Lessons learned at LLNL

Zachary Matheson

March 3, 2017

Nucleon localization function

1 March 2017

Most recently I tried using Erik's modified version of HFBTHO to run for several constraints along Q_{30} (or anywhere, really). What I'd like to do is use HFBTHO to generate densities quickly for ^{176}Pt between $Q_{20} = 241$ and $Q_{20} \approx 300$, and at $Q_{30} = 4, 18$ (something I decided semi-arbitrarily once upon a time). I'm putting that on hold for a bit while I work on this inertia thing. Erik sent me some notes for perhaps getting the code to do what I want it to do, which are in my email. The files are currently in /p/lscratchh/matheson/locali-176Pt/hfbtho (/erik for testing his version of the code). Another thought I had was to try constraining Q_{40} to something reasonable, and then releasing that constraint to find the actual density (hopefully) nearby.

1 March 2017

Impact of basis deformation on EHFB

We wanted to see how much the observables of the system would change if we used a single HO basis across an entire PES. This is based on a misunderstanding I had of something Jhilam said, where in order to use his inertia code, you need adjacent points to have the same basis deformation and other basis properties (so that you can numerically take derivatives of the densities at those points). It turns out he gets around this problem by changing the basis every 30b along Q20, but all the same we thought it would be good to check the dependence of system observables on the basis, especially since the half-life is so dependent on small deviations in the potential energy (a change of 1 MeV can affect the half-life by orders of magnitude).

To test, I took three points on the PES I had generated for ^{294}Og: (-14, 0), (72, 0), and (148, 28). According to the output file, the basis deformation chosen for each of these (by the automatic basis setting routine in HFODD) was, respectively, AL20 = 0.187, 0.424, and 0.608. I took those record files and used them to restart a new calculation, this time with the basis deformation set uniformly to AL20 = 0.61 (and AL40 = 0.10). The superdeformed asymettric shape was, understandably, least affected by the change, with the kinetic energy varying by about 3 MeV but the total energy varying by only about 0.14 MeV (-2085.878548 vs -2086.012955). Quasiparticle and canonical single-particle states were nearly identical, and fragment properties were almost the same (except for the interaction energies, which were quite different). The elongated symmetric shape differed by about 0.6 Mev (-2080.815396 vs -2080.202346). The oblate ground state didn't converge in the allotted time, but based on its last iteration it was probably going to finish with an HFB energy around -2078.649984 (compared to -2080.263986 from before), a difference of about 1.6 MeV.

3 March 2017

I've written a Python script which extracts the basis parameters from a previous run of HFODD, and uses it to initialize a new run for several neighboring points in order to facilitate an inertia calculation down the line. A problem we noticed, though, was that, while setting the α_2 deformation parameter worked fine, the basis was totally different because there was an additional constraint on α_4. It turns out that what had happened is that the code automatically sets default values of α_2 and $alpha_4$, UNLESS you set the code to choose the basis automatically, in which case it only sets a value for α_2. You can get around that by deleting the preset line in HFODD and recompiling, but I'd like for there to be an easier way. Perhaps the basis matrix is initialized to zero? So we could get around it by just setting $\alpha_4 = 0$ in the input file? $\Rightarrow$ Aha, yes. That'll work. So we're benchmarking the time on that now.

Another thing we're testing is comparing versions of the inertia code. Jhilam sent Nicolas an input and an output for ^{240}Pu. I'm running a single point now. Later, I'll run the surrounding points using both my convention and Jhilam's for how widely-spaced the points should be. I'll also compare the inertia computer with the code Jhilam sent me versus the one in Nicolas' repository.

Unfortunately, this run uses Lipkin-Nogami, and for some reason the parser doesn't write the data to the XML file, which ultimately means I'm going to have to set up the subsequent runs by hand. The question is whether to do so using Jhilam's convention (for benchmarking purposes) or the one Nicolas and I talked about (where the points are much closer and you might get a better value for the inertia). Computing the neighboring runs will probably still take a similar amount of time (it took roughly 50-60 iterations for a deviation of about 0.001 units in each direction. It'll probably be more for something farther away but if you already have those computed anyway it might not be such a big deal).

So I went ahead and did that. Once those jobs complete, we can try to analyze the inertia using each of the inertia codes, just to make sure they both give the same results. I used Jhilam's grid spacing, but we can try it again later with the narrower grid spacing once we see how long it takes for these points (which have a grid spacing of 1 in the multipole constraints, and 0.1 in the pairing constraints) to converge and then decide if using the narrower grid is economical and useful. Actually, this would be a good test case for that; if we see a noticeable improvement in accuracy, then the extra time-to-solution for the narrower grid might be worthwhile.

Of course, to do that we'll need working inertia codes. Right now, I don't undertand why, but for some reason we're getting some kind of runtime error in LAPACK:

`IntelMKLERROR : Parameter8wasincorrectonentrytoZGEMM`