CP's user guide (v.7.2)

Contents

1 Introduction

This guide covers the usage of the CP package, a core component of the Quantum ESPRESSO distribution. Further documentation, beyond what is provided in this guide, can be found in the directory CPV/Doc/, containing a copy of this guide.

This guide assumes that you know the physics that CP describes and the methods it implements. A good reference for the topic is the book D, Marx and J. Hutter, "Ab Initio Molecular Dynamics" It also assumes that you have already installed, or know how to install, Quantum ESPRESSO. If not, please read the general User's Guide for Quantum ESPRESSO, found in directory Doc/ two levels above the one containing this guide; or consult the web site: www.quantum-espresso.org.

People who want to modify or contribute to CP should read the Developer Manual: gitlab.com/QEF/q-e/-/wikis/home.

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

- Sergio Orlandini (CINECA) for completing the CUDA Fortran acceleration started by Carlo Cavazzoni
- Fabio Affinito and Mariella Ippolito (CINECA) for testing and benchmarking
- Ivan Carnimeo and Pietro Delugas (SISSA) for further openACC acceleration
- Riccardo Bertossa (SISSA) for extensive refactoring of ensemble dynamics / conjugate gradient part, contributions to the documentation
- Federico Grasselli and Riccardo Bertossa (SISSA) for bug fixes, extensions to Autopilot;
- Biswajit Santra, Hsin-Yu Ko, Marcus Calegari Andrade (Princeton) for various contribution, notably the SCAN functional;
- Robert DiStasio (Cornell), Biswajit Santra, and Hsin-Yu Ko for hybrid functionals with MLWF; (maximally localized Wannier functions);
- Manu Sharma (Princeton) and Yudong Wu (Princeton) for dynamics with MLWF;
- Paolo Umari (Univ. Padua) for finite electric fields and conjugate gradients;
- Paolo Umari and Ismaila Dabo (Penn State) for ensemble-DFT;
- Xiaofei Wang (Princeton) for META-GGA;
- The Autopilot feature was implemented by Targacept, Inc.

The original version of this guide was mostly written by Gerardo Ballabio and Carlo Cavazzoni.

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

> P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N.

Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, R. M. Wentzcovitch, J.Phys.: Condens.Matter 21, 395502 (2009)

and

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

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

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

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

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

2 Compilation

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

Typing `make cp` from the main Quantum ESPRESSO directory or `make` from the `CPV/` subdirectory produces the following codes in `CPV/src`:

- `cp.x`: Car-Parrinello Molecular Dynamics code
- `cppp.x`: postprocessing code for `cp.x`. See `Doc/INPUT_CPPP.*` for input variables.
- `wfdd.x`: utility code for finding maximally localized Wannier functions using damped dynamics.

Symlinks to executable programs will be placed in the `bin/` subdirectory.

As a final check that compilation was successful, you may want to run some or all of the tests and examples. Automated tests for `cp.x` are in directory `test-suite/` and can be run via the `Makefile` found there. Please see the general User's Guide for their setup.

You may take the tests and examples distributed with `CP` as templates for writing your own input files. Input files for tests are contained in subdirectories `test-suite/cp_*` with file type `*.in1`, `*.in2`, ... Input files for examples are produced, if you run the examples, in the `results/` subdirectories, with names ending with `.in`.

For general information on parallelism and how to run in parallel execution, please see the general User's Guide. `CP` currently can take advantage of both MPI and OpenMP parallelization and on GPU acceleration. The "plane-wave", "linear-algebra" and "task-group" parallelization levels are implemented.

3 Input data

Input data for `cp.x` is organized into several namelists, followed by other fields ("cards") introduced by keywords. The namelists are:

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

The `&CELL` namelist may be omitted for fixed-cell calculations. This depends on the value of variable `calculation` in namelist &CONTROL. Most variables in namelists have default values. Only he 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

Explanations for the meaning of variables `ibrav` and `celldm`, as well as on alternative ways to input structural data, are contained in files `Doc/INPUT_CP.*`. These files are the reference for input data and describe a large number of other variables as well. Almost 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
CELL_PARAMETERS (optional)
OCCUPATIONS (optional) AUTOPILOT (optional)

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

Comment lines in namelists can be introduced by a "!", exactly as in fortran code. Comments lines in "cards" can be introduced by either a "!" or a "#" character in the first position of a line.

3.1 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='cp_$ndw'`, where `ndw` is an integer specified in input). In order to use the data on a different machine, you may need to compile `CP` with HDF5 enabled.

The execution stops if you create a file `prefix.EXIT` either in the working directory (i.e. where the program is executed), or in the `outdir` directory. Note that with some versions of MPI, the working directory is the directory where the executable is! The advantage of this procedure is that all files are properly closed, whereas just killing the process may leave data and output files in an unusable state.

The format of arrays containing charge density, potential, etc. is described in the developer manual.

4 Output files

The `cp.x` code produces many output files, that together build up the trajectory.

You have a file for the positions, called `prefix.pos`, where `prefix` is defined in the input file, that is formatted like:

```
10     0.00157227
0.48652245874924E+01      0.38015905345591E+01      0.37361508020082E+01
0.40077990926697E+01      0.59541011690914E+01      0.34691399577808E+01
0.43874410242643E+01      0.38553718662714E+01      0.59039702898524E+01
20     0.00641004
0.49677092782926E+01      0.38629427979469E+01      0.37777995137803E+01
0.42395189282719E+01      0.55766875434652E+01      0.31291744042209E+01
0.45445534106843E+01      0.36049553522533E+01      0.55864387532281E+01
```

where the first line contains the step number and elapsed time, in ps, at this step; the following lines contain the positions, in Bohr radii, of all the atoms (3 in this examples), in the same order as in the input file (since v6.6 – previously, atoms were sorted by type; the type must be deduced from the input file). The same structure is repeated for the second step and so on. The printout is made every `iprint` steps (10 in this case, so at step 10, 20, etc.). Note that the atomic coordinates are not wrapped into the simulation cell, so it is possible that they lie outside it.

The velocities are written in a similar file named `prefix.vel`, where `prefix` is defined in the input file, that is formatted like the `.pos` file. The units are the usual Hartree atomic units (note that the velocities in the `pw.x` code are in *Rydberg* a.u. and differ by a factor 2).

The `prefix.for` file, formatted like the previous two, contains the computed forces, in Hartree atomic units as well. It is written only if a molecular dynamics calculation is performed, or if `tprnfor = .true.` is set in input.

In the `prefix.str` file you can find the stress tensor (with the ionic kinetic part included), in GPa.

The simulation cell is written in a file named `prefix.cel` with the same header as the previous described files, and the cell matrix is then listed. NB: **THE CELL MATRIX IN THE OUTPUT IS TRANSPOSED** that means that if you want to reuse it again for a new input file, you have to pick the one that you find in `prefix.cel` and write in the input file after inverting rows and columns. In the `prefix.cel` file you will find the cell vectors in the **columns** of the matrix.

The file `prefix.evp` has one line per printed step and contains some thermodynamics data. The first line of the file names the columns:

```
#   nfi  time(ps)  ekinc  Tcell(K)  Tion(K)  etot  enthal  econs  econt  Volume
Pressure(GPa)
```

where: - `ekinc` is the electrons fictitious kinetic energy, $K_{ELECTRONS}$ - `enthal` is the enthalpy, $E_{DFT}+PV$ - `etot` is the DFT (potential) energy of the system, E_{DFT} - `econs` is a physically meaningful constant of motion, $E_{DFT} + K_{NUCLEI}$, in the limit of zero electronic fictitious mass - `econt` is the constant of motion of the Lagrangian$E_{DFT} + K_{IONS} + K_{ELECTRONS}$ t. If the time step `dt` is small enough this will be up to a very good precision a constant. It is not a physical quantity, since $K_{ELECTRONS}$ has *nothing* to do with the quantum kinetic energy of the electrons.

5 Using `CP`

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. to modify the temperature or the pressure.

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

atomic positions
atomic velocities (can be zero, read from the input, sampled from Maxwell-Boltzmann)
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.*`.

The best way to initialize and run the simulation is first doing a few steps of Born-Oppenheimer (BO) molecular dynamics (MD) using the `electron_dynamic = 'cg'` input for minimizing the electronic degrees of freedom each MD step, after sampling the ionic velocities from a Maxwell-Boltzmann distribution. Then usually you run a Nosé thermostat and/or a barostat to reach some temperature and pressure that you choose. Then you can run in the microcanonical ensemble if needed. During the CP MD you must check that the fake electronic kinetic energy does not increase too much, otherwise at some point your simulation may become completely wrong.

5.1 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. The suggested method is to use the `electron_dynamics = 'cg'` to initialize the simulation.

Sometimes when you do not use the CG routine, 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. You can choose among the following:

- *conjugate gradient* (CG): direct minimization of the energy functional `electron_dynamics = 'cg'`
- damped dynamics: `electron_dynamics = 'damp'` and `electron_damping` = a number typically ranging from 0.1 and 0.5.
- Steepest descent: `electron_dynamics = 'sd'`

The CG routine computes also the wavefunction time derivative in the parallel transport gauge. The time derivative is performed by computing twice the ground state using the atomic positions at the current timestep, and the atomic positions at the next timestep, that are computed by moving the atoms with their atomic velocities. This procedure allows to start the dynamics with a wavefunction velocity consistent with the atomic velocities. This usually results in a smoother start of the simulation. An other advantage of this technique is that you can apply it also in the middle of the dynamics introducing a much lower discontinuity in the trajectory than apply a minimization only and settings again to zero the wavefunction velocity. All the other minimization routines in CP when called set the wavefunction velocity to zero. At the moment the CG routine works only with the plane wave parallelization scheme, both on CPU and GPU machines. It can run "on the fly" by using the autopilot module.

See the input description to compute the optimal damping factor of the damped dynamics routine. Steepest descent is also available but it is typically slower than damped dynamics and should be used only to start the minimization.

Note that the CG routine is used also for doing a Born-Oppenheimer dynamic, or for using the ensemble DFT molecular dynamics. For this reason the meaning of the input variable `nstep` is different. In the CG minimization `nstep` is the number of ions dynamic step: the conjugate gradient algorithm stops when the required accuracy is reached, and the maximum number of steps that the CG algorithm performs is the input variable `maxiter`. In the damped dynamics and the steepest descent the `nstep` input variable is the number of sd/damped dynamics steps performed to reach the ground state, and ions usually are kept fixed.

5.2 Relax the system

If the input atomic positions are far from the equilibrium position you need to relax the system. Note that you can consider to do that with the pw.x code that may be faster for this task. Keep in mind that a recipe that works for every case does not exist, and it is impossible to predict what will happen in an arbitrary system without running the simulation. The best approach is to try and see what happens. Here we give some ideas on a possible approach to the simulation.

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 or a few step of CG dynamic with a very small timestep to equilibrate a little the simulation, otherwise the system could break up during the CP 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. You will need to set the atomic velocities from input or let the code chose them by sampling a Maxwell-Boltzmann distribution at a given temperature. An alternative approach is also to randomize a little bit the atomic positions.

Let us consider case 1). Suppose that you decided to minimize the system with the cp.x code. There are different strategies to relax the system in cp.x, 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 conjugate-gradient.

- 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.

- 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
  ```

- 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). You can also use the CG dynamics for the electrons, that take care of fully minimizing the electronic energy.

  You can perform a ionic step every N electronic steps done with 'sd' or 'damp', 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. This process is not automatic, but is strongly based on experience, and trial and error. For this reason we suggest to use the CG minimization algorithm, that simplifies the process and usually works well, at least for the electronic part of the problem.

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.

A different approach is to use a small timestep and run the ion dynamic using ground states computed with the CG routine. The system, given a small enough timestep and a big enough number of steps, will thermalize at a defined temperature. Note that if the potential energy of the initial configuration is too big the final temperature may be too large. To run some steps of Born-Oppenheimer (BO) molecular dynamics you can set the following input variables:

```
        &control
           ! ...
           nstep = 100, ! number of MD steps. It has to be big enought
           dt    = 1.0d0, ! put a small number here...
           ! ...
        /
        !...
        &electrons
           electron_dynamics = 'cg'
        /
        !...
        &ions
          ion_dynamics = 'verlet' ! you also mix different approaches by changing
this to sd
        /
        !...
```

An additional feature of the code that you can use to perform the simulation initialization doing less restarts is the autopilot module (see the autopilot guide). For example you can play with the timestep *while the simulation is running* by writing a file called `pilot.mb` in the folder where the simulation is running with the following content:

```
PILOT
  NOW : DT = 0.5
  NOW + 10 : DT = 1.0
  NOW + 20 : DT = 5.0
ENDRULES
```

Let us now move to case 2.

If you have forces that are too small in your initial state, you can chose to initialize the simulation with random initial atomic velocities. The input variables are `ion_velocities = 'random'` and `tempw = 300.0` to sample from a 300K distribution. In this case the best approach is to perform the ground

state calculation and the small initial thermalization in a single run by performing few steps of BO dynamics using CG

```
         &control
            ! ...
            nstep = 50, ! number of MD steps. It has to be big enought
            dt    = 20.0d0, ! if the forces are small, you don't need a small
number...
            ! ...
         /
         !...
         &electrons
            electron_dynamics = 'cg'
         /
         !...
         &ions
           ion_dynamics = 'verlet' ! you also mix different approaches by changing
this to sd
           ion_velocities = 'random'
           tempw = 300.d0
         /
         !...
```

A different approach is to randomize the atomic positions, and set the ionic velocities to zero. 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 Kelvin, 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.

5.3 CP dynamics

At this point after having:

1. minimized the electrons
2. a good initial configuration of the ionic degrees of freedom (good positions and velocities)

3. optionally computed the initial wavefunction velocity with the CG routine

we are ready to start a CP dynamics.

The parameter specific to the CP method that you must choose very carefully is the electron fake mass `emass`. The fake electron mass is the parameter that change how the wavefunctions follow the minimum of the DFT energy during the dynamics. The smaller the better. In the limit where both the fake mass and the timestep are zero we recover a perfect minimization of the wavefunction at each timestep, but this would require an infinite amount of steps to simulate a trajectory of a given length in time. So we need a compromise between efficiency and precision. You should always check that the forces on the ions are well converged with respect to the emass parameter. What happens is that, the bigger the fake electron mass, the bigger the systematic error that you have on the forces. The error, intuitively, is related to the fact that the electrons have a finite classical mass, that is interacting with the ions through the DFT potential, so they have an inertia that adds up to the mass of the ion, slowing it down. This effect causes that, with a good approximation, the forces calculated with the CP method, on average, are the *true* ground state forces time a factor between 0.0 and 1.0 that depends on `emass`. You can correct the leading order of this error by changing the ionic mass by the same factor to recover the same inertia of the atom+electron classical object. Remember also that static properties of an equilibrium molecular dynamics simulation do not depend on the mass of the ions. So if only static properties like radial distribution function or lattice parameters are needed this leading order correction is not necessary, but it becomes important if you are computing, for example, a diffusion coefficient. For a detailed analysis of this aspect of the theory see for example P. Tangney, "On the theory underlying the Car-Parrinello method and the role of the fictitious mass parameter"(arXiv) or the Marx and Hutter book.

In practice, you can safely use `emass` of about 50 atomic units without affecting the dynamical properties too much.

Keep in mind that the lower the `emass` the lower the integration timestep `dt`, because the electronic degrees of freedom will move faster. You have to check that with your parameters the CP constant of motion in the output is conserved with a good approximation.

To run the CP simulation you 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. To start the simulation you must use the restart from the previous step. Restore the proper masses for the ions. In this way we will sample the microcanonical ensemble. If you did not use the CG routine (that computes electron velocities), the input section changes as follow:

```
&electrons
   emass = 50.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet',
   electron_velocities = 'zero' ! only if you did NOT use CG
/
&ions
   ion_dynamics = 'verlet',
   ion_velocities = 'zero' ! if you did NOT use CG!
/
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
```

If you used a method that computes the ionic velocities and the electronic velocities in a consistent way you can use the following:

```
&electrons
   emass = 50.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'verlet',
   electron_velocities = 'default' ! if you have wfc velocities
/
&ions
   ion_dynamics = 'verlet',
   ion_velocities = 'default' ! if you have velocities
/
ATOMIC_SPECIES
C 12.0d0 c_blyp_gia.pp
H 1.00d0 h.ps
```

If you want to change the timestep, for example because you used a big timestep in a small CG thermalization performed in the relaxation step, you must specify the following additional parameters:

```
&control
     dt    = 5.0d0, ! new integration timestep
/
&electrons
   electron_velocities = 'change_step' ! if you have wfc velocities
/
&ions
   ion_velocities = 'change_step' ! if you have velocities
   tolp = 20.d0 ! old integration timestep
/
```

If you want to specify a new set of initial velocities for ions, you have to set `ion_velocities ='from_input'`, and add the ATOMIC_VELOCITIES card, after the ATOMIC_POSITION card, with the list of velocities with time in atomic units, and length in the same units specified for the atomic positions.

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

```
&electrons
   emass = 50.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.

5.3.0.1 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.

5.3.0.2 Nośe 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."

5.4 Advanced usage

5.4.1 Autopilot features

For changing variables while the simulation is running see the autopilot guide

5.4.2 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 itself.

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,
tot_magnetization=1,
sic_alpha = 1.d0,
sic_epsilon = 1.0d0,
sic = 'sic_mac',
force_pairing = .true.,
```

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.

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)

5.4.3 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/src/ensemble_dft.f90` where you define all the quantities of interest. The subroutine `CPV/src/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 $1 \div$ `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 `PWscf`is 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.

5.4.4 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)

5.4.5 Hybrid functional calculations using maximally localized Wannier functions

In this section, we illustrate some guidelines to perform exact exchange (EXX) calculations using Wannier functions efficiently.

The references for this algorithm are:

- Theory: X. Wu , A. Selloni, and R. Car, Phys. Rev. B 79, 085102 (2009).
- Implementation: H.-Y. Ko, B. Santra, R. A. DiStasio, L. Kong, Z. Li, X. Wu, and R. Car, arxiv.

The parallelization scheme in this algorithm is based upon the number of electronic states. In the current implementation, there are certain restrictions on the choice of the number of MPI tasks. Also slightly different algorithms are employed depending on whether the number of MPI tasks used in the calculation are greater or less than the number of electronic states. We highly recommend users to follow the notes below. This algorithm can be used most efficiently if the numbers of electronic states are uniformly distributed over the number of MPI tasks. For a system having N electronic states the optimum numbers of MPI tasks (nproc) are the following:

- In case of nproc $\leq$ N, the optimum choices are N/m, where m is any positive integer.
  - Robustness: Can be used for odd and even number of electronic states.
  - OpenMP threads: Can be used.
  - Taskgroup: Only the default value of the task group (-ntg 1) is allowed.
- In case of nproc $>$ N, the optimum choices are N*m, where m is any positive integer.
  - Robustness: Can be used for even number of electronic states.
  - Largest value of m: As long as nj_max (see output) is greater than 1, however beyond m=8 the scaling may become poor. The scaling should be tested by users.
  - OpenMP threads: Can be used and highly recommended. We have tested number of threads starting from 2 up to 64. More threads are also allowed. For very large calculations (nproc $>$ 1000) efficiency can largely depend on the computer architecture and the balance between the MPI tasks and the OpenMP threads. User should test for an optimal balance. Reasonably good scaling can be achieved by using m=6-8 and OpenMP threads=2-16.
  - Taskgroup: Can be greater than 1 and users should choose the largest possible value for ntg. To estimate ntg, find the value of nr3x in the output and compute nproc/nr3x and take the integer value. We have tested the value of ntg as 2^m, where m is any positive integer. Other values of ntg should be used with caution.
  - Ndiag: Use -ndiag X option in the execution of cp.x. Without this option jobs may crash on certain architectures. Set X to any perfect square number which is equal to or less than N.
- DEBUG: The EXX calculations always work when number of MPI tasks = number of electronic states. In case of any uncertainty, the EXX energy computed using different numbers of MPI tasks can be checked by performing test calculations using number of MPI tasks = number of electronic states.

An example input is listed as following:

```
&CONTROL
  calculation       = 'cp-wf',
  title             = "(H2O)32 Molecule: electron minimization PBE0",
  restart_mode      = "from_scratch",
  pseudo_dir        = './',
  outdir            = './',
  prefix            = "water",
  nstep             = 220,
  iprint            = 100,
  isave             = 100,
```

```
  dt                = 4.D0,
  ekin_conv_thr     = 1.D-5,
  etot_conv_thr     = 1.D-5,
/
&SYSTEM
  ibrav             = 1,
  celldm(1)         = 18.6655,
  nat               = 96,
  ntyp              = 2,
  ecutwfc           = 85.D0,
  input_dft         = 'pbe0',
/
&ELECTRONS
  emass             = 400.D0,
  emass_cutoff      = 3.D0,
  ortho_eps         = 1.D-8,
  ortho_max         = 300,
  electron_dynamics = "damp",
  electron_damping  = 0.1D0,
/
&IONS
  ion_dynamics      = "none",
/
&WANNIER
  nit               = 60,
  calwf             = 3,
  tolw              = 1.D-6,
  nsteps            = 20,
  adapt             = .FALSE.
  wfdt              = 4.D0,
  wf_q              = 500,
  wf_friction       = 0.3D0,
  exx_neigh         = 60,     ! exx related optional
  exx_dis_cutoff    = 8.0D0,  ! exx related optional
  exx_ps_rcut_self  = 6.0D0,  ! exx related optional
  exx_ps_rcut_pair  = 5.0D0,  ! exx related optional
  exx_me_rcut_self  = 9.3D0,  ! exx related optional
  exx_me_rcut_pair  = 7.0D0,  ! exx related optional
  exx_poisson_eps   = 1.D-6,  ! exx related optional
/
ATOMIC_SPECIES
O 16.0D0 O_HSCV_PBE-1.0.UPF
H  2.0D0 H_HSCV_PBE-1.0.UPF
```

6 Parallel Performances

`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. 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.

We refer to the "Parallelization" section of the general User's Guide for a description of MPI and OpenMP parallelization paradigms, of the various MPI parallelization levels, and on how to activate them.

A judicious choice of the various levels of parallelization, together with the availability of suitable hardware (e.g. fast communications) is fundamental to reach good performances._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 or a bad parallelization style will yield performance degradation.

For `CP` with hybrid functionals, see the related section above this one. For all other cases, the relevant MPI parallelization levels are:

- "plane waves" (PW);
- "tasks" (activated by command-line option `-nt N`);
- "linear algebra" (`-nd N`);
- "bands" parallelization (`-nb N`), to be used only in special cases;
- "images" parallelization (`-ni N`), used only in code `manycp.x` (see the header of `CPV/src/manycp.f90` for documentation).

As a rule of thumb: - start with PW parallelization only (e.g. `mpirun -np N cp.x ...` with no other parallelization options); the code will scale well unless `N` exceeds the third FFT dimensions `nr3` and/or `nr3s`. - To further increase the number of processors, use "task groups", typically 4 to 8 (e.g. `mpirun -np N cp.x -nt 8 ...`). - Alternatively, or in addition, you may compile with OpenMP: `./configure --enable-openmp ...`, then `export OMP_NUM_THREADS=n` and run on `n` threads (4 to 8 typically). *Beware conflicts between MPI and OpenMP threads*! don't do this unless you know what you are doing. - Finally, the optimal number of processors for "linear-algebra" parallelization can be found by observing the performances of `ortho` in the final time report for different numbers of processors in the linear-algebra group (must be a square integer, not larger than the number of processoris for plane-wave parallelization). Linear-algebra parallelization distributes $M \times M$ matrices, with `M` number of bands, so it may be useful if memory-constrained.

Note: 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).