User's Guide for Quantum ESPRESSO

(version 4.2.0)

Contents

1 Introduction

This guide covers the installation and usage of QUANTUM ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization), version 4.2.0.

The QUANTUM ESPRESSO distribution contains the following core packages for the calculation of electronic-structure properties within Density-Functional Theory (DFT), using a Plane-Wave (PW) basis set and pseudopotentials (PP):

- `PWscf` (Plane-Wave Self-Consistent Field).
- `CP` (Car-Parrinello).

It also includes the following more specialized packages:

- `PHonon`: phonons with Density-Functional Perturbation Theory.
- `PostProc`: various utilities for data postprocessing.
- `PWcond`: ballistic conductance.
- `GIPAW` (Gauge-Independent Projector Augmented Waves): EPR g-tensor and NMR chemical shifts.
- `XSPECTRA`: K-edge X-ray adsorption spectra.
- `vdW`: (experimental) dynamic polarizability.
- `GWW`: (experimental) GW calculation using Wannier functions.

The following auxiliary codes are included as well:

- `PWgui`: a Graphical User Interface, producing input data files for `PWscf`.
- `atomic`: a program for atomic calculations and generation of pseudopotentials.
- `QHA`: utilities for the calculation of projected density of states (PDOS) and of the free energy in the Quasi-Harmonic Approximation (to be used in conjunction with `PHonon`).
- `PlotPhon`: phonon dispersion plotting utility (to be used in conjunction with `PHonon`).

A copy of required external libraries are included:

- `iotk`: an Input-Output ToolKit.
- PMG: Multigrid solver for Poisson equation.
- BLAS and LAPACK

Finally, several additional packages that exploit data produced by QUANTUM ESPRESSO can be installed as *plug-ins*:

- `Wannier90`: maximally localized Wannier functions (`http://www.wannier.org/`), written by A. Mostofi, J. Yates, Y.-S Lee.
- `WanT`: quantum transport properties with Wannier functions.
- `YAMBO`: optical excitations with Many-Body Perturbation Theory.

This guide documents `PWscf`, `CP`, `PHonon`, `PostProc`. The remaining packages have separate documentation.

The QUANTUM ESPRESSO codes work on many different types of Unix machines, including parallel machines using both OpenMP and MPI (Message Passing Interface). Running QUANTUM ESPRESSO on Mac OS X and MS-Windows is also possible: see section 2.2.

Further documentation, beyond what is provided in this guide, can be found in:

- the `pw_forum` mailing list (`pw_forum@pwscf.org`). You can subscribe to this list, browse and search its archives (links in `http://www.quantum-espresso.org/contacts.php`). Only subscribed users can post. Please search the archives before posting: your question may have already been answered.
- the `Doc/` directory of the QUANTUM ESPRESSO distribution, containing a detailed description of input data for most codes in files `INPUT_*.txt` and `INPUT_*.html`, plus and a few additional pdf documents; people who want to contribute to QUANTUM ESPRESSO should read the Developer Manual, `developer_man.pdf`.
- the QUANTUM ESPRESSO Wiki: `http://www.quantum-espresso.org/wiki/index.php/Main_Page`.

This guide does not explain solid state physics and its computational methods. If you want to learn that, you should read a good textbook, such as e.g. the book by Richard Martin: *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press (2004). See also the Reference Paper section in the Wiki.

This guide assume that you know the basic Unix concepts (shell, execution path, directories etc.) and utilities. If you don't, you will have a hard time running QUANTUM ESPRESSO.

1.1 What can QUANTUM ESPRESSO do

`PWscf` can currently perform the following kinds of calculations:

- ground-state energy and one-electron (Kohn-Sham) orbitals;
- atomic forces, stresses, and structural optimization;

- molecular dynamics on the ground-state Born-Oppenheimer surface, also with variable cell;
- Nudged Elastic Band (NEB) and Fourier String Method Dynamics (SMD) for energy barriers and reaction paths;
- macroscopic polarization and finite electric fields via the modern theory of polarization (Berry Phases).

All of the above works for both insulators and metals, in any crystal structure, for many exchange-correlation (XC) functionals (including spin polarization, DFT+U, hybrid functionals), for norm-conserving (Hamann-Schluter-Chiang) PPs (NCPPs) in separable form or Ultrasoft (Vanderbilt) PPs (USPPs) or Projector Augmented Waves (PAW) method. Non-collinear magnetism and spin-orbit interactions are also implemented. An implementation of finite electric fields with a sawtooth potential in a supercell is also available.

`PHonon` can perform the following types of calculations:

- phonon frequencies and eigenvectors at a generic wave vector, using Density-Functional Perturbation Theory;
- effective charges and dielectric tensors;
- electron-phonon interaction coefficients for metals;
- interatomic force constants in real space;
- third-order anharmonic phonon lifetimes;
- Infrared and Raman (nonresonant) cross section.

`PHonon` can be used whenever `PWscf` can be used, with the exceptions of DFT+U and hybrid functionals. PAW is not implemented for higher-order response calculations. Calculations, in the Quasi-Harmonic approximations, of the vibrational free energy can be performed using the `QHA` package.

`PostProc` can perform the following types of calculations:

- Scanning Tunneling Microscopy (STM) images;
- plots of Electron Localization Functions (ELF);
- Density of States (DOS) and Projected DOS (PDOS);
- Löwdin charges;
- planar and spherical averages;

plus interfacing with a number of graphical utilities and with external codes.

`CP` can perform Car-Parrinello molecular dynamics, including variable-cell dynamics.

1.2 People

In the following, the cited affiliation is either the current one or the one where the last known contribution was done.

The maintenance and further development of the QUANTUM ESPRESSO distribution is promoted by the DEMOCRITOS National Simulation Center of IOM-CNR under the coordination of Paolo Giannozzi (Univ.Udine, Italy) and Layla Martin-Samos (Democritos) with the strong support of the CINECA National Supercomputing Center in Bologna under the responsibility of Carlo Cavazzoni.

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

- Matteo Cococcioni (Univ. Minnesota) for DFT+U implementation;
- David Vanderbilt's group at Rutgers for Berry's phase calculations;
- Ralph Gebauer (ICTP, Trieste) and Adriano Mosca Conte (SISSA, Trieste) for noncolinear magnetism;
- Andrea Dal Corso for spin-orbit interactions;
- Carlo Sbraccia (Princeton) for NEB, Strings method, for improvements to structural optimization and to many other parts;
- Paolo Umari (Democritos) for finite electric fields;
- 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);
- Ismaila Dabo (INRIA, Palaiseau) for electrostatics with free boundary conditions.

For `PHonon`, we mention in particular:

- Michele Lazzeri (Univ.Paris VI) for the 2n+1 code and Raman cross section calculation with 2nd-order response;
- Andrea Dal Corso for USPP, noncollinear, spin-orbit extensions to `PHonon`.

For `PostProc`, we mention:

- Andrea Benassi (SISSA) for the `epsilon` utility;
- Norbert Nemec (U.Cambridge) for the `pw2casino` utility;
- Dmitry Korotin (Inst. Met. Phys. Ekaterinburg) for the `wannier_ham` utility.

The CP package is based on the original code written by Roberto Car and Michele Parrinello. CP was developed by Alfredo Pasquarello (IRRMA, Lausanne), Kari Laasonen (Oulu), Andrea Trave, Roberto Car (Princeton), Nicola Marzari (Univ. Oxford), Paolo Giannozzi, and others. FPMD, later merged with CP, was developed by Carlo Cavazzoni, Gerardo Ballabio (CINECA), Sandro Scandolo (ICTP), Guido Chiarotti (SISSA), Paolo Focher, and others. We quote in particular:

- Carlo Sbraccia (Princeton) for NEB;
- Manu Sharma (Princeton) and Yudong Wu (Princeton) for maximally localized Wannier functions and dynamics with Wannier functions;
- Paolo Umari (Democritos) for finite electric fields and conjugate gradients;
- Paolo Umari and Ismaila Dabo for ensemble-DFT;
- Xiaofei Wang (Princeton) for META-GGA;
- The Autopilot feature was implemented by Targacept, Inc.

Other packages in QUANTUM ESPRESSO:

- PWcond was written by Alexander Smogunov (SISSA) and Andrea Dal Corso. For an introduction, see http://people.sissa.it/~smogunov/PWCOND/pwcond.html
- GIPAW (http://www.gipaw.net) was written by Davide Ceresoli (MIT), Ari Seitsonen (Univ.Zurich), Uwe Gerstmann, Francesco Mauri (Univ. Paris VI).
- PWgui was written by Anton Kokalj (IJS Ljubljana) and is based on his GUIB concept (http://www-k3.ijs.si/kokalj/guib/).
- atomic was written by Andrea Dal Corso and it is the result of many additions to the original code by Paolo Giannozzi and others. Lorenzo Paulatto wrote the PAW extension.
- iotk (http://www.s3.infm.it/iotk) was written by Giovanni Bussi (SISSA) .
- XSPECTRA was written by Matteo Calandra (Univ. Paris VI) and collaborators.
- VdW was contributed by Huy-Viet Nguyen (SISSA).
- GWW was written by Paolo Umari and Geoffrey Stenuit (Democritos).
- QHA amd PlotPhon were contributed by Eyvaz Isaev (Moscow Steel and Alloy Inst. and Linkoping and Uppsala Univ.).

Other relevant contributions to QUANTUM ESPRESSO:

- Andrea Ferretti (MIT) contributed the qexml and sumpdos utility, helped with file formats and with various problems;
- Hannu-Pekka Komsa (CSEA/Lausanne) contributed the HSE functional;
- Dispersions interaction in the framework of DFT-D were contributed by Daniel Forrer (Padua Univ.) and Michele Pavone (Naples Univ. Federico II);

- Filippo Spiga (Univ. Milano Bicocca) contributed the mixed MPI-OpenMP parallelization;
- The initial BlueGene porting was done by Costas Bekas and Alessandro Curioni (IBM Zurich);
- Gerardo Ballabio wrote the first `configure` for QUANTUM ESPRESSO
- Audrius Alkauskas (IRRMA), Uli Aschauer (Princeton), Simon Binnie (Univ. College London), Guido Fratesi, Axel Kohlmeyer (UPenn), Konstantin Kudin (Princeton), Sergey Lisenkov (Univ.Arkansas), Nicolas Mounet (MIT), William Parker (Ohio State Univ), Guido Roma (CEA), Gabriele Sclauzero (SISSA), Sylvie Stucki (IRRMA), Pascal Thibaudeau (CEA), Vittorio Zecca, Federico Zipoli (Princeton) answered questions on the mailing list, found bugs, helped in porting to new architectures, wrote some code.

An alphabetical list of further contributors includes: Dario Alfè, Alain Allouche, Francesco Antoniella, Francesca Baletto, Mauro Boero, Nicola Bonini, Claudia Bungaro, Paolo Cazzato, Gabriele Cipriani, Jiayu Dai, Cesar Da Silva, Alberto Debernardi, Gernot Deinzer, Yves Ferro, Martin Hilgeman, Yosuke Kanai, Nicolas Lacorne, Stephane Lefranc, Kurt Maeder, Andrea Marini, Pasquale Pavone, Mickael Profeta, Kurt Stokbro, Paul Tangney, Antonio Tilocca, Jaro Tobik, Malgorzata Wierzbowska, Silviu Zilberman, and let us apologize to everybody we have forgotten.

This guide was mostly written by Paolo Giannozzi. Gerardo Ballabio and Carlo Cavazzoni wrote the section on `CP`.

1.3 Contacts

The web site for QUANTUM ESPRESSO is `http://www.quantum-espresso.org/`. Releases and patches can be downloaded from this site or following the links contained in it. The main entry point for developers is the QE-forge web site: `http://www.qe-forge.org/`.

The recommended place where to ask questions about installation and usage of QUANTUM ESPRESSO, and to report bugs, is the `pw_forum` mailing list: `pw_forum@pwscf.org`. Here you can receive news about QUANTUM ESPRESSO and obtain help from the developers and from knowledgeable users. You have to be subscribed in order to post to the list. Please browse or search the archive – links are available in the "Contacts" page of the QUANTUM ESPRESSO web site, `http://www.quantum-espresso.org/contacts.php` – before posting: many questions are asked over and over again.

NOTA BENE: only messages that appear to come from the registered user's e-mail address, in its *exact form*, will be accepted. Messages "waiting for moderator approval" are automatically deleted with no further processing (sorry, too much spam). In case of trouble, carefully check that your return e-mail is the correct one (i.e. the one you used to subscribe).

Since `pw_forum` averages ~ 10 message a day, an alternative low-traffic mailing list, `pw_users@pwscf.org`, is provided for those interested only in QUANTUM ESPRESSO-related news, such as e.g. announcements of new versions, tutorials, etc.. You can subscribe (but not post) to this list from the QUANTUM ESPRESSO web site.

If you need to contact the developers for *specific* questions about coding, proposals, offers of help, etc., send a message to the developers' mailing list: user `q-e-developers`, address `qe-forge.org`.

1.4 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 this code will contain an explicit acknowledgment and the following reference:

> 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), http://arxiv.org/abs/0906.2569

Note the form QUANTUM ESPRESSO for textual citations of the code. 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`.

2 Installation

2.1 Download

Presently, QUANTUM ESPRESSO is only distributed in source form; some precompiled executables (binary files) are provided only for `PWgui`. Stable releases of the QUANTUM ESPRESSO source package (current version is 4.2.0) can be downloaded from this URL: `http://www.quantum-espresso.org/download.php`.

Uncompress and unpack the core distribution using the command:

```
tar zxvf espresso-X.Y.Z.tar.gz
```

(a hyphen before "zxvf" is optional) where `X.Y.Z` stands for the version number. If your version of `tar` doesn't recognize the "z" flag:

```
gunzip -c espresso-X.Y.Z.tar.gz | tar xvf -
```

A directory `espresso-X.Y.Z/` will be created. Given the size of the complete distribution, you may need to download more packages and to unpack them following the same procedure (they will unpack into the same directory). Plug-ins should instead be downloaded into subdirectory `plugin/archive` but not unpacked or uncompressed: command `make` will take care of this during installation.

Occasionally, patches for the current version, fixing some errors and bugs, may be distributed as a "diff" file. In order to install a patch (for instance):

```
cd espresso-X.Y.Z/
patch -p1 < /path/to/the/diff/file/patch-file.diff
```

If more than one patch is present, they should be applied in the correct order.

Daily snapshots of the development version can be downloaded from the developers' site `qe-forge.org`: follow the link "Quantum ESPRESSO", then "SCM". Beware: the development version is, well, under development: use at your own risk! The bravest may access the development version via anonymous CVS (Concurrent Version System): see the Developer Manual (`Doc/developer_man.pdf`), section "Using CVS".

The QUANTUM ESPRESSO distribution contains several directories. Some of them are common to all packages:

`Modules/`	source files for modules that are common to all programs
`include/`	files *.h included by fortran and C source files
`clib/`	external libraries written in C
`flib/`	external libraries written in Fortran
`iotk/`	Input/Output Toolkit
`install/`	installation scripts and utilities
`pseudo/`	pseudopotential files used by examples
`upftools/`	converters to unified pseudopotential format (UPF)
`examples/`	sample input and output files
`Doc/`	general documentation

while others are specific to a single package:

`PW/`	`PWscf`: source files for scf calculations (`pw.x`)
`pwtools/`	`PWscf`: source files for miscellaneous analysis programs
`tests/`	`PWscf`: automated tests
`PP/`	`PostProc`: source files for post-processing of `pw.x` data file
`PH/`	`PHonon`: source files for phonon calculations (`ph.x`) and analysis
`Gamma/`	`PHonon`: source files for Gamma-only phonon calculation (`phcg.x`)
`D3/`	`PHonon`: source files for third-order derivative calculations (`d3.x`)
`PWCOND/`	`PWcond`: source files for conductance calculations (`pwcond.x`)
`vdW/`	`VdW`: source files for molecular polarizability calculation at finite frequency
`CPV/`	`CP`: source files for Car-Parrinello code (`cp.x`)
`atomic/`	`atomic`: source files for the pseudopotential generation package (`ld1.x`)
`atomic_doc/`	Documentation, tests and examples for `atomic`
`GUI/`	`PWGui`: Graphical User Interface

2.2 Prerequisites

To install QUANTUM ESPRESSO from source, you need first of all a minimal Unix environment: basically, a command shell (e.g., bash or tcsh) and the utilities `make`, `awk`, `sed`. MS-Windows users need to have Cygwin (a UNIX environment which runs under Windows) installed: see `http://www.cygwin.com/`. Note that the scripts contained in the distribution assume that the local language is set to the standard, i.e. "C"; other settings may break them. Use `export LC_ALL=C` (sh/bash) or `setenv LC_ALL C` (csh/tcsh) to prevent any problem when running scripts (including installation scripts).

Second, you need C and Fortran-95 compilers. For parallel execution, you will also need MPI libraries and a "parallel" (i.e. MPI-aware) compiler. For massively parallel machines, or for simple multicore parallelization, an OpenMP-aware compiler and libraries are also required.

Big machines with specialized hardware (e.g. IBM SP, CRAY, etc) typically have a Fortran-95 compiler with MPI and OpenMP libraries bundled with the software. Workstations or

"commodity" machines, using PC hardware, may or may not have the needed software. If not, you need either to buy a commercial product (e.g Portland) or to install an open-source compiler like gfortran or g95. Note that several commercial compilers are available free of charge under some license for academic or personal usage (e.g. Intel, Sun).

2.3 `configure`

To install the QUANTUM ESPRESSO source package, run the `configure` script. This is actually a wrapper to the true `configure`, located in the `install/` subdirectory. `configure` will (try to) detect compilers and libraries available on your machine, and set up things accordingly. Presently it is expected to work on most Linux 32- and 64-bit PCs (all Intel and AMD CPUs) and PC clusters, SGI Altix, IBM SP machines, NEC SX, Cray XT machines, Mac OS X, MS-Windows PCs. It may work with some assistance also on other architectures (see below).

Instructions for the impatient:

```
cd espresso-X.Y.Z/
./configure
 make all
```

Symlinks to executable programs will be placed in the `bin/` subdirectory. Note that both C and Fortran compilers must be in your execution path, as specified in the PATH environment variable.

Additional instructions for CRAY XT, NEC SX, Linux PowerPC machines with xlf:

```
./configure ARCH=crayxt4
./configure ARCH=necsx
./configure ARCH=ppc64-mn
```

`configure` Generates the following files:

`install/make.sys`	compilation rules and flags (used by `Makefile`)
`install/configure.msg`	a report of the configuration run (not needed for compilation)
`install/config.log`	detailed log of the configuration run (may be needed for debugging)
`include/fft_defs.h`	defines fortran variable for C pointer (used only by FFTW)
`include/c_defs.h`	defines C to fortran calling convention and a few more definitions used by C files

NOTA BENE: unlike previous versions, `configure` no longer runs the `makedeps.sh` shell script that updates dependencies. If you modify the sources, run `./install/makedeps.sh` or type `make depend` to update files `make.depend` in the various subdirectories.

You should always be able to compile the QUANTUM ESPRESSO suite of programs without having to edit any of the generated files. However you may have to tune `configure` by specifying appropriate environment variables and/or command-line options. Usually the tricky part is to get external libraries recognized and used: see Sec.2.4 for details and hints.

Environment variables may be set in any of these ways:

```
export VARIABLE=value; ./configure            # sh, bash, ksh
setenv VARIABLE value; ./configure            # csh, tcsh
./configure VARIABLE=value                    # any shell
```

Some environment variables that are relevant to `configure` are:

`ARCH`	label identifying the machine type (see below)
`F90, F77, CC`	names of Fortran 95, Fortran 77, and C compilers
`MPIF90`	name of parallel Fortran 95 compiler (using MPI)
`CPP`	source file preprocessor (defaults to $CC -E)
`LD`	linker (defaults to $MPIF90)
`(C,F,F90,CPP,LD)FLAGS`	compilation/preprocessor/loader flags
`LIBDIRS`	extra directories where to search for libraries

For example, the following command line:

```
./configure MPIF90=mpf90 FFLAGS="-O2 -assume byterecl" \
            CC=gcc CFLAGS=-O3 LDFLAGS=-static
```

instructs `configure`to use `mpf90` as Fortran 95 compiler with flags `-O2 -assume byterecl`, `gcc` as C compiler with flags `-O3`, and to link with flag `-static`. Note that the value of `FFLAGS` must be quoted, because it contains spaces. NOTA BENE: do not pass compiler names with the leading path included. `F90=f90xyz` is ok, `F90=/path/to/f90xyz` is not. Do not use environmental variables with `configure` unless they are needed! try `configure` with no options as a first step.

If your machine type is unknown to `configure`, you may use the `ARCH` variable to suggest an architecture among supported ones. Some large parallel machines using a front-end (e.g. Cray XT) will actually need it, or else `configure` will correctly recognize the front-end but not the specialized compilation environment of those machines. In some cases, cross-compilation requires to specify the target machine with the `--host` option. This feature has not been extensively tested, but we had at least one successful report (compilation for NEC SX6 on a PC). Currently supported architectures are:

`ia32`	Intel 32-bit machines (x86) running Linux
`ia64`	Intel 64-bit (Itanium) running Linux
`x86_64`	Intel and AMD 64-bit running Linux - see note below
`aix`	IBM AIX machines
`solaris`	PC's running SUN-Solaris
`sparc`	Sun SPARC machines
`crayxt4`	Cray XT4/5 machines
`macppc`	Apple PowerPC machines running Mac OS X
`mac686`	Apple Intel machines running Mac OS X
`cygwin`	MS-Windows PCs with Cygwin
`necsx`	NEC SX-6 and SX-8 machines
`ppc64`	Linux PowerPC machines, 64 bits
`ppc64-mn`	as above, with IBM xlf compiler

Note: `x86_64` replaces `amd64` since v.4.1. Cray Unicos machines, SGI machines with MIPS architecture, HP-Compaq Alphas are no longer supported since v.4.2.0. Finally, `configure` recognizes the following command-line options:

`--enable-parallel`	compile for parallel execution if possible (default: yes)
`--enable-openmp`	compile for openmp execution if possible (default: no)
`--enable-shared`	use shared libraries if available (default: yes)
`--disable-wrappers`	disable C to fortran wrapper check (default: enabled)
`--enable-signals`	enable signal trapping (default: disabled)

and the following optional packages:

`--with-internal-blas`	compile with internal BLAS (default: no)
`--with-internal-lapack`	compile with internal LAPACK (default: no)
`--with-scalapack`	use ScaLAPACK if available (default: yes)

If you want to modify the `configure` script (advanced users only!), see the Developer Manual.

2.3.1 Manual configuration

If `configure` stops before the end, and you don't find a way to fix it, you have to write working `make.sys`, `include/fft_defs.h` and `include/c_defs.h` files. For the latter two files, follow the explanations in `include/defs.h.README`.

If `configure` has run till the end, you should need only to edit `make.sys`. A few templates (each for a different machine type) are provided in the `install/` directory: they have names of the form `Make.`*system*, where *system* is a string identifying the architecture and compiler. The template used by `configure` is also found there as `make.sys.in` and contains explanations of the meaning of the various variables. The difficult part will be to locate libraries. Note that you will need to select appropriate preprocessing flags in conjunction with the desired or available libraries (e.g. you need to add `-D__FFTW`) to `DFLAGS` if you want to link internal FFTW). For a correct choice of preprocessing flags, refer to the documentation in `include/defs.h.README`.

NOTA BENE: If you change any settings (e.g. preprocessing, compilation flags) after a previous (successful or failed) compilation, you must run `make clean` before recompiling, unless you know exactly which routines are affected by the changed settings and how to force their recompilation.

2.4 Libraries

Quantum ESPRESSO makes use of the following external libraries:

- BLAS (`http://www.netlib.org/blas/`) and
- LAPACK (`http://www.netlib.org/lapack/`) for linear algebra
- FFTW (`http://www.fftw.org/`) for Fast Fourier Transforms

A copy of the needed routines is provided with the distribution. However, when available, optimized vendor-specific libraries should be used: this often yields huge performance gains.

BLAS and LAPACK Quantum ESPRESSO can use the following architecture-specific replacements for BLAS and LAPACK:

MKL for Intel Linux PCs
ACML for AMD Linux PCs
ESSL for IBM machines
SCSL for SGI Altix
SUNperf for Sun

If none of these is available, we suggest that you use the optimized ATLAS library: see `http://math-atlas.sourceforge.net/`. Note that ATLAS is not a complete replacement for LAPACK: it contains all of the BLAS, plus the LU code, plus the full storage Cholesky code. Follow the instructions in the ATLAS distributions to produce a full LAPACK replacement.

Sergei Lisenkov reported success and good performances with optimized BLAS by Kazushige Goto. They can be freely downloaded, but not redistributed. See the "GotoBLAS2" item at `http://www.tacc.utexas.edu/tacc-projects/`.

FFT QUANTUM ESPRESSO has an internal copy of an old FFTW version, and it can use the following vendor-specific FFT libraries:

IBM ESSL
SGI SCSL
SUN sunperf
NEC ASL
AMD ACML

`configure` will first search for vendor-specific FFT libraries; if none is found, it will search for an external FFTW v.3 library; if none is found, it will fall back to the internal copy of FFTW.

If you have recent versions of MKL installed, you may try the FFTW interface provided with MKL. You will have to compile them (only sources are distributed with the MKL library) and to modify file `make.sys` accordingly (MKL must be linked *after* the FFTW-MKL interface)

MPI libraries MPI libraries are usually needed for parallel execution (unless you are happy with OpenMP multicore parallelization). In well-configured machines, `configure` should find the appropriate parallel compiler for you, and this should find the appropriate libraries. Since often this doesn't happen, especially on PC clusters, see Sec.2.7.5.

Other libraries QUANTUM ESPRESSO can use the MASS vector math library from IBM, if available (only on AIX).

2.4.1 If optimized libraries are not found

The `configure` script attempts to find optimized libraries, but may fail if they have been installed in non-standard places. You should examine the final value of `BLAS_LIBS, LAPACK_LIBS, FFT_LIBS, MPI_LIBS` (if needed), `MASS_LIBS` (IBM only), either in the output of `configure` or in the generated `make.sys`, to check whether it found all the libraries that you intend to use.

If some library was not found, you can specify a list of directories to search in the environment variable `LIBDIRS`, and rerun `configure`; directories in the list must be separated by spaces. For example:

```
./configure LIBDIRS="/opt/intel/mkl70/lib/32 /usr/lib/math"
```

If this still fails, you may set some or all of the `*_LIBS` variables manually and retry. For example:

```
./configure BLAS_LIBS="-L/usr/lib/math -lf77blas -latlas_sse"
```

Beware that in this case, `configure` will blindly accept the specified value, and won't do any extra search.

2.5 Compilation

There are a few adjustable parameters in `Modules/parameters.f90`. The present values will work for most cases. All other variables are dynamically allocated: you do not need to recompile your code for a different system.

At your option, you may compile the complete QUANTUM ESPRESSO suite of programs (with `make all`), or only some specific programs.

`make` with no arguments yields a list of valid compilation targets. Here is a list:

- `make pw` produces `PW/pw.x`
  `pw.x` calculates electronic structure, structural optimization, molecular dynamics, barriers with NEB.

- `make ph` produces the following codes in `PH/` for phonon calculations:
  - `ph.x`: Calculates phonon frequencies and displacement patterns, dielectric tensors, effective charges (uses data produced by `pw.x`).
  - `dynmat.x`: applies various kinds of Acoustic Sum Rule (ASR), calculates LO-TO splitting at $\mathbf{q} = 0$ in insulators, IR and Raman cross sections (if the coefficients have been properly calculated), from the dynamical matrix produced by `ph.x`
  - `q2r.x`: calculates Interatomic Force Constants (IFC) in real space from dynamical matrices produced by `ph.x` on a regular $\mathbf{q}$-grid
  - `matdyn.x`: produces phonon frequencies at a generic wave vector using the IFC file calculated by `q2r.x`; may also calculate phonon DOS, the electron-phonon coefficient λ, the function $\alpha^2F(\omega)$
  - `lambda.x`: also calculates λ and $\alpha^2F(\omega)$, plus T_c for superconductivity using the McMillan formula

- `make d3` produces `D3/d3.x`: calculates anharmonic phonon lifetimes (third-order derivatives of the energy), using data produced by `pw.x` and `ph.x` (USPP and PAW not supported).

- `make gamma` produces `Gamma/phcg.x`: a version of `ph.x` that calculates phonons at $\mathbf{q} = 0$ using conjugate-gradient minimization of the density functional expanded to second-order. Only the Γ ($\mathbf{k} = 0$) point is used for Brillouin zone integration. It is faster and takes less memory than `ph.x`, but does not support USPP and PAW. tem `make pp` produces several codes for data postprocessing, in `PP/` (see list below).

- `make tools` produces several utility programs in `pwtools/` (see list below).

- `make pwcond` produces `PWCOND/pwcond.x` for ballistic conductance calculations.

- `make pwall` produces all of the above.

- `make ld1` produces code `atomic/ld1.x` for pseudopotential generation (see specific documentation in `atomic_doc/`).

- `make upf` produces utilities for pseudopotential conversion in directory `upftools/`.

- `make cp` produces the Car-Parrinello code `CPV/cp.x` and the postprocessing code `CPV/cppp.x`.

- `make all` produces all of the above.

For the setup of the GUI, refer to the `PWgui-X.Y.Z /INSTALL` file, where X.Y.Z stands for the version number of the GUI (should be the same as the general version number). If you are using the CVS sources, see the `GUI/README` file instead.

The codes for data postprocessing in `PP/` are:

- `pp.x` extracts the specified data from files produced by `pw.x`, prepares data for plotting by writing them into formats that can be read by several plotting programs.
- `bands.x` extracts and reorders eigenvalues from files produced by `pw.x` for band structure plotting
- `projwfc.x` calculates projections of wavefunction over atomic orbitals, performs Löwdin population analysis and calculates projected density of states. These can be summed using auxiliary code `sumpdos.x`.
- `plotrho.x` produces PostScript 2-d contour plots
- `plotband.x` reads the output of `bands.x`, produces PostScript plots of the band structure
- `average.x` calculates planar averages of quantities produced by `pp.x` (potentials, charge, magnetization densities,...)
- `dos.x` calculates electronic Density of States (DOS)
- `epsilon.x` calculates RPA frequency-dependent complex dielectric function
- `pw2wannier.x`: interface with Wannier90 package
- `wannier_ham.x`: generate a model Hamiltonian in Wannier functions basis
- `pmw.x` generates Poor Man's Wannier functions, to be used in DFT+U calculations
- `pw2casino.x`: interface with CASINO code for Quantum Monte Carlo calculation (`http://www.tcm.phy.cam.ac.uk/~mdt26/casino.html`). See the header of `PP/pw2casino.f90` for instructions on how to use it.

Note about Bader's analysis: on `http://theory.cm.utexas.edu/bader/` one can find a software that performs Bader's analysis starting from charge on a regular grid. The required "cube" format can be produced by Quantum ESPRESSO using `pp.x` (info by G. Lapenna who has successfully used this technique). This code should perform decomposition into Voronoi polyhedra as well, in place of obsolete code `voronoy.x` (removed from distribution since v.4.2).

The utility programs in `pwtools/` are:

- `dist.x` calculates distances and angles between atoms in a cell, taking into account periodicity
- `ev.x` fits energy-vs-volume data to an equation of state
- `kpoints.x` produces lists of k-points

- `pwi2xsf.sh`, `pwo2xsf.sh` process respectively input and output files (not data files!) for `pw.x` and produce an XSF-formatted file suitable for plotting with XCrySDen, a powerful crystalline and molecular structure visualization program (`http://www.xcrysden.org/`). BEWARE: the `pwi2xsf.sh` shell script requires the `pwi2xsf.x` executables to be located somewhere in your PATH.

- `band_plot.x`: undocumented and possibly obsolete

- `bs.awk`, `mv.awk` are scripts that process the output of `pw.x` (not data files!). Usage:

```
awk -f bs.awk < my-pw-file > myfile.bs
awk -f mv.awk < my-pw-file > myfile.mv
```

  The files so produced are suitable for use with `xbs`, a very simple X-windows utility to display molecules, available at:
  `http://www.ccl.net/cca/software/X-WINDOW/xbsa/README.shtml`

- `path_int.sh`/ `path_int.x`: utility to generate, starting from a path (a set of images), a new one with a different number of images. The initial and final points of the new path can differ from those in the original one. Useful for NEB calculations.

- `kvecs_FS.x`, `bands_FS.x`: utilities for Fermi Surface plotting using XCrySDen

Other utilities `VdW/` contains the sources for the calculation of the finite (imaginary) frequency molecular polarizability using the approximated Thomas-Fermi + von Weizäcker scheme, contributed by H.-V. Nguyen (Sissa and Hanoi University). Compile with `make vdw`, executables in `VdW/vdw.x`, no documentation yet, but an example in `examples/example34`.

2.6 Running examples

As a final check that compilation was successful, you may want to run some or all of the examples. You should first of all ensure that you have downloaded and correctly unpacked the package containing examples (since v.4.1 in a separate package). There are two different types of examples:

- automated tests (in directories `tests/` and `cptests/`). Quick and exhaustive, but not meant to be realistic, implemented only for `pw.x` and `cp.x`.

- examples (in directory `examples/`). Cover many more programs and features of the QUANTUM ESPRESSO distribution, but they require manual inspection of the results.

Let us first consider the tests. Automated tests for `pw.x` are in directory `tests/`. File `tests/README` contains a list of what is tested. To run tests, follow the directions in the header if file `check_pw.x.j`, edit variables PARA_PREFIX, PARA_POSTFIX if needed (see below). Same for `cp.x`, this time in directory `cptests/`.

Let us now consder examples. A list of examples and of what each example does is contained in `examples/README`. For details, see the `README` file in each example's directory. If you find that any relevant feature isn't being tested, please contact us (or even better, write and send us a new example yourself !).

To run the examples, you should follow this procedure:

1. Go to the `examples/` directory and edit the `environment_variables` file, setting the following variables as needed:

   BIN_DIR: directory where executables reside
   PSEUDO_DIR: directory where pseudopotential files reside
   TMP_DIR: directory to be used as temporary storage area

   The default values of BIN_DIR and PSEUDO_DIR should be fine, unless you have installed things in nonstandard places. TMP_DIR must be a directory where you have read and write access to, with enough available space to host the temporary files produced by the example runs, and possibly offering high I/O performance (i.e., don't use an NFS-mounted directory). NOTA BENE: do not use a directory containing other data, the examples wil clean it!

2. If you have compiled the parallel version of QUANTUM ESPRESSO (this is the default if parallel libraries are detected), you will usually have to specify a driver program (such as `mpirun` or `mpiexec`) and the number of processors: see Sec.3.2 for details. In order to do that, edit again the `environment_variables` file and set the PARA_PREFIX and PARA_POSTFIX variables as needed. Parallel executables will be run by a command like this:

   ```
   $PARA_PREFIX pw.x $PARA_POSTFIX < file.in > file.out
   ```

   For example, if the command line is like this (as for an IBM SP):

   ```
   poe pw.x -procs 4 < file.in > file.out
   ```

   you should set PARA_PREFIX="poe", PARA_POSTFIX="-procs 4". Furthermore, if your machine does not support interactive use, you must run the commands specified below through the batch queuing system installed on that machine. Ask your system administrator for instructions.

3. To run a single example, go to the corresponding directory (e.g. `example/example01`) and execute:

   ```
   ./run_example
   ```

   This will create a subdirectory results, containing the input and output files generated by the calculation. Some examples take only a few seconds to run, while others may require several minutes depending on your system. To run all the examples in one go, execute:

   ```
   ./run_all_examples
   ```

   from the examples directory. On a single-processor machine, this typically takes a few hours. The `make_clean` script cleans the examples tree, by removing all the results subdirectories. However, if additional subdirectories have been created, they aren't deleted.

4. In each example's directory, the `reference/` subdirectory contains verified output files, that you can check your results against. They were generated on a Linux PC using the Intel compiler. On different architectures the precise numbers could be slightly different, in particular if different FFT dimensions are automatically selected. For this reason, a plain diff of your results against the reference data doesn't work, or at least, it requires human inspection of the results.

2.7 Installation tricks and problems

2.7.1 All architectures

Working Fortran-95 and C compilers are needed in order to compile QUANTUM ESPRESSO. Most "Fortran-90" compilers actually implement the Fortran-95 standard, but older versions may not be Fortran-95 compliant. Moreover, C and Fortran compilers must be in your PATH. If `configure` says that you have no working compiler, well, you have no working compiler, at least not in your PATH, and not among those recognized by `configure`.

If you get *Compiler Internal Error*' or similar messages: your compiler version is buggy. Try to lower the optimization level, or to remove optimization just for the routine that has problems. If it doesn't work, or if you experience weird problems at run time, try to install patches for your version of the compiler (most vendors release at least a few patches for free), or to upgrade to a more recent compiler version.

If you get error messages at the loading phase that look like *file XYZ.o: unknown / not recognized/ invalid / wrong file type / file format / module version*, one of the following things have happened:

1. you have leftover object files from a compilation with another compiler: run `make clean` and recompile.

2. `make` did not stop at the first compilation error (it may happen in some software configurations). Remove the file *.o that triggers the error message, recompile, look for a compilation error.

If many symbols are missing in the loading phase: you did not specify the location of all needed libraries (LAPACK, BLAS, FFTW, machine-specific optimized libraries), in the needed order. If only symbols from `clib/` are missing, verify that you have the correct C-to-Fortran bindings, defined in `include/c_defs.h`. Note that QUANTUM ESPRESSO is self-contained (with the exception of MPI libraries for parallel compilation): if system libraries are missing, the problem is in your compiler/library combination or in their usage, not in QUANTUM ESPRESSO.

If you get mysterious errors in the provided tests and examples: your compiler, or your mathematical libraries, or MPI libraries, or a combination thereof, is very likely buggy. Although the presence of subtle bugs in QUANTUM ESPRESSO that are not revealed during the testing phase can never be ruled out, it is very unlikely that this happens on the provided tests and examples.

2.7.2 Cray XT machines

Use `./configure ARCH=crayxt4` or else `configure`will not recognize the Cray-specific software environment. Older Cray machines: T3D, T3E, X1, are no longer supported.

2.7.3 IBM AIX

On IBM machines with ESSL libraries installed, there is a potential conflict between a few LAPACK routines that are also part of ESSL, but with a different calling sequence. The appearance of run-time errors like *ON ENTRY TO ZHPEV PARAMETER NUMBER 1 HAD AN ILLEGAL VALUE* is a signal that you are calling the bad routine. If you have defined `-D__ESSL` you should load ESSL before LAPACK: see variable LAPACK_LIBS in make.sys.

2.7.4 Linux PC

Both AMD and Intel CPUs, 32-bit and 64-bit, are supported and work, either in 32-bit emulation and in 64-bit mode. 64-bit executables can address a much larger memory space than 32-bit executable, but there is no gain in speed. Beware: the default integer type for 64-bit machine is typically 32-bit long. You should be able to use 64-bit integers as well, but it will not give you any advantage and you may run into trouble.

Currently the following compilers are supported by `configure`: Intel (ifort), Portland (pgf90), g95, gfortran, Pathscale (pathf95), Sun Studio (sunf95), AMD Open64 (openf95). The ordering approximately reflects the quality of support. Both Intel MKL and AMD acml mathematical libraries are supported. Some combinations of compilers and of libraries may however require manual editing of `make.sys`.

It is usually convenient to create semi-statically linked executables (with only libc, libm, libpthread dynamically linked). If you want to produce a binary that runs on different machines, compile it on the oldest machine you have (i.e. the one with the oldest version of the operating system).

If you get errors like *IPO Error: unresolved : __svml_cos2* at the linking stage, your compiler is optimized to use the SSE version of sine, cosine etc. contained in the SVML library. Append `-lsvml` to the list of libraries in your `make.sys` file (info by Axel Kohlmeyer, oct.2007).

Linux PCs with Portland compiler (pgf90) QUANTUM ESPRESSO does not work reliably, or not at all, with many old versions (< 6.1) of the Portland Group compiler (pgf90). Use the latest version of each release of the compiler, with patches if available (see the Portland Group web site, `http://www.pgroup.com/`).

Linux PCs with Pathscale compiler Version 2.99 of the Pathscale EKO compiler (web site `http://www.pathscale.com/`) works and is recognized by `configure`, but the preprocessing command, `pathcc -E`, causes a mysterious error in compilation of iotk and should be replaced by

```
/lib/cpp -P --traditional
```

The MVAPICH parallel environment with Pathscale compilers also works. (info by Paolo Giannozzi, July 2008)

Linux PCs with gfortran gfortran v.4.1.2 and later are supported. Earlier gfortran versions used to produce nonfunctional phonon executables (segmentation faults and the like), but more recent versions should be fine.

If you experience problems in reading files produced by previous versions of QUANTUM ESPRESSO: "gfortran used 64-bit record markers to allow writing of records larger than 2

GB. Before with 32-bit record markers only records <2GB could be written. However, this caused problems with older files and inter-compiler operability. This was solved in GCC 4.2 by using 32-bit record markers but such that one can still store >2GB records (following the implementation of Intel). Thus this issue should be gone. See 4.2 release notes (item "Fortran") at `http://gcc.gnu.org/gcc-4.2/changes.html`." (Info by Tobias Burnus, March 2010).

"Using gfortran v.4.4 (after May 27, 2009) and 4.5 (after May 5, 2009) can produce wrong results, unless the environment variable GFORTRAN_UNBUFFERED_ALL=1 is set. Newer 4.4/4.5 versions (later than April 2010) should be OK. See `http://gcc.gnu.org/bugzilla/show_bug.cgi?id=43551`." (Info by Tobias Burnus, March 2010).

Linux PCs with g95 g95 v.0.91 and later versions (`http://www.g95.org`) work. The executables thet produce are however slower (let us say 20% or so) that those produced by gfortran, which in turn are slower (by another 20% or so) than those produced by ifort.

Linux PCs with Sun Studio compiler "The Sun Studio compiler, sunf95, is free (web site: `http://developers.sun.com/sunstudio/` and comes with a set of algebra libraries that can be used in place of the slow built-in libraries. It also supports OpenMP, which g95 does not. On the other hand, it is a pain to compile MPI with it. Furthermore the most recent version has a terrible bug that totally miscompiles the iotk input/output library (you'll have to compile it with reduced optimization)." (info by Lorenzo Paulatto, March 2010).

Linux PCs with AMD Open64 suite The AMD Open64 compiler suite, openf95 (web site: `http://developer.amd.com/cpu/open64/pages/default.aspx`) can be freely downloaded from the AMD site. It is recognized by `configure` but little tested. It sort of works but it fails to pass several tests. (info by Paolo Giannozzi, March 2010).

Linux PCs with Intel compiler (ifort) The Intel compiler, ifort, is available for free for personal usage (`http://software.intel.com/`) It seem to produce the faster executables, at least on Intel CPUs, but not all versions work as expected. ifort versions < 9.1 are not recommended, due to the presence of subtle and insidious bugs. In case of trouble, update your version with the most recent patches, available via Intel Premier support (registration free of charge for Linux): `http://software.intel.com/en-us/articles/intel-software-developer-support`.

If `configure` doesn't find the compiler, or if you get *Error loading shared libraries* at run time, you may have forgotten to execute the script that sets up the correct PATH and library path. Unless your system manager has done this for you, you should execute the appropriate script – located in the directory containing the compiler executable – in your initialization files. Consult the documentation provided by Intel.

The warning: *feupdateenv is not implemented and will always fail*, showing up in recent versions, can be safely ignored. Since each major release of ifort differs a lot from the previous one. compiled objects from different releases may be incompatible and should not be mixed.

ifort v.11: Segmentation faults were reported for the combination ifort 11.0.081, MKL 10.1.1.019, OpenMP 1.3.3. The problem disappeared with ifort 11.1.056 and MKL 10.2.2.025 (Carlo Nervi, Oct. 2009).

ifort v.10: on 64-bit AMD CPUs, at least some versions of ifort 10.1 miscompile subroutine `write_rho_xml` in `Module/xml_io_base.f90` with -O2 optimization. Using -O1 instead solves

the problem (info by Carlo Cavazzoni, March 2008).

"The intel compiler version 10.1.008 miscompiles a lot of codes (I have proof for CP2K and CPMD) and needs to be updated in any case" (info by Axel Kohlmeyer, May 2008).

ifort v.9: The latest (July 2006) 32-bit version of ifort 9.1 works. Earlier versions yielded *Compiler Internal Error*.

Linux PCs with MKL libraries On Intel CPUs it is very convenient to use Intel MKL libraries. They can be also used for AMD CPU, selecting the appropriate machine-optimized libraries, and also together with non-Intel compilers. Note however that recent versions of MKL (10.2 and following) do not perform well on AMD machines.

`configure` should recognize properly installed MKL libraries. By default the non-threaded version of MKL is linked, unless option `configure --with-openmp` is specified. In case of trouble, refer to the following web page to find the correct way to link MKL:
`http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/`.

MKL contains optimized FFT routines and a FFTW interface, to be separately compiled. For 64-bit Intel Core2 processors, they are slightly faster than FFTW (MKL v.10, FFTW v.3 fortran interface, reported by P. Giannozzi, November 2008).

For parallel (MPI) execution on multiprocessor (SMP) machines, set the environmental variable OMP_NUM_THREADS to 1 unless you know what you are doing. See Sec.3 for more info on this and on the difference between MPI and OpenMP parallelization.

Linux PCs with ACML libraries For AMD CPUs, especially recent ones, you may find convenient to link AMD acml libraries (can be freely downloaded from AMD web site). `configure` should recognize properly installed acml libraries, together with the compilers most frequently used on AMD systems: pgf90, pathscale, openf95, sunf95.

2.7.5 Linux PC clusters with MPI

PC clusters running some version of MPI are a very popular computational platform nowadays. Quantum ESPRESSO is known to work with at least two of the major MPI implementations (MPICH, LAM-MPI), plus with the newer MPICH2 and OpenMPI implementation. `configure` should automatically recognize a properly installed parallel environment and prepare for parallel compilation. Unfortunately this not always happens. In fact:

- `configure` tries to locate a parallel compiler in a logical place with a logical name, but if it has a strange names or it is located in a strange location, you will have to instruct `configure` to find it. Note that in many PC clusters (Beowulf), there is no parallel Fortran-95 compiler in default installations: you have to configure an appropriate script, such as mpif90.

- `configure` tries to locate libraries (both mathematical and parallel libraries) in the usual places with usual names, but if they have strange names or strange locations, you will have to rename/move them, or to instruct `configure` to find them. If MPI libraries are not found, parallel compilation is disabled.

- `configure` tests that the compiler and the libraries are compatible (i.e. the compiler may link the libraries without conflicts and without missing symbols). If they aren't and the compilation fail, `configure` will revert to serial compilation.

Apart from such problems, QUANTUM ESPRESSO compiles and works on all non-buggy, properly configured hardware and software combinations. You may have to recompile MPI libraries: not all MPI installations contain support for the fortran-90 compiler of your choice (or for any fortran-90 compiler at all!). Useful step-by-step instructions for MPI compilation can be found in the following post by Javier Antonio Montoya:
`http://www.democritos.it/pipermail/pw_forum/2008April/008818.htm`.

If QUANTUM ESPRESSO does not work for some reason on a PC cluster, try first if it works in serial execution. A frequent problem with parallel execution is that QUANTUM ESPRESSO does not read from standard input, due to the configuration of MPI libraries: see Sec.3.2.

If you are dissatisfied with the performances in parallel execution, see Sec.3 and in particular Sec.9.4. See also the following post from Axel Kohlmeyer:
`http://www.democritos.it/pipermail/pw_forum/2008-April/008796.html`

2.7.6 Intel Mac OS X

Newer Mac OS-X machines (10.4 and later) with Intel CPUs are supported by `configure`, with gcc4+g95, gfortran, and the Intel compiler ifort with MKL libraries. Parallel compilation with OpenMPI also works.

Intel Mac OS X with ifort "Uninstall darwin ports, fink and developer tools. The presence of all of those at the same time generates many spooky events in the compilation procedure. I installed just the developer tools from apple, the intel fortran compiler and everything went on great" (Info by Riccardo Sabatini, Nov. 2007)

Intel Mac OS X 10.4 with g95 and gfortran An updated version of Developer Tools (XCode 2.4.1 or 2.5), that can be downloaded from Apple, may be needed. Some tests fails with mysterious errors, that disappear if fortran BLAS are linked instead of system Atlas libraries. Use:

```
BLAS_LIBS_SWITCH = internal
BLAS_LIBS      = /path/to/espresso/BLAS/blas.a -latlas
```

(Info by Paolo Giannozzi, jan.2008, updated April 2010)

Detailed installation instructions for Mac OS X 10.6 (Instructions for 10.6.3 by Osman Baris Malcioglu, tested as of May 2010)

Summary for the hasty:

GNU: Install macports compilers, Install MPI environment, Configure QUANTUM ESPRESSO using

```
./configure CC=gcc-mp-4.3 CPP=cpp-mp-4.3 CXX=g++-mp-4.3 F77=g95 FC=g95
```

Intel: Use Version ¿11.1.088 Use 32 bit compilers Install MPI environment, install macports provided cpp (optional) Configure QUANTUM ESPRESSO using

```
./configure CC=icc CXX=icpc F77=ifort F90=ifort FC=ifort CPP=cpp-mp-4.3
```

Compilation with GNU compilers:

The following instructions use macports version of gnu compilers due to some issues in mixing gnu supplied fortran compilers with apple modified gnu compiler collection. For more information regarding macports please refer to: `http://www.macports.org/`

First install necessary compilers from macports

```
port install gcc43
port install g95
```

The apple supplied MPI environment has to be overridden since there is a new set of compilers now (and Apple provided mpif90 is just an empty placeholder since Apple does not provide fortran compilers). I have used OpenMPI for this case. Recommended minimum configuration line is:

```
./configure CC=gcc-mp-4.3 CPP=cpp-mp-4.3 CXX=g++-mp-4.3 F77=g95 FC=g95
```

of course, installation directory should be set accordingly if a multiple compiler environment is desired. The default installation directory of OpenMPI overwrites apple supplied MPI permanently!

Next step is QUANTUM ESPRESSO itself. Sadly, the Apple supplied optimized BLAS/LAPACK libraries tend to misbehave under different tests, and it is much safer to use internal libraries. The minimum recommended configuration line is (presuming the environment is set correctly):

```
./configure CC=gcc-mp-4.3 CXX=g++-mp-4.3 F77=g95 F90=g95 FC=g95 CPP=cpp-mp-4.3 --wi
```

Compilation with Intel compilers: Newer versions of Intel compiler (¿11.1.067) support Mac OS X 10.6, and furthermore they are bundled with intel MKL. 32 bit binaries obtained using 11.1.088 are tested and no problems have been encountered so far. Sadly, as of 11.1.088 the 64 bit binary misbehave under some tests. Any attempt to compile 64 bit binary using ¡11.1.088 will result in very strange compilation errors.

Like the previous section, I would recommend installing macports compiler suite.

First, make sure that you are using the 32 bit version of the compilers, i.e.

```
. /opt/intel/Compiler/11.1/088/bin/ifortvars.sh ia32

. /opt/intel/Compiler/11.1/088/bin/iccvars.sh ia32
```

will set the environment for 32 bit compilation in my case.

Then, the MPI environment has to be set up for Intel compilers similar to previous section.

The recommended configuration line for QUANTUM ESPRESSO is:

```
./configure CC=icc CXX=icpc F77=ifort F90=ifort FC=ifort CPP=cpp-mp-4.3
```

MKL libraries will be detected automatically if they are in their default locations. Otherwise, mklvars32 has to be sourced before the configuration script.

Security issues: MacOs 10.6 comes with a disabled firewall. Preparing a ipfw based firewall is recommended. Open source and free GUIs such as "WaterRoof" and "NoobProof" are available that may help you in the process.

2.7.7 SGI, Alpha

SGI Mips machines (e.g. Origin) and HP-Compaq Alpha machines are no longer supported since v.4.2.

3 Parallelism

3.1 Understanding Parallelism

Two different parallelization paradigms are currently implemented in QUANTUM ESPRESSO:

1. *Message-Passing (MPI).* A copy of the executable runs on each CPU; each copy lives in a different world, with its own private set of data, and communicates with other executables only via calls to MPI libraries. MPI parallelization requires compilation for parallel execution, linking with MPI libraries, execution using a launcher program (depending upon the specific machine). The number of CPUs used is specified at run-time either as an option to the launcher or by the batch queue system.

2. *OpenMP.* A single executable spawn subprocesses (threads) that perform in parallel specific tasks. OpenMP can be implemented via compiler directives (*explicit* OpenMP) or via *multithreading* libraries (*library* OpenMP). Explicit OpenMP require compilation for OpenMP execution; library OpenMP requires only linking to a multithreading version of mathematical libraries, e.g.: ESSLSMP, ACML_MP, MKL (the latter is natively multithreading). The number of threads is specified at run-time in the environment variable OMP_NUM_THREADS.

MPI is the well-established, general-purpose parallelization. In QUANTUM ESPRESSO several parallelization levels, specified at run-time via command-line options to the executable, are implemented with MPI. This is your first choice for execution a parallel machine.

Library OpenMP is a low-effort parallelization suitable for multicore CPUs. Its effectiveness relies upon the quality of the multithreading libraries and the availability of multithreading FFTs. If you are using MKL,[1] you may want to select FFTW3 (set `CPPFLAGS=-D__FFTW3...` in `make.sys`) and to link with the MKL interface to FFTW3. You will get a decent speedup ($\sim 25\%$) on two cores.

Explicit OpenMP is a very recent addition, still at an experimental stage, devised to increase scalability on large multicore parallel machines. Explicit OpenMP is devised to be run together with MPI and also together with multithreaded libraries. BEWARE: you have to be VERY careful to prevent conflicts between the various kinds of parallelization. If you don't know how to run MPI processes and OpenMP threads in a controlled manner, forget about mixed OpenMP-MPI parallelization.

3.2 Running on parallel machines

Parallel execution is strongly system- and installation-dependent. Typically one has to specify:

1. a launcher program (not always needed), such as `poe`, `mpirun`, `mpiexec`, with the appropriate options (if any);

2. the number of processors, typically as an option to the launcher program, but in some cases to be specified after the name of the program to be executed;

3. the program to be executed, with the proper path if needed: for instance, `pw.x`, or `./pw.x`, or `$HOME/bin/pw.x`, or whatever applies;

[1] Beware: MKL v.10.2.2 has a buggy `dsyev` yielding wrong results with more than one thread; fixed in v.10.2.4

4. other QUANTUM ESPRESSO-specific parallelization options, to be read and interpreted by the running code:
   - the number of "images" used by NEB calculations;
   - the number of "pools" into which processors are to be grouped (`pw.x` only);
   - the number of "task groups" into which processors are to be grouped;
   - the number of processors performing iterative diagonalization (for `pw.x`) or orthonormalization (for `cp.x`).

Items 1) and 2) are machine- and installation-dependent, and may be different for interactive and batch execution. Note that large parallel machines are often configured so as to disallow interactive execution: if in doubt, ask your system administrator. Item 3) also depend on your specific configuration (shell, execution path, etc). Item 4) is optional but may be important: see the following section for the meaning of the various options.

For illustration, here is how to run `pw.x` on 16 processors partitioned into 8 pools (2 processors each), for several typical cases.

IBM SP machines, batch:

```
pw.x -npool 8 < input
```

This should also work interactively, with environment variables NPROC set to 16, MP_HOSTFILE set to the file containing a list of processors.

IBM SP machines, interactive, using `poe`:

```
poe pw.x -procs 16 -npool 8 < input
```

PC clusters using `mpiexec`:

```
mpiexec -n 16 pw.x -npool 8 < input
```

SGI Altix and PC clusters using `mpirun`:

```
mpirun -np 16 pw.x -npool 8 < input
```

IBM BlueGene using `mpirun`:

```
mpirun -np 16 -exe /path/to/executable/pw.x -args "-npool 8" \
  -in /path/to/input -cwd /path/to/work/directory
```

If you want to run in parallel the examples distributed with QUANTUM ESPRESSO (see Sec.2.6), set PARA_PREFIX to everything before the executable (`pw.x` in the above examples), PARA_POSTFIX to what follows it until the first redirection sign ($<, >, |, ..$), if any. For execution using OpenMP on N threads, set PARA_PREFIX to `env OMP_NUM_THREADS=N`.

3.3 Parallelization levels

Data structures are distributed across processors. Processors are organized in a hierarchy of groups, which are identified by different MPI communicators level. The groups hierarchy is as follow:

```
                / pools _ task   groups
world _ images
                \ linear-algebra  groups
```

world: is the group of all processors (MPI_COMM_WORLD).

images: Processors can then be divided into different "images", corresponding to a point in configuration space (i.e. to a different set of atomic positions). Such partitioning is used when performing Nudged Elastic band (NEB) calculations.

pools: When k-point sampling is used, each image group can be subpartitioned into "pools", and k-points can distributed to pools. Within each pool, reciprocal space basis set (PWs) and real-space grids are distributed across processors. This is usually referred to as "PW parallelization". All linear-algebra operations on array of PW / real-space grids are automatically and effectively parallelized. 3D FFT is used to transform electronic wave functions from reciprocal to real space and vice versa. The 3D FFT is parallelized by distributing planes of the 3D grid in real space to processors (in reciprocal space, it is columns of G-vectors that are distributed to processors).

task groups: In order to allow good parallelization of the 3D FFT 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.

linear-algebra group: A further level of parallelization, independent on PW or k-point parallelization, is the parallelization of subspace diagonalization (`pw.x`) or iterative orthonormalization (`cp.x`). Both operations required the diagonalization of arrays whose dimension is the number of Kohn-Sham states (or a small multiple). All such arrays are distributed block-like across the "linear-algebra group", a subgroup of the pool of processors, organized in a square 2D grid. As a consequence the number of processors in the linear-algebra group is given by n^2, where n is an integer; n^2 must be smaller than the number of processors of a single pool. The diagonalization is then performed in parallel using standard linear algebra operations. (This diagonalization is used by, but should not be confused with, the iterative Davidson algorithm). One can choose to compile ScaLAPACK if available, internal built-in algorithms otherwise.

Communications: Images and pools are loosely coupled and processors communicate between different images and pools only once in a while, whereas processors within each pool are tightly coupled and communications are significant. This means that Gigabit ethernet (typical for cheap PC clusters) is ok up to 4-8 processors per pool, but *fast* communication hardware (e.g. Mirynet or comparable) is absolutely needed beyond 8 processors per pool.

Choosing parameters: To control the number of processors in each group, command line switches: `-nimage`, `-npools`, `-ntg`, `northo` (for `cp.x`) or `-ndiag` (for `pw.x`) are used. As an example consider the following command line:

```
mpirun -np 4096 ./pw.x -nimage 8 -npool 2 -ntg 8 -ndiag 144 -input my.input
```

This executes `PWscf` on 4096 processors, to simulate a system with 8 images, each of which is distributed across 512 processors. k-points are distributed across 2 pools of 256 processors each, 3D FFT is performed using 8 task groups (64 processors each, so the 3D real-space grid is cut into 64 slices), and the diagonalization of the subspace Hamiltonian is distributed to a square grid of 144 processors (12x12).

Default values are: `-nimage 1 -npool 1 -ntg 1` ; `ndiag` is set to 1 if ScaLAPACK is not compiled, it is set to the square integer smaller than or equal to half the number of processors of each pool.

Massively parallel calculations For very large jobs (i.e. O(1000) atoms or so) or for very long jobs to be run on massively parallel machines (e.g. IBM BlueGene) it is crucial to use in an effective way both the "task group" and the "linear-algebra" parallelization. Without a judicious choice of parameters, large jobs will find a stumbling block in either memory or CPU requirements. In particular, the linear-algebra parallelization is used in the diagonalization of matrices in the subspace of Kohn-Sham states (whose dimension is as a strict minumum equal to the number of occupied states). These are stored as block-distributed matrices (distributed across processors) and diagonalized using custom-tailored diagonalization algorithms that work on block-distributed matrices.

Since v.4.1, ScaLAPACK can be used to diagonalize block distributed matrices, yielding better speed-up than the default algorithms for large (> 1000) matrices, when using a large number of processors (> 512). If you want to test ScaLAPACK, use `configure --with-scalapack`. This will add `-D__SCALAPACK` to DFLAGS in `make.sys` and set LAPACK_LIBS to something like:

```
LAPACK_LIBS = -lscalapack -lblacs -lblacsF77init -lblacs -llapack
```

The repeated `-lblacs` is not an error, it is needed! If `configure` does not recognize ScaLAPACK, inquire with your system manager on the correct way to link them.

A further possibility to expand scalability, especially on machines like IBM BlueGene, is to use mixed MPI-OpenMP. The idea is to have one (or more) MPI process(es) per multicore node, with OpenMP parallelization inside a same node. This option is activated by `configure --with-openmp`, which adds preprocessing flag -D__OPENMP and one of the following compiler options:

ifort: `-openmp`
xlf: `-qsmp=omp`
PGI: `-mp`
ftn: `-mp=nonuma`

OpenMP parallelization is currently implemented and tested for the following combinations of FFTs and libraries:

internal FFTW copy: `-D__FFTW`
ESSL: `-D__ESSL` or `-D__LINUX_ESSL`, link with `-lesslsmp`
ACML: `-D__ACML`, link with `-lacml_mp`.

Currently, ESSL (when available) are faster than internal FFTW, which in turn are faster than ACML.

3.3.1 Understanding parallel I/O

In parallel execution, each processor has its own slice of wavefunctions, to be written to temporary files during the calculation. The way wavefunctions are written by `pw.x` is governed by variable `wf_collect`, in namelist &CONTROL If `wf_collect=.true.`, the final wavefunctions are collected into a single directory, written by a single processor, whose format is independent on the number of processors. If `wf_collect=.false.` (default) each processor writes its own slice of the final wavefunctions to disk in the internal format used by `PWscf`.

The former case requires more disk I/O and disk space, but produces portable data files; the latter case requires less I/O and disk space, but the data so produced can be read only by

a job running on the same number of processors and pools, and if all files are on a file system that is visible to all processors (i.e., you cannot use local scratch directories: there is presently no way to ensure that the distribution of processes on processors will follow the same pattern for different jobs).

`cp.x` instead always collects the final wavefunctions into a single directory. Files written by `pw.x` can be read by `cp.x` only if `wf_collect=.true.` (and if produced for $k = 0$ case). The directory for data is specified in input variables `outdir` and `prefix` (the former can be specified as well in environment variable ESPRESSO_TMPDIR): `outdir/prefix.save`. A copy of pseudopotential files is also written there. If some processor cannot access the data directory, the pseudopotential files are read instead from the pseudopotential directory specified in input data. Unpredictable results may follow if those files are not the same as those in the data directory!

IMPORTANT: Avoid I/O to network-mounted disks (via NFS) as much as you can! Ideally the scratch directory `outdir` should be a modern Parallel File System. If you do not have any, you can use local scratch disks (i.e. each node is physically connected to a disk and writes to it) but you may run into trouble anyway if you need to access your files that are scattered in an unpredictable way across disks residing on different nodes.

You can use input variable `disk_io='minimal'`, or even `'none'`, if you run into trouble (or into angry system managers) with excessive I/O with `pw.x`. The code will store wavefunctions into RAM during the calculation. Note however that this will increase your memory usage and may limit or prevent restarting from interrupted runs.

Cray XT3 On the cray xt3 there is a special hack to keep files in memory instead of writing them without changes to the code. You have to do a: module load iobuf before compiling and then add liobuf at link time. If you run a job you set the environment variable IOBUF_PARAMS to proper numbers and you can gain a lot. Here is one example:

```
env IOBUF_PARAMS='*.wfc*:noflush:count=1:size=15M:verbose,\
*.dat:count=2:size=50M:lazyflush:lazyclose:verbose,\
*.UPF*.xml:count=8:size=8M:verbose' pbsyod =\
\~{}/pwscf/pwscfcvs/bin/pw.x npool 4 in si64pw2x2x2.inp > & \
si64pw2x2x232moreiobuf.out &
```

This will ignore all flushes on the *wfc* (scratch files) using a single i/o buffer large enough to contain the whole file ($\sim$ 12 Mb here). this way they are actually never(!) written to disk. The *.dat files are part of the restart, so needed, but you can be 'lazy' since they are writeonly. .xml files have a lot of accesses (due to iotk), but with a few rather small buffers, this can be handled as well. You have to pay attention not to make the buffers too large, if the code needs a lot of memory, too and in this example there is a lot of room for improvement. After you have tuned those parameters, you can remove the 'verboses' and enjoy the fast execution. Apart from the i/o issues the cray xt3 is a really nice and fast machine. (Info by Axel Kohlmeyer, maybe obsolete)

3.4 Tricks and problems

Trouble with input files Some implementations of the MPI library have problems with input redirection in parallel. This typically shows up under the form of mysterious errors when

reading data. If this happens, use the option `-in` (or `-inp` or `-input`), followed by the input file name. Example:

```
pw.x -in inputfile npool 4 > outputfile
```

Of course the input file must be accessible by the processor that must read it (only one processor reads the input file and subsequently broadcasts its contents to all other processors).

Apparently the LSF implementation of MPI libraries manages to ignore or to confuse even the `-in/inp/input` mechanism that is present in all QUANTUM ESPRESSO codes. In this case, use the `-i` option of `mpirun.lsf` to provide an input file.

Trouble with MKL and MPI parallelization If you notice very bad parallel performances with MPI and MKL libraries, it is very likely that the OpenMP parallelization performed by the latter is colliding with MPI. Recent versions of MKL enable autoparallelization by default on multicore machines. You must set the environmental variable OMP_NUM_THREADS to 1 to disable it. Note that if for some reason the correct setting of variable OMP_NUM_THREADS does not propagate to all processors, you may equally run into trouble. Lorenzo Paulatto (Nov. 2008) suggests to use the `-x` option to `mpirun` to propagate OMP_NUM_THREADS to all processors. Axel Kohlmeyer suggests the following (April 2008): "(I've) found that Intel is now turning on multithreading without any warning and that is for example why their FFT seems faster than FFTW. For serial and OpenMP based runs this makes no difference (in fact the multi-threaded FFT helps), but if you run MPI locally, you actually lose performance. Also if you use the 'numactl' tool on linux to bind a job to a specific cpu core, MKL will still try to use all available cores (and slow down badly). The cleanest way of avoiding this mess is to either link with

`-lmkl_intel_lp64 -lmkl_sequential -lmkl_core` (on 64-bit: x86_64, ia64)
`-lmkl_intel -lmkl_sequential -lmkl_core` (on 32-bit, i.e. ia32)

or edit the `libmkl_'platform'.a` file. I'm using now a file `libmkl10.a` with:

```
GROUP (libmkl_intel_lp64.a libmkl_sequential.a libmkl_core.a)
```

It works like a charm". UPDATE: Since v.4.2, `configure` links by default MKL without multithreaded support.

Trouble with compilers and MPI libraries Many users of QUANTUM ESPRESSO, in particular those working on PC clusters, have to rely on themselves (or on less-than-adequate system managers) for the correct configuration of software for parallel execution. Mysterious and irreproducible crashes in parallel execution are sometimes due to bugs in QUANTUM ESPRESSO, but more often than not are a consequence of buggy compilers or of buggy or miscompiled MPI libraries. Very useful step-by-step instructions to compile and install MPI libraries can be found in the following post by Javier Antonio Montoya:
`http://www.democritos.it/pipermail/pw_forum/2008-April/008818.htm`.

On a Xeon quadriprocessor cluster, erratic crashes in parallel execution have been reported, apparently correlated with ifort 10.1 (info by Nathalie Vast and Jelena Sjakste, May 2008).

4 Using Quantum ESPRESSO

Input files for PWscf codes 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 CVS sources.

You may take the examples distributed with Quantum ESPRESSO as templates for writing your own input files: see Sec.2.6. In the following, whenever we mention "Example N", we refer to those. Input files are those in the results/ subdirectories, with names ending with .in (they will appear after you have run the examples).

Note about XC: the type of XC used in the calculation is read from PP files. All PPs must have been generated using the same XC. You can override this choice by setting input variable input_dft (see list of allowed values in Modules/funct.f90).

4.1 Input data

Input data for the basic codes of the Quantum ESPRESSO distribution, pw.x and ,x, is organized as several namelists, followed by other fields 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 dynamics
&EE (optional):	for density counter charge electrostatic corrections

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:

ibrav	(integer)	Bravais-lattice index
celldm	(real, dimension 6)	crystallographic constants
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.

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 width degauss and optionally the smearing type smearing. Spin-polarized systems must be treated as metallic system, except the special case of a single k-point, for which occupation numbers can be fixed (occupations='from input' and card OCCUPATIONS).

Explanations for the meaning of variables ibrav and celldm, as well as on alternative ways to input structural data, are in files Doc/INPUT_PW.* (for pw.x) and Doc/INPUT_CP.* (for cp.x). These files are the reference for input data and describe a large number of other variables as well. Almopst all variables have default values, which may or may not fit your needs.

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)
CLIMBING_IMAGES (optional)

The keywords may be followed on the same line by an option. Unknown fields (including some that are specific to `CP`) are ignored by `PWscf`(and vice versa, `CP` ignores `PWscf`-specific fields). See the files mentioned above for details on the available "cards".

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-point 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.

4.2 Data files

The output data files are written in the directory specified by variable `outdir`, with names specified by variable `prefix` (a string that is prepended to all file names, whose default value is: `prefix='pwscf'`). The `iotk` toolkit is used to write the file in a XML format, whose definition can be found in the Developer Manual. In order to use the data directory on a different machine, you need to convert the binary files to formatted and back, using the `bin/iotk` script.

The execution stops if you create a file `prefix.EXIT` in the working directory. NOTA BENE: this is the directory where the program is executed, NOT the directory `outdir` defined in input, where files are written. Note that with some versions of MPI, the working directory is the directory where the `pw.x` 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 unusable state.

4.3 Format of arrays containing charge density, potential, etc.

The index of arrays used to store functions defined on 3D meshes is actually a shorthand for three indices, following the FORTRAN convention ("leftmost index runs faster"). An example will explain this better. Suppose you have a 3D array `psi(nr1x,nr2x,nr3x)`. FORTRAN compilers store this array sequentially in the computer RAM in the following way:

```
psi(   1,   1,   1)
psi(   2,   1,   1)
...
psi(nr1x,   1,   1)
psi(   1,   2,   1)
psi(   2,   2,   1)
...
psi(nr1x,   2,   1)
...
...
psi(nr1x,nr2x,   1)
...
```

```
        psi(nr1x,nr2x,nr3x)
etc
```

Let `ind` be the position of the `(i,j,k)` element in the above list: the following relation

```
        ind = i + (j - 1) * nr1x + (k - 1) *  nr2x * nr1x
```

holds. This should clarify the relation between 1D and 3D indexing. In real space, the `(i,j,k)` point of the FFT grid with dimensions `nr1` ($\leq$`nr1x`), `nr2` ($\leq$`nr2x`), , `nr3` ($\leq$`nr3x`), is

$$r_{ijk} = \frac{i-1}{nr1}\tau_1 + \frac{j-1}{nr2}\tau_2 + \frac{k-1}{nr3}\tau_3$$

where the τ_i are the basis vectors of the Bravais lattice. The latter are stored row-wise in the `at` array: τ_1 = `at(:, 1)`, τ_2 = `at(:, 2)`, τ_3 = `at(:, 3)`.

The distinction between the dimensions of the FFT grid, `(nr1,nr2,nr3)` and the physical dimensions of the array, `(nr1x,nr2x,nr3x)` is done only because it is computationally convenient in some cases that the two sets are not the same. In particular, it is often convenient to have `nrx1`=`nr1`+1 to reduce memory conflicts.

5 Using PWscf

Code `pw.x` performs various kinds of electronic and ionic structure calculations. We may distinguish the following typical cases of usage for `pw.x`:

5.1 Electronic structure calculations

Single-point (fixed-ion) SCF calculation Set `calculation='scf'` (this is actually the default). Namelists &IONS and &CELL will be ignored. See Example 01.

Band structure calculation First perform a SCF calculation as above; then do a non-SCF calculation with the desired k-point grid and number `nbnd` of bands. Specify `calculation='bands'` if you are interested in calculating only the Kohn-Sham states for the given set of k-points; specify `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 specio 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 01, 05, 08,

NOTA BENE: until v.4.0, atomic positions for a non scf calculations were read from input, while the scf potential was read from the data file of the scf calculation. Since v.4.1, both atomic positions and the scf potential are read from the data file so that consistency is guaranteed.

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 `noncolin` must be true. 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, it makes a non collinear spin polarized calculation with spin-orbit interaction. The final spin magnetization might be zero or different from zero depending on the system.

Furthermore to make a spin-orbit calculation you must use fully relativistic pseudopotentials at least for the atoms in which you think that spin-orbit interaction is large. If all the pseudopotentials are scalar relativistic the calculation becomes equivalent to a noncolinear calculation without spin orbit. (Andrea Dal Corso, 2007-07-27) See Example 13 for non-collinear magnetism, Example 22 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. See Example 25 and references quoted therein.

Dispersion Interactions (DFT-D) For DFT-D (DFT + semiempirical dispersion interactions), see the description of input variables `london*`, sample files `tests/vdw.*`, and the comments in source file `Modules/mm_dispersion.f90`.

Hartree-Fock and Hybrid functionals Calculations in the Hartree-Fock approximation, or using hybrid XC functionals that include some Hartree-Fock exchange, currently require that `-DEXX` is added to the preprocessing options `DFLAGS` in file `make.sys` before compilation (if you change this after the first compilation, `make clean`, recompile). Documentation on usage can be found in subdirectory `examples/EXX_example/`.

The algorithm is quite standard: see for instance Chawla and Voth, JCP bf 108, 4697 (1998); Sorouri, Foulkes and Hine, JCP **124**, 064105 (2006); Spencer and Alavi, PRB **77**, 193110 (2008). Basically, one generates auxiliary densities $\rho_{-q} = \phi^*_{k+q} * \psi_k$ in real space and transforms them to reciprocal space using FFT; the Poisson equation is solved and the resulting potential is transformed back to real space using FFT, then multiplied by ϕ_{k+q} and the results are accumulated. The only tricky point is the treatment of the $q \rightarrow 0$ limit, which is described in the Appendix A.5 of the QUANTUM ESPRESSO paper mentioned in the Introduction (note the reference to the Gygi and Baldereschi paper). See also J. Comp. Chem. **29**, 2098 (2008); JACS **129**, 10402 (2007) for examples of applications.

Polarization via Berry Phase See Example 10, file `example10/README`, and the documentation in the header of `PW/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 `examples/dipole_example` for an example of application. Electric fields via modern theory of polarization are documented in example 31. The exact meaning of the related variables, for both cases, is explained in the general input documentation.

5.2 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 (since v.4.1, you do not need any longer to update the atomic positions in the input file for non scf calculation). See Example 03.

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 04.

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 (RM 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).

See also `examples/VCSexample`.

5.3 Nudged Elastic Band calculation

Specify `calculation='neb'` and add namelist &IONS.

All options for a single SCF calculation apply, plus a few others. In the namelist &IONS the number of images used to discretize the elastic band must be specified. All other variables have a default value. Coordinates of the initial and final image of the elastic band have to be specified in the ATOMIC_POSITIONS card. A detailed description of all input variables is contained in files `Doc/INPUT_PW.*`. See Example 17.

A NEB calculation will produce a number of files in the current directory (i.e. in the directory were the code is run) containing additional information on the minimum-energy path. The files are organized as following (where `prefix` is specified in the input file):

`prefix.dat` is a three-column file containig the position of each image on the reaction coordinate (arb. units), its energy in eV relative to the energy of the first image and the residual error for the image in eV/a_0.

`prefix.int` contains an interpolation of the path energy profile that pass exactly through each image; it is computed using both the image energies and their derivatives

`prefix.path` information used by QUANTUM ESPRESSO to restart a path calculation, its format depends on the input details and is undocumented

`prefix.axsf` atomic positions of all path images in the XCrySDen animation format: to visualize it, use `xcrysden --axsf prefix.axsf`

`prefix.xyz` atomic positions of all path images in the generic xyz format, used by many quantum-chemistry softwares

`prefix.crd` path information in the input format used by `pw.x`, suitable for a manual restart of the calculation

"NEB calculation are a bit tricky in general and require extreme care to be setup correctly. NEB also takes easily hundreds of iteration to converge, of course depending on the number of atoms and of images. Here is some free advice:

1. Don't use Climbing Image (CI) from the beginning. It makes convergence slower, especially if the special image changes during the convergence process (this may happen if `CI_scheme='auto'` and if it does it may mess up everything). Converge your calculation, then restart from the last configuration with CI option enabled (note that this will *increase* the barrier).

2. Carefully choose the initial path. Remember that QUANTUM ESPRESSO assumes continuity between the first and the last image at the initial condition. In other words, periodic images are NOT used; you may have to manually translate an atom by one or more unit cell base vectors in order to have a meaningful initial path. You can visualize NEB input files with XCrySDen as animations, take some time to check if any atoms overlap or get very close in the initial path (you will have to add intermediate images, in this case).

3. Try to start the NEB process with most atomic positions fixed, in order to converge the more "problematic" ones, before leaving all atoms move.

4. Especially for larger systems, you can start NEB with lower accuracy (less k-points, lower cutoff) and then increase it when it has converged to refine your calculation.

5. Use the Broyden algorithm instead of the default one: it is a bit more fragile, but it removes the problem of "oscillations" in the calculated activation energies. If these oscillations persist, and you cannot afford more images, focus to a smaller problem, decompose it into pieces.

6. A gross estimate of the required number of iterations is (number of images) * (number of atoms) * 3. Atoms that do not move should not be counted. It may take half that many iterations, or twice as many, but more or less that's the order of magnitude, unless one starts from a very good or very bad initial guess.

(Courtesy of Lorenzo Paulatto)

6 Phonon calculations

Phonon calculation is presently a two-step process. First, you have to find the ground-state atomic and electronic configuration; Second, you can calculate phonons using Density-Functional Perturbation Theory. Further processing to calculate Interatomic Force Constants, to add macroscopic electric field and impose Acoustic Sum Rules at q=0 may be needed. In the following, we will indicate by q the phonon wavevectors, whle k will indicate Bloch vectors used for summing over the Brillouin Zone.

Since version 4.0 it is possible to safely stop execution of `ph.x` code using the same mechanism of the `pw.x` code, i.e. by creating a file `prefix.EXIT` in the working directory. Execution can be resumed by setting `recover=.true.` in the subsequent input data.

6.1 Single-q calculation

The phonon code `ph.x` calculates normal modes at a given q-vector, starting from data files produced by `pw.x` with a simple SCF calculation. NOTE: the alternative procedure in which a band-structure calculation with `calculation='phonon` was performed as an intermediate step is no longer implemented since version 4.1. It is also no longer needed to specify `lnscf=.true.` for $q \neq 0$.

The output data file appear in the directory specified by variables outdir, with names specified by variable prefix. After the output file(s) has been produced (do not remove any of the files, unless you know which are used and which are not), you can run `ph.x`.

The first input line of `ph.x` is a job identifier. At the second line the namelist &INPUTPH starts. The meaning of the variables in the namelist (most of them having a default value) is described in file `Doc/INPUT_PH.*`. Variables `outdir` and `prefix` must be the same as in the input data of `pw.x`. Presently you must also specify `amass(i)` (a real variable): the atomic mass of atomic type i.

After the namelist you must specify the q-vector of the phonon mode. This must be the same q-vector given in the input of `pw.x`.

Notice that the dynamical matrix calculated by `ph.x` at $q = 0$ does not contain the non-analytic term occurring in polar materials, i.e. there is no LO-TO splitting in insulators. Moreover no Acoustic Sum Rule (ASR) is applied. In order to have the complete dynamical matrix at $q = 0$ including the non-analytic terms, you need to calculate effective charges by specifying option `epsil=.true.` to `ph.x`. This is however not possible (because not physical!) for metals (i.e. any system subject to a broadening).

At $q = 0$, use program `dynmat.x` to calculate the correct LO-TO splitting, IR cross sections, and to impose various forms of ASR. If `ph.x` was instructed to calculate Raman coefficients, `dynmat.x` will also calculate Raman cross sections for a typical experimental setup. Input documentation in the header of `PH/dynmat.f90`.

A sample phonon calculation is performed in Example 02.

6.2 Calculation of interatomic force constants in real space

First, dynamical matrices are calculated and saved for a suitable uniform grid of q-vectors (only those in the Irreducible Brillouin Zone of the crystal are needed). Although this can be done one q-vector at the time, a simpler procedure is to specify variable `ldisp=.true.` and to set variables `nq1`, `nq2`, `nq3` to some suitable Monkhorst-Pack grid, that will be automatically

generated, centered at $q = 0$. Do not forget to specify `epsil=.true.` in the input data of `ph.x` if you want the correct TO-LO splitting in polar materials.

Second, code `q2r.x` reads the dynamical matrices produced in the preceding step and Fourier-transform them, writing a file of Interatomic Force Constants in real space, up to a distance that depends on the size of the grid of q-vectors. Input documentation in the header of `PH/q2r.f90`.

Program `matdyn.x` may be used to produce phonon modes and frequencies at any q using the Interatomic Force Constants file as input. Input documentation in the header of `PH/matdyn.f90`.

For more details, see Example 06.

6.3 Calculation of electron-phonon interaction coefficients

The calculation of electron-phonon coefficients in metals is made difficult by the slow convergence of the sum at the Fermi energy. It is convenient to use a coarse k-point grid to calculate phonons on a suitable wavevector grid; a dense k-point grid to calculate the sum at the Fermi energy. The calculation proceeds in this way:

1. a scf calculation for the dense k-point grid (or a scf calculation followed by a non-scf one on the dense k-point grid); specify option `la2f=.true.` to `pw.x` in order to save a file with the eigenvalues on the dense k-point grid. The latter MUST contain all k and k+q grid points used in the subsequent electron-phonon calculation. All grids MUST be unshifted, i.e. include $k = 0$.

2. a normal scf + phonon dispersion calculation on the coarse k-point grid, specifying option `elph=.true.`. and the file name where the self-consistent first-order variation of the potential is to be stored: variable `fildvscf`). The electron-phonon coefficients are calculated using several values of Gaussian broadening (see `PH/elphon.f90`) because this quickly shows whether results are converged or not with respect to the k-point grid and Gaussian broadening.

3. Finally, you can use `matdyn.x` and `lambda.x` (input documentation in the header of `PH/lambda.f90`) to get the $\alpha^2F(\omega)$ function, the electron-phonon coefficient λ, and an estimate of the critical temperature T_c.

For more details, see Example 07.

6.4 Distributed Phonon calculations

A complete phonon dispersion calculation can be quite long and expensive, but it can be spit into a number of semi-independent calculations, using options `start_q`, `last_q`, `start_irr`, `last_irr`. An example on how to distribute the calculations and collect the results can be found in `examples/GRID_example`. Reference:
Calculation of Phonon Dispersions on the GRID using Quantum ESPRESSO, R. di Meo, A. Dal Corso, P. Giannozzi, and S. Cozzini, in *Chemistry and Material Science Applications on Grid Infrastructures*, editors: S. Cozzini, A. Laganà, ICTP Lecture Notes Series, Vol. 24, pp.165-183 (2009).

7 Post-processing

There are a number of auxiliary codes performing postprocessing tasks such as plotting, averaging, and so on, on the various quantities calculated by `pw.x`. Such quantities are saved by `pw.x` into the output data file(s). Postprocessing codes are in the `PP/` directory. All codes for which input documentation is not explicitly mentioned have documentation in the header of the fortran sources.

7.1 Plotting selected quantities

The main postprocessing code `pp.x` reads data file(s), extracts or calculates the selected quantity, writes it into a format that is suitable for plotting.

Quantities that can be read or calculated are:

charge density
spin polarization
various potentials
local density of states at E_F
local density of electronic entropy
STM images
selected squared wavefunction
ELF (electron localization function)
planar averages
integrated local density of states

Various types of plotting (along a line, on a plane, three-dimensional, polar) and output formats (including the popular cube format) can be specified. The output files can be directly read by the free plotting system Gnuplot (1D or 2D plots), or by code `plotrho.x` that comes with `PostProc` (2D plots), or by advanced plotting software XCrySDen and gOpenMol (3D plots).

See file `Doc/INPUT_PP.*` for a detailed description of the input for code `pp.x`. See Example 05 for an example of a charge density plot, Example 16 for an example of STM image simulation.

7.2 Band structure, Fermi surface

The code `bands.x` reads data file(s), extracts eigenvalues, regroups them into bands (the algorithm used to order bands and to resolve crossings may not work in all circumstances, though). The output is written to a file in a simple format that can be directly read by plotting program `plotband.x`. Unpredictable plots may results if k-points are not in sequence along lines. See Example 05 directory for a simple band plot.

The code `bands.x` performs as well a symmetry analysis of the band structure: see Example 01.

The calculation of Fermi surface can be performed using `kvecs_FS.x` and `bands_FS.x`. The resulting file in .xsf format can be read and plotted using XCrySDen. See Example 08 for an example of Fermi surface visualization (Ni, including the spin-polarized case).

7.3 Projection over atomic states, DOS

The code `projwfc.x` calculates projections of wavefunctions over atomic orbitals. The atomic wavefunctions are those contained in the pseudopotential file(s). The Löwdin population anal-

ysis (similar to Mulliken analysis) is presently implemented. The projected DOS (or PDOS: the DOS projected onto atomic orbitals) can also be calculated and written to file(s). More details on the input data are found in file `Doc/INPUT_PROJWFC.*`. The ordering of the various angular momentum components (defined in routine `flib/ylmr2.f90`) is as follows: $P_{0,0}(t)$, $P_{1,0}(t)$, $P_{1,1}(t)cos\phi$, $P_{1,1}(t)sin\phi$, $P_{2,0}(t)$, $P_{2,1}(t)cos\phi$, $P_{2,1}(t)sin\phi$, $P_{2,2}(t)cos2\phi$, $P_{2,2}(t)sin2\phi$ and so on, where $P_{l,m}$=Legendre Polynomials, $t = cos\theta = z/r$, $\phi = atan(y/x)$.

The total electronic DOS is instead calculated by code `dos.x`. See Example 08 for total and projected electronic DOS calculations.

7.4 Wannier functions

There are several Wannier-related utilities in `PostProc`:

1. The "Poor Man Wannier" code `pmw.x`, to be used in conjunction with DFT+U calculations (see Example 25)

2. The interface with Wannier90 code, `pw2wannier.x`: see the documentation in `W90/` (you have to install the Wannier90 plug-in)

3. The `wannier_ham.x` code generates a model Hamiltonian in Wannier functions basis: see `examples/WannierHam_example/`.

7.5 Other tools

Code `sumpdos.x` can be used to sum selected PDOS, produced by `projwfc.x`, by specifiying the names of files containing the desired PDOS. Type `sumpdos.x -h` or look into the source code for more details.

Code `epsilon.x` calculates RPA frequency-dependent complex dielectric function. Documentation is in `Doc/eps_man.tex`.

The code `path_int.x` is intended to be used in the framework of NEB calculations. It is a tool to generate a new path (what is actually generated is the restart file) starting from an old one through interpolation (cubic splines). The new path can be discretized with a different number of images (this is its main purpose), images are equispaced and the interpolation can be also performed on a subsection of the old path. The input file needed by `path_int.x` can be easily set up with the help of the self-explanatory `path_int.sh` shell script.

8 Using CP

This section is intended to explain how to perform basic Car-Parrinello (CP) simulations using the `CP` package.

It is important to understand that a CP simulation is a sequence of different runs, some of them used to "prepare" the initial state of the system, and other performed to collect statistics, or to modify the state of the system itself, i.e. modify the temperature or the pressure.

To prepare and run a CP simulation you should first of all define the system:

atomic positions
system cell
pseudopotentials

cut-offs
number of electrons and bands (optional)
FFT grids (optional)

An example of input file (Benzene Molecule):

```
&control
   title = 'Benzene Molecule',
   calculation = 'cp',
   restart_mode = 'from_scratch',
   ndr = 51,
   ndw = 51,
   nstep = 100,
   iprint = 10,
   isave = 100,
   tstress = .TRUE.,
   tprnfor = .TRUE.,
   dt    = 5.0d0,
   etot_conv_thr = 1.d-9,
   ekin_conv_thr = 1.d-4,
   prefix = 'c6h6',
   pseudo_dir='/scratch/benzene/',
   outdir='/scratch/benzene/Out/'
/
&system
   ibrav = 14,
   celldm(1) = 16.0,
   celldm(2) = 1.0,
   celldm(3) = 0.5,
   celldm(4) = 0.0,
   celldm(5) = 0.0,
   celldm(6) = 0.0,
   nat = 12,
   ntyp = 2,
   nbnd = 15,
   ecutwfc = 40.0,
   nr1b= 10, nr2b = 10, nr3b = 10,
   input_dft = 'BLYP'
/
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'sd'
/
&ions
   ion_dynamics = 'none'
/
&cell
```

```
  cell_dynamics = 'none',
  press = 0.0d0,
/
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
ATOMIC_POSITIONS (bohr)
C     2.6 0.0 0.0
C     1.3 -1.3 0.0
C    -1.3 -1.3 0.0
C    -2.6 0.0 0.0
C    -1.3 1.3 0.0
C     1.3 1.3 0.0
H     4.4 0.0 0.0
H     2.2 -2.2 0.0
H    -2.2 -2.2 0.0
H    -4.4 0.0 0.0
H    -2.2 2.2 0.0
H     2.2 2.2 0.0
```

You can find the description of the input variables in file `Doc/INPUT_CP.*`.

8.1 Reaching the electronic ground state

The first run, when starting from scratch, is always an electronic minimization, with fixed ions and cell, to bring the electronic system on the ground state (GS) relative to the starting atomic configuration. This step is conceptually very similar to self-consistency in a `pw.x` run.

Sometimes a single run is not enough to reach the GS. In this case, you need to re-run the electronic minimization stage. Use the input of the first run, changing `restart_mode = 'from_scratch'` to `restart_mode = 'restart'`.

NOTA BENE: Unless you are already experienced with the system you are studying or with the internals of the code, you will usually need to tune some input parameters, like `emass`, `dt`, and cut-offs. For this purpose, a few trial runs could be useful: you can perform short minimizations (say, 10 steps) changing and adjusting these parameters to fit your needs. You can specify the degree of convergence with these two thresholds:

`etot_conv_thr`: total energy difference between two consecutive steps
`ekin_conv_thr`: value of the fictitious kinetic energy of the electrons.

Usually we consider the system on the GS when `ekin_conv_thr` $< 10^{-5}$. You could check the value of the fictitious kinetic energy on the standard output (column EKINC).

Different strategies are available to minimize electrons, but the most used ones are:

- steepest descent: `electron_dynamics = 'sd'`
- damped dynamics: `electron_dynamics = 'damp'`, `electron_damping` = a number typically ranging from 0.1 and 0.5

See the input description to compute the optimal damping factor.

8.2 Relax the system

Once your system is in the GS, depending on how you have prepared the starting atomic configuration:

1. if you have set the atomic positions "by hand" and/or from a classical code, check the forces on atoms, and if they are large ($\sim 0.1 \div 1.0$ atomic units), you should perform an ionic minimization, otherwise the system could break up during the dynamics.

2. if you have taken the positions from a previous run or a previous ab-initio simulation, check the forces, and if they are too small ($\sim 10^{-4}$ atomic units), this means that atoms are already in equilibrium positions and, even if left free, they will not move. Then you need to randomize positions a little bit (see below).

Let us consider case 1). There are different strategies to relax the system, but the most used are again steepest-descent or damped-dynamics for ions and electrons. You could also mix electronic and ionic minimization scheme freely, i.e. ions in steepest-descent and electron in with damped-dynamics or vice versa.

(a) suppose we want to perform steepest-descent for ions. Then we should specify the following section for ions:

```
&ions
  ion_dynamics = 'sd'
/
```

Change also the ionic masses to accelerate the minimization:

```
ATOMIC_SPECIES
 C 2.0d0 c_blyp_gia.pp
 H 2.00d0 h.ps
```

while leaving other input parameters unchanged. *Note* that if the forces are really high (> 1.0 atomic units), you should always use steepest descent for the first (~ 100 relaxation steps.

(b) As the system approaches the equilibrium positions, the steepest descent scheme slows down, so is better to switch to damped dynamics:

```
&ions
  ion_dynamics = 'damp',
  ion_damping = 0.2,
  ion_velocities = 'zero'
/
```

A value of `ion_damping` around 0.05 is good for many systems. It is also better to specify to restart with zero ionic and electronic velocities, since we have changed the masses.

Change further the ionic masses to accelerate the minimization:

```
ATOMIC_SPECIES
C 0.1d0 c_blyp_gia.pp
H 0.1d0 h.ps
```

(c) when the system is really close to the equilibrium, the damped dynamics slow down too, especially because, since we are moving electron and ions together, the ionic forces are not properly correct, then it is often better to perform a ionic step every N electronic steps, or to move ions only when electron are in their GS (within the chosen threshold).

This can be specified by adding, in the ionic section, the `ion_nstepe` parameter, then the &IONS namelist become as follows:

```
&ions
  ion_dynamics = 'damp',
  ion_damping = 0.2,
  ion_velocities = 'zero',
  ion_nstepe = 10
/
```

Then we specify in the &CONTROL namelist:

```
etot_conv_thr = 1.d-6,
ekin_conv_thr = 1.d-5,
forc_conv_thr = 1.d-3
```

As a result, the code checks every 10 electronic steps whether the electronic system satisfies the two thresholds `etot_conv_thr`, `ekin_conv_thr`: if it does, the ions are advanced by one step. The process thus continues until the forces become smaller than `forc_conv_thr`.

Note that to fully relax the system you need many runs, and different strategies, that you should mix and change in order to speed-up the convergence. The process is not automatic, but is strongly based on experience, and trial and error.

Remember also that the convergence to the equilibrium positions depends on the energy threshold for the electronic GS, in fact correct forces (required to move ions toward the minimum) are obtained only when electrons are in their GS. Then a small threshold on forces could not be satisfied, if you do not require an even smaller threshold on total energy.

Let us now move to case 2: randomization of positions.

If you have relaxed the system or if the starting system is already in the equilibrium positions, then you need to displace ions from the equilibrium positions, otherwise they will not move in a dynamics simulation. After the randomization you should bring electrons on the GS again, in order to start a dynamic with the correct forces and with electrons in the GS. Then you should switch off the ionic dynamics and activate the randomization for each species, specifying the amplitude of the randomization itself. This could be done with the following &IONS namelist:

```
&ions
  ion_dynamics = 'none',
  tranp(1) = .TRUE.,
  tranp(2) = .TRUE.,
  amprp(1) = 0.01
  amprp(2) = 0.01
/
```

In this way a random displacement (of max 0.01 a.u.) is added to atoms of species 1 and 2. All other input parameters could remain the same. Note that the difference in the total energy (etot) between relaxed and randomized positions can be used to estimate the temperature that will be reached by the system. In fact, starting with zero ionic velocities, all the difference is potential energy, but in a dynamics simulation, the energy will be equipartitioned between kinetic and potential, then to estimate the temperature take the difference in energy (de), convert it in Kelvins, divide for the number of atoms and multiply by 2/3. Randomization could be useful also while we are relaxing the system, especially when we suspect that the ions are in a local minimum or in an energy plateau.

8.3 CP dynamics

At this point after having minimized the electrons, and with ions displaced from their equilibrium positions, we are ready to start a CP dynamics. We need to specify `'verlet'` both in ionic and electronic dynamics. The threshold in control input section will be ignored, like any parameter related to minimization strategy. The first time we perform a CP run after a minimization, it is always better to put velocities equal to zero, unless we have velocities, from a previous simulation, to specify in the input file. Restore the proper masses for the ions. In this way we will sample the microcanonical ensemble. The input section changes as follow:

```
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet',
   electron_velocities = 'zero'
/
&ions
   ion_dynamics = 'verlet',
   ion_velocities = 'zero'
/
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
```

If you want to specify the initial velocities for ions, you have to set `ion_velocities ='from_input'`, and add the IONIC_VELOCITIES card, after the ATOMIC_POSITION card, with the list of velocities in atomic units.

NOTA BENE: in restarting the dynamics after the first CP run, remember to remove or comment the velocities parameters:

```
&electrons
   emass = 400.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet'
   ! electron_velocities = 'zero'
/
&ions
   ion_dynamics = 'verlet'
   ! ion_velocities = 'zero'
/
```

otherwise you will quench the system interrupting the sampling of the microcanonical ensemble.

Varying the temperature It is possible to change the temperature of the system or to sample the canonical ensemble fixing the average temperature, this is done using the Nosé thermostat. To activate this thermostat for ions you have to specify in namelist &IONS:

```
&ions
   ion_dynamics = 'verlet',
   ion_temperature = 'nose',
   fnosep = 60.0,
   tempw = 300.0
/
```

where `fnosep` is the frequency of the thermostat in THz, that should be chosen to be comparable with the center of the vibrational spectrum of the system, in order to excite as many vibrational modes as possible. `tempw` is the desired average temperature in Kelvin.

Note: to avoid a strong coupling between the Nosé thermostat and the system, proceed step by step. Don't switch on the thermostat from a completely relaxed configuration: adding a random displacement is strongly recommended. Check which is the average temperature via a few steps of a microcanonical simulation. Don't increase the temperature too much. Finally switch on the thermostat. In the case of molecular system, different modes have to be thermalized: it is better to use a chain of thermostat or equivalently running different simulations with different frequencies.

Noše thermostat for electrons It is possible to specify also the thermostat for the electrons. This is usually activated in metals or in systems where we have a transfer of energy between ionic and electronic degrees of freedom. Beware: the usage of electronic thermostats is quite delicate. The following information comes from K. Kudin:

"The main issue is that there is usually some "natural" fictitious kinetic energy that electrons gain from the ionic motion ("drag"). One could easily quantify how much of the fictitious energy comes from this drag by doing a CP run, then a couple of CG (same as BO) steps, and then going back to CP. The fictitious electronic energy at the last CP restart will be purely due to the drag effect."

"The thermostat on electrons will either try to overexcite the otherwise "cold" electrons, or it will try to take them down to an unnaturally cold state where their fictitious kinetic energy is even below what would be just due pure drag. Neither of this is good."

"I think the only workable regime with an electronic thermostat is a mild overexcitation of the electrons, however, to do this one will need to know rather precisely what is the fictitious kinetic energy due to the drag."

8.4 Advanced usage

8.4.1 Self-interaction Correction

The self-interaction correction (SIC) included in the `CP` package is based on the Constrained Local-Spin-Density approach proposed my F. Mauri and coworkers (M. D'Avezac et al. PRB 71, 205210 (2005)). It was used for the first time in QUANTUM ESPRESSO by F. Baletto, C. Cavazzoni and S.Scandolo (PRL 95, 176801 (2005)).

This approach is a simple and nice way to treat ONE, and only one, excess charge. It is moreover necessary to check a priori that the spin-up and spin-down eigenvalues are not too different, for the corresponding neutral system, working in the Local-Spin-Density Approximation (setting `nspin = 2`). If these two conditions are satisfied and you are interest in charged systems, you can apply the SIC. This approach is a on-the-fly method to correct the self-interaction with the excess charge with itself.

Briefly, both the Hartree and the XC part have been corrected to avoid the interaction of the excess charge with tself.

For example, for the Boron atoms, where we have an even number of electrons (valence electrons = 3), the parameters for working with the SIC are:

```
&system
nbnd= 2,
total_magnetization=1,
sic_alpha = 1.d0,
sic_epsilon = 1.0d0,
sic = 'sic_mac',
force_pairing = .true.,

&ions
ion_dynamics = 'none',
ion_radius(1) = 0.8d0,
sic_rloc = 1.0,

ATOMIC_POSITIONS (bohr)
B 0.00 0.00 0.00 0 0 0 1
```

The two main parameters are:

`force_pairing = .true.`, which forces the paired electrons to be the same;
`sic='sic_mac'`, which instructs the code to use Mauri's correction.

Remember to add an extra-column in ATOMIC_POSITIONS with "1" to activate SIC for those atoms.

Warning: This approach has known problems for dissociation mechanism driven by excess electrons.

Comment 1: Two parameters, `sic_alpha` and `sic_epsilon'`, have been introduced following the suggestion of M. Sprik (ICR(05)) to treat the radical (OH)-H_2O. In any case, a complete ab-initio approach is followed using `sic_alpha=1`, `sic_epsilon=1`.

Comment 2: When you apply this SIC scheme to a molecule or to an atom, which are neutral, remember to add the correction to the energy level as proposed by Landau: in a neutral system, subtracting the self-interaction, the unpaired electron feels a charged system, even if using a compensating positive background. For a cubic box, the correction term due to the Madelung energy is approx. given by $1.4186/L_{box} - 1.047/(L_{box})^3$, where L_{box} is the linear dimension of your box (=celldm(1)). The Madelung coefficient is taken from I. Dabo et al. PRB 77, 115139 (2007). (info by F. Baletto, francesca.baletto@kcl.ac.uk)

8.4.2 ensemble-DFT

The ensemble-DFT (eDFT) is a robust method to simulate the metals in the framework of "ab-initio" molecular dynamics. It was introduced in 1997 by Marzari et al.

The specific subroutines for the eDFT are in `CPV/ensemble_dft.f90` where you define all the quantities of interest. The subroutine `CPV/inner_loop_cold.f90` called by `cg_sub.f90`, control the inner loop, and so the minimization of the free energy A with respect to the occupation matrix.

To select a eDFT calculations, the user has to set:

```
calculation = 'cp'
occupations= 'ensemble'
tcg = .true.
passop= 0.3
maxiter = 250
```

to use the CG procedure. In the eDFT it is also the outer loop, where the energy is minimized with respect to the wavefunction keeping fixed the occupation matrix. While the specific parameters for the inner loop. Since eDFT was born to treat metals, keep in mind that we want to describe the broadening of the occupations around the Fermi energy. Below the new parameters in the electrons list, are listed.

- `smearing`: used to select the occupation distribution; there are two options: Fermi-Dirac smearing='fd', cold-smearing smearing='cs' (recommended)
- `degauss`: is the electronic temperature; it controls the broadening of the occupation numbers around the Fermi energy.
- `ninner`: is the number of iterative cycles in the inner loop, done to minimize the free energy A with respect the occupation numbers. The typical range is 2-8.
- `conv_thr`: is the threshold value to stop the search of the 'minimum' free energy.
- `niter_cold_restart`: controls the frequency at which a full iterative inner cycle is done. It is in the range 1÷`ninner`. It is a trick to speed up the calculation.
- `lambda_cold`: is the length step along the search line for the best value for A, when the iterative cycle is not performed. The value is close to 0.03, smaller for large and complicated metallic systems.

NOTE: degauss is in Hartree, while in PWscfis in Ry (!!!). The typical range is 0.01-0.02 Ha.
The input for an Al surface is:

```
&CONTROL
 calculation = 'cp',
 restart_mode = 'from_scratch',
 nstep  = 10,
 iprint = 5,
 isave  = 5,
 dt    = 125.0d0,
 prefix = 'Aluminum_surface',
 pseudo_dir = '~/UPF/',
 outdir = '/scratch/'
 ndr=50
 ndw=51
/
&SYSTEM
 ibrav=  14,
 celldm(1)= 21.694d0, celldm(2)= 1.00D0, celldm(3)= 2.121D0,
 celldm(4)= 0.0d0,   celldm(5)= 0.0d0, celldm(6)= 0.0d0,
 nat= 96,
 ntyp= 1,
 nspin=1,
 ecutwfc= 15,
 nbnd=160,
 input_dft = 'pbe'
 occupations= 'ensemble',
 smearing='cs',
 degauss=0.018,
/
&ELECTRONS
 orthogonalization = 'Gram-Schmidt',
 startingwfc = 'random',
 ampre = 0.02,
 tcg = .true.,
 passop= 0.3,
 maxiter = 250,
 emass_cutoff = 3.00,
 conv_thr=1.d-6
 n_inner = 2,
 lambda_cold = 0.03,
 niter_cold_restart = 2,
/
&IONS
 ion_dynamics  = 'verlet',
 ion_temperature = 'nose'
 fnosep = 4.0d0,
```

```
 tempw = 500.d0
/
ATOMIC_SPECIES
 Al 26.89 Al.pbe.UPF
```

NOTA1 remember that the time step is to integrate the ionic dynamics, so you can choose something in the range of 1-5 fs.
NOTA2 with eDFT you are simulating metals or systems for which the occupation number is also fractional, so the number of band, `nbnd`, has to be chosen such as to have some empty states. As a rule of thumb, start with an initial occupation number of about 1.6-1.8 (the more bands you consider, the more the calculation is accurate, but it also takes longer. The CPU time scales almost linearly with the number of bands.)
NOTA3 the parameter `emass_cutoff` is used in the preconditioning and it has a completely different meaning with respect to plain CP. It ranges between 4 and 7.

All the other parameters have the same meaning in the usual `CP` input, and they are discussed above.

8.4.3 Treatment of USPPs

The cutoff `ecutrho` defines the resolution on the real space FFT mesh (as expressed by `nr1`, `nr2` and `nr3`, that the code left on its own sets automatically). In the USPP case we refer to this mesh as the "hard" mesh, since it is denser than the smooth mesh that is needed to represent the square of the non-norm-conserving wavefunctions.

On this "hard", fine-spaced mesh, you need to determine the size of the cube that will encompass the largest of the augmentation charges - this is what `nr1b`, `nr2b`, `nr3b` are. hey are independent of the system size, but dependent on the size of the augmentation charge (an atomic property that doesn't vary that much for different systems) and on the real-space resolution needed by augmentation charges (rule of thumb: `ecutrho` is between 6 and 12 times `ecutwfc`).

The small boxes should be set as small as possible, but large enough to contain the core of the largest element in your system. The formula for estimating the box size is quite simple:

$$\texttt{nr1b} = 2R_c/L_x \times \texttt{nr1}$$

and the like, where R_{cut} is largest cut-off radius among the various atom types present in the system, L_x is the physical length of your box along the x axis. You have to round your result to the nearest larger integer. In practice, `nr1b` etc. are often in the region of 20-24-28; testing seems again a necessity.

The core charge is in principle finite only at the core region (as defined by some R_{rcut}) and vanishes out side the core. Numerically the charge is represented in a Fourier series which may give rise to small charge oscillations outside the core and even to negative charge density, but only if the cut-off is too low. Having these small boxes removes the charge oscillations problem (at least outside the box) and also offers some numerical advantages in going to higher cut-offs." (info by Nicola Marzari)

9 Performances

9.1 Execution time

Since v.4.2 QUANTUM ESPRESSO prints real (wall) time instead of CPU 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

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

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`; $N_{r1} N_{r2} N_r 3 \sim 8 N_g$, 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.

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).

9.2 Memory requirements

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

$$O = mMN + PN + pN_1 N_2 N_3 + qN_{r1} N_{r2} N_{r3}$$

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.

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

9.3 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:

$$O = N_k MN + qN_{r1} N_{r2} N_{r3}$$

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.

9.4 Parallelization issues

`pw.x` and `cp.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.3.2);
- 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.

As a general rule, image parallelization:

- may give good scaling, but the slowest image will determine the overall performances ("load balancing" may be a problem);
- requires very little communications (suitable for ethernet communications);
- does not reduce the required memory per processor (unsuitable for large-memory jobs).

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);
- does not reduce the required memory per processor (unsuitable for large-memory jobs).

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` (ćp.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.

10 Troubleshooting

Almost all problems in QUANTUM ESPRESSO arise from incorrect input data and result in error stops. Error messages should be self-explanatory, but unfortunately this is not always true. If the code issues a warning messages and continues, pay attention to it but do not assume that something is necessarily wrong in your calculation: most warning messages signal harmless problems.

10.1 pw.x problems

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 the " " in the definitions of 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 -inp` and the input file name (see Sec.3.2), 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 just reports *Error while reading XXX namelist* and no further useful information. Here are some more subtle sources of trouble:

- Out-of-bound indices in dimensioned variables read in the namelists;
- Input data files containing ^M (Control-M) characters at the end of lines, or non-ASCII characters (e.g. non-ASCII quotation marks, that at a first glance may look the same as the ASCII character). Typically, this happens with files coming from Windows or produced with "smart" editors.

Both 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.
- 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. Known cases: HP-Compaq alphas with cxml libraries, Mac OS-X with system BLAS/LAPACK. 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.
- reduce `nbnd` to the strict minimum, or reduce the cutoffs, or the cell size , or a combination of them
- use conjugate-gradient (`diagonalization='cg'`: slow but very robust): it requires less memory than the default Davidson algorithm. If you stick to the latter, use `diago_david_ndim=2`.
- 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: parallelization with respect to R- (and G-) space does.
- IBM only (32-bit machines): if you need more than 256 MB you must specify it at link time (option `-bmaxdata`).
- buggy or weird-behaving compiler. Some versions of the Portland and Intel compilers on Linux PCs or clusters have this problem. For Intel ifort 8.1 and later, the problem seems to be due to the allocation of large automatic arrays that exceeds the available stack. Increasing the stack size (with command `limits` or `ulimit`) may solve the problem. Versions > 3.2 try to avoid this problem by removing the stack size limit at startup. See: `http://www.democritos.it/pipermail/pw_forum/2007-September/007176.html`, `http://www.democritos.it/pipermail/pw_forum/2007-September/007179.html`.

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.
- 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 '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 resutls 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 numers (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 `Modules/fft_scalar.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/eqvect.f90`, the variable `accep` is used to decide whether a rotation is a symmetry operation. Its current value (10^{-5}) 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.

Warning: symmetry operation # N not allowed This is not an error. If a symmetry operation contains a fractional translation that is incompatible with the FFT grid, it is discarded in order to prevent problems with symmetrization. Typical fractional translations are 1/2 or 1/3 of a lattice vector. If the FFT grid dimension along that direction is not divisible respectively by 2 or by 3, the symmetry operation will not transform the FFT grid into itself.

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.

10.2 PostProc

Some postprocessing codes complain that they do not find some files For Linux PC clusters in parallel execution: in at least some versions of MPICH, the current directory is set to the directory where the executable code resides, instead of being set to the directory where the code is executed. This MPICH weirdness may cause unexpected failures in some postprocessing codes that expect a data file in the current directory. Workaround: use symbolic links, or copy the executable to the current directory.

***error in davcio* in postprocessing codes** Most likely you are not reading the correct data files, or you are not following the correct procedure for postprocessing. In parallel execution: if you did not set `wf_collect=.true.`, the number of processors and pools for the phonon run should be the same as for the self-consistent run; all files must be visible to all processors.

10.3 ph.x errors

ph.x stops with *error reading file* The data file produced by `pw.x` is bad or incomplete or produced by an incompatible version of the code. In parallel execution: if you did not set `wf_collect=.true.`, the number of processors and pools for the phonon run should be the same as for the self-consistent run; all files must be visible to all processors.

ph.x mumbles something like *cannot recover* or *error reading recover file* You have a bad restart file from a preceding failed execution. Remove all files `recover*` in `outdir`.

ph.x says *occupation numbers probably wrong* and continues You have a metallic or spin-polarized system but occupations are not set to `'smearing'`.

ph.x does not yield acoustic modes with zero frequency at $q = 0$ This may not be an error: the Acoustic Sum Rule (ASR) is never exactly verified, because the system is never exactly translationally invariant as it should be. The calculated frequency of the acoustic mode is typically less than 10 cm^{-1}, but in some cases it may be much higher, up to 100 cm^{-1}. The ultimate test is to diagonalize the dynamical matrix with program `dynmat.x`, imposing the ASR. If you obtain an acoustic mode with a much smaller ω (let us say $< 1\mathrm{cm}^{-1}$) with all other modes virtually unchanged, you can trust your results.

"The problem is [...] in the fact that the XC energy is computed in real space on a discrete grid and hence the total energy is invariant (...) only for translation in the FFT grid. Increasing the charge density cutoff increases the grid density thus making the integral more exact thus reducing the problem, unfortunately rather slowly...This problem is usually more severe for GGA than with LDA because the GGA functionals have functional forms that vary more strongly with the position; particularly so for isolated molecules or system with significant portions of "vacuum" because in the exponential tail of the charge density a) the finite cutoff (hence there is an effect due to cutoff) induces oscillations in rho and b) the reduced gradient is diverging."(info by Stefano de Gironcoli, June 2008)

ph.x yields really lousy phonons, with bad or negative frequencies or wrong symmetries or gross ASR violations Possible reasons

- if this happens only for acoustic modes at $q = 0$ that should have $\omega = 0$: Acoustic Sum Rule violation, see the item before this one.
- wrong data file read.
- wrong atomic masses given in input will yield wrong frequencies (but the content of file fildyn should be valid, since the force constants, not the dynamical matrix, are written to file).
- convergence threshold for either SCF (`conv_thr`) or phonon calculation (`tr2_ph`) too large: try to reduce them.
- maybe your system does have negative or strange phonon frequencies, with the approximations you used. A negative frequency signals a mechanical instability of the chosen structure. Check that the structure is reasonable, and check the following parameters:

– The cutoff for wavefunctions, `ecutwfc`

– For USPP: the cutoff for the charge density, `ecutrho`

– The k-point grid, especially for metallic systems.

Note that "negative" frequencies are actually imaginary: the negative sign flags eigenvalues of the dynamical matrix for which $\omega^2 < 0$.

***Wrong degeneracy* error in star_q** Verify the q-vector for which you are calculating phonons. In order to check whether a symmetry operation belongs to the small group of q, the code compares q and the rotated q, with an acceptance tolerance of 10^{-5} (set in routine `PW/eqvect.f90`). You may run into trouble if your q-vector differs from a high-symmetry point by an amount in that order of magnitude.

11 Frequently Asked Questions (FAQ)

11.1 General

If you search information on QUANTUM ESPRESSO, the best starting point is the web site `html://www.quantum-espresso.org`. See in particular the links "learn" for documentation, "contacts" if you need somebody to talk with. The mailing list `pw_forum` is the typical place where to ask questions about QUANTUM ESPRESSO.

11.2 Installation

Most installation problems have obvious origins and can be solved by reading error messages and acting accordingly. Sometimes the reason for a failure is less obvious. In such a case, you should look into Sec.2.2, and into the `pw_forum` archive to see if a similar problem (with solution) is described. If you get really weird error messages during installation, look for them with your preferred Internet search engine (such as Google): very often you will find an explanation and a workaround.

What Fortran compiler do I need to compile QUANTUM ESPRESSO? Any non-buggy, or not-too-buggy, fortran-95 compiler should work, with minimal or no changes to the code. `configure`may not be able to recognize your system, though.

Why is `configure` saying that I have no fortran compiler? Because you haven't one (really!); or maybe you have one, but it is not in your execution path; or maybe it has been given an unusual name by your system manager. Install a compiler if you have none; if you have one, fix your execution path, or define an alias if it has a strange name.

Why is `configure` saying that my fortran compiler doesn't work? Because it doesn't work (really!); more exactly, `configure` has tried to compile a small test program and didn't succeed. Your compiler may not be properly installed. For Intel compiler on PC's: you may have forgotten to run the required initialization script for the compiler.

`configure` doesn't recognize my system, what should I do? If compilation/linking works, never mind, Otherwise, try to supply a suitable supported architecture, or/and manually edit the `make.sys` file. Detailed instructions in Sec.2.2.

Why doesn't `configure` recognize that I have a parallel machine? You need a properly configured complete parallel environment. If any piece is missing, `configure` will revert to serial compilation. Detailed instructions in Sec.2.2.

Compilation fails with *internal error*, what should I do? Any message during compilation saying something like *internal compiler error* and the like means that your compiler is buggy. You should report the problem to the compiler maker – especially if you paid real money for it. Sometimes reducing the optimization level, or rearranging the code in a strategic place, will make the problem disappear. In other cases you will need to move to a different compiler, or to a less buggy version (or buggy in a different way that doesn't bug you) of the same compiler.

Compilation fails at linking stage: *symbol ... not found* If the missing symbols (i.e. routines that are called but not found) are in the code itself: most likely the fortran-to-C conventions used in file `include/c_defs.h` are not appropriate. Edit this file and retry.

If the missing symbols are in external libraries (BLAS, LAPACK, FFT, MPI libraries): there is a name mismatch between what the compiler expects and what the library provides. See Sec.2.2).

If the missing symbols aren't found anywhere either in the code or in the libraries: they are system library symbols. i) If they are called by external libraries, you need to add a missing system library, or to use a different set of external libraries, compiled with the same compiler you are using. ii) If you are using no external libraries and still getting missing symbols, your compiler and compiler libraries are not correctly installed.

11.3 Pseudopotentials

Can I mix USPP/NCPP/PAW ? Yes, you can (if implemented, of course: a few kinds of calculations are not available with USPP, a few more are not for PAW). A small restrictions exists in `cp.x`, expecting atoms with USPP listed before those with NCPP, which in turn are expected before local PP's (if any). Otherwise you can mix and match, as long as the XC functional used in the generation of the PP is the same for all PPs. Note that it is the hardest atom that determines the cutoff.

Where can I find pseudopotentials for atom X? First, a general rule: when you ask for a pseudopotential, you should always specify which kind of PP you need (NCPP, USPP PAW, full- or scalar-relativistic, for which XC functional, and for many elements, with how many electrons in valence). If you do not find anything suitable in the "pseudo" page of the web site links, we have bad news for you: you have to produce it by yourself. You can use the `atomic` code: have a look first at the contents of the library of input data in `atomic_doc/pseudo_gen`. Otherwise, you can use any other code producing a file format that is either recognized by Quantum ESPRESSO or for which a converter to the UPF format exists. New contributions to the PP table are very appreciated (and very scarce).

Where can I find pseudopotentials for rare-earth X? Please consider first if DFT is suitable for your system! In many cases, it isn't (at least "plain" DFT: GGA and the like). If you are still convinced that it is, see above.

Is there a converter from format XYZ to UPF? What is available (no warranty) is in directory `upftools/`. You are most welcome to contribute a new converter.

11.4 Input data

A large percentage of the problems reported to the mailing list are caused by incorrect input data. Before reporting a problem with strange crashes or strange results, *please* have a look at your structure with XCrySDen. XCrySDen can directly visualise the structure from both `PWscf` input data:

```
xcrysden --pwi "input-data-file"
```

and from `PWscf` output as well:

```
xcrysden --pwo "output-file".
```

Unlike most other visualizers, XCrySDen is periodicity-aware: you can easily visualize periodically repeated cells. You are advised to always use XCrySDen to check your input data!

Where can I find the crystal structure/atomic positions of XYZ? The following site contains a lot of crystal structures: `http://cst-www.nrl.navy.mil/lattice`.
"Since this seems to come up often, I'd like to point out that the American Mineralogist Crystal Structure Database (`http://rruff.geo.arizona.edu/AMS/amcsd`) is another excellent place to find structures, though you will have to use it in conjunction with the Bilbao crystallography server (`http://www.cryst.ehu.es`), and have some understanding of space groups and Wyckoff positions".

How can I generate a supercell? If you need to create a supercell and are too lazy to create a small program to translate atoms, you can

- "use the 'spacegroup' program in EXCITING package (http://exciting-code.org) to generate the supercell, use 'fropho' (http://fropho.sourceforge.net) to check the symmetry" (Kun Yin, April 2009)
- "use the PHON code: http://chianti.geol.ucl.ac.uk/~dario/" (Eyvaz Isaev, April 2009).

Where can I find the Brillouin Zone/high-symmetry points/irreps for XYZ? "You might find this web site useful: `http://www.cryst.ehu.es/cryst/get_kvec.html`" (info by Cyrille Barreteau, nov. 2007). Or else: in textbooks, such as e.g. *The mathematical theory of symmetry in solids* by Bradley and Cracknell.

Where can I find Monkhorst-Pack grids of k-points? Auxiliary code `kpoints.x`, found in `pwtools/` and produced by `make tools`, generates uniform grids of k-points that are equivalent to Monkhorst-Pack grids.

11.5 Parallel execution

Effective usage of parallelism requires some basic knowledge on how parallel machines work and how parallelism is implemented in QUANTUM ESPRESSO. If you have no experience and no clear ideas (or not idea at all), consider reading Sec.3.

How do I choose the number of processors/how do I setup my parallel calculation? Please see above.

Why is my parallel job running in such a lousy way? A frequent reason for lousy parallel performances is a conflict between MPI parallelization (implemented in QUANTUM ESPRESSO) and the autoparallelizing feature of MKL libraries. Set the environment variable `OPEN_MP_THREADS` to 1. See Sec.3 for more info.

Why is my parallel job crashing when reading input data / doing nothing? If the same data work in serial execution, use `code -inp input_file` instead of `code < input_file`. Some MPI libraries do not properly handle input redirection.

The code stops with an *error reading namelist xxxx* Most likely there is a misspelled variable in namelist xxxx. If there isn't any (have you looked carefully? really?? REALLY???), beware control characters like DOS control-M: they can confuse the namelist-reading code. If this happens to the first namelist to be read (usually "&CONTROL") in parallel execution, see above.

Why is my parallel job crashing with mysterious errors? Mysterious, unpredictable, erratic errors in parallel execution are almost always coming from bugs in the compiler or/and in the MPI libraries and sometimes even to flacky hardware. Sorry, not our fault.

11.6 Frequent errors during execution

Why is the code saying *Wrong atomic coordinates*? Because they are: two or more atoms in the list of atoms have overlapping, or anyway too close, positions. Can't you see why? look better (or use XCrySDen: see above) and remember that the code checks periodic images as well.

The code stops with an *error in davcio* Possible reasons: disk is full; `outdir` is not writable for any reason; you changed some parameter(s) in the input (like `wf_collect`, or the number of processors/pools) without doing a bit of cleanup in your temporary files; you were running more than one instance of `pw.x` in the same temporary directory with the same file names.

The code stops with a *wrong charge* error In most cases: you are treating a metallic system as if it were insulating.

11.7 Self Consistency

What are the units for quantity XYZ? Unless otherwise specified, all `PWscf` input and output quantities are in atomic "Rydberg" units, i.e. energies in Ry, lengths in Bohr radii, etc.. Note that `CP` uses instead atomic "Hartree" units: energies in Ha, lengths in Bohr radii.

Self-consistency is slow or does not converge at all In most cases: your input data is bad, or else your system is metallic and you are treating it as an insulator. If this is not the case: reduce `mixing_beta` to $\sim 0.3 \div 0.1$ or smaller, try the `mixing_mode` value that is more appropriate for your problem.

What is the difference between total and absolute magnetization? The total magnetization is the integral of the magnetization in the cell:

$$M_T = \int (n_{up} - n_{down}) d^3r.$$

The absolute magnetization is the integral of the absolute value of the magnetization in the cell:

$$M_A = \int |n_{up} - n_{down}| d^3r.$$

In a simple ferromagnetic material they should be equal (except possibly for an overall sign)'. In simple antiferromagnets (like FeO, NiO) M_T is zero and M_A is twice the magnetization of each of the two atoms. (info by Stefano de Gironcoli)

How can I calculate magnetic moments for each atom? There is no 'right' way of defining the local magnetic moment around an atom in a multi-atom system. However an approximate way to define it is via the projected density of states on the atomic orbitals (code projwfc.x, see example08 for its use as a postprocessing tool). This code generate many files with the density of states projected on each atomic wavefunction of each atom and a BIG amount of data on the standard output, the last few lines of which contain the decomposition of Lowdin charges on angular momentum and spin component of each atom.

What is the order of Y_{lm} components in projected DOS / projection of atomic wavefunctions? See input data documentation for `projwfc.x`.

Why is the sum of partial Lowdin charges not equal to the total charge? "Lowdin charges (as well as other conventional atomic charges) do not satisfy any sum rule. You can easily convince yourself that ths is the case because the atomic orbitals that are used to calculate them are arbitrary to some extent. If yu like, you can think that the missing charge is "delocalized" or "bonding" charge, but this would be another way of naming the conventional (to some extent) character of Lowdin charge." (Stefano Baroni, Sept. 2008).

See also the definition of "spilling parameter": Sanchez-Portal et al., Sol. State Commun. 95, 685 (1995). The spilling parameter measures the ability of the basis provided by the pseudo-atomic wfc to represent the PW eigenstates, by measuring how much of the subspace of the Hamiltonian eigenstates falls outside the subspace spanned by the atomic basis.

I cannot find the Fermi energy, where is it? It is printed in the output. If not, the information on Gaussian smearing, needed to calculate a sensible Fermi energy, was not provided in input. In this case, `pw.x` prints instead the highest occupied and lowest unoccupied levels. If not, the number of bands to be calculated was not provided in input and `pw.x` calculates occupied bands only.

What is the reference level for Kohn-Sham energies? Why do I get positive values for Kohn-Sham levels? The reference level is an ill-defined quantity in calculations in solids with periodic boundary conditions. Absolute values of Kohn-Sham eigenvalues are meaningless.

Why do I get a strange value of the Fermi energy? "The value of the Fermi energy (as well as of any energy, for that matter) depends of the reference level. What you are referring to is probably the "Fermi energy referred to the vacuum level" (i.e. the work function). In order to obtain that, you need to know what the vacuum level is, which cannot be said from a bulk calculation only" (Stefano Baroni, Sept. 2008).

Why I don't get zero pressure/stress at equilibrium? It depends. If you make a calculation with fixed cell parameters, you will never get exactly zero pressure/stress, unless you use the cell that yields perfect equilibrium for your pseudopotentials, cutoffs, k-points, etc.. Such cell will anyway be slightly different from the experimental one. Note however that pressures/stresses in the order of a few KBar correspond to very small differences in terms of lattice parameters.

If you obtain the equilibrium cell from a variable-cell optimization, do not forget that the pressure/stress calculated with the modified kinetic energy functional (very useful for variable-cell calculations) slightly differ from those calculated without it. Also note that the PW basis set used during variable-cell calculations is determined by the given cutoff and the *initial* cell. If you make a calculation with the final geometry at the same cutoff, you may get slightly different results. The difference should be small, though, unless you are using a too low cutoff for your system.

Why do I get *negative starting charge*? Self-consistency requires an initial guess for the charge density in order to bootstrap the iterative algorithm. This first guess is usually built from a superposition of atomic charges, constructed from pseudopotential data.

More often than not, this charges are a slightly too hard to be expanded very accurately in PWs, hence some aliasing error will be introduced. Especially if the unit cell is big and mostly empty, some local low negative charge density will be produced.

"This is NOT harmful at all, the negative charge density is handled properly by the code and will disappear during the self-consistent cycles", but if it is very high (let's say more than 0.001*number of electrons) it may be a symptom that your charge density cutoff is too low. (L. Paulatto - November 2008)

How do I calculate the work function? Work function = (average potential in the vacuum) - (Fermi Energy). The former is estimated in a supercell with the slab geometry, by looking at the average of the electrostatic potential (typically without the XC part). See the example in examples/WorkFct_example.

11.8 Phonons

Is there a simple way to determine the symmetry of a given phonon mode? A symmetry analyzer was added in v.3.2 by Andrea Dal Corso. Other packages that perform symmetry analysis of phonons and normal modes:
ISOTROPY package: http://stokes.byu.edu/iso/isotropy.html
ACKJ, ACMI packages: http://www.cpc.cs.qub.ac.uk.

I am not getting zero acoustic mode frequencies, why? Because the Acoustic Sum Rule (ASR), i.e. the translational invariance, is violated in approximated calculations. In PW calculations, the main and most irreducible violation comes from the discreteness of the FFT grid. There may be other reasons, though, notably insufficient convergence: "Recently I found that the parameters `tr2_ph` for the phonons and `conv_thr` for the ground state can affect the quality of the phonon calculation, especially the "vanishing" frequencies for molecules." (Info from Katalyn Gaal-Nagy). Anyway: if the nonzero frequencies are small, you can impose the ASR to the dynamical matrix, usually with excellent results.

Nonzero frequencies for rotational modes of a molecule are a fictitious effect of the finite supercell size, or else, of a less than perfect convergence of the geometry of the molecule.

Why do I get negative phonon frequencies? "Negative" frequencies actually are "imaginary" frequencies ($\omega^2 < 0$). If these occur for acoustic frequencies at Gamma point, or for rotational modes of a molecule, see above. In all other cases: it depends. It may be a problem of bad convergence (see above), or it may signal a real instability.

Why do I get a message *no elec. field with metals*? If you want to calculate the contribution of macroscopic electric fields to phonons – a quantity that is well-defined in insulators only — you cannot use smearing in the scf calculation, or else the code will complain.

How can I calculate Raman/IR coefficients in metals? You cannot: they are well defined only for insulators.

How can I calculate the electron-phonon coefficients in insulators? You cannot: the current implementation is for metals only.