PWscf User's Guide (v.7.4)

Contents

1 Introduction

This guide covers the usage of the `PWscf` (Plane-Wave Self-Consistent Field) package, a core component of the QUANTUM ESPRESSO distribution. Further documentation, beyond what is provided in this guide, can be found in the directory `PW/Doc/`, containing a copy of this guide (*note: all directories are relative to the root directory of* QUANTUM ESPRESSO).

This guide assumes that you know the physics that `PWscf` describes and the methods it implements. It also assumes that you have already installed, or know how to install, QUANTUM ESPRESSO. If not, please read the general User's Guide in directory `Doc/`; or consult the web site: `http://www.quantum-espresso.org`.

People who want to modify or contribute to `PWscf` should read the Wiki pages on GitLab: `https://gitlab.com/QEF/q-e/-/wikis`.

1.1 What can `PWscf` do

`PWscf` performs many different kinds of self-consistent calculations of electronic-structure properties within Density-Functional Theory (DFT), using a Plane-Wave (PW) basis set and pseudopotentials (PP). In particular:

- ground-state energy and one-electron (Kohn-Sham) orbitals, atomic forces, stresses;
- structural optimization, also with variable cell;
- molecular dynamics on the Born-Oppenheimer surface, also with variable cell;
- macroscopic polarization (and orbital magnetization) via Berry Phases;
- various forms of finite electric fields, with a sawtooth potential or with the modern theory of polarization;
- Effective Screening Medium (ESM) method;
- self-consistent continuum solvation (SCCS) model, if patched with ENVIRON (`http://www.quant`

`PWscf` works for both insulators and metals, in any crystal structure, for many exchange-correlation (XC) functionals (including spin polarization, DFT+U, meta-GGA, nonlocal and hybrid functionals), for norm-conserving (Hamann-Schluter-Chiang) PPs (NCPPs) in separable form or Ultrasoft (Vanderbilt) PPs (USPPs) or Projector Augmented Waves (PAW) method. Noncollinear magnetism and spin-orbit interactions are also implemented.

Please note that NEB calculations are no longer performed by `pw.x`, but are instead carried out by `neb.x` (see main user guide), a dedicated code for path optimization which can use `PWscf` as computational engine.

1.2 People

The `PWscf` package (which in earlier releases included `PHonon` and `PostProc`) was originally developed by Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso (SISSA), Paolo Giannozzi (Univ. Udine), and many others. We quote in particular:

- David Vanderbilt's group at Rutgers for Berry's phase calculations;

- Paolo Umari (Univ. Padua) for finite electric fields;
- Ralph Gebauer (ICTP, Trieste) and Adriano Mosca Conte (SISSA, Trieste) for non-collinear magnetism;
- Andrea Dal Corso for spin-orbit interactions;
- Carlo Sbraccia (Princeton) for improvements to structural optimization and to many other parts;
- Dario Alfè (University College London) for implementation of Born-Oppenheimer molecular dynamics;
- Renata Wentzcovitch and collaborators (Univ. Minnesota) for variable-cell molecular dynamics;
- Lorenzo Paulatto (Univ.Paris VI) for PAW implementation, built upon previous work by Guido Fratesi (Univ.Milano Bicocca) and Riccardo Mazzarello (ETHZ-USI Lugano);
- Matteo Cococcioni (Univ. Minnesota) for DFT+U implementation;
- Timo Thonhauser (WFU) for vdW-DF, svdW-DF, and variants;

and the more recent contributors:

- Sophie Beck (CCQ Flatiron Institute) for interface with DMFT;
- Minkyu Park (KAIST) for magnetic symmetry improvement;
- Elena de Paoli (IOM-CNR) for porting to GPU of the RMM-DIIS algorithm by Satomichi Nisihara;
- Gabriel S. Gusmão (Georgia Tech) and Johannes Voss (ANL) for BEEF support;
- Miha Gunde for interface with Grimme's DFT-D3 code, as repackaged by Bàlint Aradi
- Pietro Bonfà (CINECA) for memory estimator;
- Michele Ceriotti and Riccardo Petraglia (EPFL Lausanne) for interfacing with i-PI;
- Taylor Barnes (LBL) and Nicola Varini (CSCS) for improvements to hybrid functionals;
- Robert DiStasio (Cornell), Biswajit Santra (Princeton), Hsin-Yu Ko (Princeton), Thomas Markovich (Harvard) for Tkatchenko-Scheffler stress in PWscf;
- Jong-Won Song (RIKEN) for Gau-PBE functional;
- Alberto Otero de la Roza (Merced Univ.) for XDM (exchange-hole dipole moment) model of dispersions, PW86 (unrevised) and B86B functionals;
- Hannu-Pekka Komsa (CSEA/Lausanne) for the HSE functional;
- Gabriele Sclauzero (IRRMA Lausanne) for DFT+U with on-site occupations obtained from pseudopotential projectors;

- Alexander Smogunov (CEA) for DFT+U with noncollinear magnetization and for calculation of Magnetic Anisotropy Energy using the Force Theorem;
- Burak Himmetoglou (UCSB) for DFT+U+J;
- Xiaochuan Ge (SISSA) for Smart MonteCarlo Langevin dynamics;
- Andrei Malashevich (Univ. Berkeley) for calculation of orbital magnetization;
- Minoru Otani (AIST), Yoshio Miura (Tohoku U.), Nicephore Bonet (MIT), Nicola Marzari (Univ. Oxford), Brandon Wood (LLNL), Tadashi Ogitsu (LLNL), for ESM, Effective Screening Method (PRB 73, 115407 [2006]);
- Dario Alfè, Mike Towler (University College London), Norbert Nemec (U.Cambridge) for the interface with `CASINO`.

This guide was mostly written by Paolo Giannozzi. Mike Towler wrote the `PWscf` to `CASINO` subsection. Michele Ceriotti and Riccardo Petraglia wrote the subsection on i-PI interface.

1.3 Terms of use

Quantum ESPRESSO is free software, released under the GNU General Public License. See `http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt`, or the file License in the distribution).

We shall greatly appreciate if scientific work done using the Quantum ESPRESSO distribution will contain an acknowledgment to the following references:

> P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, R. M. Wentzcovitch, J.Phys.: Condens.Matter 21, 395502 (2009)

and

> P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M .Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, S. Baroni, J.Phys.: Condens.Matter 29, 465901 (2017)

Users of the GPU-enabled version should also cite the following paper:

> P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, J. Chem. Phys. 152, 154105 (2020)

Note the form Quantum ESPRESSO for textual citations of the code. Please also see package-specific documentation for further recommended citations. Pseudopotentials should be cited as (for instance)

> [ ] We used the pseudopotentials C.pbe-rrjkus.UPF and O.pbe-vbc.UPF from `http://www.quantum-espresso.org`.

References for all exchange-correlation functionals can be found inside file `Modules/funct.f90`.

2 Compilation

`PWscf` is included in the core Quantum ESPRESSO distribution. Instruction on how to install it can be found in the general documentation (User's Guide) for Quantum ESPRESSO.

Typing `make pw` from the main Quantum ESPRESSO directory or `make` from the `PW/` subdirectory produces the `pw.x` executable in `PW/src` and a link to the `bin/` directory. In addition, the following utility programs, and related links in `bin/`, are produced in `PW/src`:

- `dist.x` symbolic link to `pw.x`: reads input data for `PWscf`, calculates distances and angles between atoms in a cell, taking into account periodicity,

and in `PW/tools`:

- `ev.x` fits energy-vs-volume data to an equation of state
- `kpoints.x` produces lists of k-points
- `ibrav2cell.x` and `cell2ibrav.x` convert from variables used in Quantum ESPRESSO to specify the unit cell to primitive lattice translations, and vice versa
- `scan_ibrav.x` works as `cell2ibrav.x` but tries to figure out whether the axis are rotated with respect to those assumed by Quantum ESPRESSO
- `pwi2xsf.sh`, `pwo2xsf.sh` process respectively input and output files (not data files!) for `pw.x` and `neb.x` (the latter, courtesy of Pietro Bonfà) and produce an XSF-formatted file suitable for plotting with XCrySDen: `http://www.xcrysden.org/`, a powerful crystalline and molecular structure visualization program. BEWARE: the `pwi2xsf.sh` shell script requires the `pwi2xsf.x` executables to be located somewhere in your PATH.
- `cif2qe.sh`: script converting from CIF (Crystallographic Information File) to a format suitable for Quantum ESPRESSO. Courtesy of Carlo Nervi (Univ. Torino, Italy).

The other auxiliary codes contain their own documentation in the source files.

3 Using PWscf

Input files for pw.x may be either written by hand or produced via the PWgui graphical interface by Anton Kokalj, included in the Quantum ESPRESSO distribution. See PWgui-x.y.z/INSTALL (where x.y.z is the version number) for more info on PWgui, or GUI/README if you are using sources from the repository.

You may take the tests (in test-suite/) and examples (in PW/examples/) distributed with Quantum ESPRESSO as templates for writing your own input files. You may find input files (typically with names ending with .in) either in test-suite/pw_*/ or in the various PW/examples/*/results/ subdirectories, after you have run the examples. All examples contain a README file.

3.1 Input data

Input data is organized as several namelists, followed by other fields ("cards") introduced by keywords. The namelists are

&CONTROL:	general variables controlling the run
&SYSTEM:	structural information on the system under investigation
&ELECTRONS:	electronic variables: self-consistency, smearing
&IONS (optional):	ionic variables: relaxation, dynamics
&CELL (optional):	variable-cell optimization or dynamics

Optional namelist may be omitted if the calculation to be performed does not require them. This depends on the value of variable calculation in namelist &CONTROL. Most variables in namelists have default values. Only the following variables in &SYSTEM must always be specified:

nat	(integer)	number of atoms in the unit cell
ntyp	(integer)	number of types of atoms in the unit cell
ecutwfc	(real)	kinetic energy cutoff (Ry) for wavefunctions.

plus the variables needed to describe the crystal structure, e.g.:

ibrav	(integer)	Bravais-
celldm	(real, dimension 6)	crystallo

Alternative ways to input structural data are described in files PW/Doc/INPUT_PW.*. For metallic systems, you have to specify how metallicity is treated in variable occupations. If you choose occupations='smearing', you have to specify the smearing type smearing and the smearing width degauss. Spin-polarized systems are as a rule treated as metallic system, unless the total magnetization, tot_magnetization is set to a fixed value, or if occupation numbers are fixed (occupations='from input' and card OCCUPATIONS).

Detailed explanations of the meaning of all variables are found in files PW/Doc/INPUT_PW.*. Almost all variables have default values, which may or may not fit your needs.

Comment lines in namelists can be introduced by a "!", exactly as in fortran code.

After the namelists, you have several fields ("cards") introduced by keywords with self-explanatory names:

ATOMIC_SPECIES
ATOMIC_POSITIONS
K_POINTS
CELL_PARAMETERS (optional)
OCCUPATIONS (optional)

The keywords may be followed on the same line by an option. Unknown fields are ignored. See the files mentioned above for details on the available "cards".

Comments lines in "cards" can be introduced by either a "!" or a "#" character in the first position of a line.

Note about k-points: The k-point grid can be either automatically generated or manually provided as a list of k-points and a weight in the Irreducible Brillouin Zone only of the Bravais lattice of the crystal. The code will generate (unless instructed not to do so: see variable `nosym`) all required k-points and weights if the symmetry of the system is lower than the symmetry of the Bravais lattice. The automatic generation of k-points follows the convention of Monkhorst and Pack.

3.2 Data files

The output data files are written in the directory `outdir/prefix.save`, as specified in variable `prefix` (a string that is prepended to all file names, whose default value is `prefix='pwscf'`). `outdir` is specified via environment variable `ESPRESSO_TMPDIR`. The usage of variable `outdir` is still possible but *deprecated*. The `FoX` library is used to write a "head" data file in a XML format. This file has a "schema" that can be found on `https://github.com/QEF/qeschemas`.

In case of multi-step calculations such as: `'md', 'relax', 'vc-md ', 'vc-relax'` the XML files contains also elements reporting the intermediate configurations. By default includes a maximum of 250 intermediate elements uniformly distributed along the trajectory and including first and last step. If one want to change the maximum number of intermediate steps described in the XML file it is sufficient to set the `MAX_XML_STEPS` variable to the desired value.

For more information about the XML file contents see the Developer Manual. The data directory contains binary files that are not guaranteed to be readable on different machines. If you need file portability, compile the code with HDF5 (see the general User Guide).

The execution stops if you create an "EXIT" file `prefix.EXIT` either in the working directory (i.e. where the program is executed), or in the `outdir` directory. Note that with some versions of MPI, the working directory is the directory where the executable is! The advantage of this procedure is that all files are properly closed, whereas just killing the process may leave data and output files in an unusable state. If you start the execution with the EXIT file already in place, the code will stop after initialization. Alternatively: set `nstep` to 0 in input. This is useful to have a quick check of the correctness of the input.

3.3 Electronic structure calculations

Single-point (fixed-ion) SCF calculation Set `calculation='scf'` (this is actually the default). Namelists &IONS and &CELL will be ignored. For LSDA spin-polarized calculations (that is: with a fixed quantization axis for magnetization), set `nspin=2`. Note that the number of k-points will be internally doubled (one set of k-points for spin-up, one set for spin-down). See example 1 (that is: `PW/examples/example01`).

Band structure calculation First perform a SCF calculation as above; then do a non-SCF calculation (at fixed potential, computed in the previous step) with the desired k-point grid and number `nbnd` of bands. Use `calculation='bands'` if you are interested in calculating only the Kohn-Sham states for the given set of k-points (e.g. along symmetry lines: see for instance `http://www.cryst.ehu.es/cryst/get_kvec.html`). Specify instead `calculation='nscf'` if you are interested in further processing of the results of non-SCF calculations (for instance, in DOS calculations). In the latter case, you should specify a uniform grid of points. For DOS calculations you should choose `occupations='tetrahedra'`, together with an automatically generated uniform k-point grid (card K_POINTS with option "automatic"). Specify `nosym=.true.` to avoid generation of additional k-points in low-symmetry cases. Variables `prefix` and `outdir`, which determine the names of input or output files, should be the same in the two runs. See examples 1, 6, 7.

NOTA BENE: in non-scf calculations, the atomic positions are read by default from the data file of the scf step, not from input.

Noncollinear magnetization, spin-orbit interactions The following input variables are relevant for noncollinear and spin-orbit calculations:

`noncolin`
`lspinorb`
`starting_magnetization` (one for each type of atoms)

To make a spin-orbit calculation both `noncolin` and `lspinorb` must be true. Furthermore you must use fully relativistic pseudopotentials at least for one atom. If all pseudopotentials are scalar-relativistic, the calculation is noncollinear but there is no spin-orbit coupling.

If `starting_magnetization` is set to zero (or not given) the code makes a spin-orbit calculation without spin magnetization (it assumes that time reversal symmetry holds and it does not calculate the magnetization). The states are still two-component spinors but the total magnetization is zero.

If `starting_magnetization` is different from zero, the code makes a noncollinear spin polarized calculation with spin-orbit interaction. The final spin magnetization might be zero or different from zero depending on the system. Note that the code will look only for symmetries that leave the starting magnetization unchanged.

See example 6 for noncollinear magnetism, example 7 (and references quoted therein) for spin-orbit interactions.

DFT+U DFT+U (formerly known as LDA+U) calculation can be performed within a simplified rotationally invariant form of the U Hubbard correction. Note that for all atoms having a U value there should be an item in function `Modules/set_hubbard_l.f90` and one in subroutine `PW/src/tabd.f90`, defining respectively the angular momentum and the occupancy of the orbitals with the Hubbard correction. If your Hubbard-corrected atoms are not there, you need to edit these files and to recompile.

See example 8 and its README.

Dispersion Interactions (DFT-D) For DFT-D (DFT + semiempirical dispersion interactions), see the description of input variable `vdw_corr` and related input variables; sample input files can be found in `test-suite/pw_vdw/vdw-d*.in`. For DFT-D2, see also the comments in source file `Modules/mm_dispersion.f90`. For DFT-D3, see the README in the `dft-d3/` directory.

Hartree-Fock and Hybrid functionals Hybrid functionals do not require anything special to be done, but note that: 1) they are much slower than plain GGA calculations, 2) non-scf and band calculations are not presently implemented, and 3) there are no pseudopotentials generated for hybrid functionals. See `EXX_example/` and its README file, and tests `pw_b3lyp`, `pw_pbe`, `pw_hse`.

Dispersion interaction with non-local functional (vdW-DF) See example `vdwDF_example`, references quoted in file `README` therein, tests `pw_vdW`.

Polarization via Berry Phase See example 4, its file README, the documentation in the header of `PW/src/bp_c_phase.f90`.

Finite electric fields There are two different implementations of macroscopic electric fields in `pw.x`: via an external sawtooth potential (input variable `tefield=.true.`) and via the modern theory of polarizability (`lelfield=.true.`). The former is useful for surfaces, especially in conjunction with dipolar corrections (`dipfield=.true.`): see the web page – courtesy Christoph Wolf – `https://christoph-wolf.at/tag/dipfield`, and `PP/examples/dipole_example` for an example of application. Electric fields via modern theory of polarization are documented in example 10. The exact meaning of the related variables, for both cases, is explained in the general input documentation.

Orbital magnetization Modern theory of orbital magnetization [Phys. Rev. Lett. 95, 137205 (2005)] for insulators. The calculation is performed by setting input variable `lorbm=.true.` in nscf run. If finite electric field is present (`lelfield=.true.`) only Kubo terms are computed [see New J. Phys. 12, 053032 (2010) for details].

3.4 Optimization and dynamics

Structural optimization For fixed-cell optimization, specify `calculation='relax'` and add namelist &IONS. All options for a single SCF calculation apply, plus a few others. You may follow a structural optimization with a non-SCF band-structure calculation. See example 2.

Molecular Dynamics Specify `calculation='md'`, the time step `dt`, and possibly the number of MD stops `nstep`. Use variable `ion_dynamics` in namelist &IONS for a fine-grained control of the kind of dynamics. Other options for setting the initial temperature and for thermalization using velocity rescaling are available. Remember: this is MD on the electronic ground state, not Car-Parrinello MD. See example 3.

Variable-cell optimization Variable-cell calculations (both optimization and dynamics) are performed with plane waves and G-vectors *calculated for the starting cell.* Only the last step, after convergence has been achieved, is performed for the converged structure, with plane waves and G-vectors *calculated for the final cell.* Small differences between the two last steps are thus to be expected and give an estimate of the convergence of the variable-cell optimization with respect to the plane-wave basis. A large difference means that you are far from convergence in the plane-wave basis set and you need to increase the cutoff(s) `ecutwfc` and/or (if applicable) `ecutrho`.

Variable-cell molecular dynamics "A common mistake many new users make is to set the time step `dt` improperly to the same order of magnitude as for CP algorithm, or not setting `dt` at all. This will produce a "not evolving dynamics". Good values for the original RMW (Wentzcovitch) dynamics are `dt` $= 50 \div 70$. The choice of the cell mass is a delicate matter. An off-optimal mass will make convergence slower. Too small masses, as well as too long time steps, can make the algorithm unstable. A good cell mass will make the oscillation times for internal degrees of freedom comparable to cell degrees of freedom in non-damped Variable-Cell MD. Test calculations are advisable before extensive calculation. I have tested the damping algorithm that I have developed and it has worked well so far. It allows for a much longer time step (dt=$100 \div 150$) than the RMW one and is much more stable with very small cell masses, which is useful when the cell shape, not the internal degrees of freedom, is far out of equilibrium. It also converges in a smaller number of steps than RMW." (Info from Cesar Da Silva: the new damping algorithm is the default since v. 3.1).

3.5 Direct interface with `CASINO`

`PWscf` supports the Quantum Monte Carlo program CASINO directly. For more information on the `CASINO` code, see `https://vallico.net/casinoqmc`. `CASINO` may take the output of `PWSCF` and 'improve it' giving considerably more accurate total energies and other quantities than DFT is capable of.

`PWscf` users wishing to learn how to use CASINO may like to attend one of the annual `CASINO` summer schools in Mike Towler's "Apuan Alps Centre for Physics" in Tuscany, Italy. More information can be found at `http://www.vallico.net/tti/tti.html`

Practicalities The interface between `PWscf` and `CASINO` is provided through a file with a standard format containing geometry, basis set, and orbital coefficients, which `PWscf` will produce on demand. For SCF calculations, the name of this file may be `pwfn.data`, `bwfn.data` or `bwfn.data.b1` depending on user requests (see below). If the files are produced from an MD run, the files have a suffix `.0001`, `.0002`, `.0003` etc. corresponding to the sequence of timesteps.

`CASINO` support is implemented by three routines in the `PW` directory of the espresso distribution:

- `pw2casino.f90` : the main routine
- `pw2casino_write.f90` : writes the `CASINO` `xwfn.data` file in various formats
- `pw2blip.f90` : does the plane-wave to blip conversion, if requested

Relevant behavior of `PWscf` may be modified through an optional auxiliary input file, named `pw2casino.dat` (see below).

How to generate `xwfn.data` files with `PWscf` Use the '-pw2casino' option when invoking `pw.x`, e.g.:

```
pw.x -pw2casino < input_file > output_file
```

The `xfwn.data` file will then be generated automatically.

PWscf is capable of doing the plane wave to blip conversion directly (the 'blip' utility provided in the CASINO distribution is not required) and so by default, PWscf produces the 'binary blip wave function' file bwfn.data.b1

Various options may be modified by providing a file pw2casino.dat in outdir with the following format:

```
&inputpp
blip_convert=.true.
blip_binary=.true.
blip_single_prec=.false.
blip_multiplicity=1.d0
n_points_for_test=0
/
```

Some or all of the 5 keywords may be provided, in any order. The default values are as given above (and these are used if the pw2casino.dat file is not present.

The meanings of the keywords are as follows:

blip_convert : reexpand the converged plane-wave orbitals in localized blip functions prior to writing the CASINO wave function file. This is almost always done, since wave functions expanded in blips are considerably more efficient in quantum Monte Carlo calculations. If `blip_convert=.false.` a pwfn.data file is produced (orbitals expanded in plane waves); if `blip_convert=.true.`, either a `bwfn.data file` or a `bwfn.data.b1` file is produced, depending on the value of `blip_binary` (see below).

blip_binary : if true, and if `blip_convert` is also true, write the blip wave function as an unformatted binary `bwfn.data.b1` file. This is much smaller than the formatted `bwfn.data` file, but is not generally portable across all machines.

blip_single_prec : if `.false.` the orbital coefficients in `bwfn.data(.b1)` are written out in double precision; if the user runs into hardware limits `blip_single_prec` can be set to `.true.` in which case the coefficients are written in single precision, reducing the memory and disk requirements at the cost of a small amount of accuracy..

blip_multiplicity : the quality of the blip expansion (i.e., the fineness of the blip grid) can be improved by increasing the grid multiplicity parameter given by this keyword. Increasing the grid multiplicity results in a greater number of blip coefficients and therefore larger memory requirements and file size, but the CPU time should be unchanged. For very accurate work, one may want to experiment with grid multiplicity larger that 1.0. Note, however, that it might be more efficient to keep the grid multiplicity to 1.0 and increase the plane wave cutoff instead.

n_points_for_test : if this is set to a positive integer greater than zero, PWscf will sample the wave function, the Laplacian and the gradient at a large number of random points in the simulation cell and compute the overlap of the blip orbitals with the original plane-wave orbitals:

$$\alpha = \frac{< BW|PW >}{\sqrt{< BW|BW >< PW|PW >}}$$

The closer α is to 1, the better the blip representation. By increasing `blip_multiplicity`, or by increasing the plane-wave cutoff, one ought to be able to make α as close to 1 as desired. The number of random points used is given by `n_points_for_test`.

Finally, note that DFT trial wave functions produced by `PWSCF` must be generated using the same pseudopotential as in the subsequent QMC calculation. This requires the use of tools to switch between the different file formats used by the two codes.

`CASINO` uses the '`CASINO` tabulated format', `PWSCF` uses the UPF format. See `upflib/README.md` for instructions on how to convert between these formats.

An alternative converter 'casinogon' is included in the `CASINO` distribution which produces the deprecated GON format but which can be useful when using non-standard grids.

3.6 Socket interface with i-PI

The i-PI universal force engine performs advanced Molecular Dynamics (MD) (such as Path Integral Molecular Dynamics, Thermodynamic Integration, Suzuki-Chin path integral, Multiple Time Step molecular dynamics) and other force related computations (see `ipi-code.org` for more information about i-PI).

`PWscf` users wishing to learn how to use i-PI should refer to the i-PI website.

Practicalities The communication between `PWscf` and i-PI relies on a socket interface. This allows running i-PI and `PWscf` on different computers provided that the two computers have an Internet connection. Basically, i-PI works as a server waiting for a connection of a suitable software (for example `PWscf`). When this happens, i-PI injects atomic positions and cell parameters into the software, that will return forces and stress tensor to i-PI.

The file containing the interface is `run_driver.f90`. The files `socket.c` and `fsocket.f90` provide the necessary infrastructure to the socket interface.

How to use the i-PI inteface Since the communication goes through the Internet, the `PWscf` instance needs to know the address of the i-PI server that can be specified with the command line option `--ipi` (or `-ipi`) followed by the address of the computer running i-PI and the port number where i-PI is listening, e.g.

```
pw.x --ipi localhost:3142 -in pw.input > pw.out
```

If i-PI and `PWscf` are running on the same machine, a UNIX socket is preferable since allows faster communications, e.g.

```
pw.x --ipi socketname:UNIX -in pw.input > pw.out
```

In the last case, `UNIX` is a keyword that tells to `PWscf` to look for an UNIX socket connection instead of an INET one. More extensive examples and tutorials can be found at `ipi-code.org`. The `PWscf` input file must contain all the information to perform a single point calculation (`calculation = "scf"`) which are also used to initialize the `PWscf` run. Thus, it is important that the `PWscf` input contains atomic positions and cell parameters which are as close as possible to those specified in the i-PI input.

Advanced i-PI usage For more advanced users, calculation flags can be changed on-the-fly by parsing a single binary-encoded integer to QE through the i-PI socket. That gives users the flexibility to define what properties to be calculated. For example, if only a single SCF cycle is needed, traditionally `run_driver.f90` would be set to calculate not only the potential energy, but also forces, stresses and initialize g-vectors. With the binary-integer encoded flags, now one can turn flags on and off as necessary to speed up their code flow.

The sequence of flags that is currently accepted is: SCF, forces, stresses, variable-cell and ensembles. The latter is only available if QE has been compiled against BEEF-vdW XC. For a SCF and forces-only calculation, that would corresponds to a `11000` sequence, which has a `24` decimal representation. The QE side of the i-PI socket expects the equivalent-decimal`+1`; therefore, for a `11000` calculation, the integer `25` would have to be parsed to the `driver_init` subroutine in `run_driver.f90`. If any number less-than or equal-to `1` is parsed to QE, it falls back to its standard i-PI mode.

Currently, the QE i-PI interface can only reside in three different states: ”NEEDINIT”, ”READY” or ”HAVEDATA”. Whenever the socket sends a ”STATUS” message to QE, it responds back with its current status. A simple calculation sequence of events would be: (1) an ”STATUS” message is received, QE sends back ”NEEDINIT”, (2) an ”INIT” message is received, QE waits for three data packets, (i) an integer that identifies the client on the other side of the socket, (ii) the flag-encoded integer mentioned above, which can be used to change calculation settings, and (iii) an initialization string. QE then changes its status to ”READY”. (3) The server sends a ”POSDATA” message and QE then expects a sequence of variables depending on the calculation settings; the default being: a 3-by-3 matrix with cell and 3-by-3 marix with its inverse (if `lmovecell` is `.TRUE.`) and a (# of atoms)-by-3 position matrix. QE proceeds and computes all active properties (e.g. SCF, forces, stresses, etc.) and change its status to ”HAVEDATA” and expects a (4) ”GETFORCE” message from the socket. Once it is received, (5) QE sends back (i) a float with the potential energy, (ii) an integer with the total number of atoms, (iii) a (# of atoms)-by-3 matrix with forces (if `lforce` is `.TRUE.`), (iv) a 9-element-virial tensor (if `lstres` is `.TRUE.`). QE goes back to ”NEEDINIT” status. The other side of socket should be able to compute new positions and cell coordinates (if `lmovecell` is `.TRUE.`) and start the cycle again from (1).

4 Performances

4.1 Execution time

The following is a rough estimate of the complexity of a plain scf calculation with `pw.x`, for NCPP. USPP and PAW give raise additional terms to be calculated, that may add from a few percent up to 30-40% to execution time. For phonon calculations, each of the $3N_{at}$ modes requires a time of the same order of magnitude of self-consistent calculation in the same system (possibly times a small multiple). For `cp.x`, each time step takes something in the order of $T_h + T_{orth} + T_{sub}$ defined below.

The time required for the self-consistent solution at fixed ionic positions, T_{scf} , is:

$$T_{scf} = N_{iter} T_{iter} + T_{init}$$

where N_{iter} = number of self-consistency iterations (`niter`), T_{iter} = time for a single iteration, T_{init} = initialization time (usually much smaller than the first term).

The time required for a single self-consistency iteration T_{iter} is:

$$T_{iter} = N_k T_{diag} + T_{rho} + T_{scf}$$

where N_k = number of k-points, T_{diag} = time per Hamiltonian iterative diagonalization, T_{rho} = time for charge density calculation, T_{scf} = time for Hartree and XC potential calculation.

The time for a Hamiltonian iterative diagonalization T_{diag} is:

$$T_{diag} = N_h T_h + T_{orth} + T_{sub}$$

where N_h = number of $H\psi$ products needed by iterative diagonalization, T_h = time per $H\psi$ product, T_{orth} = CPU time for orthonormalization, T_{sub} = CPU time for subspace diagonalization.

The time T_h required for a $H\psi$ product is

$$T_h = a_1 MN + a_2 MN_1N_2N_3 log(N_1N_2N_3) + a_3 MPN.$$

The first term comes from the kinetic term and is usually much smaller than the others. The second and third terms come respectively from local and nonlocal potential. a_1, a_2, a_3 are prefactors (i.e. small numbers $\mathcal{O}(1)$), M = number of valence bands (`nbnd`), N = number of PW (basis set dimension: `npw`), N_1, N_2, N_3 = dimensions of the FFT grid for wavefunctions (`nr1s`, `nr2s`, `nr3s`; $N_1N_2N_3 \sim 8N$), P = number of pseudopotential projectors, summed on all atoms, on all values of the angular momentum l, and $m = 1, ..., 2l + 1$.

The time T_{orth} required by orthonormalization is

$$T_{orth} = b_1 NM_x^2$$

and the time T_{sub} required by subspace diagonalization is

$$T_{sub} = b_2 M_x^3$$

where b_1 and b_2 are prefactors, M_x = number of trial wavefunctions (this will vary between M and $2 \div 4M$, depending on the algorithm).

The time T_{rho} for the calculation of charge density from wavefunctions is

$$T_{rho} = c_1 MN_{r1}N_{r2}N_{r3} log(N_{r1}N_{r2}N_{r3}) + c_2 MN_{r1}N_{r2}N_{r3} + T_{us}$$

where c_1, c_2, c_3 are prefactors, N_{r1}, N_{r2}, N_{r3} = dimensions of the FFT grid for charge density (`nr1`, `nr2`, `nr3`; $N_{r1}N_{r2}N_{r3} \sim 8N_g$, where N_g = number of G-vectors for the charge density, `ngm`), and T_{us} = time required by PAW/USPPs contribution (if any). Note that for NCPPs the FFT grids for charge and wavefunctions are the same.

The time T_{scf} for calculation of potential from charge density is

$$T_{scf} = d_2 N_{r1}N_{r2}N_{r3} + d_3 N_{r1}N_{r2}N_{r3} log(N_{r1}N_{r2}N_{r3})$$

where d_1, d_2 are prefactors.

For hybrid DFTs, the dominant term is by far the calculation of the nonlocal ($V_x\psi$) product, taking as much as

$$T_{exx} = eN_kN_qM^2N_1N_2N_3 log(N_1N_2N_3)$$

where N_q is the number of points in the $k + q$ grid, determined by options `nqx1,nqx2,nqx3`, e is a prefactor.

The above estimates are for serial execution. In parallel execution, each contribution may scale in a different manner with the number of processors (see below).

4.2 Memory requirements

A typical self-consistency or molecular-dynamics run requires a maximum memory in the order of O double precision complex numbers, where

$$O = mMN + PN + pN_1N_2N_3 + qN_{r1}N_{r2}N_{r3}$$

with m, p, q = small factors; all other variables have the same meaning as above. Note that if the $\Gamma-$point only ($k = 0$) is used to sample the Brillouin Zone, the value of N will be cut into half.

For hybrid DFTs, additional storage of O_x double precision complex numbers is needed (for Fourier-transformed Kohn-Sham states), with $O_x = xN_qMN_1N_2N_3$ and $x = 0.5$ for $\Gamma-$only calculations, $x = 1$ otherwise.

The memory required by the phonon code follows the same patterns, with somewhat larger factors m, p, q.

4.3 File space requirements

A typical `pw.x` run will require an amount of temporary disk space in the order of O double precision complex numbers:

$$O = N_kMN + qN_{r1}N_{r2}N_{r3}$$

where $q = 2\times$ `mixing_ndim` (number of iterations used in self-consistency, default value = 8) if `disk_io` is set to 'high'; q = 0 otherwise.

4.4 Parallelization issues

`pw.x` can run in principle on any number of processors. The effectiveness of parallelization is ultimately judged by the "scaling", i.e. how the time needed to perform a job scales with the number of processors, and depends upon:

- the size and type of the system under study;
- the judicious choice of the various levels of parallelization (detailed in Sec.4.4);
- the availability of fast interprocess communications (or lack of it).

Ideally one would like to have linear scaling, i.e. $T \sim T_0/N_p$ for N_p processors, where T_0 is the estimated time for serial execution. In addition, one would like to have linear scaling of the RAM per processor: $O_N \sim O_0/N_p$, so that large-memory systems fit into the RAM of each processor.

Parallelization on k-points:

- guarantees (almost) linear scaling if the number of k-points is a multiple of the number of pools;
- requires little communications (suitable for ethernet communications);
- reduces the required memory per processor by distributing wavefunctions (but not other quantities like the charge density). Does not hold if you set `disk_io='high'`.

Parallelization on PWs:

- yields good to very good scaling, especially if the number of processors in a pool is a divisor of N_3 and N_{r3} (the dimensions along the z-axis of the FFT grids, `nr3` and `nr3s`, which coincide for NCPPs);
- requires heavy communications (suitable for Gigabit ethernet up to 4, 8 CPUs at most, specialized communication hardware needed for 8 or more processors);
- yields almost linear reduction of memory per processor with the number of processors in the pool.

A note on scaling: optimal serial performances are achieved when the data are as much as possible kept into the cache. As a side effect, PW parallelization may yield superlinear (better than linear) scaling, thanks to the increase in serial speed coming from the reduction of data size (making it easier for the machine to keep data in the cache).

VERY IMPORTANT: For each system there is an optimal range of number of processors on which to run the job. A too large number of processors will yield performance degradation. If the size of pools is especially delicate: N_p should not exceed N_3 and N_{r3}, and should ideally be no larger than $1/2 \div 1/4 N_3$ and/or N_{r3}. In order to increase scalability, it is often convenient to further subdivide a pool of processors into "task groups". When the number of processors exceeds the number of FFT planes, data can be redistributed to "task groups" so that each group can process several wavefunctions at the same time.

The optimal number of processors for "linear-algebra" parallelization, taking care of multiplication and diagonalization of $M \times M$ matrices, should be determined by observing the performances of `cdiagh/rdiagh` (`pw.x`) or `ortho` (`cp.x`) for different numbers of processors in the linear-algebra group (must be a square integer).

Actual parallel performances will also depend on the available software (MPI libraries) and on the available communication hardware. For PC clusters, OpenMPI (`http://www.openmpi.org/`) seems to yield better performances than other implementations (info by Kostantin Kudin). Note however that you need a decent communication hardware (at least Gigabit ethernet) in order to have acceptable performances with PW parallelization. Do not expect good scaling with cheap hardware: PW calculations are by no means an "embarrassing parallel" problem.

Also note that multiprocessor motherboards for Intel Pentium CPUs typically have just one memory bus for all processors. This dramatically slows down any code doing massive access to memory (as most codes in the QUANTUM ESPRESSO distribution do) that runs on processors of the same motherboard.

4.5 Understanding the time report

The time report printed at the end of a `pw.x` run contains a lot of useful information that can be used to understand bottlenecks and improve performances.

4.5.1 Serial execution

The following applies to calculations taking a sizable amount of time (at least minutes): for short calculations (seconds), the time spent in the various initializations dominates. Any discrepancy with the following picture signals some anomaly.

- For a typical job with norm-conserving PPs, the total (wall) time is mostly spent in routine "electrons", calculating the self-consistent solution.

- Most of the time spent in "electrons" is used by routine "c_bands", calculating Kohn-Sham states. "sum_band" (calculating the charge density), "v_of_rho" (calculating the potential), "mix_rho" (charge density mixing) should take a small fraction of the time.
- Most of the time spent in "c_bands" is used by routines "cegterg" (k-points) or "regterg" (Gamma-point only), performing iterative diagonalization of the Kohn-Sham Hamiltonian in the PW basis set.
- Most of the time spent in "*egterg" is used by routine "h_psi", calculating $H\psi$ products. "cdiaghg" (k-points) or "rdiaghg" (Gamma-only), performing subspace diagonalization, should take only a small fraction.
- Among the "general routines", most of the time is spent in FFT on Kohn-Sham states: "fftw", and to a smaller extent in other FFTs, "fft" and "ffts", and in "calbec", calculating $\langle\psi|\beta\rangle$ products.
- Forces and stresses typically take a fraction of the order of 10 to 20% of the total time.

For PAW and Ultrasoft PP, you will see a larger contribution by "sum_band" and a nonnegligible "newd" contribution to the time spent in "electrons", but the overall picture is unchanged. You may drastically reduce the overhead of Ultrasoft PPs by using input option "tqr=.true.".

4.5.2 Parallel execution

The various parallelization levels should be used wisely in order to achieve good results. Let us summarize the effects of them on CPU:

- Parallelization on FFT speeds up (with varying efficiency) almost all routines, with the notable exception of "cdiaghg" and "rdiaghg".
- Parallelization on k-points speeds up (almost linearly) "c_bands" and called routines; speeds up partially "sum_band"; does not speed up at all "v_of_rho", "newd", "mix_rho".
- Linear-algebra parallelization speeds up (not always) "cdiaghg" and "rdiaghg".
- "task-group" parallelization speeds up "fftw".
- OpenMP parallelization speeds up "fftw", plus selected parts of the calculation, plus (depending on the availability of OpenMP-aware libraries) some linear algebra operations.

and on RAM:

- Parallelization on FFT distributes most arrays across processors (i.e. all G-space and R-spaces arrays) but not all of them (in particular, not subspace Hamiltonian and overlap matrices).
- Linear-algebra parallelization also distributes subspace Hamiltonian and overlap matrices.
- All other parallelization levels do not distribute any memory.

In an ideally parallelized run, you should observe the following:

- CPU and wall time do not differ by much, if OpenMP is not active, or: CPU time approaches wall time times the number of OpenMP threads, if OpenMP is active.
- Time usage is still dominated by the same routines as for the serial run.
- Routine "fft_scatter" (called by parallel FFT) takes a sizable part of the time spent in FFTs but does not dominate it.

Quick estimate of parallelization parameters You need to know

- the number of k-points, N_k
- the third dimension of the (smooth) FFT grid, N_3
- the number of Kohn-Sham states, M

These data allow to set bounds on parallelization:

- k-point parallelization is limited to N_k processor pools: `-nk Nk`
- FFT parallelization shouldn't exceed N_3 processors, i.e. if you run with `-nk Nk`, use $N = N_k \times N_3$ MPI processes at most (`mpirun -np N ...`)
- Unless M is a few hundreds or more, don't bother using linear-algebra parallelization

You will need to experiment a bit to find the best compromise. In order to have good load balancing among MPI processes, the number of k-point pools should be an integer divisor of N_k; the number of processors for FFT parallelization should be an integer divisor of N_3.

Automatic guess of parallelization parameters Since v.7.1, the code tries to guess a reasonable set of parameters for the k-point, linear-algebra, and task-group parallelizations, if they are not explicitly provided in the command line. The logic is as follows:

- if the number of processors N_p exceeds N_3, one uses k-point parallelization on the smallest number N_k such that $N_p/N_k \leq N_3/2$;
- if the number of processors N_p/N_k still exceeds N_3, one uses task-group parallelization on the smallest N_t that ensures $N_p/N_k/N_t \leq N_3/4$;
- linear-algebra parallelization is done on n_d^2 processors ($n_d^2 \leq N_p/N_k/N_t$) with n_d such that $M/n_d \sim 100$.

Typical symptoms of bad/inadequate parallelization

- *a large fraction of time is spent in "v_of_rho", "newd", "mix_rho"*, or
  the time doesn't scale well or doesn't scale at all by increasing the number of processors for k-point parallelization. Solution:
  - use (also) FFT parallelization if possible
- *a disproportionate time is spent in "cdiaghg"/"rdiaghg".* Solutions:

  - use (also) k-point parallelization if possible
  - use linear-algebra parallelization, with scalapack if possible.

- *a disproportionate time is spent in "fft_scatter"*, or *in "fft_scatter" the difference between CPU and wall time is large.* Solutions:
  - if you do not have fast (better than Gigabit ethernet) communication hardware, do not try FFT parallelization on more than 4 or 8 procs.
  - use (also) k-point parallelization if possible

- *the time doesn't scale well or doesn't scale at all by increasing the number of processors for FFT parallelization.* Solutions:
  - use "task groups": try command-line option `-ntg 4` or `-ntg 8`. This may improve your scaling.

4.6 Restarting

Since QE 5.1 restarting from an arbitrary point of the code is no more supported.

The code must terminate properly in order for restart to be possible. A clean stop can be triggered by one the following three conditions:

1. The amount of time specified by the input variable max_seconds is reached
2. The user creates a file named "$prefix.EXIT" either in the working directory or in output directory "$outdir" (variables $outdir and $prefix as specified in the control namelist)
3. (experimental) The code is compiled with signal-trapping support and one of the trapped signals is received (see the next section for details).

After the condition is met, the code will try to stop cleanly as soon as possible, which can take a while for large calculation. Writing the files to disk can also be a long process. In order to be safe you need to reserve sufficient time for the stop process to complete.

If the previous execution of the code has stopped properly, restarting is possible setting restart_mode="restart" in the control namelist.

4.6.1 Signal trapping (experimental!)

In order to compile signal-trapping add "-D__TERMINATE_GRACEFULLY" to MANUAL_DFLAGS in the make.doc file. Currently the code intercepts SIGINT, SIGTERM, SIGUSR1, SIGUSR2, SIGXCPU; signals can be added or removed editing the file `clib/custom_signals.c`.

Common queue systems will send a signal some time before killing a job. The exact behaviour depends on the queue systems and could be configured. Some examples:

With PBS:

- send the default signal (SIGTERM) 120 seconds before the end:
  `#PBS -l signal=@120`

- send signal SIGUSR1 10 minutes before the end:
  `#PBS -l signal=SIGUSR1@600`

- you cand also send a signal manually with qsig
- or send a signal and then stop:
  `qdel -W 120 jobid`
  will send SIGTERM, wait 2 minutes than force stop.

With LoadLeveler (untested): the SIGXCPU signal will be sent when wall *softlimit* is reached, it will then stop the job when *hardlimit* is reached. You can specify both limits as:
```
# @ wall_clock_limit = hardlimit,softlimit
```
e.g. you can give pw.x thirty minutes to stop using:
```
# @ wall_clock_limit = 5:00,4:30
```

5 Troubleshooting

pw.x says 'error while loading shared libraries' or 'cannot open shared object file' and does not start Possible reasons:

- If you are running on the same machines on which the code was compiled, this is a library configuration problem. The solution is machine-dependent. On Linux, find the path to the missing libraries; then either add it to file `/etc/ld.so.conf` and run `ldconfig` (must be done as root), or add it to variable LD_LIBRARY_PATH and export it. Another possibility is to load non-shared version of libraries (ending with .a) instead of shared ones (ending with .so).
- If you are *not* running on the same machines on which the code was compiled: you need either to have the same shared libraries installed on both machines, or to load statically all libraries (using appropriate `configure` or loader options). The same applies to Beowulf-style parallel machines: the needed shared libraries must be present on all PCs.

errors in examples with parallel execution If you get error messages in the example scripts – i.e. not errors in the codes – on a parallel machine, such as e.g.: *run example: -n: command not found* you may have forgotten to properly define PARA_PREFIX and PARA_POSTFIX.

pw.x prints the first few lines and then nothing happens (parallel execution) If the code looks like it is not reading from input, maybe it isn't: the MPI libraries need to be properly configured to accept input redirection. Use `pw.x -i` and the input file name (see Sec.4.4), or inquire with your local computer wizard (if any). Since v.4.2, this is for sure the reason if the code stops at *Waiting for input...*.

pw.x stops with error while reading data There is an error in the input data, typically a misspelled namelist variable, or an empty input file. Unfortunately with most compilers the code often reports *Error while reading XXX namelist* and no further useful information. Here are some more subtle sources of trouble:

- Input files should be plain ASCII text. The presence of CRLF line terminators (may appear as ˆM, Control-M, characters at the end of lines), tabulators, or non-ASCII characters (e.g. non-ASCII quotation marks, that at a first glance may look the same as the ASCII character) is a frequent source of trouble. Typically, this happens with files coming from Windows or produced with ”smart” editors. Verify with command `file` and convert with command `iconv` if needed.

- The input file ends at the last character (there is no end-of-line character).

- Out-of-bound indices in dimensioned variables read in the namelists.

These reasons may cause the code to crash with rather mysterious error messages. If none of the above applies and the code stops at the first namelist (&CONTROL) and you are running in parallel, see the previous item.

pw.x mumbles something like *cannot recover* or *error reading recover file* You are trying to restart from a previous job that either produced corrupted files, or did not do what you think it did. No luck: you have to restart from scratch.

pw.x stops with *inconsistent DFT* error As a rule, the flavor of DFT used in the calculation should be the same as the one used in the generation of pseudopotentials, which should all be generated using the same flavor of DFT. This is actually enforced: the type of DFT is read from pseudopotential files and it is checked that the same DFT is read from all PPs. If this does not hold, the code stops with the above error message. Use – at your own risk – input variable `input_dft` to force the usage of the DFT you like.

pw.x stops with error in cdiaghg or rdiaghg Possible reasons for such behavior are not always clear, but they typically fall into one of the following cases:

- serious error in data, such as bad atomic positions or bad crystal structure/supercell;

- a bad pseudopotential, typically with a ghost, or a USPP giving non-positive charge density, leading to a violation of positiveness of the S matrix appearing in the USPP formalism;

- a failure of the algorithm performing subspace diagonalization. The LAPACK algorithms used by `cdiaghg` (for generic k-points) or `rdiaghg` (for $\Gamma-$only case) are very robust and extensively tested. Still, it may seldom happen that such algorithms fail. Try to use conjugate-gradient diagonalization (`diagonalization='cg'`), a slower but very robust algorithm, and see what happens; or, newer diagonalization 'paro'.

- buggy libraries. Machine-optimized mathematical libraries are very fast but sometimes not so robust from a numerical point of view. Suspicious behavior: you get an error that is not reproducible on other architectures or that disappears if the calculation is repeated with even minimal changes in parameters. Try to use compiled BLAS and LAPACK (or better, ATLAS) instead of machine-optimized libraries.

pw.x crashes with no error message at all This happens quite often in parallel execution, or under a batch queue, or if you are writing the output to a file. When the program crashes, part of the output, including the error message, may be lost, or hidden into error files where nobody looks into. It is the fault of the operating system, not of the code. Try to run interactively and to write to the screen. If this doesn't help, move to next point.

pw.x crashes with *segmentation fault* or similarly obscure messages Possible reasons:

- too much RAM memory or stack requested (see next item).
- if you are using highly optimized mathematical libraries, verify that they are designed for your hardware.
- If you are using aggressive optimization in compilation, verify that you are using the appropriate options for your machine
- the executable was not properly compiled, or was compiled on a different and incompatible environment.
- buggy compiler or libraries: this is the default explanation if you have problems with the provided tests and examples.

pw.x works for simple systems, but not for large systems or whenever more RAM is needed Possible solutions:

- Increase the amount of RAM you are authorized to use (which may be much smaller than the available RAM). Ask your system administrator if you don't know what to do. In some cases the stack size can be a source of problems: if so, increase it with command `limits` or `ulimit`).
- Reduce `nbnd` to the strict minimum (for insulators, the default is already the minimum, though).
- Reduce the work space for Davidson diagonalization to the minimum by setting `diago_david_ndim` also consider using conjugate gradient diagonalization (`diagonalization='cg'`), slow but very robust, which requires almost no work space.
- If the charge density takes a significant amount of RAM, reduce `mixing_ndim` from its default value (8) to 4 or so.
- In parallel execution, use more processors, or use the same number of processors with less pools. Remember that parallelization with respect to k-points (pools) does not distribute memory: only parallelization with respect to R- (and G-) space does.
- If none of the above is sufficient or feasible, you have to either reduce the cutoffs and/or the cell size, or to use a machine with more RAM.

pw.x crashes with *error in davcio* `davcio` is the routine that performs most of the I/O operations (read from disk and write to disk) in `pw.x`; *error in davcio* means a failure of an I/O operation.

- If the error is reproducible and happens at the beginning of a calculation: check if you have read/write permission to the scratch directory specified in variable `outdir`. Also: check if there is enough free space available on the disk you are writing to, and check your disk quota (if any).
- If the error is irreproducible: your might have flaky disks; if you are writing via the network using NFS (which you shouldn't do anyway), your network connection might be not so stable, or your NFS implementation is unable to work under heavy load
- If it happens while restarting from a previous calculation: you might be restarting from the wrong place, or from wrong data, or the files might be corrupted. Note that, since QE 5.1, restarting from arbitrary places is no more supported: the code must terminate cleanly.
- If you are running two or more instances of `pw.x` at the same time, check if you are using the same file names in the same temporary directory. For instance, if you submit a series of jobs to a batch queue, do not use the same `outdir` and the same `prefix`, unless you are sure that one job doesn't start before a preceding one has finished.

pw.x crashes in parallel execution with an obscure message related to MPI errors Random crashes due to MPI errors have often been reported, typically in Linux PC clusters. We cannot rule out the possibility that bugs in QUANTUM ESPRESSO cause such behavior, but we are quite confident that the most likely explanation is a hardware problem (defective RAM for instance) or a software bug (in MPI libraries, compiler, operating system).

Debugging a parallel code may be difficult, but you should at least verify if your problem is reproducible on different architectures/software configurations/input data sets, and if there is some particular condition that activates the bug. If this doesn't seem to happen, the odds are that the problem is not in QUANTUM ESPRESSO. You may still report your problem, but consider that reports like *it crashes with...(obscure MPI error)* contain 0 bits of information and are likely to get 0 bits of answers.

pw.x stops with error message *the system is metallic, specify occupations* You did not specify state occupations, but you need to, since your system appears to have an odd number of electrons. The variable controlling how metallicity is treated is `occupations` in namelist &SYSTEM. The default, `occupations='fixed'`, occupies the lowest (N electrons)/2 states and works only for insulators with a gap. In all other cases, use `'smearing'` (`'tetrahedra'` for DOS calculations). See input reference documentation for more details.

pw.x stops with *internal error: cannot bracket Ef* Possible reasons:

- serious error in data, such as bad number of electrons, insufficient number of bands, absurd value of broadening;

- the Fermi energy is found by bisection assuming that the integrated DOS $N(E)$ is an increasing function of the energy. This is not guaranteed for Methfessel-Paxton smearing of order 1 and can give problems when very few k-points are used. Use some other smearing function: simple Gaussian broadening or, better, Marzari-Vanderbilt-DeVita-Payne 'cold smearing'.

pw.x yields *internal error: cannot bracket Ef* message but does not stop This may happen under special circumstances when you are calculating the band structure for selected high-symmetry lines. The message signals that occupations and Fermi energy are not correct (but eigenvalues and eigenvectors are). Remove `occupations='tetrahedra'` in the input data to get rid of the message.

pw.x runs but nothing happens Possible reasons:

- in parallel execution, the code died on just one processor. Unpredictable behavior may follow.
- in serial execution, the code encountered a floating-point error and goes on producing NaNs (Not a Number) forever unless exception handling is on (and usually it isn't). In both cases, look for one of the reasons given above.
- maybe your calculation will take more time than you expect.

pw.x yields weird results If results are really weird (as opposed to misinterpreted):

- if this happens after a change in the code or in compilation or preprocessing options, try `make clean`, recompile. The `make` command should take care of all dependencies, but do not rely too heavily on it. If the problem persists, recompile with reduced optimization level.
- maybe your input data are weird.

FFT grid is machine-dependent Yes, they are! The code automatically chooses the smallest grid that is compatible with the specified cutoff in the specified cell, and is an allowed value for the FFT library used. Most FFT libraries are implemented, or perform well, only with dimensions that factors into products of small numbers (2, 3, 5 typically, sometimes 7 and 11). Different FFT libraries follow different rules and thus different dimensions can result for the same system on different machines (or even on the same machine, with a different FFT). See function allowed in `FFTXlib/fft_support.f90`.

As a consequence, the energy may be slightly different on different machines. The only piece that explicitly depends on the grid parameters is the XC part of the energy that is computed numerically on the grid. The differences should be small, though, especially for LDA calculations.

Manually setting the FFT grids to a desired value is possible, but slightly tricky, using input variables `nr1`, `nr2`, `nr3` and `nr1s`, `nr2s`, `nr3s`. The code will still increase them if not acceptable. Automatic FFT grid dimensions are slightly overestimated, so one may try *very carefully* to reduce them a little bit. The code will stop if too small values are required, it will waste CPU time and memory for too large values.

Note that in parallel execution, it is very convenient to have FFT grid dimensions along z that are a multiple of the number of processors.

pw.x does not find all the symmetries you expected `pw.x` determines first the symmetry operations (rotations) of the Bravais lattice; then checks which of these are symmetry operations of the system (including if needed fractional translations). This is done by rotating (and translating if needed) the atoms in the unit cell and verifying if the rotated unit cell coincides with the original one.

Assuming that your coordinates are correct (please carefully check!), you may not find all the symmetries you expect because:

- the number of significant figures in the atomic positions is not large enough. In file `PW/src/eqvect.f90`, the variable `accep` is used to decide whether a rotation is a symmetry operation. Its current value (10^{-5}), set in module `PW/src/symm_base.f90`, is quite strict: a rotated atom must coincide with another atom to 5 significant digits. You may change the value of `accep` and recompile.
- they are not acceptable symmetry operations of the Bravais lattice. This is the case for C_{60}, for instance: the I_h icosahedral group of C_{60} contains 5-fold rotations that are incompatible with translation symmetry.
- the system is rotated with respect to symmetry axis. For instance: a C_{60} molecule in the fcc lattice will have 24 symmetry operations (T_h group) only if the double bond is aligned along one of the crystal axis; if C_{60} is rotated in some arbitrary way, `pw.x` may not find any symmetry, apart from inversion.
- they contain a fractional translation that is incompatible with the FFT grid (see next paragraph). Note that if you change cutoff or unit cell volume, the automatically computed FFT grid changes, and this may explain changes in symmetry (and in the number of k-points as a consequence) for no apparent good reason (only if you have fractional translations in the system, though).
- a fractional translation, without rotation, is a symmetry operation of the system. This means that the cell is actually a supercell. In this case, all symmetry operations containing fractional translations are disabled. The reason is that in this rather exotic case there is no simple way to select those symmetry operations forming a true group, in the mathematical sense of the term.

Self-consistency is slow or does not converge at all Bad input data will often result in bad scf convergence. Please carefully check your structure first, e.g. using XCrySDen.

Assuming that your input data is sensible :

1. Verify if your system is metallic or is close to a metallic state, especially if you have few k-points. If the highest occupied and lowest unoccupied state(s) keep exchanging place during self-consistency, forget about reaching convergence. A typical sign of such behavior is that the self-consistency error goes down, down, down, than all of a sudden up again, and so on. Usually one can solve the problem by adding a few empty bands and a small broadening.

2. Reduce `mixing_beta` to $\sim 0.3 \div 0.1$ or smaller. Try the `mixing_mode` value that is more appropriate for your problem. For slab geometries used in surface problems or for elongated cells, `mixing_mode='local-TF'` should be the better choice, dampening "charge sloshing". You may also try to increase `mixing_ndim` to more than 8 (default value). Beware: this will increase the amount of memory you need.

3. Specific to USPP: the presence of negative charge density regions due to either the pseudization procedure of the augmentation part or to truncation at finite cutoff may give convergence problems. Raising the `ecutrho` cutoff for charge density will usually help.

I do not get the same results in different machines! If the difference is small, do not panic. It is quite normal for iterative methods to reach convergence through different paths as soon as anything changes. In particular, between serial and parallel execution there are operations that are not performed in the same order. As the numerical accuracy of computer numbers is finite, this can yield slightly different results.

It is also normal that the total energy converges to a better accuracy than its terms, since only the sum is variational, i.e. has a minimum in correspondence to ground-state charge density. Thus if the convergence threshold is for instance 10^{-8}, you get 8-digit accuracy on the total energy, but one or two less on other terms (e.g. XC and Hartree energy). It this is a problem for you, reduce the convergence threshold for instance to 10^{-10} or 10^{-12}. The differences should go away (but it will probably take a few more iterations to converge).

Execution time is time-dependent! Yes it is! On most machines and on most operating systems, depending on machine load, on communication load (for parallel machines), on various other factors (including maybe the phase of the moon), reported execution times may vary quite a lot for the same job.

Warning : N eigenvectors not converged This is a warning message that can be safely ignored if it is not present in the last steps of self-consistency. If it is still present in the last steps of self-consistency, and if the number of unconverged eigenvector is a significant part of the total, it may signal serious trouble in self-consistency (see next point) or something badly wrong in input data.

***Warning : negative or imaginary charge...*, or *...core charge ...*, or *npt with rhoup*< 0... or *rho dw*< 0...** These are warning messages that can be safely ignored unless the negative or imaginary charge is sizable, let us say of the order of 0.1. If it is, something seriously wrong is going on. Otherwise, the origin of the negative charge is the following. When one transforms a positive function in real space to Fourier space and truncates at some finite cutoff, the positive function is no longer guaranteed to be positive when transformed back to real space. This happens only with core corrections and with USPPs. In some cases it may be a source of trouble (see next point) but it is usually solved by increasing the cutoff for the charge density.

Structural optimization is slow or does not converge or ends with a mysterious bfgs error Typical structural optimizations, based on the BFGS algorithm, converge to the default thresholds (etot_conv_thr and forc_conv_thr) in 15-25 BFGS steps (depending on the starting configuration). This may not happen when your system is characterized by "floppy" low-energy modes, that make very difficult (and of little use anyway) to reach a well converged structure, no matter what. Other possible reasons for a problematic convergence are listed below.

Close to convergence the self-consistency error in forces may become large with respect to the value of forces. The resulting mismatch between forces and energies may confuse the line minimization algorithm, which assumes consistency between the two. The code reduces the starting self-consistency threshold conv thr when approaching the minimum energy configuration, up to a factor defined by `upscale`. Reducing `conv_thr` (or increasing `upscale`) yields a smoother structural optimization, but if `conv_thr` becomes too small, electronic self-consistency may not converge. You may also increase variables `etot_conv_thr` and `forc_conv_thr` that determine the threshold for convergence (the default values are quite strict).

A limitation to the accuracy of forces comes from the absence of perfect translational invariance. If we had only the Hartree potential, our PW calculation would be translationally invariant to machine precision. The presence of an XC potential introduces Fourier components in the potential that are not in our basis set. This loss of precision (more serious for gradient-corrected functionals) translates into a slight but detectable loss of translational invariance (the energy changes if all atoms are displaced by the same quantity, not commensurate with the FFT grid). This sets a limit to the accuracy of forces. The situation improves somewhat by increasing the `ecutrho` cutoff.

pw.x stops during variable-cell optimization in checkallsym with *non orthogonal operation* error Variable-cell optimization may occasionally break the starting symmetry of the cell. When this happens, the run is stopped because the number of k-points calculated for the starting configuration may no longer be suitable. Possible solutions:

- start with a nonsymmetric cell;
- use a symmetry-conserving algorithm: the Wentzcovitch algorithm (`cell dynamics='damp-w'`) should not break the symmetry.