" their development over large G vectors (introducing a "controlled" error). The scheme is governed by 3 parameters: G_{max}, γ and W_l. The efficiency of Real Space Optimization strongly depends on the non-local projectors (it can sometimes be detrimental); only experienced users should use it.

`ecut`: (Rydberg)
Optional argument, but if present must follow `rsoptim` in the input.
Define the cut-off energy (E_{cut}) used to optimize the projectors ($G_{max}=E_{cut}$^2).
Default (if missing) is **10. Rydberg**

`gfact`:
Optional argument, but if present must follow `ecut` in the input.
Define the factor γ/G_{max} used to optimize the projectors.
Default (if missing) is **2**

`werror`:
Optional argument, but if present must follow `gfact` in the input.
Define the error W_l used to optimize the projectors.
Default (if missing) is **0.0001**

Default (if missing) is **nooptim**

Note:
If you just want to produce a PAW dataset for ***abinit***, without any additional data treatment, you can use the default keyword:

```
2
Default
Or
ABINITOUT
Default
```

To obtain a file formatted for ***PWscf*** code *(Unified Pseudopotential Format)*, enter the following lines:

```
3
upfdx upfxmin upfzmesh
```

Enter **3 or PWSCFOUT** to activate output in ***UPF*** format

`UPF grid_keywords` *(all are optional and may occur in any order)*

For the PAW mode, the *pwscf* code needs a logarithmic grid of the form:

$$r(i) = \frac{1}{\text{upfzmesh}}\left(e^{\text{upfxmin}}\ e^{\text{upfdx}\bullet(i-1)}\right)$$

upfzmesh:
Optional argument.
Define the inverse of the radial step of the grid..
Default (if missing) is **1.0 a.u.** $^{-1}$

upfxmin:
Optional argument.
`exp(upfxmin)` is the minimum radius given by the grid (i=1).
Default (if missing) is **-9.0**

upfdx:
Optional argument.
Define the logarithmic step of the grid.
Default (if missing) is **0.005**

Note for the generation of PAW datasets in UPF format:

The ***PWscf*** code uses the Kresse treatment [PRB **59**, 1758 (1999)] of the exchange-correlation functional which can lead to inaccuracies as explained in [Comp. Phys. Comm. **181**, 1862 (2010)]. To prevent these inaccuracies, we recommend using the BESSELSHAPE option for the compensation charge and choosing $r_{shape} = r_{paw}/1.2$.
For example, an input for Li including all electrons in the valence suitable for use with ***PWscf*** is as follows:

```
Li 3
GGA-PBE loggrid 2001
2 2 0 0 0 0
2 1 0
2 0 1
0 0 0
v
v
v
1
1.6    1.3    1.6    1.6
n
n
vanderbilt  besselshape
2 0
1.4
1.6
1.6
```

Other output formats:
4 or PWPAWOUT: output dataset for use in **pwpaw** or **Socorro** codes
5 or XMLOUT – output dataset in xml format

c. "explore" mode of the program
The EXPLORE (or 10) keyword puts allows the user to run the pseudofunction portion of the program multiple times to search for optimal parameter values. See the ATOMPAW_Explore_Userguide.pdf for more details.

Examples

A "minimal" input file

Boron [$1s^2$] $2s^2$ $2p^1$
4 partial-waves in basis

```
B 5.
LDA-PW
2 2 0 0 0
2 1 1.0
0 0 0
c
v
v
1
1.7
y
3.
n
y
3.
n
vanderbilt
2 0.
1.5
1.5
1.7
1.7
```

A "complete" input file

Nickel [$1s^2\ 2s^2\ 2s^6\ 3s^2\ 3p^6$] $3d^9\ 4s^1\ 4p^0$
6 partial-waves in basis

```
Nickel 28.
GGA-PBE  scalarrelativistic  point-nucleus  loggrid 1500 80. 2.3 loggderivrange -10. 10. 300
4 4 3 0 0           ! Up to 4s, 4p and 3d
1 0 2.0             ! Electronic configuration 3d^9 4s^1 4p^0
2 0 2.0
2 1 6.0
3 0 2.0
3 1 6.0
3 2 9.0
4 0 1.0
4 1 0.0
0 0 0
c                   ! 1s
c                   ! 2s
c                   ! 3s
v                   ! 4s valence
c                   ! 2p
c                   ! 3p
v                   ! 4p valence
v                   ! 3d valence
2                   ! Basis contains s, p and d partial-waves
2.3 2.3 1.1 2.2     ! rpaw=2.3, rshape=2.3, rveff=1.1, rcore=2.2
y                   ! Additional s partial-wave
4.                  ! at Eref=4.0 Ry
n
y                   ! Additional p partial-wave
4.                  ! at Eref=4.0 Ry
n
y                   ! Additional d partial-wave
2.5                 ! at Eref=2.5 Ry
n
custom rrkj gramschmidtortho sinc  ! RRKJ PW + sinc shape func.
Bessel              ! Simple Bessel Vloc
2.3                 ! Matching radius for Phi1 (l=0)
2.3                 ! Matching radius for Phi2 (l=0)
2.3                 ! Matching radius for Phi3 (l=1)
2.3                 ! Matching radius for Phi4 (l=1)
2.3                 ! Matching radius for Phi5 (l=2)
2.3                 ! Matching radius for Phi6 (l=2)
1
1 0 2.0             ! Test configuration 3d^8 4s^2
2 0 2.0
2 1 6.0
3 0 2.0
3 1 6.0
3 2 8.0
4 0 2.0
0 0 0
2                                                          ! Output for abinit
prtcorewf noxcnhat rsoptim 12. 2. 0.00001 logspline 500 0.03   ! abinit options
3                                                          ! Output for PWscf
upfdx 0.005  upfxmin -9.0  upfzmesh 1.0                    ! PWscf options
0                   ! END
```

Advice for use

In the following we give some keys for non-experienced users so that they can build input files for ***atompaw*** for new materials.

Short write-up

The first advice is to begin with a simple expression of the input file, setting most of the keywords to their default values.

- Concerning the all-electrons atomic computation, prefer a logarithmic grid; test the influence of the number of grid points and, in case of difficulties, choose a regular grid. Begin with a scalar-relativistic solution of the wave equation. If the system shows convergence problems, try non-relativistic choice (not recommended when Z becomes high).

- Concerning the partial-waves basis generation, simply begin with:
  - an unique radius r_{paw}
  - 2 partial-waves per l angular momentum (if r_{paw} is small enough, 1 wave per l may suffice)
  - "`bloechl`" choice for `projector_keyword`
  - a norm-conserving *Troullier-Martins* pseudopotential at $l_{loc}=l_{max}+1$ and $E_{loc}=0$.

  This choice should give a "stable" PAW dataset with correct physical results ; but Blöchl's scheme for projectors can produce "inefficient" datasets (in the sense that they may need a large number of plane waves to converge the DFT calculation). To increase performance, choose the "`vanderbilt`" option for `projector_keyword`. The gain can be noticeable. But, generally, the best choice (for performance) would be "`custom rrkj`" projectors.

- Concerning the pseudopotential $V^{PS}(r)$, norm-conserving *Troullier-Martins* is generally the best choice but it can produce "ghost states" for *d* and *f* materials. If this happens, a simple "`Bessel`" pseudopotential can solve the problem. But, in the later case, one has to noticeably decrease the matching radius r_{vloc} (try $0.6*r_{paw}$ first).

- The other keywords in the input file can be adjusted (by experienced users) in order to obtain better results on physical properties (by comparison with all-electrons calculations)…

Detailed write-up

Here is a proposal for using ***atompaw*** to build new PAW datasets (from scratch). The procedure detailed here should help the user to generate optimal datasets in most cases.

☑ **In a first stage, edit a simple input file for *atompaw*.**

- *In the all-electrons atomic computation part*
  - Define the material in the first line
  - Choose the exchange-correlation functional (LDA-PW or GGA-PBE) and select a scalar-relativistic wave equation and a (2000 points) logarithmic grid (second line).

    "`scalarrelativistic`" is recommended for high Z materials
  - Then define the electronic configuration; an excited configure may be useful if the PAW dataset is intended for use in a context where the material is charged (such as oxides). *Although, in our experience, the results are not highly dependent on the chosen electronic configuration.*
  - Select the core and valence electrons: in a first approach, select only electrons from outer shells. But, if particular thermo dynamical conditions are to be simulated, it is generally needed to include "semi-core states" in the set of valence electrons. Semi-core states are generally needed with transition metal and rare-earth materials. There are also some cases (such as P) where physical conditions do not indicate a need for semi-core states, but the use of semi-core states are needed to avoid the appearance of the dreaded ghost states.

    Note that all wave functions designated as valence electrons will be used in the partial-wave basis.

- *In the partial-waves basis generation part*

  Begin with a simple scheme. Select most of the keywords at their default values.
  - Enter only one matching radius (r_{paw}). Select it to be slightly less than half the inter-atomic distance in the solid (as a first choice).
  - Add additional partial-waves if needed: choose to have 2 partial-waves per angular momentum in the basis (this choice is not necessarily optimal but this is the most common one; if r_{paw} is small enough, 1 partial-wave per l may suffice). As a first guess, put all reference energies for additional partial-waves to 0 Rydberg.
  - Select a "`bloechl`" projector scheme and a norm-conserving *Troullier-Martins* pseudopotential at $l_{loc}=lmax+1$ and $E_{loc}=0$. "`bloechl`" will probably be changed later to make the PAW dataset more efficient.

- *In the test configuration part*
  - Add one test configuration; a good idea is to test (at least) the electronic configuration used in the all-electrons atomic computation part.

☑ **At this stage, run *atompaw* !**

The generated PAW dataset is a first draft. Several parameters have to be adjusted, in order to get accurate results and efficient DFT calculations.

1. **The sensitivity of results to some parameters has to be checked.**

   - The radial grid:
     Try to select 700 points in the logarithmic grid and check if any noticeable difference in the results appears. If yes, adjust the size of the grid (else, keep 700 points). If the results are difficult to get converged, try a regular grid…

     *For use with the **pwpaw** code, linear or logarithmic grids can be used.*
     *For use with the **abinit** code, the logarithmic grid is preferred.*

   - The relativistic approximation of the wave equation:
     `scalarrelativistic` option should give better results than non-relativistic one, but it sometimes produces difficulties for the convergence of the atomic problem (either at the all-electrons resolution step or at the PAW Hamiltonian solution step). If convergence cannot be reached, try a `nonrelativistic` calculation (not recommended for high Z materials).

   - A summary of the atomic all-electrons computation and the PAW dataset properties can be found in the Atom_name file (*Atom_name* is the first parameter of the input file). A look at the different values of *evale* (valence energy) is important. All-electron value has to be as close to others as possible. *evale* has to be insensitive to grids parameters.

2. **Have a look at the partial-waves, PS partial-waves and projectors.**

   Plot the wfn.i files in a graphical tool of your choice. You should get 3 curves per file: $\varphi_i(r)$, $\tilde{\varphi}_i(r)$ *and* $\tilde{p}_i(r)$

   - The $\tilde{\varphi}_i(r)$ should meet the $\varphi_i(r)$ near or after the last maximum (or minimum). If not, it is preferable to change the value of the matching radius rc_i.

   - The $\tilde{\varphi}_i(r)$ and $\tilde{p}_i(r)$ should have the same order of magnitude. If not, you can try to get this in three ways:
     - Change the matching radius rc_i for this partial-wave; but this is not always possible… spheres cannot have a large overlap in the solid…
     - Change the pseudopotential scheme (see later).
     - If there are two (or more) partial waves for the considered l angular momentum, including additional partial waves (unbound states):
       Decreasing the magnitude of projector is possible by displacing the references energies. Moving the energies away from each other generally reduce the magnitude of projectors, but a too big difference between energies can lead to wrong logarithmic derivatives (see following chapter).

   - The two first values of *evale* (valence energy) in the Atom_name file have to be close. If not, choices for projectors and/or partial waves certainly are not judicious.

   - Example of difficulty with $\tilde{p}_i(r)$: when the amplitude of projectors becomes too large, *atompaw* can produce an error with the following message:

     ```
     No convergence in boundsep
     ```

     Followed by

     ```
     Best guess of eig, dele = xxxxx   yyyyy
     ```

     This happens during the PAW Hamiltonian resolution (which cannot be achieved). One can bypass the difficulty by generating “softer” projectors as explained just above.

3. **Have a look at the logarithmic derivatives.**

   They are printed in the logderiv.l *files. Each* logderiv.l *file correspond to l quantum number and contains the logarithmic derivative of the l-state,* $d\log(\psi(E))/dr$*, computed for exact atomic problem and with the PAW dataset.*

   - The 2 curves should be superimposed as much as possible. By construction, they are superimposed at the two energies corresponding to the two *l* partial-waves. If the superimposition is not good enough, the reference energy for the second *l* partial-wave should be changed.
   - Generally a discontinuity in the logarithmic derivative curve appears at $0<=E_0<=4$ Rydberg. A reasonable choice is to choose the 2 reference energies so that E_0 is in between (if possible, i.e. if one the 2 partial-waves correspond to an unbound state).
   - Too close reference energies produce "hard" projector functions. But moving reference energies away from each other can damage accuracy of logarithmic derivatives.
   - Another possible problem is the presence of a discontinuity in the PAW logarithmic derivative curve at an energy where the exact logarithmic derivative is continuous. This generally shows the presence of a "ghost state".

     First, try to change to value of reference energies; this sometimes can make the ghost state disappear.

     If not, it can be useful to:

     - Change the pseudopotential scheme. Norm-conserving pseudopotentials are sometimes so deep (attractive near $r=0$) that they produce ghost states. A first solution is to change the *l* quantum number used to generate the norm-conserving pseudopotential. But this is generally not sufficient. Changing the pseudopotential scheme is (in most cases) the only efficient cure.

       Select a simple "`bessel`" pseudopotential can solve the problem. But, in that case, one has to noticeably decrease the matching radius r_{vloc} if one wants to keep reasonable physical results. Loosing to much norm for the wave function associated to the pseudopotential can have dramatic effects on the results.

       Selecting a value of r_{vloc} between $0.6*r_{paw}$ and $0.8*r_{paw}$ is a good choice; but the best way to adjust r_{vloc} value is to have a look at the two first values of *evale* in Atom_name file. They have to be as equal as possible and are sensitive to the choice of r_{vloc}.

     - Change the matching radius rc_i for one (or both) *l* partial-wave(s). In some cases, changing rc_i can remove ghost states …

   In most cases (changing pseudopotential or matching radius), one has to ***restart the procedure from step 2.*** *(only for l partial-waves).*

4. **Now, one has to test the efficiency of the generated PAW dataset.**

   Run a DFT computation and determine the size of the plane wave basis needed to get a given accuracy. If the cut-off energy defining the plane waves basis is too high (higher than 20 Hartree, if matching radius has a reasonable value), some changes have to be made in the input file.

- First possibility: change `projector_keyword="bloechl"` by `projector_keyword="vanderbilt"`. Vanderbilt projectors generally are more localized in reciprocal space than Bloechl ones.

  Recheck the plane waves cut-off (in a DFT calculation)… it should have decrease (but this is not a general rule).

- Second possibility: use `RRKJ` pseudization for PS partial-waves (put `projector_keyword="custom"` and `ps_keyword="rrkj"`). This pseudization is particularly efficient and gives highly localized projectors (in reciprocal space). This choice has, in most cases, the best influence on the plane wave basis.

One has to note that:

- The localization of projectors in reciprocal space can (generally) be predicted by a look at tprod.i files. Such a file contains the curve of $q^2 \cdot \tilde{p}_i(q) \cdot \tilde{\phi}_i(q)$ as a function of q (reciprocal space variable). q is given in Bohr^{-1} units; it can be connected to the plane waves cut-off energy (in Hartree units) by: $E_{cut}=q_{cut}^2/2$. These quantities are only calculated for the bound states, since the Fourier transform of an extended function is not well-defined.
- Generating projectors with Blöchl's scheme often gives the guaranty to have stable calculations. *atompaw* ends without any convergence problem and DFT calculations run without any divergence (but they need high plane wave cut-off). Vanderbilt projectors (and even more "custom" projectors) sometimes produce instabilities during the PAW dataset generation process and/or the DFT calculations…

In most cases, after having changed the projector generation scheme, one has to ***restart the procedure from step 2.***

5. Finally, have a careful look at physical quantities obtained with the PAW dataset.

It can be useful to test their sensitivity to some input parameters:

- The analytical form and the cut-off radius r_{shape} of the shape function used in compensation charge density definition. By default a "`sinc`" function is used but "`gaussian`" shapes can have an influence on results. "`Bessel`" shapes are efficient and generally need a smaller cut-off radius (~0.8*r_{paw}).
- The matching radius r_{core} used to get pseudo core density from atomic core density.
- The integration of additional ("semi-core") states in the set of valence electrons.
- The pseudization scheme used to get $V^{PS}(r)$.

All these parameters have to be meticulously checked, especially if the PAW dataset is used for non-standard solid structures or thermo dynamical domains.

Appendix

Appendix A: use of LibXC library

LibXC is a library (available from the web) under *GNU-LGPL* written by M. Marques, that contains a large set of very varied exchange-correlations functionals.
Provided that it is linked to *LibXC* library, ***atompaw*** can use these exchange-correlation functionals (at present only LDA and GGA).

*Note : at present, only **abinit** code can use LibXC functionals.*

LibXC is available at : `http://www.tddft.org/programs/octopus/wiki/index.php/Libxc`

How to build *atompaw* with LibXC support

- ✓ Download *LibXC* tarball from
  `http://www.tddft.org/programs/octopus/wiki/index.php/Libxc:download`
- ✓ Build *LibXC* and install it with the standard *Linux* procedure (see *LibXC* manual) :
  `./configure ; make ; make install`

  Let's suppose in the following that *LibXC* is installed in **`~libxc`**
- ✓ Build ***atompaw*** with *LibXC* support
  At configure process, add the following options:
  ```
  --enable-libxc --with-libxc-incs="-I~libxc/include" \
  --with-libxc-libs="-L~libxc/lib -lxc"
  ```
  Then build and install atompaw :
  ```
  make ; make install
  ```

How to use *atompaw* with a LibXC functional

- ✓ Choose an *Exchange* functional and a *Correlation* functional, or directly one single *Exchange-Correlation* functional, in *LibXC* list:
  `http://www.tddft.org/programs/octopus/wiki/index.php/Libxc:manual#Available_functionals`
- ✓ If your choice is an *Exchange-Correlation* functional, put its name as `XC_functional` keyword in ***atompaw*** input file.

  If your choice is an *Exchange* and a *Correlation* functional, put the `XC_functional` keyword as a merge of the two names separated by a "+" (plus).

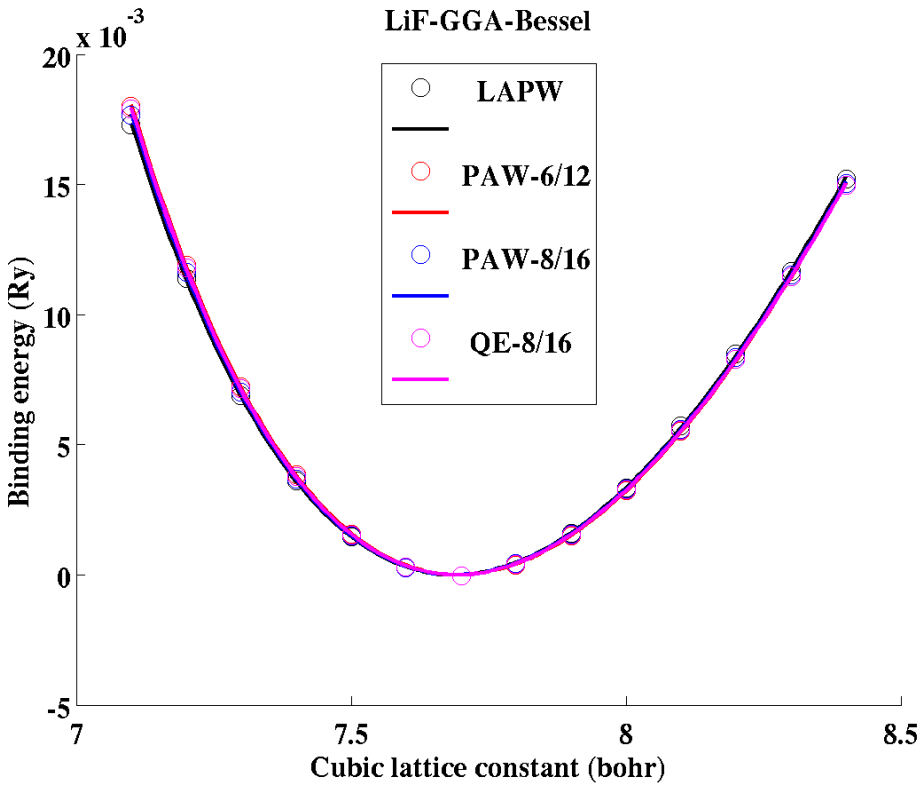
  Examples: if you want to use `XC_GGA_XC_B97`, put: `XC_GGA_XC_B97`
  if you want to use `XC_LDA_X` and `XC_LDA_C_PW`, put: `XC_LDA_X+XC_LDA_C_PW`

Example of input file using *LibXC* (GGA-PBE functional):

```
B 5.
XC_GGA_X_PBE+XC_GGA_C_PBE loggrid 2001
2 2 0 0 0
2 1 1.0
0 0 0
c
v
v
1
1.7
y
3.
n
y
3.
n
```

Note for experts: you also can address *LibXC* functionals in input file by their numerical identifier:
`XC_functional = LIBXC_101+LIBXC_130`

Appendix B: comparison between DFT codes

```
Li 3
GGA-PBE loggrid 2001
2 2 0 0 0 0
2 1 0
2 0 1
0 0 0
v
v
v
1
1.6    1.3    1.6    1.6
n
n
vanderbilt  besselshape
2 0
1.4
1.6
1.6
2
default
3
```

The results of the illustrated *atompaw* input file result in nearly identical results among the 3 PAW codes *pwpaw*, *abinit*, and *PWscf* and with the all-electron code WIEN2k which uses the LAPW method as shown in the following binding energy curve: