Tutorial: CPMD

Riccardo Bertossa

May 2021

1 Introduction: BOMD and CPMD

So, what is a Born-Oppenheimer molecular dynamics (from now on: *BOMD*)? Suppose you what to know how atoms move in a big molecule. The first useful thing to assume is that the atomic nuclei are classical particle. Se we can calculate the forces on the nuclei at some discretized instants of time and use those forces to integrate the atomic positions by using, for example, the verlet algorithm. And you want to calculate those forces with DFT. The second thing that is very useful to assume is that the forces depend only on the atomic positions. This is the adiabaticity assumption: we neglect every contributions to the dynamics that comes from "the history" of the electron's wave functions. Every electronic configurations depends only on the current atomic positions. In this way, we can simply run a DFT ground state calculation for each atomic configuration and then calculate the forces via the Hellman-Feynman theorem.

Now think for a second about the variations of the electronic wave functions state during the dynamic. At each timestep the wavefunctions are the ones that minimize the energy of the system. Can we find a more efficient way of finding this ground state? Is it necessary to recompute it from scratch at every timestep? Of course not. First of all, we can think to use the result of the previous timestep as a starting point for the solution of the current one. But we can do better. We can calculate "forces" over each point in the space of the wavefunctions and treat them as a classical fluid that is running after a potential that changes at each step, and propagate everything with the verlet algorithm. Sort of *dynamical simulated annealing*, as explained in the original Car and Parrinello paper `https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.55.2471` while introducing the famous Car-Parrinello molecular dynamics technique (from now on *CPMD*). The famous *classical* lagrangian is

$$\mathcal{L} = \sum_v \frac{1}{2}\mu \int_\Omega d^3r |\dot{\psi_v}|^2 + \sum_I \frac{1}{2} M_I \dot{R}_I^2 - E_{DFT}[\{\psi_v\}, \{R_I\}] + \sum_{i,j} \Lambda_{ij} \left(\int_\Omega d^3r \psi_i^* \psi_j - \delta_{ij} \right) \quad (1)$$

where μ is the "fake" mass of the electronic fluid, E_{DFT} is the energy functional and the last term are the orthonormality constraints that keeps the electronic orbitals orthogonal to each other. Λ_{ij} are lagrangian multipliers. Note again that this is a classical lagrangian, from which we can derive classical equation of motion. But note here that the coordinates that we are using to derive the dynamics are the electron wavefuntions coordinates (one coordinate for the value and one for the time derivative for each point in the discretized space) and the atomic coordinates (usual velocity and positions), and the lagrangian multipliers to keep the wavefunctions orthogonal. From the Euler-Lagrange equation (I repeat it again: the equations and the dynamics here is governed by classical laws) $\frac{\partial \mathcal{L}}{\partial x_i} = \frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{x}_i})$ (were the x_i are all the generalized coordinates that we use in the lagrangian) we can derive the equation of motions:

$$\mu \ddot{\psi_v}(r,t) = -\frac{\delta E}{\delta \psi_v^*}(r,t) + \sum_j \Lambda_{vj} \psi_j \quad (2)$$

$$M_I \ddot{R}_I = -\nabla_{R_I} E \quad (3)$$

Those are the equation that the code is going to integrate with the verlet algorithm. So we will have to choose a suitable dt to have a correct conservation of the constant of motion of the CP lagrangian. We have to choose μ to keep the electronic degrees of freedom as cold as possible, to allow them to gracefully oscillate around the true BOMD ground state.

2 Ground state and initial BOMD

So, we have our molecular system. Let's choose water. The first thing to do is to choose the initial position of the atoms. Since both BOMD and CMPD are by far more expensive than pure classical molecular dynamics with a force field, a good idea would be doing a preliminary simulation with a classical force field to get a more or less thermalized initial state and use this as a starting point. But suppose that we don't have any accurate force field, or that we are able to face the extra computational cost. It is not a good idea to start immediately with a CPMD simulation because if you start say assigning random velocities to the atoms, you can have some sharp changes in the atomic velocities because they could be not consistent with the current atomic positions, or because two atoms could be too near to each other. This will reflect in little kinks to the nuclei and by consequence little kinks to the electronic degrees of freedom, that will cause a little more kinetic energy to the electronic system (that we want as cold as possible) and a little (or big, depending on how much the initial configuration was bad) rise in the electronic temperature.

For this reason is a good idea to start with few steps of BOMD, to allow the system to relax at the correct temperature for the system's initial energy and thermalize a little bit.

2.1 BOMD cp.x's input file

To run a calculation you have to write the following file and run `mpirun -np 12 cp.x -in water10.in.00` for example, where you have to put the correct number of mpi processes, put the correct path for your `cp.x` executable and `water10.in.00` is the input file that you wrote on the filesystem:

```
&control
   title = 'Water 8 molecules',
   calculation = 'cp',
   restart_mode = 'from_scratch',
   ndr = 50,
   ndw = 50,
   nstep  = 100,
   iprint = 10,
   isave  = 1000,
   ! tstress = .TRUE.,
   ! tprnfor = .TRUE.,
   dt    = 5.0d0,
   prefix = 'h2o',
   pseudo_dir='../pseudo',
   outdir='./',
/
&system
   ibrav=1, celldm(1)=13.00000
   nat = 24,
   ntyp = 2,
   ecutwfc = 50.0,
/
&electrons
   emass = 50.d0,
   emass_cutoff = 2.5d0,
   electron_dynamics = 'cg',
```

```
 /
 &ions
    ion_dynamics = 'verlet',
    !ion_velocities = 'zero',
    ion_velocities = 'random',
    tempw=600.d0
 /
 &cell
    cell_dynamics = 'none',
 /
ATOMIC_SPECIES
 O  16.0d0   O_ONCV_PBE-1.2.upf
 H  1.0079d0 H_ONCV_PBE-1.2.upf
ATOMIC_POSITIONS (bohr)
O   0.48425102952101E+01     0.37961987444088E+01     0.37327381430173E+01
H   0.40045904816720E+01     0.59867799653268E+01     0.35330129791271E+01
H   0.43651175783709E+01     0.38723218001876E+01     0.58797734074100E+01
O   0.37356574289384E+01    -0.44747002555544E+01     0.29242739490968E+01
H   0.60591480525558E+01    -0.37950308878052E+01     0.31534025213325E+01
H   0.28749619712861E+01    -0.39586722357631E+01     0.46751869989712E+01
O  -0.45466860308386E+01    -0.35422621276003E+01     0.30607442672430E+01
H  -0.37488382797211E+01    -0.58116264134953E+01     0.33207580642326E+01
H  -0.40539478128360E+01    -0.25743644332842E+01     0.47683540965636E+01
O  -0.34365057953046E+01     0.48042682147233E+01     0.33772341318046E+01
H  -0.59041512457399E+01     0.42174303155281E+01     0.38014151466890E+01
H  -0.23030769008515E+01     0.46720049038444E+01     0.50821358987130E+01
O   0.36144195243782E+01    -0.42827457674593E+01    -0.39899980086654E+01
H   0.31198656062526E+01    -0.45610696424210E+01    -0.20022000203335E+01
H   0.59144800748620E+01    -0.39529825813534E+01    -0.38666276232141E+01
O  -0.44370946792819E+01    -0.40253658651776E+01    -0.40851101377693E+01
H  -0.39508218126754E+01    -0.36848090700940E+01    -0.21027197127076E+01
H  -0.39403988307857E+01    -0.61904746485483E+01    -0.41653506412705E+01
O  -0.35587892994654E+01     0.38773012775526E+01    -0.38192334238418E+01
H  -0.32332194882106E+01     0.34352421960554E+01    -0.18180551968315E+01
H  -0.56904752202074E+01     0.34796638592131E+01    -0.37894354717772E+01
O   0.38377828950880E+01     0.38204092706037E+01    -0.45079974049078E+01
H   0.20131356918203E+01     0.30922405697794E+01    -0.39131118473731E+01
H   0.36593535000291E+01     0.62003173674732E+01    -0.43870392213130E+01

AUTOPILOT
  on_step=10 : dt = 20.d0
  on_step=90 : dt = 5.d0
ENDRULES
```

This input file is organized first in some fortran namelists and some cards, as in the PW code. You can see that many of the variables and cards are common in both codes. One important difference (that is there for historical reasons) is that the units of measure are different in the codes: in the CP's input description page `https://www.quantum-espresso.org/Doc/INPUT_CP.html` you found the following

```
dt  REAL
    time step for molecular dynamics, in Hartree atomic units
    (1 a.u.=2.4189 * 10^-17 s : beware, PW code use
    Rydberg atomic units, twice that much!!!)
```

that results in a factor 2 in both energies and velocities.

Here I start the simulation by doing 50 steps of BOMD. The initial ionic velocities are sampled from the Boltzmann distribution. It is only a guess: later the temperature of the system will almost always equilibrate to a different value, and a Nose-Hover thermostat will be necessary.

Note that at the end of the file there is a special card that tells the code to change some variables during the simulation. Here for example we are changing the timestep at the 10th steps to 25 a.u. and then back to 5.0 a.u. at the last step of the simulation. This can be useful specially at the beginning of the simulation since we save the time of writing multiple input files and running a lot more times the code.

The code produces some output file. In the specified `outdir` you will find many new files where the data will be appended at each run. The most important file, which determines the trajectory, are the `.pos`, the `.vel` and the `.cel` files that contains the positions, the velocities of the atoms and the the time series of the cell vectors (you can choose to have a variable cell calculation). Then you have the `.evp` file, that contains some useful quantities, like all the parts of the energy, the temperature, number of timesteps and time in picoseconds. You can visualize the data with gnuplot, for example:

```
$ gnuplot
gnuplot> pl 'h2o.evp' u 2:3
```

will plot electron's kinetic energy on the y axis and the time in ps on the x axis. In this case `ekinc` is zero because we are not running yet the verlet algorithm, and there is no lagrangian yet that can be used to define an electronic classical (fake) kinetic energy.

2.2 OPTIONAL

- Try to see what changes with different values of dt. For example you could set a very big timestep and see how much the energy of the system is conserved
- Try to modify the coordinate of one atom so that it approaches an other one. Is there some distance at which the simulation stops working?

3 CPMD start

Now we start the verlet algorithm. First of all it is necessary to modify the input file

- set `calculation = 'restart'` to start from a previously stopped calculation
- set `ndw = 51` (increase by one the number of the folder where the code will write the restart file)
- set `nstep = 1000` if you want to run for 1000 steps
- set `electron_dynamics = 'verlet'` to set the verlet algorithm to integrate the Car-Parrinello equation of morion
- set `ion_velocities = 'default'` to read the velocity from the specified restart file
- remove the `AUTOPILOT` card

Then note the cg code does a special thing at the last step: it computes the wavefunction velocity with the projector trick. Essentially the derivative is moved from the wavefunction (where numerically is ill defined, since two consecutive and independent cg minimization do not need to share the same phase) to the projector, which has a well defined numerical derivative:

$$|\dot{\psi}^{\perp}\rangle = \dot{P}|\psi\rangle \tag{4}$$

where $P = \sum_v |\psi_v\rangle\langle\psi_v|$ is the projector over the occupied manifold is an object that does not depend on the gauge that the KS solver chose and thus can be differentiated numerically. In this way the verlet algorithm can start with a wfc velocity that is consistent with the velocity of the ionic nuclei.

Now we can check few things:

- check that the constant of motion is conserved. The constant of motion is plotted at the column 9
- try to see what happens if you increase dt too much in this case (you can use autopilot to speed up the tests), or what happens if you use a smaller dt

4 Nose-Hoover thermostat

Now that we see that the simulation is working we notice that the average temperature is not what we want. So let's assume that we are fine with the density, but we want a temperature of 600K. We have to modify the input as follows:

- set the number of steps to 1000 with `nstep = 1000`
- increase by one `ndr` and `ndw`
- set the Nose-Hoover thermostat (namelist `IONS`):
  - set nose: `ion_temperature = 'nose'`
  - temperature: `tempw = 600.d0`
  - nose frequency: `fnosep = 5.d0`
  - number of thermostat in the NH chain: `nhpcl = 3`

run again the code and see how the temperature changes. Plot the temperature with gnuplot (column 5). Remember to keep an eye on electronic kinetic energy!

5 NVE simulation

5.1 re-cooling of the electronic degrees of freedom

As always, you have to change the input file. Now we want to use CG to cool down the electrons and we use the projector trick to initialize again the wfc velocity. To do so you have to:

- set the number of steps to 1 with `nstep = 1`
- increase by one both `ndr` and `ndw`
- change `electron_dynamics = 'cg'`
- change `ion_temperature = 'not_controlled'`

5.2 verlet again without nose

- increase by one both `ndr` and `ndw`
- set `nstep = 10000`
- set again verlet for electrons `electron_dynamics = 'verlet'`

This run will be a little bit longer.

6 Simulation accuracy

6.1 Forces error

So far everything look great (is it?). But are the CP forces the same as the forces computed by a standard DFT solver? If you think carefully, you don't have to expect that. First of all because you are not minimizing the electronic energy at each timestep, but the electrons are following a classical dynamics inside a potential that keeps changing at each step, so we expect that the forces oscillate around the true BOMD forces. Then, remember that electrons have a classical mass, that is many times bigger than the true electron mass. This mass has nothing to do with the physical electron mass. So, if they have a mass, to accelerate them you have to spend some energy: they have an inertia. Because of that, there is a drag effect that result in a systematically lower computed forces on the atoms with respect to BOMD, as if atoms were heavier. In some sense it is true, there is some electronic fluid that is moving with the atoms, it is attached to them, so the result is like the atoms were heavier.

So is this method useless? Absolutely not. First, you can check by yourself, this effect most of the time is negligible if you choose an electronic mass μ low enough, like the one that we chose. This approximation can be controlled. Then, it affects only dynamical quantities, like diffusion coefficient and not static properties like $g(r)$.

The higher the μ the lower the force ratio. Try to verify that by selecting a snapshot and calculating the forces, with the same cutoff, with the PW code. Then compare the forces to the one saved in the `.for` file. Remember the factor 2 between the two codes.

6.2 Electronic gap

Verify that the electronic gap is always higher than few's k_bT. This check is necessary to make sure that the electronic system is decoupled as much as possible from the ionic system. Electrons are usually faster than ions and the lowest electronic frequency is of the order of

$$\omega_e^{min} \propto \sqrt{E_{gap}/\mu} \tag{5}$$

If this frequency overlaps with the frequency of the ionic system, energy will flow from the hottest to the colder subsystem, resulting in a massive heating up of the electrons. In very few steps the forces will become totally meaningless making your simulation a very expensive way to generate not so good random numbers.

6.3 ooops. Everything exploded

try to see what happens when you increase μ too much, and make some expensive random numbers.

7 VMD visualization

You can use `https://github.com/lorisercole/fancy-scripts/blob/master/QuantumEspresso/cp.x/cp2lammpstrj.py` to convert the `cp.x` output format to the lammps trajectory format that can be visualized by VMD:

```
$ $./cp2lammpstrj.py input_trajectory_prefix -n 24 --minimal --not-ordered  \
-t 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 -o outfile
```

where the sequence of 0 and 1 are the atomic type associated with the atom, in the same order that the code uses in the output.

8 OPTIONAL: let's calculate something from the trajectory

You can use the package `analisi` that is available on the conda package manager or on `https://github.com/rikigigi/analisi`

To begin you have to convert the `cp.x` trajectory format to the lammps binary format, this task can be performed by the code `https://github.com/rikigigi/analisi/blob/public/cp2analisi.py` with the command

```
$ python3 cp2analisi.py input_trajectory_prefix output \
0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
```

where the sequence of 0 and 1 are the atomic type associated with the atom, in the same order that the code uses in the output. This python script works after installing the python package, following the instructions or using conda. Then the usage is very simple.

8.1 Pair correlation function

To estimate the $g(r)$ from the converted trajectory simply input the following command

```
$ analisi -i h2o.bin -F 1.0 10.0 -g 100 -S 1 > gofr
```

and then visualize it on gnuplot by typing

```
$ gnuplot
gnuplot>  pl 'gofr' u 2:3 w l, '' u 2:5 w l, '' u 2:7 w l
```

You will recognize the shape of the g(r) of water

8.2 Mean square displacement

To generate a plot of the mean square displacement, you can use the command

```
$ analisi -i h2o.bin -q -B 2 > msd
```

and then visualize it with gnuplot

```
$ gnuplot
gnuplot> pl 'msd' u 0:1 w l, '' u 0:3 w l
```