the APE manual

Atomic calculations
Pseudopotentials generation
August 2013

By Micael Oliveira and Fernando Nogueira.
Coimbra (Portugal)

This manual is for APE (Atomic Pseudopotentials Engine) 2.2.1, a density functional theory atomic program and pseudopotentials generator.

Copyright © 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Micael Oliveira and Fernando Nogueira

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation.

1 Copying

This program is “free”; this means that everyone is free to use it and free to redistribute it on a free basis. What is not allowed is to try to prevent others from further sharing any version of this program that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the program, that you receive source code or else can get it if you want it, that you can change this program or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the program, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for this program. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license are found in the General Public Licenses that accompany it.

2 Authors, Collaborators and Acknowledgments.

The main developing team of this program is composed of:

- Micael Oliveira (Universidade de Coimbra, Coimbra, Portugal),
- Fernando Nogueira (Universidade de Coimbra, Coimbra, Portugal)

Other people who made significant contributions to this program:

- Tiago Cerqueira (Universidade de Coimbra, Coimbra, Portugal),

3 Introduction

`APE` is a computer package designed to generate and test norm-conserving pseudopotentials within Density Functional Theory. The generated pseudopotentials can be either non-relativistic, scalar relativistic or fully relativistic and can explicitly include semi-core states. A wide range of exchange-correlation functionals is included.

More information about `APE` can be found in its homepage and in the following article:

M. J. T. Oliveira and F. Nogueira, *Generating relativistic pseudo-potentials with explicit incorporation of semi-core states using APE, the Atomic Pseudo-potentials Engine*, Comp. Phys. Comm. *178*, 524 (2008).

4 Installation

4.1 Quick instructions

If you are feeling lucky:

```
prompt> tar xzf ape<-version>.tar.gz
prompt> cd ape-<version>
prompt> ./configure
prompt> make
prompt> make install
```

This may not work but, before giving up, just read the following paragraphs.

4.2 Long instructions

The code is written in standard Fortran 95, with some routines written in C. To build it you will need both a C compiler (`gcc` works just fine), and a Fortran 95 compiler.

Besides the compiler, you will also need:

1. `make`: most computers have it installed, otherwise just grab and install the GNU `make`.
2. `cpp`: GNU `cpp` is just fine, but any `cpp` that accepts the `-C` flag (preserve comments) should work just as well.
3. GSL: The GSL is heavily used in `APE`. If you don't have it already installed in your system, you can obtain GSL from here. You will need version 1.0 or higher.
4. `libxc`: `libxc` is a library of exchange and correlation functionals. If you don't have it already installed in your system, you can obtain it here.

First you should obtain the code file, `ape<-version>.tar.gz`, (you probably have already done this). The code is freely available, and can be downloaded from `http://www.tddft.org/programs/APE`. There exists a `svn` server, which you can browse at `http://www.tddft.org/trac/APE/browser/`. The sources of the svn version (in general more unstable than the *official* distribution) may be downloaded by anonymous svn access:

```
prompt> svn co http://www.tddft.org/svn/APE/trunk ape
```

Uncompress and untar it (`tar xzf ape<-version>.tar.gz`). In the following, `APE-HOME` denotes the home directory of APE, created by the `tar` command.

`APE-HOME` contains the following subdirectories:

- `autom4te.cache, build`: contains files related to the building system. May actually not be there. Not of real interest for the plain user.
- `doc`: The documentation of `APE` in *texinfo* format.
- `liboct_parser`: The parser used by `APE`. This is the parser from `Octopus`.
- `sample`: Sample input files for `APE`.
- `src`: Fortran 90/95 source files. Note that these have to be preprocessed before being fed to the Fortran compiler, so do not be scared by the # directives.
- `testsuite`: `APE` regression testsuite.

Before configuring you can (should) setup a couple of options. Although the configure script tries to guess your system settings for you, we recommend that you set explicitly the default Fortran 95 compiler and the compiler options. For example, in `bash` you would typically do:

```
export FC=gfortran
export FCFLAGS="-O3 -Wall"
```

if you are using the GNU Fortran compiler on a Linux machine. Also, if you have some of the required libraries in some unusual directories, these directories may be placed in the variable `LDFLAGS` (e.g., `export LDFLAGS="-L/opt/lib/"`).

You can now run the configure script (`./configure`).[1]

There are some options you can use with the configure script. To obtain a full list just type `./configure --help`. Some commonly used options include:

- `--prefix=PREFIX`: Change the base installation dir of `APE` to `PREFIX`. The executable will be installed in `PREFIX/bin`, the libraries in `PREFIX/lib` and the documentation in `PREFIX/info`. `PREFIX` defaults to the home directory of the user who runs `configure`.
- `--with-gsl-prefix=DIR`: Installation directory of the GSL library. The libraries are expected to be in `DIR/lib` and the include files in `DIR/include`. The value of `DIR` is usually found by issuing the command `gsl-config --prefix`. (If the GSL library is installed, the program `gsl-config` should be somewhere.)
- `--with-libxc-prefix=DIR`: Installation directory of the `libxc` library. The libraries are expected to be in `DIR/lib` and the include files in `DIR/include`.

Run `make`, and then `make check`. This last command will launch the `APE` testsuite. If all tests were passed, you can then install the code with `make install`. If everything went fine, you should now be able to run `APE`.

The program has been tested in the following platforms:

- `i686-linux-gnu`: with the Intel and GNU Fortran compilers.
- `opteron`: with the PathScale compiler.

If you manage to compile/run `APE` on a different platform or with a different compiler, please let us know so we can update the list. Patches to solve compiler issues are also welcome.

Build the documentation in the format you prefer. Since you are reading this, you already have it in some format. Due to the power of `texinfo`, a series of formats are available, namely `dvi`, `html`, `pdf` and `info`. The `APE.texi` source code of this document is in the `APE-HOME/doc` directory.

[1] If you downloaded the svn version, you will not find the `configure` script. In order to compile the development version you will first have to run the GNU autotools. This may be done by executing the command `autoreconf -i`. Note that you need to have working versions of the `automake` (1.11), `autoconf` (2.68) and `libtool` (2.4) programs (the versions we currently use are between parentheses). Note that `autoreconf` will likely not work if you have (much) older versions of the autotools.

4.3 Troubleshooting

So, something went wrong. Here is a list of things that might have gone wrong.

Some tests of the testsuite fail: When running the testsuite, one or more tests fail.

- Is the test failing because of a very small numerical difference? While running a test, if a test case is failed, the code will print some extra information about the failure, namely the calculated value, the reference value, and the allowed tolerance. This should look like this:

```
Match Failed
 Calculated value : -24.344302
 Reference value  : -24.344198
 Difference       : 0.000104
 Tolerance        : 8e-5
```

  If the difference is slightly larger than the tolerance, like in the previous example, and if the same happens for all the failed test cases, then it is likely that there is nothing wrong with your compilation.

Could not find GSL library: We assume that you have already installed `GSL` but, for some reason, you were not able to compile the code.

- Did you pass the correct `--with-gsl-prefix` to the configure script? If your library is installed in a non-standard directory (like `/opt/gsl`), you will have to pass the script the location of the library (in this example, you could try `./configure --with-gsl-prefix=/opt/gsl`.

Could not find `libxc` library: We assume that you have already installed `libxc` but, for some reason, you were not able to compile the code.

- Did you pass the correct `--with-libxc-prefix` to the configure script? If your library is installed in a non-standard directory (like `/opt/libxc`), you will have to pass the script the location of the library (in this example, you could try `./configure --with-libxc-prefix=/opt/libxc`.
- Did you use the same Fortran compiler when compiling `libxc`? Unfortunately the compilation of Fortran modules generates compiler dependent files (usually with the .mod extension). Since APE uses the Fortran interface of `libxc`, it is required that you use the same Fortran compiler for `APE` and `libxc`.

Error while loading shared libraries: You have already compiled the code, but it fails to run, giving an error message about a failure in locating some shared object file.

- Did you set the `LD_LIBRARY_PATH` environment variable? If some of the libraries used to compile `APE` are installed in a non-standard directory (like `/opt/libxc`), you will have to add the path to the directory where the library objects are installed to the `LD_LIBRARY_PATH` environment variable. If you are using `bash`, this is done with the following command:

```
export LD_LIBRARY_PATH=/path/to/libxc/lib:$LD_LIBRARY_PATH
```

Fatal Error: Sometimes the code stops after issuing a `Fatal Error` message. Here is a list of some of the most common errors and a brief explanation of their origin and some possible solutions.

- *Unable to bracket eigenvalue for state xxx.* This error occurs when the eigensolver is unable to find the eigenvalue for a given orbital. Most of the times this will happen when an occupied orbital becomes unbound.
- *Generation of pseudopotentials from spin-polarized calculations is not implemented!* You are trying to generate a pseudopotential from a spin-polarized calculation, but this is not possible in `APE`. Although there are schemes to generate spin-dependent pseudopotentials, for most applications, spin-polarized calculations can be performed using pseudopotentials generated for a spin-unpolarized atom. This is because spin-polarization should come from the valence electrons. Furthermore, spin-dependent pseudopotentials cannot be used in the same way as spin-independent ones and most DFT codes don't know how to handle them.

Whatever went wrong...: If something else went wrong, please send an e-mail to the APE-users mailing list (ape-users at tddft.org). Note that you have to be subscribed to the mailing list in order to be able to post a message.

5 The parser

All input options should be in a file called "inp.ape", in the directory APE is run from. It is also possible to use a file with a different name. In that case the standard input should redirect to that file. For example, in bash:

```
prompt> ape < filename
```

For fairly comprehensive examples, just look at the files in the APE_HOME/samples directory.

At the beginning of the program the oct_parser library reads the inp.ape file, parses it, and generates a list of variables that will be read by APE (note that the input is case independent). There are two kind of variables, scalar values (strings or numbers), and blocks (that you may view as matrices). A scalar variable var can be defined by:

```
var = exp
```

var can contain any alphanumeric character plus "_", and exp can be a quote delimited string, a number (integer, real, or complex), a variable name, or a mathematical expression. In the expressions all arithmetic operators are supported ("a+b", "a-b", "a*b", "a/b"; for exponentiation the C syntax "a^b" is used), and the following functions can be used:

- sqrt(x): The square root of x.
- exp(x): The exponential of x.
- log(x) or ln(x): The natural logarithm of x.
- log10(x): Base 10 logarithm of x.
- sin(x), cos(x), tan(x), cot(x), sec(x), csc(x): The sinus, co-sinus, tangent, co-tangent, secant and co-secant of x.
- asin(x), acos(x), atan(x), acot(x), asec(x), acsc(x): The inverse (arc-) sinus, co-sinus, tangent, co-tangent, secant and co-secant of x.
- sinh(x), cosh(x), tanh(x), coth(x), sech(x), csch(x): The hyperbolic sinus, co-sinus, tangent, co-tangent, secant and co-secant of x.
- asinh(x), acosh(x), atanh(x), acoth(x), asech(x), acsch(x): The inverse hyperbolic sinus, co-sinus, tangent, co-tangent, secant and co-secant of x.

You can also use any of the predefined variables:

- pi: 3.141592653589793, what else is there to say?
- e: The base of the natural logarithms.
- false or f or no: False in all its flavors. For the curious, false is defined as 0.
- true or t or yes: The truthful companion of false. For the curious, true is defined as 1.

Blocks are defined as a collection of values, organized in row and column format. The syntax is the following:

```
%var
  exp | exp | exp | ...
  exp | exp | exp | ...
  ...
%
```

Rows in a block are separated by a newline, while columns are separated by the character "`|`" or by a tab. There may be any number of lines and any number of columns in a block. Note also that each line can have a different number of columns.

If `APE` tries to read a variable that is not defined in the `inp.ape` file, it automatically assigns to it a default value. All variables read are output to the file "`parser.log`". If you are not sure of what the program is reading, just take a look at it. Everything following the character "#" until the end of the line is considered a comment and is simply cast into oblivion.

6 Input file options

APE has quite a few options, that we will subdivide in different groups. After the name of the option, its type and default value (when applicable) are given in parenthesis. Do not be scared by the amount of options! For many calculations you can safely rely on the default values of most of the options.

6.1 Generalities

6.1.1 CalculationMode (flag, ae + pp)

It defines the type of calculation to perform. Options are:

- ae: Atomic calculation.
- pp: Pseudopotential generation.
- pp_test: Pseudopotential test.
- xc: One-shot evaluation of exchange and correlation energies and potentials and/or kinetic energy density functionals.
- ip: Calculation of ionization energy.
- nt: Numerical tests. Should only be useful to developers.

6.1.2 Units (integer, 1)

Internally, the code works in atomic units. However, you can use this option to define the units used in the input and output files.

Valid options are:

- 1: atomic units.
- 2: atomic Rydberg units.
- 3: electron-volts/ångström.

6.1.3 UnitsOutput (integer, 1)

Same as Units, but only refers to the values in the output files.

6.1.4 UnitsInput (integer, 1):

Same as Units, but only refers to the values in the input file.

6.1.5 Verbose (integer, 30)

Verbosity level of the program. The higher, the more verbose APE is. Current levels are:

- verbose <= 0: Silent mode. No output except fatal errors.
- verbose > 0: Warnings only.
- verbose > 10: Important program info only.
- verbose > 20: Normal program info.
- verbose > 30: Normal program info plus detailed info about the SCF cycle.

6.2 Hamiltonian

These parameters control what Hamiltonian to use when solving the Khon-Sham equations.

6.2.1 `WaveEquation` (integer, schrodinger)

When performing atomic calculations `APE` can solve either the Kohn-Sham equations, the Dirac-Kohn-Sham equations or the scalar-relativistic Khon-Sham equations. Valid options are:

- `schrodinger`: Kohn-Sham equations.
- `dirac`: Dirac-Kohn-Sham equations.
- `scalar_rel`: scalar-relativistic Kohn-Sham equations.

6.2.2 `SpinMode` (integer, unpolarized)

Defines the spin mode `APE` will run in. Valid modes are:

- `unpolarized`: Spin-unpolarized calculation.
- `polarized`: Spin-polarized calculation.

6.2.3 `XCFunctional` (integer, lda_x+lda_c_pw)

The possible values are (note that, depending on the version of `libxc` used, some of the following functionals might not be available):

- `none`: No exchange-correlation.
- `lda_x`: Slater exchange.
- `lda_c_wigner`: Wigner parametrization.
- `lda_c_rpa`: Random Phase Approximation.
- `lda_c_hl`: Hedin & Lundqvist.
- `lda_c_gl`: Gunnarson & Lundqvist.
- `lda_c_xalpha`: Slater's Xalpha.
- `lda_c_vwn`: Vosko, Wilk, & Nussair.
- `lda_c_vwn_rpa`: Vosko, Wilk, & Nussair (fit to the RPA correlation energy).
- `lda_c_pz`: Perdew & Zunger.
- `lda_c_pz_mod`: Perdew & Zunger (Modified to improve the matching between the high and the low rs region).
- `lda_c_ob_pz`: Ortiz & Ballone (PZ-type parametrization).
- `lda_c_pw`: Perdew & Wang.
- `lda_c_pw_mod`: Perdew & Wang (Modified to match the original PBE routine).
- `lda_c_ob_pw`: Ortiz & Ballone (PW-type parametrization).
- `lda_c_vbh`: von Barth & Hedin.
- `lda_c_ml1`: Modified LSD (version 1) of Proynov and Salahub.
- `lda_c_ml2`: Modified LSD (version 2) of Proynov and Salahub.
- `lda_xc_teter93`: Teter 93.
- `gga_x_pbe`: Perdew, Burke & Ernzerhof.

- gga_x_pbe_r: Perdew, Burke & Ernzerhof (revised).
- gga_x_p86: Becke 86 Xalpha,beta,gamma.
- gga_x_b86_r: Becke 86 Xalpha,beta,gamma reoptimized.
- gga_x_b86_mgc: Becke 86 Xalpha,beta,gamma (with mod. grad. correction).
- gga_x_b88: Becke 88.
- gga_x_g96: Gill 96.
- gga_x_pw86: Perdew & Wang 86.
- gga_x_pw91: Perdew & Wang 91.
- gga_x_optx: Handy & Cohen OPTX 01.
- gga_x_dk87_r1: dePristo & Kress 87 (version R1).
- gga_x_dk87_r2: dePristo & Kress 87 (version R2).
- gga_x_lg93: Lacks & Gordon 93.
- gga_x_ft97_a: Filatov & Thiel 97 (version A).
- gga_x_ft97_b: Filatov & Thiel 97 (version B).
- gga_x_pbe_sol: Perdew, Burke & Ernzerhof exchange (solids).
- gga_x_rpbe: Hammer, Hansen & Norskov (PBE-like).
- gga_x_wc: Wu & Cohen
- gga_x_mpw91: mPW91 of Adamo & Barone.
- gga_x_am05: Armiento & Mattsson 05 exchange.
- gga_x_pbea: Madsen 07.
- gga_x_mpbe: Adamo & Barone modification to PBE.
- gga_x_xpbe: Extended PBE by Xu & Goddard III.
- gga_x_bayesian: Bayesian best fit for the enhancement factor.
- gga_x_pbe_jsjr:
- gga_x_optb88_vdw: opt-Becke 88 for vdW.
- gga_x_pbek1_vdw: Reparametrized PBE for vdW.
- gga_x_optpbe_vdw: Reparametrized PBE for vdW.
- gga_x_rge2: Regularized PBE.
- gga_c_pbe: Perdew, Burke & Ernzerhof correlation.
- gga_c_lyp: Lee, Yang, & Parr LDA.
- gga_c_pbe_sol: Perdew, Burke & Ernzerhof correlation SOL.
- gga_c_p86: Perdew 86.
- gga_c_pw91: Perdew & Wang 91.
- gga_c_am05: Armiento & Mattsson 05 correlation.
- gga_c_xpbe: Extended PBE by Xu & Goddard III.
- gga_c_lm: Langreth & Mehl.
- gga_c_pbe_jrgx: Reparametrized PBE by Pedroza, Silva & Capelle.
- gga_c_rge2: Regularized PBE.
- gga_xc_lb: van Leeuwen & Baerends.

- `gga_xc_hcth_93`: HCTH functional fitted to 93 molecules.
- `gga_xc_hcth_120`: HCTH functional fitted to 120 molecules.
- `gga_xc_hcth_147`: HCTH functional fitted to 147 molecules.
- `gga_xc_hcth_407`: HCTH functional fitted to 407 molecules.
- `gga_xc_edf1`: Empirical functionals from Adamson, Gill, and Pople.
- `gga_xc_xlyp`: XLYP functional.
- `gga_xc_b97`: Becke 97.
- `gga_xc_b97_1`: Becke 97-1.
- `gga_xc_b97_2`: Becke 97-2.
- `gga_xc_b97_d`: Becke 97-D (Grimme functional to be used with C6 vdW term).
- `gga_xc_b97_k`: Becke 97-K (Boese-Martin for Kinetics).
- `gga_xc_b97_3`: Becke 97-3.
- `gga_xc_pbe1w`: PBE1W (functional fitted for water).
- `gga_xc_mpwlyp1w`: mPWLYP1w (functional fitted for water).
- `gga_xc_pbelyp1w`: PBELYP1W (functional fitted for water).
- `gga_xc_sb98_1a`: SB98 (1a).
- `gga_xc_sb98_1b`: SB98 (1b).
- `gga_xc_sb98_1c`: SB98 (1c).
- `gga_xc_sb98_2a`: SB98 (2a).
- `gga_xc_sb98_2b`: SB98 (2b).
- `gga_xc_sb98_2c`: SB98 (2c).
- `mgga_x_bj06`: Becke & Johnson 06.
- `mgga_x_tb09`: Tran & Blaha 09.
- `mgga_x_rpp09`: Rasanen, Pittalis & Proetto 09.

6.2.4 `XCCorrections` (integer, none)

The possible values are:

- `rel_x`: relativistic correction to the exchange functional (implementd only for LDA exchange).
- `rel_c`: relativistic correction to the correlation functional (not yet implemented).
- `adsic`: averaged density self-interaction correction.

6.2.5 `KEDFunctional` (integer, none)

Kinetic energy density functional. This is only used when running the "xc" calculation mode. For a complete list of available functionals, check the `libxc` documentation.

6.2.6 `TheoryLevel` (integer, dft)

The possible values are:

- `independent_particles`: particles are considered as independent, i.e. as non-interacting.
- `dft`: this is the default density-functional theory scheme.

6.3 Specie

6.3.1 NuclearCharge (real)

Nuclear charge of the specie.

6.3.2 Orbitals (block data)

This options sets the electronic configuration of the specie. To each line corresponds an orbital, which is defined by some quantum numbers and its occupancy.

The first and second columns are the main quantum number n and the angular momentum quantum number l. The meaning of the remaining columns depends if we are running a fully-relativistic spin-polarized calculation or not. In the later case, the remaining columns are used to specify the occupancies. One or two columns can be used to specify the occupancies and there are four possible cases:

- The calculation is spin-unpolarized and only the third column is set: the occupancy of the orbital is simply set by the third column.
- The calculation is spin-unpolarized and the third and fourth columns are set: the occupancy of the orbital is equal to the sum of both columns.
- The calculation is spin-polarized and only the third column is set: the occupancies of the two spin channels are equal to half the value of the third column.
- The calculation is spin-polarized and the third and fourth columns are set: the occupancies of the two spin channels are set by the two columns.

When running a fully-relativistic spin-polarized calculation (`WaveEquation = dirac` and `SpinMode = polarized`), besides the n and l quantum numbers, it is also necessary to specify the m quantum number, which will always be the third column of the block. This quantum number runs from *-j* to *j*, where *j* can have two values: $j = l$ *- 1/2* and $j = l$ *+ 1/2*. This means that, when $|m| < l$*+1/2* there will be two orbitals with the same m quantum number. To differentiate these two orbitals, a fourth quantum number is introduced, which can have two values: *up* or *down* (these are represented in APE by *0.5* and *-0.5*). The meaning of the columns beyond the third is thus the following:

- If $|m| = 2l$*+1*: the fourth column is just the occupancy.
- If $|m| < 2l$*+1*: the fourth column is the occupancy of the *up* orbital and the fifth column is the occupancy of the *down* orbital. If only the fourth column is set, both orbitals are equally occupied with half the specified value.

Note that the occupancies can be set to zero.

Here is an example for the Lithium ground state electronic configuration:

```
%Orbitals
  1 | 0 | 1 | 1
  2 | 0 | 1 | 0
  2 | 1 | 0 | 0
%
```

The first column sets the quantum number n, the second the quantum number l and the last two the occupancies for both spin channels. In case of a fully-relativistic spin-polarized calculation, the previous example would become:

```
%Orbitals
  1 | 0 | -1/2 | 1
  1 | 0 |  1/2 | 1
  1 | 1 | -1/2 | 1
  1 | 1 |  1/2 | 0
  2 | 1 | -3/2 | 0
  2 | 1 | -1/2 | 0 | 0
  2 | 1 |  1/2 | 0 | 0
  2 | 1 |  3/2 | 0
%
```

To make things easier, the core configuration can be replaced by a string with the chemical symbol of the rare gas that has the same configuration. For example, to run Sodium, it is possible to use the following input:

```
%Orbitals
  "Ne"
  3 | 0 | 1
%
```

6.4 Pseudopotentials generation

6.4.1 `PPComponents` (block data)

This block sets which pseudopotential components will be generated and the corresponding core radii. The first column is the main quantum number n, the second column is the angular momentum quantum number l, the one before the last is the core radius, and the last is the scheme to be used for constructing the pseudopotentials.

When generating relativistic pseudopotentials, the third column will be the total angular momentum quantum number j. Nevertheless, it is possible to omit j, in which case both *j=l-1/2* and *j=l+1/2* components will use the same core radius.

Here is an example:

```
%PPComponents
  5 | 0 | 1.0 | tm
  5 | 1 | 0.5 | 1.4 | tm
  5 | 1 | 1.5 | 1.6 | tm
  4 | 2 | 2.0 | tm
%
```

In this case, the *j=l-1/2* and *j=l+1/2* components of *l=1* will have different core radius, while for *l=2* they will have the same core radius.

Note that it is possible to set the core-radii to zero. In that case, `APE` will use some default value that depends on which scheme is used for constructing the pseudopotentials. Right now, this feature only works with the Hamann scheme. In that case the default core-radius is determined using the value of the outermost maximum of the all-electron wavefunction. If there are core states present with the same angular momentum the default core-radius is equal to 0.6 times the outermost maximum. Otherwise, the core radius is 0.4 times the outermost maximum.

The possible values for the scheme used for constructing the pseudopotentials are:

- `ham`: Hamann scheme.
- `tm`: Troullier-Martins scheme.
- `rtm`: Relativistic extension of the Troullier-Martins scheme.
- `mrpp`: MRPP scheme.

Note that when using the MRPP scheme, the quantum numbers should correspond to a semi-core orbital, i.e., there should be at least another orbital with the same l and j quantum numbers, but higher in energy.

6.4.2 `PPCalculationTolerance` (real, 1e-6)

This tolerance is used during the pseudopotential generation. What it does exactly depends on the pseudopotential generation scheme used.

6.4.3 `PPOutputFileFormat` (integer, upf)

`APE` can write the pseudopotentials in different formats compatible with other programs. Programs currently supported are:

- `siesta`: SIESTA format.
- `fhi`: fhi98PP pseudopotential generator format.
- `abinit5`: ABINIT format 5.
- `abinit6`: ABINIT format 6.
- `upf`: PWscf unified pseudopotential format.
- `parsec`: parsec format. This is basically the SIESTA format plus the pseudo-wavefunctions.
- `latepp_so` : LATEPP format. Spin-orbit difference potentials and respective wavefunctions for the spin-orbit treatment in the LATEPP code.

6.4.4 `CoreCorrection` (integer, 1)

Scheme for constructing the partial core density. Possible values are:

- `0`: no non-linear core corrections.
- `1`: scheme implemented in José Luis Martins atom code.
- `2`: scheme implemented in the fhi98PP code.

6.4.5 `LLocal` (integer, -1)

Angular momentum component of the pseudo-potentential to take as local component when building the Kleinman-Bylander projectros. If set to -1 a Vanderbilt function is used.

6.5 Pseudopotentials tests

6.5.1 `PPTests` (integer, ld+dm)

What tests should be performed. Possible values are:

- `ld`: logarithmic derivatives.
- `dm`: transition dipole moments.
- `ip_test`: ionization potentials.

6.5.2 PPTestSCF (logical, false)

If set to "yes", run a SCF calculation for the pseudostates before performing the tests.

6.5.3 PPTestOrbitals (block date)

Orbitals configuration to be used for the SCF calculation performed when PPTestSCF is set ot "yes". This allows to change the occupancies of the valence states or to add other states to be tested.

6.5.4 PPTestAEDir (string, "ae")

Directory where to find the all-electron calculations that should be used during the tests. The default is the "ae" directory.

6.5.5 LogDerivativeRadius (real, 0.0)

Diagnostic radius where the logarithmic derivatives are computed. If set to zero, APE will use the element covalent radius.

6.5.6 LogDerivativeEnergyMax (real)

Upper bound of the energy interval where the logarithmic derivatives are computed. If it is not set, then the value to be used will be equal to the largest eigenvalue plus 1 a.u.

6.5.7 LogDerivativeEnergyMin (real)

Lower bound of the energy interval where the logarithmic derivatives are computed. If it is not set, then the value to be used will be equal to the smallest eigenvalue minus 1 a.u.

6.5.8 LogDerivativeEnergyStep (real, 0.0 a.u.)

Step in energy interval. If set to zero, an adaptive step will be used.

6.6 Mesh

APE uses a mesh to store functions. In order to change the mesh parameters you can use the following options:

6.6.1 MeshType (inteager, log1)

Possible values are:

- lin: Linear
- log1: Logarithmic [ri = b*exp(a*i)]
- log2: Logarithmic [ri = b*(exp(a*i) - 1)]

6.6.2 MeshStartingPoint (real, sqrt(NuclearCharge)*1e-5 a.u.)

Sets the starting point of the mesh.

6.6.3 MeshOutmostPoint (real, sqrt(NuclearCharge)*30 a.u.)

Sets the outmost point of the mesh.

6.6.4 MeshNumberOfPoints (integer, sqrt(NuclearCharge)*200)

Sets the mesh number of points.

6.6.5 MeshDerivMethod (int, cubic_spline)

Numerical method used to compute derivatives. Possible choices are:

- `cubic_spline`: Use cubic splines to interpolate the function and then uses this interpolation to compute the derivatives.
- `finite_diff`: Finite differences. The number of points used is controled by the `MeshFiniteDiffOrder` variable.

6.6.6 MeshFiniteDiffOrder (int, 4)

This variable controls the numbers of points used for the finite differences discretization. To compute the derivative at a given point, the finite difference formula will use `MeshFiniteDiffOrder` points to the left and `MeshFiniteDiffOrder` points to the right. This means that in total `MeshFiniteDiffOrder`*2 + 1 points will be used.

6.7 SCF

The self consistent field procedure will stop when one of the convergence criteria is fulfilled. At each iteration the new guess potential is built mixing the input and output potentials.

6.7.1 MaximumIter (integer, 300)

Maximum number of SCF iterations. When that number is reached the SCF procedure will stop, even if none of the criteria are fulfilled, and the calculation will continue as normal. `0` means unlimited.

6.7.2 ConvAbsDens (real, 0.0)

Absolute convergence of the density. `0` means do not use this criterion.

6.7.3 ConvRelDens (real, 1e-8)

Relative convergence of the density. `0` means do not use this criterion.

6.7.4 ConvAbsEnergy (real, 0.0)

Absolute convergence of the total energy. `0` means do not use this criterion.

6.7.5 ConvRelEnergy (real, 0.0)

Relative convergence of the total energy. `0` means do not use this criterion.

6.7.6 SmearingFunction (integer, fixed_occ)

Select how the states should be occupied. The option are:

- `fixed_occ`: the occupancies of the states are fixed.
- `semiconducting`: semiconducting occupancies, i.e., the lowest lying states are occupied until no more electrons are left
- `averill_painter`: F.W. Averill and G.S. Painter, Phys. Rev. B 46, 2498 (1992)

6.7.7 MixingScheme (integer, broyden)

Selects the mixing procedure to be used during the SCF cycle. Possible values are:

- `linear`: Linear mixing.

– `broyden`: Broyden mixing.

6.7.8 `Mixing` (real, 0.3)

Determines the amount of the new potential which is to be mixed with the old one.

6.7.9 `MixingSteps` (integer, 3)

Number of steps used by Broyden mixing to extrapolate the new potential.

6.8 Wave-equations solver

`APE` solves the Schrodinger and Dirac equations using the ODE solver from GSL.

6.8.1 `EigenSolverTolerance` (real, 1.0e-8 a.u.)

The eigensolver will improve the eigenvalues untill the error estimate of each eigenvalue is smaller than this tolerance.

6.8.2 `ODEIntTolerance` (real, 1.0e-12)

6.8.3 `ODESteppingFunction` (integer, rkpd8)

Possible values are:

– `rk2`: Embedded 2nd order Runge-Kutta method
– `rk4`: 4th order (classical) Runge-Kutta method
– `rkf4`: Embedded 4th order Runge-Kutta-Fehlberg method
– `rkck4`: Embedded 4th order Runge-Kutta Cash-Karp method
– `rkpd8`: Embedded 8th order Runge-Kutta Prince-Dormand method

For more information about the stepping functions please refer to the GSL documentation.

6.8.4 `ODEMaxSteps` (integer, 500000)

Sometimes it may happen that the ODE solver takes too many steps, because of some problem. To avoid that, `APE` will issue a fatal error if the number of steps is greater than `ODEMaxSteps`.

7 Examples

7.1 Lithium

As a first example, we will take a Lithium atom. Using a text editor, create the following input "inp.ape":

```
Title = "Lithium"
CalculationMode = ae
Verbose = 30

NuclearCharge = 3

%Orbitals
 "He"
  2  |  0  |  1
  2  |  1  |  0
%
```

Then run APE. The output should look like this:

```
                  APE - Atomic Pseudopotentials Engine

               Program started on 2012/06/20 at 12:01:41

Compilation Info
  Version: 2.x
  Revision:
  Build time: Wed Jun 20 12:00:49 WEST 2012
  C compiler: gcc
  C compiler flags: -g -O2 -I/usr/include
  Fortran compiler: gfortran
  Fortran compiler flags: -Wall -fbounds-check

Calculation Type:
  Atomic Calculation

Setting units
  Input units system:  Atomic Units
  Output units system: Atomic Units

Eigensolver Info
  Method:    Brent's method
  Tolerance:  1.000E-08
```

```
                    -- All Electron Calculation --

Initializing ODE Integrator
  ODE Stepping function: Embedded 8th order Runge-Kutta Prince-Dormand
                         method with 9th order error estimate
  ODE Integrator tolerance:  1.000E-12
  ODE Integrator maximum number of steps: 500000

General Information about the atom:
  Symbol: Li
  Theory Level: DFT
  Wave-equation: Schrodinger
  Spin mode: unpolarized
  Nuclear charge:   3.00
  Total charge:     0.00
  Configuration    : State   Occupation
                        1s       2.00
                        2s       1.00
                        2p       0.00

Exchange-Correlation model:
  Correlation: Perdew & Wang (LDA)
  Exchange:    Slater exchange (LDA)

Mesh information:
  Type: logarithmic [ri = b*exp(a*i)]
  Mesh starting point:   1.73E-05 b
  Mesh outmost point:     51.962 b
  Mesh parameters (a, b):  7.49453E-02,  1.60699E-05
  Mesh number of points:    200
  Derivatives Method: Cubic slines

Starting SCF process
  Convergence tolerance: ConvAbsDens ConvRelDens
                          0.000E+00   1.000E-08
                         ConvAbsEnergy ConvRelEnergy
                          0.000E+00   0.000E+00
  Smearing: fixed occupancies
  Maximum number of iterations:    300
  Mixing scheme: Modified Broyden's Method
  Mixing: 0.300

Performing SCF Cycle

  Final results for SCF procedure:

              Program finished on 2012/06/20 at 12:01:42
```

Now take a look at the working directory. It should include the following files and directories:

```
drwxrwxr-x    2 user   group       4096 2012-06-20 12:01 ae
-rw-rw-r--    1 user   group        198 2012-06-20 12:01 inp.ape
-rw-rw-r--    1 user   group       1022 2012-06-20 12:01 parser.log
```

Besides the inp.ape and parser.log files there is now a new directory named `ae`. In that directory you will find the following files:

```
-rw-r--r-- 1 user group 24885 2012-06-20 12:01 data
-rw-r--r-- 1 user group 14665 2012-06-20 12:01 density
-rw-r--r-- 1 user group  1549 2012-06-20 12:01 info
-rw-r--r-- 1 user group  7396 2012-06-20 12:01 tau
-rw-r--r-- 1 user group  7567 2012-06-20 12:01 v_c
-rw-r--r-- 1 user group  7361 2012-06-20 12:01 v_ext
-rw-r--r-- 1 user group  7361 2012-06-20 12:01 v_hxc
-rw-r--r-- 1 user group  7567 2012-06-20 12:01 v_x
-rw-r--r-- 1 user group 10938 2012-06-20 12:01 wf-1s
-rw-r--r-- 1 user group 10938 2012-06-20 12:01 wf-2p
-rw-r--r-- 1 user group 10966 2012-06-20 12:01 wf-2s
```

The file `data` is a binary file containing the all-electron calculation data. This file is necessary if you want to generate pseudopotentials without having to solve again the all-electron equations. The file `info` contains some information about the all-electron calculation and it should be self-explanatory. The files `density` and `tau` contain the eletronic density and the kinetic energy density in a format suitable for plotting. The files whose name start with `v_` contain the different components of the Kohn-Sham potential. Finally, the `wf` files contain the radial wavefuntions.

Next we will generate some pseudopotentials using the Hamann scheme. Create the following "inp.ape" file:

```
Title = "Lithium"
CalculationMode = pp
Verbose = 30

%PPComponents
  2  |  0  | 0.00 | ham
  2  |  1  | 0.00 | ham
%
```

This time the output should look like this:

```
                    APE - Atomic Pseudopotentials Engine
```

```
              Program started on 2012/06/20 at 12:06:13

Compilation Info
  Version: 2.x
  Revision:
  Build time: Wed Jun 20 12:00:49 WEST 2012
  C compiler: gcc
  C compiler flags: -g -O2 -I/usr/include
  Fortran compiler: gfortran
  Fortran compiler flags: -Wall -fbounds-check

Calculation Type:
  PseudoPotential Generation

Setting units
  Input units system:  Atomic Units
  Output units system: Atomic Units

Eigensolver Info
  Method:    Brent's method
  Tolerance:  1.000E-08

Initializing ODE Integrator
  ODE Stepping function: Embedded 8th order Runge-Kutta Prince-Dormand
                         method with 9th order error estimate
  ODE Integrator tolerance:  1.000E-12
  ODE Integrator maximum number of steps: 500000


                  -- Pseudopotential Generation --

Pseudo atom information:
  Wavefunction info:
    State   Occupation    Node radius   Peak radius   Default core radius
     2s        1.00          0.842         3.012             1.807
     2p        0.00          0.000         3.771             1.509

Pseudopotential Generation:
  State: 2s
    Scheme: Hamann
    Core radius:   1.807     Matching Radius:   5.486
    cl  =     0.1270413205
  State: 2p
    Scheme: Hamann
    Core radius:   1.509     Matching Radius:   4.381
    cl  =    -0.9176124519
```

```
Pseudopotentials Self-Consistency:
  State  Eigenvalue [H ]    Norm Test   Slope Test
    2s        -0.10560      1.0000049   0.9999925
    2p        -0.04140      0.9998835   0.9998173

Kleinman & Bylander Atom
  Local potential is a Vanderbilt function
       z     rcmax      v0          v1          v2          v3
     1.00    5.09   -0.491383   -0.057906    0.001118   -0.000010
  Non-local components:
    State   KB Energy [H ]   KB Cosine
      2s        1.0840         0.0761
      2p       -0.4609        -0.2732

  Ghost state analysis:
    State: 2s
      KB energy > 0; E0 < Eref < E1  =>  No ghost states
      Local potential eigenvalues:   -0.1215 (E0)    -0.0050 (E1)
      Reference energy:              -0.1056 (Eref)
    State: 2p
      KB energy < 0; Eref < E0       =>  No ghost states
      Local potential eigenvalues:   -0.0248 (E0)     0.0000 (E1)
      Reference energy:              -0.0414 (Eref)

  Localization radii [b]:
    Local:  3.77
    l = 0:  4.06
    l = 1:  3.77

              Program finished on 2012/06/20 at 12:06:14
```

At the end of the run there should be a new file named `Li.UPF` and two new directories named `pp` and `kb` in the working directory. The file `Li.UPF` contains the pseudopotential information in the `PWscf` unified pseudopotential format. The `pp` directory should include the following files:

```
-rw-r--r-- 1 user group 26463 2012-06-20 12:06 data
-rw-r--r-- 1 user group 14665 2012-06-20 12:06 density
-rw-r--r-- 1 user group   696 2012-06-20 12:06 info
-rw-r--r-- 1 user group  7361 2012-06-20 12:06 pp-p
-rw-r--r-- 1 user group  7361 2012-06-20 12:06 pp-s
-rw-r--r-- 1 user group  7396 2012-06-20 12:06 tau
-rw-r--r-- 1 user group  7567 2012-06-20 12:06 v_c
-rw-r--r-- 1 user group  7361 2012-06-20 12:06 v_hxc
-rw-r--r-- 1 user group  7567 2012-06-20 12:06 v_x
```

```
-rw-r--r-- 1 user group 10938 2012-06-20 12:06 wf-2p
-rw-r--r-- 1 user group 10938 2012-06-20 12:06 wf-2s
```

The file `data` is a binary file containing the pseudopotential calculation data. This file is necessary if you want to perform further tests on the pseudopotential transferability. The file `info` contains some information about the pseudopotential generation and it should be self-explanatory. The files `density` and `tau` contain the valence eletronic density and kinetic energy density in a format suitable for plotting. The `wf` files contain the radial pseudo-wavefuntions and the `pp` files contain the ionic pseudopotentials. Finally, the files whose name start with `v_` contain the different components of the Kohn-Sham potential evaluated for the pseudo valence states.

As for the `kb` directory should include the following files:

```
-rw-r--r-- 1 user group  819 2012-06-20 12:06 info
-rw-r--r-- 1 user group 7361 2012-06-20 12:06 kb-local
-rw-r--r-- 1 user group 7364 2012-06-20 12:06 kb-p
-rw-r--r-- 1 user group 7364 2012-06-20 12:06 kb-s
```

The `info` contains some information about the Kleinman-Bylander projectors and it should be self-explanatory. The file `kb-local` contains the local component used to generated the Kleinman-Bylander projectors, while the other `kb` files contain the projectors.

After generating the pseudopotentials, one should test them. One simple test is to compare the logarithmic derivative of the wavefunctions as a function of the orbital energy at a given diagnostic radius. The following input file will to precisely that:

```
Title = "Lithium"
CalculationMode = pp_test
Verbose = 30

PPTests = ld
```

In this case the output should look like this:

```
                    APE - Atomic Pseudopotentials Engine

                Program started on 2012/06/20 at 13:55:25

  Compilation Info
    Version: 2.x
    Revision:
    Build time: Wed Jun 20 12:00:49 WEST 2012
    C compiler: gcc
    C compiler flags: -g -O2 -I/usr/include
    Fortran compiler: gfortran
    Fortran compiler flags: -Wall -fbounds-check
```

```
Calculation Type:
  PseudoPotential Test

Setting units
  Input units system:  Atomic Units
  Output units system: Atomic Units

Eigensolver Info
  Method:    Brent's method
  Tolerance:  1.000E-08

Initializing ODE Integrator
  ODE Stepping function: Embedded 8th order Runge-Kutta Prince-Dormand
                         method with 9th order error estimate
  ODE Integrator tolerance:  1.000E-12
  ODE Integrator maximum number of steps: 500000


                    -- Pseudopotential Testing --

Initializing ODE Integrator
  ODE Stepping function: Embedded 8th order Runge-Kutta Prince-Dormand
                         method with 9th order error estimate
  ODE Integrator tolerance:  1.000E-12
  ODE Integrator maximum number of steps: 500000

Logarithmic Derivatives:
  Diagnostic radius:  2.320 b
  Energy step:    Adaptive
  Computing logarithmic derivative for states: s
    Minimum energy: -1.106 H
    Maximum energy:  0.894 H
  Computing logarithmic derivative for states: p
    Minimum energy: -1.041 H
    Maximum energy:  0.959 H

              Program finished on 2012/06/20 at 13:55:25
```

Notice that the choice of energy range and diagnostic radius was done automatically (check the description of the corresponding variable for more information about how this is done). At the end of the run there should be a new directory named `tests`. This directory should contain the following files:

```
-rw-r--r-- 1 user group  297 2012-06-20 13:55 info
-rw-r--r-- 1 user group 3916 2012-06-20 13:55 ld-p
```

```
-rw-r--r-- 1 user group 3700 2012-06-20 13:55 ld-s
```

The file `info` contains some information about the tests and it should be self-explanatory. The files `ld-` contain the all-electron and pseudo wavefunction logarithmic derivatives as a function of the orbital energy.

Options Index

Table of Contents