pbqff tutorial

Brent R. Westbrook

January 4, 2021

1 Introduction

pbqff is a program for making the generation of quartic force fields (QFFs) as easy as possible. Its name was inspired by the C&E News article "As DFT matures, will it become a push-button technology?" by Sam Lemonick. The consensus among the Tschumper and Fortenberry groups was that "DFT" or computational chemistry more generally could never be fully "push-button" because of the need for expert analysis of the data. Using pbqff is no different; you still need to be that expert to make sure the output is reasonable and to write it up into a paper. However, pbqff should make the experience of actually running the QFF as painless as possible. If you have ever run a QFF "by-hand," you know the complexity of the procedure. These tedious and repetitive steps lend themselves perfectly to being done by the computer, freeing you to think about something more interesting. If you haven't, I will prepare a separate document giving a general QFF tutorial. To understand what pbqff is doing, you should have a good grasp on what *you* would be doing if you had to its job by hand. It's not strictly necessary to understand pbqff though, and the goal for me was really to make the implementation details irrelevant to the end user. If you really want to understand the entirety of pbqff and all its implementation details, you should just read the source code, which can be found at github.com/ntBre/pbqff. Assuming you want to get to work, I hope this tutorial document will give you all the information you could possibly want for actually using pbqff effectively. I will also prepare a separate man page to use for quicker reference once you have a basic understanding.

2 Mental Framework

2.1 Internal Coordinates

As I alluded to in the introduction, the idea behind pbqff is to follow the exact steps a person would by hand in running a QFF, but to do so quickly and exactly via the computer. As it stands, I think of pbqff as "replacing" an undergraduate researcher because we usually give undergraduates all of the template input files they need. pbqff requires that you give it template intder, spectro,

anpass, and molpro files for a symmetry internal coordinate (SIC) run, which it can then modify as needed. This modification assumes that the template input files correspond to molecules with the same coordinate system as the target molecule, so the automation at that stage is fairly limited. Graduate students (and advanced undergraduates) usually learn how to generate or more substantially modify intder, anpass, and spectro files, so once pbqff can do that, it will have "replaced" them as well. Also as mentioned above, these replacements are not total, as it still requires some thought to evaluate the program output and to come up with new molecules for input. In a section on mental models, think of the replacing as a rough model for how much pbqff can do.

2.2 Cartesian Coordinates

The deal for Cartesian (or XYZ) coordinate QFFs is a little better because intder and anpass are not used. Consequently, you only need template spectro and molpro files. This is also true for gradient calculations since they are also based on Cartesian coordiantes. Even more fortunately, I think spectro input files are the easiest to generate, so the earliest full automation will come to these types of calculations as well. If you are familiar with our traditional SIC QFFs, you may be surprised that intder and anpass are not used. Intder is used for coordinate transformations from internal coordinates to Cartesian coordinates and back again, so working directly in Cartesian coordinates from the start obviates any need for it. Anpass is used for least-squares fitting of the potential energy surface, whence we gather the force constants. For the Cartesian QFF, pbqff numerically computes each force constant directly, so there is no need to do a fitting. These facts should not change your mental model of the whole procedure, however. The big idea is still to take a geometry, displace its atoms, compute single-point energies at each of the displaced points, use those energies to obtain force constants, and then jam those force constants into spectro to get spectroscopic data. In Cartesian coordinates you just get to take shortcuts in the displacement and force constant steps. Like with SICs, these are the exact steps you would take in doing a Cartesian QFF by hand, but there are so many points involved in Cartesian QFFs that we never do them by hand.

2.3 Job Submission

The other slightly tricky part about pbqff if you are more used to our conventional QFF schemes is that it does not submit, or even generate input files for, all of the jobs at once. Doing everything at once is obviously convenient when you are doing it by hand because you can submit all of the computations to the supercomputer and then go do something else. Instead, pbqff watches the running jobs, continuously checking for ones that finish, and writes and submits more calculations as the old ones finish. It also deletes the files associated with the old jobs to save space. Space is not typically a major concern with SIC QFFs since even the largest systems we have worked with have only 10000 or so points. In contrast, even water has half that many points in Cartesian co-

ordinates, and larger molecules that we've actually run have had over 200000. Eventually we would like to extend this to millions of points, so keeping all of the files around is not really feasible. Again, you should be able to picture this in the same way as doing it by hand, but it's easier to convince the computer to sit and constantly refresh qstat than to make a human do it.

A final wrinkle is the use of GNU parallel to reduce the number of individual PBS jobs that need to be submitted. parallel is basically a miniature version of the whole queuing system in that it takes a list of jobs to run and dispatches them as resources are available. While this is yet another departure from how you would likely submit these jobs by hand, you can certainly use parallel when submitting jobs by hand. It should also have no bearing on your interaction with the program, but if you are curious why there are so few jobs in your queue this is the explanation.

3 Program Input

In addition to the template files addressed in the previous and upcoming sections, pbqff takes its own input file. This section will walk through all of the accepted input options and offer some example inputs for the various supported calculation types.

Unlike some programs, the input directives are totally order-agnostic. As long as they are in the input file, they will be recognized. Available keywords are shown in Table 1. The case of the keywords is ignored by the parser, but each keyword must be followed by an equals sign (=). Comments can only start at the beginning of a line, by including "#" as the first character on the line. Geometry input is unique in that it expects to look like `geometry={...}`, beginning with an equals sign and opening curly brace and terminating with a closing curly brace. As shown in the options for the GeomType, the geometry can be input as either a Cartesian or XYZ geometry or a Z-matrix. If the Cartesian coordinates are used, the program expects a fully-formed XYZ geometry, including the number of atoms line and the comment line. These lines are skipped, so it's not important that they be accurate, but they must be present.

The following examples can be found in my home area on hpcwoods under Programs/pbqff/examples. The files embedded in this document should be synced with the ones found there, and they all should be tested to run correctly.

3.1 SIC Example

Below is an example input file for an SIC run. Since much of the information for the SICs is found in the other template input files, the pbqff file is about as minimal as it gets. Based on the defaults given in Table 1, even the `program`, `queue`, and `geomType` lines are technically redundant, but it's nice to include some of these for future reference. Example:

```
program=molpro
queueType=maple
```

```
geomType=zmat
geometry={
O
H 1 oh
H 1 oh 2 hoh
oh=1.0 ANG
hoh=109.5 DEG
}
intder=/ddn/home6/r2533/programs/intder/Intder2005.x
anpass=/ddn/home6/r2533/programs/anpass/anpass_cerebro.x
spectro=/ddn/home6/r2533/programs/spec3jm.ifort-O0.static.x
```

One thing to note is that there is no brace between the Z-matrix itself and the values of the parameters, as you might expect if you are used to Molpro. Just remember that this is the format expected by pbqff, and it will convert it to the Molpro format when necessary.

3.2 Cartesian Example

The Cartesian example is a bit more involved since you have to specify some non-default values, and you can't rely on the intder file for the step sizes or derivative level. Again, many of these options are technically optional since they are the same as the defaults, but it's nice to be explicit when possible. Example:

```
program=gocart
queueType=maple
geomType=xyz
geometry={
 3
 Comment
 H          0.0000000000          0.7574590974          0.5217905143
 O          0.0000000000          0.0000000000         -0.0657441568
 H          0.0000000000         -0.7574590974          0.5217905143
}
delta=0.005
deltas=1:0.075,4:0.075,7:0.075
flags=noopt
deriv=4
sleepint=5
joblimit=8000
chunksize=64
numjobs=8
intder=/ddn/home6/r2533/programs/intder/Intder2005.x
anpass=/ddn/home6/r2533/programs/anpass/anpass_cerebro.x
spectro=/ddn/home6/r2533/programs/spec3jm.ifort-O0.static.x
```

Of note here is a first example of an XYZ geometry. You can see that the number of atoms line and comment are present. Alignment and spacing are not important, so you can freely paste the geometry in however you think looks best. Another important aspect of this example is the demonstration of the `deltas` input. In this example, all of the steps in the x direction will be larger (of size 0.075 Å), while the rest will be 0.005 Å.

3.3 Gradient Example

The gradient version is virtually identical to the Cartesian version, except that the `program` is specified as `grad`. You may also notice that the comment line says that the reference energy was computed at the DF-CCSD(T)-F12 level rather than regular CCSD(T)-F12. Molpro only has analytic gradients for density-fitted coupled cluster, so if you want to use gradients keep that in mind. Based on some forthcoming research from our group, you probably don't want to use gradients though. Example:

```
program=grad
queueType=maple
geomType=xyz
geometry={
     3
 DF-CCSD(T)-F12/CC-PVTZ-F12   ENERGY=-76.36827708
 H            0.0000000000          0.7578204038          0.5219210812
 O            0.0000000000         -0.0000000012         -0.0660052896
 H            0.0000000000         -0.7578204026          0.5219210802
}
flags=noopt
deriv=4
joblimit=8000
chunksize=64
intder=/ddn/home6/r2533/programs/intder/Intder2005.x
anpass=/ddn/home6/r2533/programs/anpass/anpass_cerebro.x
spectro=/ddn/home6/r2533/programs/spec3jm.ifort-O0.static.x
```

4 Template Files

4.1 molpro

4.2 spectro

4.3 intder

4.4 anpass

5 Running the Program

Now that you have all of the requisite input files for any type of calculation, you are ready to run the program! In this section I will show the basic input command, along with an explanation of the parts that aren't as basic as you would expect, and also describe each of the flags you can use to modify the program's behavior.

6 Troubleshooting

Table 1: Keywords

Keyword	Type: Available values	Default	Description
QueueType	String: sequoia, maple	maple	Specify which queue to target. Basically choose the internal PBS template to use for submitting jobs.
Program	String: cccr, cart, gocart, grad, molpro	molpro	Specify the subprogram to use. cccr is for CcCR SICs; cart or gocart is for Cartesians; grad is for gradients; and molpro is for normal SICs
Queue	String: workq, r410	Both	Select the queue name to use. Default behavior is to use whatever is available.
Delta	Float: any	0.005	Specify the step size to use for the geometry displacements of a Cartesian QFF.
Deltas	Int:Float pairs: any	0.005	Specify step sizes for individual coordinates in a Cartesian QFF. Format is index:value, where index starts from 1. Pairs are separated by commas. Any indices not specified take their default value from Delta.
GeomType	String: xyz, zmat	zmat	Specify the type of geometry. Currently this is only used to compute the number of coordinates.
Flags	String: noopt	None	Specify command line flags in the input file. Only noopt is currently supported.
Deriv	Int: 2, 3, 4	4	Specify the derivative level to be computed for a Cartesian QFF.
JobLimit	Int: any	1000	Specify the maximum number of jobs to have submitted at once.
ChunkSize	Int: any	64	Specify the number of jobs to submit in a GNU parallel "chunk." This also determines how often files are deleted.
CheckInt	String/Int: no/any	100	Specify the checkpoint interval. An input of "no" disables checkpointing, while an integer value sets the interval.
SleepInt	Int: any	1	Specify the interval at which to poll running jobs in seconds.
NumJobs	Int: any	8	Specify the number of jobs to run per GNU parallel job. Each parallel job requests 64gb of memory, so NumJobs should evenly divide 64.
IntderCmd	String: any	None	Specify the path to the intder executable for SIC QFFs.
AnpassCmd	String: any	None	Specify the path to the anpass executable for SIC QFFs.
SpectroCmd	String: any	None	Specify the path to the spectro executable.
Geometry	String Block: any	None	Specify the geometry. See below for details.