How to implement new constraints into Quantum ESPRESSO

Carlo Sbraccia

March 3, 2017

The two basic ingredients that are required to implement a new type of constraint into the Quantum ESPRESSO distribution are:

- the analytical expression for the constraint $\sigma(\{\mathbf{R}^{3N}\})$ (it must be a function of the ionic coordinates $\{\mathbf{R}^{3N}\}$ only);
- the analytical expression for the gradients of the constraint $\boldsymbol{\nabla}_{\mathbf{R}_i}\sigma(\{\mathbf{R}^{3N}\})$ with respect to the ionic coordinates.

Given these expressions one has simply to follow what has already been done for the standard constraint types.[1] No detailed knowledge of the algorithm used to impose the constraints (SHAKE) is necessary since the implementation is designed to work for any possible kind of constraint, provided it is defined by an analytical expression. In the following I describe the three basic steps that are strictly necessary to implement a new constraint type.

One first has to modify the routine that reads the `CONSTRAINTS` input card (this input card contains the parameters specified at run-time by the user to define the constraint). The name of this routine is `card_constraints()` and it is located in the module `Modules/read_cards.f90`. Note the maximum allowed number of input parameters used to define a single constraint is 6; if the new constraint type requires additional input parameters one has to

[1] At present the constraint types implemented in Quantum ESPRESSO are:

- coordination numbers;
- distances;
- planar angles (linear angles included);
- torsional angles.

modify the dimension of the array `constr_inp(:,:)` defined in the module `Modules/input_parameters.f90`. All the other arrays are dynamically allocated.

Then one has to copy the input arrays into the internal ones (this is automatically done and should not not require any additional tuning) and to initialise the target value of each constraint (the target corresponds to the initial value of $\sigma_i(\{\mathbf{R}_0^{3N}\}$; this is the quantity that is kept constant during the simulation). All this is done in the routine `init_constraint()` which is located in the module `Modules/constraints_module.f90`. One has to define the new variables that are needed to calculate the value of the constraint (possibly recicling those that are already there) and then implement the equation defining the constraint (following what is done for the other constraint types).

The last step consists in the implementation of the constraint's gradient $\boldsymbol{\nabla}\sigma(\{\mathbf{R}^{3N}\})$. This is done in the routine `constraint_grad()` located in the module `Modules/constraints_module.f90`. Again one has to define the new variables and implement the equations that define both the constraint violation and the constraint gradients (respectively stored in `g` and `dg(:,:)`). This is done for a single constraint σ_i (identified by the input variable `index`) since the routine is externally called by other drivers as many times as the number of constraints. Note that for each constraint the sum of the gradients must be zero: $\sum_i \boldsymbol{\nabla}_{\mathbf{R}_i}\sigma(\{\mathbf{R}^{3N}\}) = 0$. This is usually imposed by defining one of the gradients to be equal to minus the sum of all the others.

Finally, one should not forget to test the new constraint on both PWscf and CP by monitoring the energy conservation and, of course, the conservation of the constraint itself.