3D Position User Guide

Alexander Kenny

August 2019

1 Introduction

Summary of research project and instructions for codes.

Notes, presentation and comments in codes available.

2 Research Project

2.1 Stones Implementation

- Code takes a potential and produces probability distributions by reducing action with a Metropolis algorithm.
- Implemented Force Bias in Frederic's 3D Path Integral code. Speeds convergence of action quickly for nice analytical potentials.
- Works with interaction, different acceptance algorithms for either system as a whole or each individual particle if no interaction.
- Silicon mass anisotropy taken into account with masses in longitudinal (z) direction and transverse (x/y) direction defined in a vector. Kinetic energy calculated in different directions with dot product.
- Tunnelling rate is a crude implementation that simply counts the number of times the path of a particle crosses a defined local maximum between dots.
- Code is adapted to accept non-analytic potentials. For example it interpolates discrete external potentials. This increases computational time.
- Turn features off/on with bias/interaction/mass = 0/1.

2.2 Pederson Implementation

- Runs Stones PIMC simulations for constant time step, changing number of time slices N. Analyses the system for both particles that swap and stay in their dots.

- Compares ratio of crossing and staying actions for each N, calculates exchange coupling from $J = -\frac{1}{\beta} ln(\frac{S_X}{S_{||}})$.
- Found results 1-2 orders of magnitude off expected value with time step $\Delta\tau = 10^{-6} ns (1fs)$. Wanted to check similar time steps but large computational time has cut short this analysis.

3 NCI Simulations

I used the NCI to get final actions before analysing results on my own computer. Can run Pederson simulations on NCI supercomputer. SSH into raijin. Run the following startjob.pbs code (adjusting for CPUs etc). pedersonmp.py is an adapted code that uses the multiprocessing module to run simulations for different N across different CPUs. This doesn't spread the jobs across different nodes, which would make things more efficient. Can use the module mpi4py or parallel to do it.

3.1 startjob.pbs

```
#!/bin/bash

#PBS -q express
#PBS -l ncpus=64
#PBS -l mem=128GB
#PBS -l jobfs=1GB
#PBS -l walltime=24:00:00
#PBS -l software=python
#PBS -l wd

module load python3/3.6.7
python3 file.py > /path_to_output
```