NAME

pbqff − push-button quartic force fields

SYNOPSIS

pbqff [**−c**] [**−count**] [**−cpuprofile** *name*] [**−debug**] [**−fmt**] [**−freqs**] [**−irdy** *atoms*] [**−nodel**] [**−o**] [**−pts**] [**−r**] *input_file*

DESCRIPTION

This document describes the **pbqff** program, a package for running quartic force fields in symmetry internal (SIC) and Cartesian coordinates. SICs require template intder, anpass, spectro, and Molpro files in addition to the **pbqff** input file containing the molecular geometry inside a **geometry={...}** block. The input file must also contain the **intder=** , **anpass=** , and **spectro=** directives specifying the path to the corresponding executables.

The intder template should be named **intder.in** and be a points-generating intder input file and have a geometry of the same symmetry to use as a reference. The anpass template should be named **anpass.in** and be a first-run anpass file, not a stationary point. The spectro template must be named **spectro.in** and should not contain any resonance information in the body or input directives. The Molpro template should be named **molpro.in** and have the geometry removed, along with its terminal curly brace. For gradient calculations, the **forces,varsav** directive should be included in the Molpro template after the DF-HF and DF-CCSD(T) lines. Following that should be **show[f20.15],gradx** , **show[f20.15],grady** , and **show[f20.15],gradz** to print the gradient with sufficient precision.

OPTIONS

The command line is parsed according to the rules of the Go **flag** package. This means that options cannot be grouped behind a single ’-’ (minus character). Either whitespace or an ’=’ (equals sign) can appear between a command option and its argument.

−c Resume from checkpoint; requires the **−o** flag to overwrite existing directory.

−count Read the input file for a Cartesian QFF, print the number of calculations needed, and exit.

−cpuprofile
Write a CPU profile to the supplied filename.

−debug
Print additional information for debugging purposes.

−fmt Parse existing single point output files and print them in anpass format

−freqs Start an SIC QFF from running anpass on the pts output. This requires that all of the points have been preserved.

−irdy *atoms*
Ignore the geometry in the input file and use the intder.in file as is. This implies that you should use noopt. *atoms* is a space-delimited list of atomic symbols to pair with the geometry.

−nodel Preserve output files instead of deleting them after use.

−o Allow existing directories created by the program to be overwritten.

−pts Resume an SIC QFF by generating the points from the optimized geometry in the opt directory.

−r Read the reference energy for a Cartesian QFF from an existing pts/inp/ref.out file.

EXAMPLES

Basic example

The following is a basic example of running **pbqff** on an input file input.in.

```
pbqff input.in & disown -h
```

It is necessary to run the program in the background (&) if you want to use your terminal for anything else while it runs, and you should **disown** it with the **−h** flag if you want it to survive you logging out of your SSH session. This will create the files input.out and input.err. input.out contains information about the available queues, the number of jobs that have to run, and the CPU time limits. input.err contains progress about the running jobs.

Checking progress

Before these are updated, the geometry optimization for an SIC calculation (if requested) will run in opt/opt.out, so you may want to check the overall progress with the command

```
tail -F input.err input.out opt/opt.out
```

For a Cartesian calculation the optimization or reference energy is run in pts/inp/ref.out, so the command becomes

```
tail -F input.err input.out pts/inp/ref.out
```

To make sure **pbqff** is still running you can use the command

```
ps axo pid,user,comm,time | grep pbqff
```

which will also give you information about the CPU time used by the process.

Resuming from checkpoints

If the CPU time used by **pbqff** exceeds the limits printed in the output file, the process will be killed. To resume from a checkpoint, use the **−o** and **−c** flags to overwrite the old input directory while loading the previous progress from the JSON files. The full command to do this is

```
pbqff -o -c input.in & disown -h
```

This should be run in the same directory as the initial **pbqff** run since that is where the input and checkpoint files are.

SEE ALSO

Example input files in ~r2518/Programs/pbqff/examples.