QMCPACK

User's Guide and Developer's Manual
Development Version
May 13, 2019

QMCPACK website: http://www.qmcpack.org
Releases and source code: https://github.com/QMCPACK
Google Group: https://groups.google.com/forum/#!forum/qmcpack
Latest manual: https://docs.qmcpack.org/qmcpack_manual.pdf

Contents

1 Introduction **11**
- 1.1 Quickstart and a first QMCPACK calculation . . . 11
- 1.2 Authors and History . . . 13
- 1.3 Support and Contacting the Developers . . . 14
- 1.4 Performance . . . 14
- 1.5 Open source license . . . 14
- 1.6 Contributing to QMCPACK . . . 15
- 1.7 QMCPACK Roadmap . . . 15
  - 1.7.1 Code . . . 16
  - 1.7.2 Documentation . . . 16

2 Features of QMCPACK **17**
- 2.1 Production-level features . . . 17
- 2.2 SoA optimizations and improved algorithms . . . 18
- 2.3 Supported GPU features . . . 18
- 2.4 Beta test features . . . 19
  - 2.4.1 Auxiliary-Field Quantum Monte Carlo . . . 19
  - 2.4.2 Sharing of spline data across multiple GPUs . . . 20

3 Obtaining, installing, and validating QMCPACK **21**
- 3.1 Installation steps . . . 21
- 3.2 Obtaining the latest release version . . . 22
- 3.3 Obtaining the latest development version . . . 22
- 3.4 Prerequisites . . . 22
- 3.5 C++ 14 standard library . . . 23
  - 3.5.1 Intel compiler . . . 23
- 3.6 Building with CMake . . . 24
  - 3.6.1 Quick build instructions (try first) . . . 24
  - 3.6.2 Environment variables . . . 24
  - 3.6.3 Configuration options . . . 25
  - 3.6.4 Installation from CMake . . . 27
  - 3.6.5 Role of QMC_DATA . . . 27
  - 3.6.6 Configure and build using CMake and make . . . 27
  - 3.6.7 Example configure and build . . . 28
  - 3.6.8 Build scripts . . . 28
  - 3.6.9 Using vendor-optimized numerical libraries (e.g., Intel MKL . . . 29
  - 3.6.10 Cross compiling . . . 29

3.7 Installation instructions for common workstations and supercomputers 30
3.7.1 Installing on Ubuntu Linux or other apt-get–based distributions 30
3.7.2 Installing on CentOS Linux or other yum-based distributions 30
3.7.3 Installing on Mac OS X using Macports 31
3.7.4 Installing on Mac OS X using Homebrew (brew) 32
3.7.5 Installing on ANL ALCF Mira/Cetus IBM Blue Gene/Q 32
3.7.6 Installing on ALCF Theta, Cray XC40 33
3.7.7 Installing on ORNL OLCF Titan Cray XK7 (NVIDIA GPU accelerated) 33
3.7.8 Installing on ORNL OLCF Titan Cray XK7 (CPU version) 33
3.7.9 Installing on ORNL OLCF Eos Cray XC30 34
3.7.10 Installing on ORNL OLCF Summit 34
3.7.11 Installing on NERSC Edison Cray XC30 35
3.7.12 Installing on NERSC Cori, Haswell Partition, Cray XC40 36
3.7.13 Installing on NERSC Cori, Xeon Phi KNL partition, Cray XC40 37
3.7.14 Installing on systems with ARMv8-based processors 37
3.7.15 Installing on Windows 38
3.8 Installing via Spack 38
3.8.1 Known Limitations 38
3.8.2 Setting up the Spack Environment 39
3.8.3 Building QMCPACK 40
3.8.4 Loading QMCPACK into your environment 42
3.8.5 Installing QMCPACK with Spack on Linux 42
3.8.6 Installing QMCPACK with Spack on Mac OS X 43
3.8.7 Installing QMCPACK with Spack on IBM Blue Gene 43
3.8.8 Installing QMCPACK with Spack on Cray Supercomputers 43
3.8.9 Reporting Bugs 43
3.9 Testing and validation of QMCPACK 43
3.9.1 Deterministic and unit tests 45
3.9.2 Integration tests with Quantum Espresso 45
3.9.3 Performance tests 46
3.9.4 Troubleshooting tests 46
3.9.5 Slow testing with OpenMPI 47
3.10 Automated testing of QMCPACK 47
3.11 Building ppconvert, a pseudopotential format converter 48
3.12 Installing and patching Quantum ESPRESSO 48
3.13 How to build the fastest executable version of QMCPACK 49
3.14 Troubleshooting the installation 50

4 Running QMCPACK **51**
4.1 Command line options 51
4.2 Input files 51
4.3 Output files 52
4.4 Running in parallel with MPI 52
4.5 Using OpenMP threads 52
4.5.1 Nested OpenMP threads 52
4.5.2 Performance considerations 53
4.5.3 Memory considerations 54
4.6 Running on GPU machines 54

4.6.1 Performance considerations . . . 54
4.6.2 Memory considerations . . . 55

5 Units used in QMCPACK 56

6 Input file overview 57
6.1 Project . . . 59
6.2 Random number initialization . . . 59

7 Specifying the system to be simulated 60
7.1 Specifying the simulation cell . . . 60
7.1.1 Lattice . . . 60
7.1.2 Boundary conditions . . . 61
7.1.3 Vacuum . . . 61
7.1.4 LR_dim_cutoff . . . 61
7.2 Specifying the particle set . . . 62
7.2.1 Input specification . . . 62
7.2.2 Detailed attribute description . . . 63
7.2.3 Example use cases . . . 64

8 Trial wavefunction specification 67
8.1 Introduction . . . 67
8.2 Single determinant wavefunctions . . . 67
8.3 Single-particle orbitals . . . 68
8.3.1 Spline basis sets . . . 69
8.3.2 Gaussian basis sets . . . 72
8.3.3 Hybrid orbital representation . . . 77
8.3.4 Plane-wave basis sets . . . 80
8.3.5 Homogeneous electron gas . . . 81
8.4 Jastrow Factors . . . 82
8.4.1 One-body Jastrow functions . . . 82
8.4.2 Two-body Jastrow functions . . . 87
8.4.3 Long-ranged Jastrow factors . . . 90
8.4.4 Three-body Jastrow functions . . . 93
8.5 Multideterminant wavefunctions . . . 94
8.6 Backflow wavefunctions . . . 95
8.6.1 Input Specifications . . . 95
8.6.2 Example Use Case . . . 96
8.6.3 Optimization Tips . . . 97
8.7 Finite-difference linear response wave functions . . . 98
8.7.1 Input Specifications . . . 99
8.7.2 Example Use Case . . . 99
8.8 Gaussian Product Wavefunction . . . 100
8.8.1 Example Use Case . . . 101

9 Hamiltonian and Observables **102**
9.1 The Hamiltonian . . . 102
9.2 Pair potentials . . . 103
9.2.1 Coulomb potentials . . . 104
9.2.2 Pseudopotentials . . . 106
9.2.3 Modified periodic Coulomb interaction/correction . . . 108
9.3 General estimators . . . 109
9.3.1 Chiesa-Ceperley-Martin-Holzmann kinetic energy correction . . . 110
9.3.2 Density estimator . . . 111
9.3.3 Spin density estimator . . . 112
9.3.4 Pair correlation function, $g(r)$ . . . 113
9.3.5 Static structure factor, $S(k)$ . . . 115
9.3.6 Species kinetic energy . . . 116
9.3.7 Lattice deviation estimator . . . 116
9.3.8 Energy density estimator . . . 117
9.3.9 One body density matrix . . . 121
9.4 Forward Walking Estimators . . . 125
9.5 "Force" estimators . . . 127

10 Quantum Monte Carlo Methods **129**
10.1 Variational Monte Carlo . . . 130
10.2 Wavefunction Optimization . . . 132
10.2.1 VMC run for the optimization . . . 133
10.2.2 Correlated sampling and Cost function . . . 134
10.2.3 Optimizers . . . 134
10.2.4 General recommendations . . . 140
10.3 Diffusion Monte Carlo . . . 140
10.4 Reptation Monte Carlo . . . 144

11 Output overview **148**
11.1 The .scalar.dat file . . . 148
11.2 The .opt.xml file . . . 149
11.3 The .qmc.xml file . . . 149
11.4 The .dmc.dat file . . . 149
11.5 The .bandinfo.dat file . . . 149
11.6 Checkpoint and restart files . . . 149
11.6.1 The .cont.xml file . . . 149
11.6.2 The .config.h5 file . . . 150
11.6.3 The .random.h5 file . . . 150

12 Analyzing QMCPACK data **151**
12.1 Using the qmca tool to obtain total energies and related quantities . . . 151
12.1.1 Obtaining a statistically correct mean and error bar . . . 152
12.1.2 Judging wavefunction optimization . . . 154
12.1.3 Judging diffusion Monte Carlo runs . . . 155
12.1.4 Obtaining other quantities . . . 160
12.1.5 Processing multiple files . . . 161
12.1.6 Twist averaging . . . 163

12.1.7 Setting output units . . . 165
12.1.8 Speeding up trace plotting . . . 165
12.1.9 Short usage examples . . . 166
12.1.10 Production quality checklist . . . 167
12.2 Using the qmc-fit tool for statistical timestep extrapolation and curve fitting . . . 168
12.2.1 The jack-knife statistical technique . . . 168
12.2.2 Performing timestep extrapolation . . . 169
12.3 Densities and spin-densities . . . 170

13 Periodic LCAO for solids **171**
13.1 Introduction . . . 171
13.2 Generating and using periodic gaussian trial wavefunctions using PySCF . . . 172

14 Selected Configuration Interaction **177**
14.1 Theoretical Background . . . 177
14.1.1 CIPSI wavefunction interface . . . 179

15 Auxiliary-Field Quantum Monte Carlo **186**
15.1 Theoretical Background . . . 186
15.2 Input . . . 186
15.3 File formats . . . 191
15.4 Advice/Useful Information . . . 191
15.5 Using PySCF to generate integrals for AFQMC . . . 192

16 Examples **195**
16.1 Using QMCPACK directly . . . 195
16.2 Using Nexus . . . 195

17 Lab 1: Monte Carlo Statistical Analysis **196**
17.1 Topics covered in this Lab . . . 196
17.2 Lab directories and files . . . 196
17.3 Atomic units . . . 198
17.4 Reviewing statistics . . . 199
17.5 Inspecting Monte Carlo data . . . 200
17.6 Averaging quantities in the MC data . . . 201
17.7 Evaluating MC simulation quality . . . 202
17.7.1 Tracing MC quantities . . . 202
17.7.2 Blocking away autocorrelation . . . 204
17.7.3 Balancing autocorrelation and acceptance ratio . . . 206
17.7.4 Considering variance . . . 207
17.8 Reducing statistical error bars . . . 207
17.8.1 Increasing MC sampling . . . 207
17.8.2 Improving the basis set . . . 209
17.8.3 Adding a Jastrow factor . . . 210
17.9 Scaling to larger numbers of electrons . . . 210
17.9.1 Calculating the efficiency . . . 210
17.9.2 Scaling up . . . 210

18 Lab 2: QMC Basics **212**
18.1 Topics covered in this Lab . . . 212
18.2 Lab outline . . . 212
18.3 Lab directories and files . . . 213
18.4 Obtaining and converting a pseudopotential for oxygen . . . 213
18.5 DFT with Quantum ESPRESSO to obtain the orbital part of the wavefunction . . . 214
18.6 Optimization with QMCPACK to obtain the correlated part of the wavefunction . . . 216
18.7 DMC timestep extrapolation I: neutral O atom . . . 220
18.8 DMC timestep extrapolation II: O atom ionization potential . . . 223
18.9 DMC workflow automation with Nexus . . . 225
18.10 Automated binding curve of the oxygen dimer . . . 227
18.11 (Optional) Running your system with QMCPACK . . . 233
18.12 Appendix A: Basic Python constructs . . . 235

19 Lab 3: Advanced Molecular Calculations **240**
19.1 Topics covered in this Lab . . . 240
19.2 Lab directories and files . . . 240
19.3 Exercise #1: Basics . . . 241
19.3.1 Generation of a Hatree-Fock wave-function with GAMESS . . . 241
19.3.2 Optimize the wave-function . . . 242
19.3.3 Time-step Study . . . 243
19.3.4 Walker Population Study . . . 244
19.4 Exercise #2 Slater-Jastrow Wave-Function Options . . . 246
19.4.1 Influence of Jastrow on VMC energy with HF wave-function . . . 246
19.4.2 Generation of wave-functions from DFT using GAMESS . . . 247
19.4.3 Optimization and DMC calculations with DFT wave-functions . . . 247
19.5 Exercise #3: Multi-Determinant Wave-Functions . . . 248
19.5.1 Generation of a CISD wave-functions using GAMESS . . . 248
19.5.2 Optimization of Multi-Determinant wave-function . . . 248
19.5.3 CISD, CASCI and SOCI . . . 250
19.6 Appendix A: GAMESS input . . . 251
19.6.1 HF input . . . 251
19.6.2 DFT calculations . . . 251
19.6.3 Multi-Configuration Self-Consistent Field (MCSCF) . . . 252
19.6.4 Configuration Interaction (CI) . . . 252
19.6.5 GUGA: Unitary Group CI package . . . 252
19.6.6 ECP . . . 253
19.7 Appendix B: convert4qmc . . . 254
19.7.1 Useful notes . . . 255
19.8 Appendix C: Wave-function Optimization XML block . . . 256
19.9 Appendix D: VMC and DMC XML block . . . 258
19.10 Appendix E: Wave-function XML block . . . 260

20 Lab 4: Condensed Matter Calculations **264**
20.1 Topics covered in this Lab . . . 264
20.2 Lab directories and files . . . 264
20.3 Preliminaries . . . 265
20.4 Total energy of BCC beryllium . . . 266

20.5 Handling a 2D system: graphene . . . 269
20.6 Conclusion . . . 269

21 Lab 5: Excited State Calculations **270**
21.1 Topics covered in this Lab . . . 270
21.2 Lab directories and files . . . 270
21.3 Basics and excited state experiments . . . 270
21.4 Preparation for the excited state calculations . . . 272
21.4.1 Identifying high-symmetry k-points . . . 272
21.4.2 DFT band structure calculation along high symmetry paths . . . 273
21.4.3 Finding a supertwist which includes all the k-points of interest . . . 274
21.4.4 Identifying the indexing of k-points of interest in the supertwist . . . 276
21.5 Quasiparticle (electronic) gap calculations . . . 278
21.6 Optical gap calculations . . . 279

22 Additional Tools **281**
22.1 Initialization Tools . . . 281
22.1.1 qmc-get-supercell . . . 281
22.2 Post-Processing . . . 281
22.2.1 qmca . . . 281
22.2.2 qmc-fit . . . 281
22.2.3 qmcfinitesize . . . 281
22.3 Converters . . . 281
22.3.1 convert4qmc . . . 281
22.3.2 pw2qmcpack.x . . . 293
22.3.3 ppconvert . . . 294
22.4 Obtaining pseudopotentials . . . 295
22.4.1 Pseudopotentiallibrary.org . . . 295
22.4.2 Opium . . . 295
22.4.3 Burkatzki-Filippi-Dolg . . . 295
22.4.4 CASINO . . . 296

23 External Tools **297**
23.1 LLVM Sanitizer Libraries . . . 297
23.2 Intel VTune . . . 297
23.2.1 VTune API . . . 297
23.3 NVIDIA Tools Extensions (NVTX) . . . 298
23.3.1 NVTX API . . . 298
23.3.2 Timers as Tasks . . . 298
23.4 Scitools Understand . . . 298

24 Contributing to the Manual **299**

25 Unit Testing **303**
25.1 Unit testing framework . . . 303
25.2 Unit test organization . . . 303
25.3 Example . . . 304
25.3.1 Expected output . . . 304

25.4 Adding tests . . . 305
25.4.1 Adding a test to existing file . . . 305
25.4.2 Adding a test file . . . 305
25.4.3 Adding a test directory . . . 305
25.5 Testing with random numbers . . . 305

26 QMCPACK Design and Feature Documentation **307**
26.1 QMCPACK Design . . . 307
26.2 Feature: Optimized Long-Ranged Breakup (Ewald) . . . 308
26.2.1 The Long-Range Problem . . . 308
26.2.2 Reciprocal-Space Sums . . . 308
26.2.3 Combining Terms . . . 312
26.2.4 Computing the Reciprocal Potential . . . 312
26.2.5 The Coulomb Potential . . . 312
26.2.6 Efficient calculation methods . . . 313
26.2.7 Gaussian Charge Screening Breakup . . . 313
26.2.8 Optimized Breakup Method . . . 314
26.3 Feature: Optimized Long-Ranged Breakup (Ewald) 2 . . . 321
26.3.1 Basis functions . . . 322
26.3.2 Fourier components of the basis functions in 3D . . . 322
26.3.3 Fourier components of the basis functions in 2D . . . 324
26.3.4 Construction of the matrix elements . . . 325
26.4 Feature: Cubic Spline Interpolation . . . 326
26.4.1 One Dimension . . . 326
26.4.2 Bicubic Splines . . . 328
26.4.3 Tricubic Splines . . . 329
26.5 Feature: B-spline Orbital Tiling (Band Unfolding) . . . 331
26.5.1 The mathematics . . . 331
26.5.2 Example: FeO . . . 332
26.6 Feature: Hybrid orbital representation . . . 334
26.6.1 Real Spherical Harmonics . . . 334
26.6.2 Projecting to atomic orbitals . . . 334
26.7 Feature: Electron-electron-ion Jastrow factor . . . 336
26.8 Feature: Reciprocal-Space Jastrow Factors . . . 338
26.8.1 Two-body Jastrow . . . 338
26.8.2 One-body Jastrow . . . 338
26.8.3 Symmetry considerations . . . 339
26.8.4 Gradients and Laplacians . . . 339

27 Development Guide **340**
27.1 QMCPACK Coding Standards . . . 340
27.2 Files . . . 340
27.2.1 File organization . . . 341
27.2.2 File Names . . . 341
27.2.3 Header files . . . 341
27.2.4 Includes . . . 341
27.3 Naming . . . 342
27.3.1 Namespace Names . . . 342

27.3.2 Type and Class Names . . . 342
27.3.3 Variable Names . . . 342
27.3.4 Class Data Members . . . 342
27.3.5 (Member) Function Names . . . 342
27.3.6 Lambda Expressions . . . 342
27.3.7 Macro Names . . . 342
27.3.8 Test Case and Test Names . . . 343
27.4 Comments . . . 343
27.4.1 Comment Style . . . 343
27.4.2 Documentation . . . 343
27.5 Formatting and "Style" . . . 344
27.5.1 Indentation . . . 344
27.5.2 Line Length . . . 344
27.5.3 Horizontal Spacing . . . 344
27.5.4 Preprocessor Directives . . . 344
27.5.5 Binary Operators . . . 345
27.5.6 Unary Operators . . . 345
27.5.7 Types . . . 345
27.5.8 Pointers and References . . . 345
27.5.9 Templates . . . 345
27.5.10 Vertical Spacing . . . 345
27.5.11 Variable Declarations and Definitions . . . 345
27.5.12 Function Declarations and Definitions . . . 346
27.5.13 Conditionals . . . 346
27.5.14 Switch statement . . . 347
27.5.15 Loops . . . 347
27.5.16 Class Format . . . 348
27.5.17 Namespace Formatting . . . 349
27.6 QMCPack C++ Guidance . . . 349
27.6.1 Encapsulation . . . 350
27.6.2 Casting . . . 350
27.6.3 Pre-increment and pre-decrement . . . 350
27.6.4 Alternative Operator Representations . . . 351
27.6.5 Use of const . . . 351
27.7 Scalar Estimator Implementation . . . 351
27.7.1 Introduction: Life of a Specialized QMCHamiltonianBase . . . 351
27.7.2 Single Scalar Estimator Implementation Guide . . . 353
27.7.3 Multiple Scalars . . . 358
27.7.4 HDF5 Output . . . 360
27.8 Estimator Output . . . 361
27.8.1 Estimator Definition . . . 361
27.8.2 Class Relations . . . 362
27.8.3 Estimator Output Stages . . . 362
27.8.4 Estimator Use Cases . . . 366
27.8.5 Summary . . . 367
27.8.6 Appendix: dmc.dat . . . 367
27.9 Slater-Backflow Wavefunction Implementation Details . . . 368
27.9.1 Value . . . 368

27.9.2 Gradient . . . 368
27.9.3 Laplacian . . . 369
27.9.4 Wavefunction Parameter Derivative . . . 371

References **372**

A Derivation of twist averaging efficiency **376**

B QMCPACK papers **379**

Chapter 1

Introduction

QMCPACK is an open-source, high-performance electronic structure code that implements numerous Quantum Monte Carlo (QMC) algorithms. Its main applications are electronic structure calculations of molecular, periodic 2D, and periodic 3D solid-state systems. Variational Monte Carlo (VMC), diffusion Monte Carlo (DMC), and a number of other advanced QMC algorithms are implemented. By directly solving the Schrodinger equation, QMC methods offer greater accuracy than methods such as density functional theory but at a trade-off of much greater computational expense. Distinct from many other correlated many-body methods, QMC methods are readily applicable to both bulk (periodic) and isolated molecular systems.

QMCPACK is written in C++ and is designed with the modularity afforded by object-oriented programming. It makes extensive use of template metaprogramming to achieve high computational efficiency. Because of the modular architecture, the addition of new wavefunctions, algorithms, and observables is relatively straightforward. For parallelization, QMCPACK uses a fully hybrid (OpenMP,CUDA)/MPI approach to optimize memory usage and to take advantage of the growing number of cores per SMP node or graphical processing units (GPUs) and accelerators. High parallel and computational efficiencies are achievable on the largest supercomputers. Finally, QMCPACK uses standard file formats for input and output in XML and HDF5 to facilitate data exchange.

This manual currently serves as an introduction to the essential features of QMCPACK and as a guide to installing and running it. Over time this manual will be expanded to include a fuller introduction to QMC methods in general and to include more of the specialized features in QMCPACK.

1.1 Quickstart and a first QMCPACK calculation

In case you are keen to get started, this section describes how to quickly build and run QMCPACK on a standard UNIX or Linux-like system. The autoconfiguring build system usually works without much fuss on these systems. If C++, MPI, BLAS/LAPACK, FFTW, HDF5, and CMake are already installed, QMCPACK can be built and run within five minutes. For supercomputers, cross-compilation systems, and other computer clusters, the build system might require hints on the locations of libraries and which versions to use, typical of any code; see Chapter 3. Section 3.7 includes complete examples of installations for common workstations and supercomputers that you can reuse.

To build QMCPACK:

1. Download the latest QMCPACK distribution from `http://www.qmcpack.org`.

2. Untar the archive (e.g., `tar xvf qmcpack_v1.3.tar.gz`).

3. Check the instructions in the README file.

4. Run CMake in a suitable build directory to configure QMCPACK for your system: **`cd`** `qmcpack/build; cmake ..`

5. If CMake is unable to find all needed libraries, see Chapter 3 for instructions and specific build instructions for common systems.

6. Build QMCPACK: `make` or `make -j 16`; use the latter for a faster parallel build on a system using, for example, 16 processes.

7. The QMCPACK executable is `bin/qmcpack`.

QMCPACK is distributed with examples illustrating different capabilities. Most of the examples are designed to run quickly with modest resources. We'll run a short diffusion Monte Carlo calculation of a water molecule:

1. Go to the appropriate example directory: **`cd`** `../examples/molecules/H2O`.

2. (Optional) Put the QMCPACK binary on your path:
   `export` `PATH=$PATH:location-of-qmcpack/build/bin`.

3. Run QMCPACK: `../../../build/bin/qmcpack simple-H2O.xml` or `qmcpack simple-H2O.xml` if you followed the step above.

4. The run will output to the screen and generate a number of files:

```
$ls H2O*
H2O.HF.wfs.xml      H2O.s001.scalar.dat H2O.s002.cont.xml
H2O.s002.qmc.xml    H2O.s002.stat.h5    H2O.s001.qmc.xml
H2O.s001.stat.h5    H2O.s002.dmc.dat    H2O.s002.scalar.dat
```

5. Partially summarized results are in the standard text files with the suffixes scalar.dat and dmc.dat. They are viewable with any standard editor.

If you have Python and matplotlib installed, you can use the `qmca` analysis utility to produce statistics and plots of the data. See Chapter 12 for information on analyzing QMCPACK data.

```
export PATH=$PATH:location-of-qmcpack/nexus/bin
export PYTHONPATH=$PYTHONPATH:location-of-qmcpack/nexus/library
qmca H2O.s002.scalar.dat         # For statistical analysis of the DMC data
qmca -t -q e H2O.s002.scalar.dat # Graphical plot of DMC energy
```

The last command will produce a graph as per Fig. **??**. This shows the average energy of the DMC walkers at each timestep. In a real simulation we would have to check equilibration, convergence with walker population, time step, etc.

Congratulations, you have completed a DMC calculation with QMCPACK!

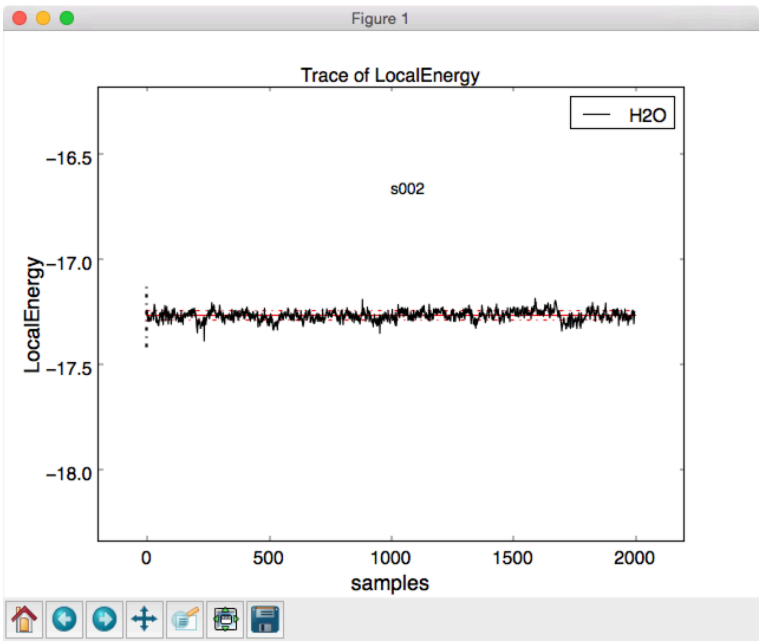


Figure 1.1: Trace of walker energies produced by the qmca tool for a simple water molecule example.

1.2 Authors and History

QMCPACK was initially written by Jeongnim Kim while in the group of Professor David Ceperley at the University of Illinois at Urbana-Champaign, with later contributations being made at Oak Ridge National Laboratory (ORNL). Over the years, many others have contributed, particularly students and researchers in the groups of Professor David Ceperley and Professor Richard M. Martin, as well as staff at Lawrence Livermore National Laboratory, Sandia National Laboratories, Argonne National Laboratory, and ORNL.

Additional developers, contributors, and advisors include Anouar Benali, Mark A. Berrill, David M. Ceperley, Simone Chiesa, Raymond C. III Clay, Bryan Clark, Kris T. Delaney, Kenneth P. Esler, Paul R. C. Kent, Jaron T. Krogel, Ying Wai Li, Ye Luo, Jeremy McMinis, Miguel A. Morales, William D. Parker, Nichols A. Romero, Luke Shulenburger, Norman M. Tubman, and Jordan E. Vincent.

If you should be added to this list, please let us know.

Development of QMCPACK has been supported financially by several grants, including the following:

- "Network for ab initio many-body methods: development, education and training" supported through the Predictive Theory and Modeling for Materials and Chemical Science program by the U.S. Department of Energy Office of Science, Basic Energy Sciences
- "QMC Endstation," supported by Accelerating Delivery of Petascale Computing Environment at the DOE Leadership Computing Facility at ORNL
- PetaApps, supported by the US National Science Foundation
- Materials Computation Center (MCC), supported by the US National Science Foundation

1.3 Support and Contacting the Developers

Questions about installing, applying, or extending QMCPACK can be posted on the QMCPACK Google group at https://groups.google.com/forum/#!forum/qmcpack. You may also email any of the developers, but we recommend checking the group first. Particular attention is given to any problem reports.

1.4 Performance

QMCPACK implements modern Monte Carlo (MC) algorithms, is highly parallel, and is written using very efficient code for high per-CPU or on-node performance. In particular, the code is highly vectorizable, giving high performance on modern central processing units (CPUs) and GPUs. We believe QMCPACK delivers performance either comparable to or better than other QMC codes when similar calculations are run, particularly for the most common QMC methods and for large systems. If you find a calculation where this is not the case, or you simply find performance slower than expected, please post on the Google group or contact one of the developers. These reports are valuable. If your calculation is sufficiently mainstream we will optimize QMCPACK to improve the performance.

1.5 Open source license

QMCPACK is distributed under the University of Illinois at Urbana-Champaign/National Center for Supercomputing Applications (UIUC/NCSA) Open Source License.

```
          permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS WITH THE SOFTWARE.
```

Copyright is generally believed to remain with the authors of the individual sections of code. See the various notations in the source code as well as the code history.

1.6 Contributing to QMCPACK

QMCPACK is fully open source, and we welcome contributions. If you are planning a development, early discussions are encouraged. Please post on the QMCPACK Google group or contact the developers. We can tell you whether anyone else is working on a similar feature or whether any related work has been done in the past. Credit for your contribution can be obtained, for example, through citation of a paper or by becoming one of the authors on the next version of the standard QMCPACK reference citation.

A guide to developing for QMCPACK, including instructions on how to work with GitHub and make pull requests (contributions) to the main source are listed on the QMCPACK GitHub wiki: https://github.com/QMCPACK/qmcpack/wiki.

Contributions are made under the same license as QMCPACK, the UIUC/NCSA open source license. If this is problematic, please discuss with a developer.

Please note the following guidelines for contributions:

- Additions should be fully synchronized with the latest release version and ideally the latest develop branch on github. Merging of code developed on older versions is error prone.
- Code should be cleanly formatted, commented, portable, and accessible to other programmers. That is, if you need to use any clever tricks, add a comment to note this, why the trick is needed, how it works, etc. Although we like high performance, ease of maintenance and accessibility are also considerations.
- Comment your code. You are not only writing it for the compiler for also for other humans! (We know this is a repeat of the previous point, but it is important enough to repeat.)
- Write a brief description of the method, algorithms, and inputs and outputs suitable for inclusion in this manual.
- Develop some short tests that exercise the functionality that can be used for validation and for examples. We can help with this and their integration into the test system.

1.7 QMCPACK Roadmap

A general outline of the QMCPACK roadmap is given in Sections 1.7.1 and 1.7.2 . Suggestions for improvements are welcome, particularly those that would facilitate new scientific applications. For example, if an interface to a particular quantum chemical or density functional code would help, this would be given strong consideration.

1.7.1 Code

We will continue to improve the accessibility and usability of QMCPACK through combinations of more convenient input parameters, improved workflow, integration with more quantum chemical and density functional codes, and a wider range of examples.

In terms of methodological development, we expect to significantly increase the range of QMC algorithms in QMCPACK in the near future.

Computationally, we are porting QMCPACK to the next generation of supercomputer systems. The internal changes required to run efficiently on these systems are expected to benefit *all* platforms due to improved vectorization, cache utilization, and memory performance.

1.7.2 Documentation

This manual describes the core features of QMCPACK that are required for routine research calculations, i.e., the VMC and DMC methods, how to obtain and optimize trial wavefunctions, and simple observables. Over time this manual will be expanded to include a broader introduction to QMC methods and to describe more features of the code.

Because of its history as a research code, QMCPACK contains a variety of additional QMC methods, trial wavefunction forms, potentials, etc., that, although not critical, might be very useful for specialized calculations or particular material or chemical systems. These "secret features" (every code has these) are not actually secret but simply lack descriptions, example inputs, and tests. You are encouraged to browse and read the source code to find them. New descriptions will be added over time but can also be prioritized and added on request (e.g., if a specialized Jastrow factor would help or a historical Jastrow form is needed for benchmarking).

Chapter 2

Features of QMCPACK

2.1 Production-level features

The following list contains the main production-level features of QMCPACK. If you do not see a specific feature that you are interested in, see the remainder of this manual and ask whether that specific feature is available or can be quickly brought to the full production level.

- Variational Monte Carlo (VMC)
- Diffusion Monte Carlo (DMC)
- Reptation Monte Carlo
- Single and multideterminant Slater Jastrow wavefunctions
- Wavefunction updates using optimized multideterminant algorithm of Clark et al.
- Backflow wavefunctions
- One, two, and three-body Jastrow factors
- Excited state calculations via flexible occupancy assignment of Slater determinants
- All electron and nonlocal pseudopotential calculations
- Casula T-moves for variational evaluation of nonlocal pseudopotentials (non-size-consistent and size-consistent variants)
- Wavefunction optimization using the "linear method" of Umrigar and coworkers, with arbitrary mix of variance and energy in the objective function
- Blocked, low memory adaptive shift optimizer of Zhao and Neuscamman
- Gaussian, Slater, plane-wave, and real-space spline basis sets for orbitals
- Interface and conversion utilities for plane-wave wavefunctions from Quantum Espresso (Plane-Wave Self-Consistent Field package [PWSCF])
- Interface and conversion utilities for Gaussian-basis wavefunctions from GAMESS

- Easy extension and interfacing to other electronic structure codes via standardized XML and HDF5 inputs
- MPI parallelism
- Fully threaded using OpenMP
- GPU (NVIDIA CUDA) implementation (limited functionality)
- HDF5 input/output for large data
- Nexus: advanced workflow tool to automate all aspects of QMC calculation from initial DFT calculations through to final analysis
- Analysis tools for minimal environments (Perl only) through to Python-based environments with graphs produced via matplotlib (included with Nexus)

2.2 SoA optimizations and improved algorithms

The Structure-of-Arrays (SoA) optimizations [17] are a set of improved data layouts facilitating vectorization on modern CPUs with wide SIMD units. **For many calculations and architectures, the SoA implementation more than doubles the speed of the code.** This so-called SoA implementation replaces the older, less efficient Array-of-Structures (AoS) code and can be enabled or disabled at compile time. The memory footprint is also reduced in the SoA implementation by better algorithms, enabling more systems to be run.

The SoA build was made the default for QMCPACK v3.7.0. As described in Section 3.6.3, the SoA implementation can be disabled by configuring with `-DENABLE_SOA=0`.

The SoA code path currently does *not* support:

- Backflow wavefunctions
- Many observables

The code should abort with a message referring to AoS vs SoA features if any unsupported feature is invoked. In this case the AoS build should be used by configuring with `-DENABLE_SOA=0`. In addition, please inform the developers via GitHub or Google Groups so that porting these features can be prioritized.

Core features are heavily tested in both SoA and AoS versions. If using untested and noncore features in the SoA code, please compare the AoS and SoA results carefully.

2.3 Supported GPU features

The GPU implementation supports multiple GPUs per node, with MPI tasks assigned in a round-robin order to the GPUs. Currently, for large runs, 1 MPI task should be used per GPU per node. For smaller calculations, use of multiple MPI tasks per GPU might yield improved performance. Supported GPU features:

- VMC, wavefunction optimization, DMC.
- Periodic and open boundary conditions. Mixed boundary conditions are not yet supported.

- Wavefunctions:
    1. Single Slater determinants with 3D B-spline orbitals. Twist-averaged boundary conditions and complex wavefunctions are fully supported. Gaussian type orbitals are not yet supported.
    2. Hybrid mixed basis representation in which orbitals are represented as 1D splines times spherical harmonics in spherical regions (muffin tins) around atoms and 3D B-splines in the interstitial region.
    3. One-body and two-body Jastrows represented as 1D B-splines. Three-body Jastrow functions are not yet supported.
- Semilocal (nonlocal and local) pseudopotentials, Coulomb interaction (electron-electron, electron-ion), and model periodic Coulomb (MPC) interaction.

2.4 Beta test features

This section describes developmental features in QMCPACK that might be ready for production but that require additional testing, features, or documentation to be ready for general use. We describe them here because they offer significant benefits and are well tested in specific cases.

2.4.1 Auxiliary-Field Quantum Monte Carlo

The orbital-space Auxiliary-Field Quantum Monte Carlo (AFMQC) method is now available in QMCPACK. The main input for the code is the matrix elements of the Hamiltonian in a given single particle basis set, which must be produced from mean-field calculations such as Hartree-Fock or density functional theory. The code and many features are in development. Check the latest version of QMCPACK for an up-to-date description of available features. A partial list of the current capabilities of the code follows. For a detailed description of the available features, see chapter 15.

- Phaseless AFQMC algorithm of Zhang et al. (S. Zhang and H. Krakauer. 2003. "Quantum Monte Carlo Method using Phase-Free Random Walks with Slater Determinants." *PRL* 90: 136401).
- "Hybrid" and "local energy" propagation schemes.
- Hamiltonian matrix elements from (1) Molpro's FCIDUMP format (which can be produced by Molpro, PySCF, and VASP) and (2) internal HDF5 format produced by PySCF (see AFQMC section below).
- AFQMC calculations with RHF (closed-shell doubly occupied), ROHF (open-shell doubly occupied), and UHF (spin polarized broken symmetry) symmetry.
- Single and multideterminant trial wavefunctions. Multideterminant expansions with either orthogonal or nonorthogonal determinants.
- Fast update scheme for orthogonal multideterminant expansions.
- Distributed propagation algorithms for large systems. Enables calculations where data structures do not fit on a single node.

- Complex implementation for PBC calculations with complex integrals.
- Sparse representation of large matrices for reduced memory usage.
- Mixed and back-propagated estimators.
- Specialized implementation for solids with k-point symmetry (e.g. primitive unit cells with kpoints).
- Efficient GPU implementation (currently limited to solids with k-point symmetry).

2.4.2 Sharing of spline data across multiple GPUs

Sharing of GPU spline data enables distribution of the data across multiple GPUs on a given computational node. For example, on a two-GPU-per-node system, each GPU would have half of the orbitals. This allows use of larger overall spline tables than would fit in the memory of individual GPUs and potentially up to the total GPU memory on a node. To obtain high performance, large electron counts or a high-performing CPU-GPU interconnect is required.

To use this feature, the following needs to be done:

- The CUDA Multi-Process Service (MPS) needs to be used (e.g., on OLCF Summit/SummitDev use "-alloc_flags gpumps" for bsub). If MPI is not detected, sharing will be disabled.
- CUDA_VISIBLE_DEVICES needs to be properly set to control each rank's visible CUDA devices (e.g., on OLCF Summit/SummitDev create a resource set containing all GPUs with the respective number of ranks with "jsrun –task-per-rs Ngpus -g Ngpus").
- In the determinant set definition of the <wavefunction> section, the "gpusharing" parameter needs to be set (i.e., <determinantset gpusharing="yes">). See Section 8.3.1.

Chapter 3

Obtaining, installing, and validating QMCPACK

This chapter describes how to obtain, build, and validate QMCPACK. This process is designed to be as simple as possible and should be no harder than building a modern plane-wave density functional theory code such as Quantum ESPRESSO, QBox, or VASP. Parallel builds enable a complete compilation in under 2 minutes on a fast multicore system. If you are unfamiliar with building codes we suggest working with your system administrator to install QMCPACK.

3.1 Installation steps

To install QMCPACK, follow the steps below. Full details of each step are given in the referenced sections.

1. Download the source code (Section 3.2 or 3.3).
2. Verify that you have the required compilers, libraries, and tools installed (Section 3.4).
3. Run the cmake configure step and build with make (Sections 3.6 and 3.6.1). Examples for common systems are given in Section 3.7.
4. Run the tests to verify QMCPACK (Section 3.9).
5. Build the ppconvert utility in QMCPACK (Section 3.11).
6. Download and patch Quantum ESPRESSO. This patch adds the pw2qmcpack utility (Section 3.12).

Hints for high performance are in Section 3.13. Troubleshooting suggestions are in Section 3.14.

Note that there are two different QMCPACK executables that can be produced: the general one, which is the default, and the "complex" version, which supports periodic calculations at arbitrary twist angles and k-points. This second version is enabled via a cmake configuration parameter (see Section 3.6.3). The general version supports only wavefunctions that can be made real. If you run a calculation that needs the complex version, QMCPACK will stop and inform you.

3.2 Obtaining the latest release version

Major releases of QMCPACK are distributed from http://www.qmcpack.org. Because these versions undergo the most testing, we encourage using them for all production calculations unless there are specific reasons not to do so.

Releases are usually compressed tar files that indicate the version number, date, and often the source code revision control number corresponding to the release. To obtain the latest release:

- Download the latest QMCPACK distribution from http://www.qmcpack.org.
- Untar the archive (e.g., `tar xvf qmcpack_v1.3.tar.gz`).

Releases can also be obtained from the 'master' branch of the QMCPACK git repository, similar to obtaining the development version (Section 3.3).

3.3 Obtaining the latest development version

The most recent development version of QMCPACK can be obtained anonymously via

```
git clone https://github.com/QMCPACK/qmcpack.git
```

Once checked out, updates can be made via the standard `git pull`.

The 'develop' branch of the git repository contains the day-to-day development source with the latest updates, bug fixes, etc. This version might be useful for updates to the build system to support new machines, for support of the latest versions of Quantum ESPRESSO, or for updates to the documentation. Note that the development version might not be fully consistent with the online documentation. We attempt to keep the development version fully working. However, please be sure to run tests and compare with previous release versions before using for any serious calculations. We try to keep bugs out, but occasionally they crawl in! Reports of any breakages are appreciated.

3.4 Prerequisites

The following items are required to build QMCPACK. For workstations, these are available via the standard package manager. On shared supercomputers this software is usually installed by default and is often accessed via a modules environment—check your system documentation.

Use of the latest versions of all compilers and libraries is strongly encouraged but not absolutely essential. Generally, newer versions are faster; see Section 3.13 for performance suggestions.

- C/C++ compilers such as GNU, Clang, Intel, and IBM XL. C++ compilers are required to support the C++ 14 standard. Use of recent ("current year version") compilers is strongly encouraged.
- An MPI library such as OpenMPI (http://open-mpi.org) or a vendor-optimized MPI.
- BLAS/LAPACK, numerical, and linear algebra libraries. Use platform-optimized libraries where available, such as Intel MKL. ATLAS or other optimized open source libraries can also be used (http://math-atlas.sourceforge.net).

- CMake, build utility (http://www.cmake.org).
- Libxml2, XML parser (http://xmlsoft.org).
- HDF5, portable I/O library (http://www.hdfgroup.org/HDF5/). Good performance at large scale requires parallel version >= 1.10.
- BOOST, peer-reviewed portable C++ source libraries (http://www.boost.org). Minimum version is 1.61.0.
- FFTW, FFT library (http://www.fftw.org/).

To build the GPU accelerated version of QMCPACK, an installation of NVIDIA CUDA development tools is required. Ensure that this is compatible with the C and C++ compiler versions you plan to use. Supported versions are included in the NVIDIA release notes.

Many of the utilities provided with QMCPACK use Python (v2). The numpy and matplotlib libraries are required for full functionality.

Note that the standalone einspline library used by previous versions of QMCPACK is no longer required. A more optimized version is included inside. The standalone version should *not* be on any standard search paths because conflicts between the old and new include files can result.

3.5 C++ 14 standard library

The C++ standard consists of language features—which are implemented in the compiler—and library features—which are implemented in the standard library. GCC includes its own standard library and headers, but many compilers do not and instead reuse those from an existing GCC install. Depending on setup and installation, some of these compilers might not default to using a GCC with C++ 14 headers (e.g., GCC 4.8 is common as a base system compiler, but its standard library only supports C++ 11).

The symptom of having header files that do not support the C++ 14 standard is usually compile errors involving standard include header files. Look for the GCC library version, which should be present in the path to the include file in the error message, and ensure that it is 5.0 or greater. To avoid these errors occurring at compile time, QMCPACK tests for a C++ 14 standard library during configuration and will halt with an error if one is not found.

At sites that use modules, running `module load gcc` is often sufficient to load a newer GCC and resolve the issue.

3.5.1 Intel compiler

The Intel compiler version must be 18 or newer. The version 17 compiler cannot compile some of the C++ 14 constructs in the code.

If a newer GCC is needed, the `-cxxlib` option can be used to point to a different GCC installation. (Alternately, the `-gcc-name` or `-gxx-name` options can be used.) Be sure to pass this flag to the C compiler in addition to the C++ compiler. This is necessary because CMake extracts some library paths from the C compiler, and those paths usually also contain to the C++ library. The symptom of this problem is C++ 14 standard library functions not found at link time.

3.6 Building with CMake

The build system for QMCPACK is based on CMake. It will autoconfigure based on the detected compilers and libraries. The most recent version of CMake has the best detection for the greatest variety of systems. The minimum required version of CMake is 3.6, which is the oldest version to support correct application of C++ 14 flags for the Intel compiler. Most computer installations have a sufficiently recent CMake, though it might not be the default.

If no appropriate version CMake is available, building it from source is straightforward. Download a version from https://cmake.org/download/ and unpack the files. Run `./boostrap` from the CMake directory, and then run `make` when that finishes. The resulting CMake executable will be in the `bin/` directory. The executable can be run directly from that location.

Previously, QMCPACK made extensive use of toolchains, but the build system has since been updated to eliminate the use of toolchain files for most cases. The build system is verified to work with GNU, Intel, and IBM XLC compilers. Specific compile options can be specified either through specific environment or CMake variables. When the libraries are installed in standard locations (e.g., /usr, /usr/local), there is no need to set environment or CMake variables for the packages.

3.6.1 Quick build instructions (try first)

If you are feeling lucky and are on a standard UNIX-like system such as a Linux workstation, the following might quickly give a working QMCPACK:

The safest quick build option is to specify the C and C++ compilers through their MPI wrappers. Here we use Intel MPI and Intel compilers. Move to the build directory, run CMake, and make

```
cd build
cmake -DCMAKE_C_COMPILER=mpiicc -DCMAKE_CXX_COMPILER=mpiicpc ..
make -j 8
```

You can increase the "8" to the number of cores on your system for faster builds. Substitute mpicc and mpicxx or other wrapped compiler names to suit your system. For example, with OpenMPI use

```
cd build
cmake -DCMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx ..
make -j 8
```

If you are feeling particularly lucky, you can skip the compiler specification:

```
cd build
cmake ..
make -j 8
```

The complexities of modern computer hardware and software systems are such that you should check that the autoconfiguration system has made good choices and picked optimized libraries and compiler settings before doing significant production. That is, check the following details. We give examples for a number of common systems in Section 3.7.

3.6.2 Environment variables

A number of environment variables affect the build. In particular they can control the default paths for libraries, the default compilers, etc. The list of environment variables is given below:

```
CXX             C++ compiler
CC              C Compiler
MKL_ROOT        Path for MKL
LIBXML2_HOME    Path for libxml2
HDF5_ROOT       Path for HDF5
BOOST_ROOT      Path for Boost
FFTW_HOME       Path for FFTW
```

3.6.3 Configuration options

In addition to reading the environment variables, CMake provides a number of optional variables that can be set to control the build and configure steps. When passed to CMake, these variables will take precedent over the environment and default variables. To set them, add -D FLAG=VALUE to the configure line between the CMake command and the path to the source directory.

- Key QMCPACK build options

```
QMC_CUDA              Enable CUDA and GPU acceleration (1:yes, 0:no)
QMC_COMPLEX           Build the complex (general twist/k-point) version (1:yes,
    0:no)
QMC_MIXED_PRECISION   Build the mixed precision (mixing double/float) version
                      (1:yes (GPU default), 0:no (CPU default)).
                      The CPU support is experimental.
                      Use float and double for base and full precision.
                      The GPU support is quite mature.
                      Use always double for host side base and full precision
                      and use float and double for CUDA base and full precision.
ENABLE_SOA            Enable data layout and algorithm optimizations using
                      Structure-of-Array (SoA) datatypes (1:yes (default), 0:no).
ENABLE_TIMERS         Enable fine-grained timers (1:yes, 0:no (default)).
                      Timers are off by default to avoid potential slowdown in
    small
                      systems. For large systems (100+ electrons) there is no
    risk.
```

- General build options

```
CMAKE_BUILD_TYPE     A variable which controls the type of build
                     (defaults to Release). Possible values are:
                     None (Do not set debug/optmize flags, use
                     CMAKE_C_FLAGS or CMAKE_CXX_FLAGS)
                     Debug (create a debug build)
                     Release (create a release/optimized build)
                     RelWithDebInfo (create a release/optimized build with debug
    info)
                     MinSizeRel (create an executable optimized for size)
CMAKE_C_COMPILER     Set the C compiler
CMAKE_CXX_COMPILER   Set the C++ compiler
CMAKE_C_FLAGS        Set the C flags.  Note: to prevent default
                     debug/release flags from being used, set the
    CMAKE_BUILD_TYPE=None
                     Also supported: CMAKE_C_FLAGS_DEBUG,
                     CMAKE_C_FLAGS_RELEASE, and CMAKE_C_FLAGS_RELWITHDEBINFO
CMAKE_CXX_FLAGS      Set the C++ flags.  Note: to prevent default
                     debug/release flags from being used, set the
    CMAKE_BUILD_TYPE=None
```

```
                     Also supported: CMAKE_CXX_FLAGS_DEBUG,
                     CMAKE_CXX_FLAGS_RELEASE, and CMAKE_CXX_FLAGS_RELWITHDEBINFO
CMAKE_INSTALL_PREFIX Set the install location (if using the optional install
    step)
INSTALL_NEXUS        Install Nexus alongside QMCPACK (if using the optional
    install step)
```

- Additional QMCPACK build options

```
QMC_INCLUDE            Add extra include paths
QMC_EXTRA_LIBS         Add extra link libraries
QMC_BUILD_STATIC       Add -static flags to build
QMC_DATA               Specify data directory for QMCPACK performance and
    integration tests
QE_BIN                 Location of Quantum Espresso binaries including
    pw2qmcpack.x
QMC_SYMLINK_TEST_FILES Set to zero to require test files to be copied. Avoids
    space
                       saving default use of symbolic links for test files.
    Useful
                       if the build is on a separate filesystem from the source,
    as
                       required on some HPC systems.
```

- Intel MKL related

```
ENABLE_MKL          Enable Intel MKL libraries (1:yes (default for intel
    compiler),
                                                0:no (default otherwise)).
MKL_ROOT            Path to MKL libraries (only necessary for non intel compilers
                    or intel without standard environment variables.)
                    One of the above environment variables can be used.
```

- libxml2 related

```
Libxml2_INCLUDE_DIRS   Specify include directories for libxml2
Libxml2_LIBRARY_DIRS   Specify library directories for libxml2
```

- HDF5 related

```
ENABLE_PHDF5    1(default)/0, enables/disable parallel collective IO.
```

- FFTW related

```
FFTW_INCLUDE_DIRS   Specify include directories for FFTW
FFTW_LIBRARY_DIRS   Specify library directories for FFTW
```

- CTest related

```
MPIEXEC                Specify the mpi wrapper, e.g. srun, aprun, mpirun, etc.
MPIEXEC_NUMPROC_FLAG   Specify the number of mpi processes flag,
                       e.g. "-n", "-np", etc.
```

- LLVM/Clang Developer Options

```
LLVM_SANITIZE_ADDRESS     link with the Clang address sanitizer library
LLVM_SANITIZE_MEMORY      link with the Clang memory sanitizer library
```

See Section 23.1 for more information.

3.6.4 Installation from CMake

Installation is optional. The QMCPACK executable can be run from the `bin` directory in the build location. If the install step is desired, run the `make install` command to install the QMCPACK executable, the converter, and some additional executables. Also installed is the `qmcpack.settings` file that records options used to compile QMCPACK. Specify the `CMAKE_INSTALL_PREFIX` CMake variable during configuration to set the install location.

3.6.5 Role of QMC_DATA

QMCPACK includes a variety of optional performance and integration tests that use research quality wavefunctions to obtain meaningful performance and to more thoroughly test the code. The necessarily large input files are stored in the location pointed to by QMC_DATA (e.g., scratch or long-lived project space on a supercomputer). These files are not included in the source code distribution to minimize size. The tests are activated if CMake detects the files when configured. See tests/performance/NiO/README, tests/solids/NiO_afqmc/README, and tests/performance/C-graphite/README for details of the current tests and input files and to download them.

Currently the files must be downloaded via
https://anl.box.com/s/pveyyzrc2wuvg5tmxjzzwxeo561vh3r0 and
https://anl.box.com/s/j5d6jeazpgx5441s04ajtv77iogoyjsh.

Current complete set of files:

```
QMC_DATA/C-graphite/lda.pwscf.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S1.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S2.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S4.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S8.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S16.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S32.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S64.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S128.h5
QMC_DATA/NiO/NiO-fcc-supertwist111-supershift000-S256.h5
QMC_DATA/NiO/NiO_afm_fcidump.h5
QMC_DATA/NiO/NiO_afm_wfn.dat
QMC_DATA/NiO/NiO_nm_choldump.h5
```

3.6.6 Configure and build using CMake and make

To configure and build QMCPACK, move to build directory, run CMake, and make

```
cd build
cmake ..
make -j 8
```

As you will have gathered, CMake encourages "out of source" builds, where all the files for a specific build configuration reside in their own directory separate from the source files. This

allows multiple builds to be created from the same source files, which is very useful when the file system is shared between different systems. You can also build versions with different settings (e.g., QMC_COMPLEX) and different compiler settings. The build directory does not have to be called build—use something descriptive such as build_machinename or build_complex. The ".." in the CMake line refers to the directory containing CMakeLists.txt. Update the ".." for other build directory locations.

3.6.7 Example configure and build

- Set the environments (the examples below assume bash, Intel compilers, and MKL library)

```
export CXX=icpc
export CC=icc
export LIBXML2_HOME=/usr/local
export MKL_ROOT=/usr/local/intel/mkl/10.0.3.020
export HDF5_ROOT=/usr/local
export BOOST_ROOT=/usr/local/boost
export FFTW_HOME=/usr/local/fftw
```

- Move to build directory, run CMake, and make

```
cd build
cmake -D CMAKE_BUILD_TYPE=Release ..
make -j 8
```

3.6.8 Build scripts

We recommended creating a helper script that contains the configure line for CMake. This is particularly useful when avoiding environment variables, packages are installed in custom locations, or the configure line is long or complex. In this case it is also recommended to add "rm -rf CMake*" before the configure line to remove existing CMake configure files to ensure a fresh configure each time the script is called. Deleting all the files in the build directory is also acceptable. If you do so we recommend adding some sanity checks in case the script is run from the wrong directory (e.g., checking for the existence of some QMCPACK files).

Some build script examples for different systems are given in the config directory. For example, on Cray systems these scripts might load the appropriate modules to set the appropriate programming environment, specific library versions, etc.

An example script build.sh is given below. It is much more complex than usually needed for comprehensiveness:

```
export CXX=mpic++
export CC=mpicc
export ACML_HOME=/opt/acml-5.3.1/gfortran64
export HDF5_ROOT=/opt/hdf5
export BOOST_ROOT=/opt/boost

rm -rf CMake*

cmake                                              \
  -D CMAKE_BUILD_TYPE=Debug                        \
  -D Libxml2_INCLUDE_DIRS=/usr/include/libxml2     \
  -D Libxml2_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu \
  -D FFTW_INCLUDE_DIRS=/usr/include                \
```

```
  -D FFTW_LIBRARY_DIRS=/usr/lib/x86_64-linux-gnu    \
  -D QMC_EXTRA_LIBS="-ldl ${ACML_HOME}/lib/libacml.a -lgfortran" \
  -D QMC_DATA=/projects/QMCPACK/qmc-data            \
  ..
```

3.6.9 Using vendor-optimized numerical libraries (e.g., Intel MKL

)

Although QMC does not make extensive use of linear algebra, use of vendor-optimized libraries is strongly recommended for highest performance. BLAS routines are used in the Slater determinant update, the VMC wavefunction optimizer, and to apply orbital coefficients in local basis calculations. Vectorized math functions are also beneficial (e.g., for the phase factor computation in solid-state calculations). CMake is generally successful in finding these libraries, but specific combinations can require additional hints, as described in the following:

Using Intel MKL with non-Intel compilers

To use Intel MKL with, e.g. an MPICH wrapped gcc:

```
cmake \
 -DCMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx \
 -DENABLE_MKL=1 -DMKL_ROOT=$MKLROOT/lib \
 ..
```

MKLROOT is the directory containing the MKL binary, examples, and lib directories (etc.) and is often /opt/intel/mkl.

Serial or multithreaded library

Vendors might provide both serial and multithreaded versions of their libraries. Using the right version is critical to QMCPACK performance. QMCPACK makes calls from both inside and outside threaded regions. When being called from outside an OpenMP parallel region, the multithreaded version is preferred for the possibility of using all the available cores. When being called from every thread inside an OpenMP parallel region, the serial version is preferred for not oversubscribing the cores. Fortunately, nowadays the multithreaded versions of many vendor libraries (MKL, ESSL) are OpenMP aware. They use only one thread when being called inside an OpenMP parallel region. This behavior meets exactly both QMCPACK needs and thus is preferred. If the multithreaded version does not provide this feature of dynamically adjusting the number of threads, the serial version is preferred. In addition, thread safety is required no matter which version is used.

3.6.10 Cross compiling

Cross compiling is often difficult but is required on supercomputers with distinct host and compute processor generations or architectures. QMCPACK tried to do its best with CMake to facilitate cross compiling.

- On a machine using a Cray programming environment, we rely on compiler wrappers provided by Cray to correctly set architecture-specific flags. The CMake configure log should indicate that a Cray machine was detected.
- If not on a Cray machine, by default we assume building for the host architecture (e.g., -xHost is added for the Intel compiler and -march=native is added for GNU/Clang compilers).

- If -x/-ax or -march is specified by the user in CMAKE_C_FLAGS and CMAKE_CXX_FLAGS, we respect the user's intention and do not add any architecture-specific flags.

The general strategy for cross compiling should therefore be to manually set CMAKE_C_FLAGS and CMAKE_CXX_FLAGS for the target architecture. Using `make VERBOSE=1` is a useful way to check the final compilation options. If on a Cray machine, selection of the appropriate programming environment should be sufficient.

3.7 Installation instructions for common workstations and supercomputers

This section describes how to build QMCPACK on various common systems including multiple Linux distributions, Apple OS X, and various supercomputers. The examples should serve as good starting points for building QMCPACK on similar machines. For example, the software environment on modern Crays is very consistent. Note that updates to operating systems and system software might require small modifications to these recipes. See Section 3.13 for key points to check to obtain highest performance and Section 3.14 for troubleshooting hints.

3.7.1 Installing on Ubuntu Linux or other apt-get–based distributions

The following is designed to obtain a working QMCPACK build on, for example, a student laptop, starting from a basic Linux installation with none of the developer tools installed. Fortunately, all the required packages are available in the default repositories making for a quick installation. Note that for convenience we use a generic BLAS. For production, a platform-optimized BLAS should be used.

```
apt-get cmake g++ openmpi-bin libopenmpi-dev libboost-dev
apt-get libatlas-base-dev liblapack-dev libhdf5-dev libxml2-dev fftw3-dev
export CXX=mpiCC
cd build
cmake ..
make -j 8
ls -l bin/qmcpack
```

For qmca and other tools to function, we install some Python libraries:

```
sudo apt-get install python-numpy python-matplotlib
```

3.7.2 Installing on CentOS Linux or other yum-based distributions

The following is designed to obtain a working QMCPACK build on, for example, a student laptop, starting from a basic Linux installation with none of the developer tools installed. CentOS 7 (Red Hat compatible) is using gcc 4.8.2. The installation is complicated only by the need to install another repository to obtain HDF5 packages that are not available by default. Note that for convenience we use a generic BLAS. For production, a platform-optimized BLAS should be used.

```
sudo yum install make cmake gcc gcc-c++ openmpi openmpi-devel fftw fftw-devel \
                 boost boost-devel libxml2 libxml2-devel
sudo yum install blas-devel lapack-devel atlas-devel
module load mpi
```

To setup repoforge as a source for the HDF5 package, go to http://repoforge.org/use. Install the appropriate up-to-date release package for your operating system. By default, CentOS Firefox will offer to run the installer. The CentOS 6.5 settings were still usable for HDF5 on CentOS 7 in 2016, but use CentOS 7 versions when they become available.

```
sudo yum install hdf5 hdf5-devel
```

To build QMCPACK:

```
module load mpi/openmpi-x86_64
which mpirun
# Sanity check; should print something like   /usr/lib64/openmpi/bin/mpirun
export CXX=mpiCC
cd build
cmake ..
make -j 8
ls -l bin/qmcpack
```

3.7.3 Installing on Mac OS X using Macports

These instructions assume a fresh installation of macports and use the gcc 6.1 compiler. Older versions are fine, but it is vital to ensure that matching compilers and libraries are used for all packages and to force use of what is installed in /opt/local. Performance should be very reasonable. Note that we use the Apple-provided Accelerate framework for optimized BLAS.

Follow the Macports install instructions at https://www.macports.org/.

- Install Xcode and the Xcode Command Line Tools.
- Agree to Xcode license in Terminal: sudo xcodebuild -license.
- Install MacPorts for your version of OS X.

Install the required tools:

```
sudo port install gcc6
sudo port select gcc mp-gcc6
sudo port install openmpi-devel-gcc6
sudo port select --set mpi openmpi-devel-gcc61-fortran

sudo port install fftw-3 +gcc6
sudo port install libxml2
sudo port install cmake
sudo post install boost +gcc6
sudo port install hdf5 +gcc6

sudo port select --set python python27
sudo port install py27-numpy +gcc6
sudo port install py27-matplotlib  #For graphical plots with qmca
```

QMCPACK build:

```
cd build
cmake -DCMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpiCXX ..
make -j 6 # Adjust for available core count
ls -l bin/qmcpack
```

Cmake should pickup the versions of HDF5 and libxml (etc.) installed in /opt/local by macports. If you have other copies of these libraries installed and wish to force use of a specific version, use the environment variables detailed in Section 3.6.2.

This recipe was verified on July 1, 2016, on a Mac running OS X 10.11.5 "El Capitain."

3.7.4 Installing on Mac OS X using Homebrew (brew)

Homebrew is a package manager for OS X that provides a convenient route to install all the QMCPACK dependencies. The following recipe will install the latest available versions of each package. This was successfully tested under OS X 10.12 "Sierra" in December 2017. Note that it is necessary to build the MPI software from source to use the brew-provided gcc instead of Apple CLANG.

1. Install Homebrew from http://brew.sh/:

```
/usr/bin/ruby -e "$(curl -fsSL
    https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Install the prerequisites:

```
brew install gcc # installs gcc 7.2.0 on 2017-12-19
export HOMEBREW_CXX=g++-7
export HOMEBREW_CC=gcc-7
brew install mpich2 --build-from-source
# Build from source required to use homebrew compiled compilers as
# opposed to Apple CLANG. Check "mpicc -v" indicates Homebrew gcc
brew install cmake
brew install fftw
brew install boost
brew install homebrew/science/hdf5
#Note: Libxml2 is not required via brew since OS X already includes it.
```

3. Configure and build QMCPACK:

```
cmake -DCMAKE_C_COMPILER=/usr/local/bin/mpicc \
      -DCMAKE_CXX_COMPILER=/usr/local/bin/mpicxx ..
make -j 12
```

4. Run the short tests. When MPICH is used for the first time, OS X will request approval of the network connection for each executable.

```
ctest -R short -LE unstable
```

3.7.5 Installing on ANL ALCF Mira/Cetus IBM Blue Gene/Q

Mira/Cetus is a Blue Gene/Q supercomputer at Argonne National Laboratory's Argonne Leadership Computing Facility (ANL ALCF). Mira has 49,152 compute nodes, and each node has a 16-core PowerPC A2 processor with 16 GB DDR3 memory. Because the login nodes and the compute nodes have different processors with distinct instruction sets, cross compiling is required on this platform. See details about using Blue Gene/Q at http://www.alcf.anl.gov/user-guides/compiling-linking. On Mira, compilers are loaded via softenv, and users need to add +mpiwrapper-bgclang-mpi3 and +cmake-3.8.1 in $HOME/.soft. To build QMCPACK, a toolchain file is provided for setting up CMake. **BGClang is required for C++ 14 support. IBM XL C/C++ compiler should not be used.**

```
cd build
cmake -DCMAKE_TOOLCHAIN_FILE=../config/BGQ_Clang_ToolChain.cmake ..
make -j 16
ls -l bin/qmcpack
```

3.7.6 Installing on ALCF Theta, Cray XC40

Theta is a 9.65 petaflops system manufactured by Cray with 3,624 compute nodes. Each node features a second-generation Intel Xeon Phi 7230 processor and 192 GB DDR4 RAM.

```
export CRAYPE_LINK_TYPE=dynamic
# Do not use cmake 3.9.1, it causes trouble with parallel HDF5.
module load cmake/3.11.4
module unload cray-libsci
module load cray-hdf5-parallel
module load gcc   # Make C++ 14 standard library available to the Intel compiler
export BOOST_ROOT=/soft/libraries/boost/1.64.0/intel
cmake ..
make -j 24
ls -l bin/qmcpack
```

3.7.7 Installing on ORNL OLCF Titan Cray XK7 (NVIDIA GPU accelerated)

Titan is a GPU-accelerated supercomputer at Oak Ridge National Laboratory's Oak Ridge Leadership Computing Facility (ORNL OLCF). Each compute node has a 16 core AMD 2.2GHz Opteron 6274 (Interlagos) and an NVIDIA Kepler accelerator. The standard Cray software environment is available, with libraries accessed via modules. The only extra settings required to build the GPU version are the cudatoolkit module and specifying -DQMC_CUDA=1 on the CMake configure line.

Note that on Crays, the compiler wrappers "CC" and "cc" are used. The build system checks for these and does not (should not) use the compilers directly.

```
export CRAYPE_LINK_TYPE=dynamic
module swap PrgEnv-pgi PrgEnv-gnu # Use gnu compilers
module load cudatoolkit           # CUDA for GPU build
module load cray-hdf5-parallel
module load cmake3
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
mkdir build_titan_gpu
cd build_titan_gpu
export CC=cc
export CXX=CC
cmake -DQMC_CUDA=1 ..             # Must enable CUDA capabilities
make -j 8
ls -l bin/qmcpack
```

3.7.8 Installing on ORNL OLCF Titan Cray XK7 (CPU version)

As noted in Section 3.7.7 for the GPU, building on Crays requires only loading the appropriate library modules.

```
export CRAYPE_LINK_TYPE=dynamic
module swap PrgEnv-pgi PrgEnv-gnu # Use gnu compilers
module unload cudatoolkit         # No CUDA for CPU build
module load cray-hdf5-parallel
module load cmake3
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
mkdir build_titan_cpu
cd build_titan_cpu
export CC=cc
export CXX=CC
cmake ..
make -j 8
ls -l bin/qmcpack
```

3.7.9 Installing on ORNL OLCF Eos Cray XC30

Eos is a Cray XC30 with 16 core Intel Xeon E5-2670 processors connected by the Aries interconnect. The build process is identical to Titan except that we use the default Intel programming environment. This is usually preferred to GNU.

```
export CRAYPE_LINK_TYPE=dynamic
module unload cray-libsci
module load cray-hdf5
module load cmake3/3.6.1   # Or newer
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
module swap gcc gcc/6.3.0  # Make C++ 14 standard library available to the Intel
    compiler
mkdir build_eos
cd build_eos
cmake ..
make -j 8
ls -l bin/qmcpack
```

3.7.10 Installing on ORNL OLCF Summit

Summit is an IBM system at the ORNL OLCF built with IBM Power System AC922 nodes. They have two IBM Power 9 processors and six NVIDIA Volta V100 accelerators.

Building QMCPACK

Note that these build instructions are preliminary as the software environment is subject to change. As of December 2018, the IBM XL compiler does not support C++14, so we currently use the gnu compiler.

```
module load gcc
module load essl
module load netlib-lapack #because ESSL does not provided needed LAPACK functionality
module load hdf5
module load fftw
export FFTW_HOME=$OLCF_FFTW_ROOT
module load cmake
```

```
module load boost
module load cuda
module load python/2.7.15-anaconda2-5.3.0
mkdir build_summit
cd build_summit
cmake -DCMAKE_C_COMPILER="mpicc" \
      -DCMAKE_CXX_COMPILER="mpicxx" \
      -DBUILD_LMYENGINE_INTERFACE=0 \
      -DQMC_CUDA=1 \
      -DCUDA_ARCH="sm_70" \
      ..
make -j 8
ls -l bin/qmcpack
```

Building Quantum Espresso

The v6.3 release of Quantum Espresso (QE) does not officially support the Power 9 architecture, and several steps of the configuration process require updating to build successfully. v6.4 is expected to officially support the new architecture. The following can be used to build a CPU version of QE on Summit, placing the script in the external_codes/quantum_espresso directory. Note that performance is not yet optimized although vendor libraries are used. Alternatively, the wavefunction files can be generated on another system and the converted HDF5 files copied over.

```
#!/usr/bin/bash
module load gcc/6.4.0 #6.4.0 was default on 2019-01-03
module load essl
module load netlib-lapack #because ESSL does not provided needed LAPACK functionality
module load hdf5
module load fftw
export FFTW_HOME=$OLCF_FFTW_ROOT
module -t list
./download_and_patch_qe6.3.sh
cd qe-6.3
cd archive
mv fox.tgz fox.tgz_orig
wget https://gitlab.com/QEF/q-e/raw/develop/archive/fox.tgz # Latest FoX on develop
    is patched for PPC64 architectures
cd ..
./configure --with-hdf5=$OLCF_HDF5_ROOT
cp -p make.inc make.inc_orig
sed -e 's/libhdf5.a /libhdf5.a -L/g' make.inc_orig >make.inc # Incorrect awk in
    install/configure drops library dir for zlib
sed -i 's/-D__FFTW3/-D__LINUX_ESSL/g' make.inc
sed -i "s|-lblas|-L${OLCF_ESSL_ROOT}/lib64 -lessl|g" make.inc
sed -i 's/-lfftw3/ /g' make.inc # Use the ESSL FFTW
sed -i 's/^LAPACK_LIBS .*=.*/LAPACK_LIBS = -lessl -llapack -lessl/g' make.inc
echo --- Starting build `date`
make all
echo --- Finished build `date`
```

3.7.11 Installing on NERSC Edison Cray XC30

Edison is a Cray XC30 with dual 12-core Intel "Ivy Bridge" nodes installed at the National Energy Research Scientific Computing Center (NERSC). The build settings are identical to Eos.

```
export CRAYPE_LINK_TYPE=dynamic
module unload cray-libsci
module load boost
module load cmake/3.11.4
module load libxml2
module load cray-hdf5-parallel
module load gcc   # Make C++ 14 standard library available to the Intel compiler
cmake ..
make -j 8
ls -l bin/qmcpack
```

When the preceding was tested on October 30, 2018, the following module and software versions were present:

```
build> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                                    14)
    alps/6.6.43-6.0.7.0_26.4__ga796da3.ari
  2) nsg/1.2.0                                           15)
    rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari
  3) intel/18.0.1.163                                    16) atp/2.1.1
  4) craype-network-aries                                17) PrgEnv-intel/6.0.4
  5) craype/2.5.14                                       18) craype-ivybridge
  6) udreg/2.3.2-6.0.7.0_33.18__g5196236.ari             19) cray-mpich/7.7.0
  7) ugni/6.0.14.0-6.0.7.0_23.1__gea11d3d.ari            20) altd/2.0
  8) pmi/5.0.13                                          21) darshan/3.1.4
  9) dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari              22) boost/1.63
 10) gni-headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari     23) cmake/3.11.4
 11) xpmem/2.2.15-6.0.7.1_5.8__g7549d06.ari              24) libxml2/2.9.4
 12) job/2.2.3-6.0.7.0_44.1__g6c4e934.ari                25) cray-hdf5-parallel/1.10.1.1
 13) dvs/2.7_2.2.112-6.0.7.1_6.4__ge96a422               26) gcc/7.3.0
```

3.7.12 Installing on NERSC Cori, Haswell Partition, Cray XC40

Cori is a Cray XC40 with 16-core Intel "Haswell" nodes installed at NERSC.

```
export CRAYPE_LINK_TYPE=dynamic
module unload cray-libsci
module load boost
module load cray-hdf5-parallel
module load cmake/3.11.4
module load gcc   # Make C++ 14 standard library available to the Intel compiler
mkdir build_cori_hsw
cd build_cori_hsw
cmake ..
make -j 16
ls -l bin/qmcpack
```

When the preceding was tested on October 30, 2018, the following module and software versions were present:

```
build_cori_hsw> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                                    14)
    alps/6.6.43-6.0.7.0_26.4__ga796da3.ari
  2) nsg/1.2.0                                           15)
    rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari
```

```
  3) intel/18.0.1.163                                    16) atp/2.1.1
  4) craype-network-aries                                17) PrgEnv-intel/6.0.4
  5) craype/2.5.14                                       18) craype-haswell
  6) udreg/2.3.2-6.0.7.0_33.18__g5196236.ari             19) cray-mpich/7.7.0
  7) ugni/6.0.14.0-6.0.7.0_23.1__gea11d3d.ari            20) altd/2.0
  8) pmi/5.0.13                                          21) darshan/3.1.4
  9) dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari              22) boost/1.63
 10) gni-headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari     23) cray-hdf5-parallel/1.10.1.1
 11) xpmem/2.2.15-6.0.7.1_5.8__g7549d06.ari              24) cmake/3.11.4
 12) job/2.2.3-6.0.7.0_44.1__g6c4e934.ari                25) gcc/7.3.0
 13) dvs/2.7_2.2.113-6.0.7.1_7.1__g1bbc03e
```

3.7.13 Installing on NERSC Cori, Xeon Phi KNL partition, Cray XC40

The second phase of NERSC's Cori uses Intel Xeon Phi Knight's Landing (KNL) nodes. The following build recipe ensures that the code generation is appropriate for the KNL nodes:

```
export CRAYPE_LINK_TYPE=dynamic
module swap craype-haswell craype-mic-knl
module unload cray-libsci
module load boost
module load cray-hdf5-parallel
module load cmake/3.11.4
module load gcc   # Make C++ 14 standard library available to the Intel compiler
mkdir build_cori_knl
cd build_cori_knl
cmake ..
make -j 16
ls -l bin/qmcpack
```

When the preceding was tested on October 30, 2018, the following module and software versions were present:

```
build_cori_knl> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                                    14)
    alps/6.6.43-6.0.7.0_26.4__ga796da3.ari
  2) nsg/1.2.0                                           15)
    rca/2.2.18-6.0.7.0_33.3__g2aa4f39.ari
  3) intel/18.0.1.163                                    16) atp/2.1.1
  4) craype-network-aries                                17) PrgEnv-intel/6.0.4
  5) craype/2.5.14                                       18) craype-mic-knl
  6) udreg/2.3.2-6.0.7.0_33.18__g5196236.ari             19) cray-mpich/7.7.0
  7) ugni/6.0.14.0-6.0.7.0_23.1__gea11d3d.ari            20) altd/2.0
  8) pmi/5.0.13                                          21) darshan/3.1.4
  9) dmapp/7.1.1-6.0.7.0_34.3__g5a674e0.ari              22) boost/1.63
 10) gni-headers/5.0.12.0-6.0.7.0_24.1__g3b1768f.ari     23) cray-hdf5-parallel/1.10.1.1
 11) xpmem/2.2.15-6.0.7.1_5.8__g7549d06.ari              24) cmake/3.11.4
 12) job/2.2.3-6.0.7.0_44.1__g6c4e934.ari                25) gcc/7.3.0
 13) dvs/2.7_2.2.113-6.0.7.1_7.1__g1bbc03e
```

3.7.14 Installing on systems with ARMv8-based processors

The following build recipe was verified using the 'Arm Compiler for HPC' on the ANL JLSE Comanche system with Cavium ThunderX2 processors on November 6, 2018.

```
# load armclang compiler
module load Generic-AArch64/RHEL/7/arm-hpc-compiler/18.4
# load Arm performance libraries
module load ThunderX2CN99/RHEL/7/arm-hpc-compiler-18.4/armpl/18.4.0
# define path to pre-installed packages
export HDF5_ROOT=</path/to/hdf5/install/>
export BOOST_ROOT=</path/to/boost/install> # header-only, no need to build
```

Then using the following command:

```
mkdir build_armclang
cd build_armclang
cmake -DCMAKE_C_COMPILER=armclang -DCMAKE_CXX_COMPILER=armclang++ -DQMC_MPI=0 \
      -DLAPACK_LIBRARIES="-L$ARMPL_DIR/lib -larmpl_mp" \
      -DFFTW_INCLUDE_DIR="$ARMPL_DIR/include" \
      -DFFTW_LIBRARIES="$ARMPL_DIR/lib/libarmpl_mp.a" \
      ..
make -j 56
```

Note that armclang is recognized as an 'unknown' compiler by CMake v3.13* and below. In this case, we need to force it as clang to apply necessary flags. To do so, pass the following additionals option to CMake:

```
      -DCMAKE_C_COMPILER_ID=Clang -DCMAKE_CXX_COMPILER_ID=Clang \
      -DCMAKE_CXX_COMPILER_VERSION=5.0 -DCMAKE_CXX_STANDARD_COMPUTED_DEFAULT=98 \
```

3.7.15 Installing on Windows

Install the Windows Subsystem for Linux and Bash on Windows. Open a bash shell and follow the install directions for Ubuntu in Section 3.7.1.

3.8 Installing via Spack

Spack is a package manager for scientific software. One of the primary goals of Spack is to reduce the barrier for the users to install scientific software. Spack is intended to work on everything from laptop computers to high-end supercomputers. More information about Spack can be found at https://spack.readthedocs.io/en/latest. The major advantage of installation with Spack is that all dependencies are automatically built, potentially including all the compilers and libraries, and different versions of QMCPACK can easily coexist with each other. The QMCPACK Spack package also knows how to automatically build and patch QE. In principle, QMCPACK can be installed with a single Spack command.

3.8.1 Known Limitations

The QMCPACK Spack package inherits the limitations of the underlying Spack infrastructure and its dependencies. The main limitation is that installation typically fails when building a dependency such as HDF5, MPICH, etc. For `spack install qmcpack` to succeed, it is very important to leverge preinstalled packages on your computer or supercomputer. The other frequently encountered challenge is that the compiler configuration is nonintuitive. This is especially the case with the Intel compiler.

Here are some additional limitations of the QMCPACK Spack package that will be resolved in future releases:

- Because of a conflict in the QE 6.3 package, the QE variant cannot build in serial. In other words, `qmcpack install qmcpack~mpi` will issue a conflict. This will not be a problem with the next release of QE.
- Robust CUDA support in Spack is a work-in-progress. It will catch only some compiler-CUDA conflicts.
- CUDA support is not leveraging the Spack CUDA infrastructure; thus, compiler conflicts will not be caught until a build is attempted.
- AFQMC is not part of the Spack package at this time.
- The Spack package will install Nexus as part of the installation, but actual use of Nexus from within the Spack environment is untested.

3.8.2 Setting up the Spack Environment

Begin by cloning Spack from GitHub and configuring your shell as described at https://spack.readthedocs.io/en/latest/getting_started.html.

The goal of the next several steps is to set up the Spack environment for building. First, we highly recommend limiting the number of build jobs to a reasonable value for your machine. This can be accomplished by modifying your `~/.spack/config.yaml` file as follows:

```
config:
  build_jobs: 16
```

Make sure any existing compilers are properly detected. For many architectures, compilers are properly detected with no additional effort.

```
your-laptop> spack compilers
==> Available compilers
-- gcc sierra-x86_64 -------------------------------------------
gcc@7.2.0  gcc@6.4.0  gcc@5.5.0  gcc@4.9.4  gcc@4.8.5  gcc@4.7.4  gcc@4.6.4
```

However, if your compiler is not automatically detected, it is straightforward to add one:

```
your-laptop> spack compiler add <path-to-compiler>
```

The Intel compiler that requires checkout a license is particular tricky. First go ahead and add the compiler by modifying `~/.spack/linux/compilers.yaml`. Here is an example of a typical configuration:

```
- compiler:
    environment:
      set:
        INTEL_LICENSE_FILE: server@national-lab.doe.gov
    extra_rpaths:
    ['/soft/com/packages/intel/18/u3/compilers_and_libraries_2018.3.222/linux/compiler/lib/intel64',
    '/soft/apps/packages/gcc/gcc-6.2.0/lib64']
    flags:
      cflags: -gcc-name=/soft/apps/packages/gcc/gcc-6.2.0/bin/gcc
      fflags: -gcc-name=/soft/apps/packages/gcc/gcc-6.2.0/bin/gcc
      cxxflags: -gxx-name=/soft/apps/packages/gcc/gcc-6.2.0/bin/g++
    modules: []
    operating_system: ubuntu14.04
    paths:
```

```
      cc:
    /soft/com/packages/intel/18/u3/compilers_and_libraries_2018.3.222/linux/bin/intel64/icc
      cxx:
    /soft/com/packages/intel/18/u3/compilers_and_libraries_2018.3.222/linux/bin/intel64/icpc
      f77:
    /soft/com/packages/intel/18/u3/compilers_and_libraries_2018.3.222/linux/bin/intel64/ifort
      fc:
    /soft/com/packages/intel/18/u3/compilers_and_libraries_2018.3.222/linux/bin/intel64/ifort
    spec: intel@18.0.3
    target: x86_64
```

This last step is the most troublesome. Pre-installed packages are not automatically detected. If vendor optimized libraries are already installed, you will need to manually add them to your `~/.spack/packages.yaml`. For example, this works on Mac OS X for the Intel MKL package.

```
your-laptop> cat \~/.spack/packages.yaml
packages:
    intel-mkl:
        paths:
            intel-mkl@2018.0.128:
    /opt/intel/compilers_and_libraries_2018.0.104/mac/mkl
        buildable: False
```

Some trial-and-error might be involved to get the directory correct. If you do not include enough of the tree path, Spack will not be able to register the package in its database. More information about system packages can be found at
http://spack.readthedocs.io/en/latest/getting_started.html#system-packages

3.8.3 Building QMCPACK

The QMCPACK Spack package has a number of variants to support different compile time options and different versions of the application. A full list can be displayed by typing:

```
your laptop> spack info qmcpack
Description:
    QMCPACK, is a modern high-performance open-source Quantum Monte Carlo
    (QMC) simulation code.

Homepage: http://www.qmcpack.org/

Tags:
    None

Preferred version:
    3.6.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.6.0

Safe versions:
    develop  [git] https://github.com/QMCPACK/qmcpack.git
    3.6.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.6.0
    3.5.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.5.0
    3.4.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.4.0
    3.3.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.3.0
    3.2.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.2.0
    3.1.1      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.1.1
    3.1.0      [git] https://github.com/QMCPACK/qmcpack.git at tag v3.1.0
```

```
Variants:
    Name [Default]                 Allowed values          Description


    build_type [RelWithDebInfo]    Debug, Release,         CMake build type
                                   RelWithDebInfo,
                                   MinSizeRel
    complex [off]                  True, False             Build the complex (general
                                                           twist/k-point) version
    cuda [off]                     True, False             Enable CUDA and GPU
                                                           acceleration
    da [off]                       True, False             Install with support for
    basic
                                                           data analysis tools
    debug [off]                    True, False             Build debug version
    gui [off]                      True, False             Install with Matplotlib
    (long
                                                           installation time)
    mixed [off]                    True, False             Build the mixed precision
                                                           (mixture of single and
    double
                                                           precision) version for gpu
    and
                                                           cpu
    mpi [on]                       True, False             Build with MPI support
    phdf5 [on]                     True, False             Build with parallel
    collective
                                                           I/O
    qe [on]                        True, False             Install with patched
    Quantum
                                                           Espresso 6.3.0
    soa [off]                      True, False             Build with
    Structure-of-Array
                                                           instead of
    Array-of-Structure
                                                           code. Only for CPU codeand
                                                           only in mixed precision
    timers [off]                   True, False             Build with support for
    timers

Installation Phases:
    cmake    build    install

Build Dependencies:
    blas  boost  cmake  cuda  espresso  fftw-api  hdf5  lapack  libxml2  mpi

Link Dependencies:
    blas  boost  cuda  espresso  fftw  hdf5  lapack  libxml2  mpi

Run Dependencies:
    py-matplotlib  py-numpy

Virtual Packages:
    None
```

For example, to install the complex version of QMCPACK in mixed-precision use:

```
your-laptop> spack install qmcpack+mixed+complex%gcc@7.2.0 ^intel-mkl
```

where

```
%gcc@7.2.0
```

specifies the compiler version to be used and

```
^intel-mkl
```

specifies that the Intel MKL should be used as the BLAS and LAPACK provider.

It is also possible to run the QMCPACK regression tests as part of the installation process, for example:

```
your-laptop> spack install --test=root qmcpack+mixed+complex%gcc@7.2.0 ^intel-mkl
```

will run the unit and short tests. The current behavior of the QMCPACK Spack package is to complete the install as long as all the unit tests pass. If the short tests fail, a warning is issued at the command prompt.

For CUDA, you will need to specify and extra `cuda_arch` parameter otherwise, it will default to `cuda_arch=61`.

```
your-laptop> spack install qmcpack+cuda%intel@18.0.3 cuda_arch=61 ^intel-mkl
```

Due to limitations in the Spack CUDA package, if your compiler and CUDA combination give you a conflict you will need to specify a specific verison of CUDA that is compatible with your compiler on the command line. For example,

```
your-laptop> spack install qmcpack+cuda%intel@18.0.3 cuda_arch=61 ^cuda@10.0.130
    ^intel-mkl
```

3.8.4 Loading QMCPACK into your environment

Spack does not set up an environment to automatically find its packages. A few additional steps are needed. First, install the modules package by executing:

```
your-laptop> spack install environment-modules
```

Then modify your `~/.cshrc` file with something like

```
source ${HOME}/spack/opt/spack/darwin-sierra-x86_64/\\

    clang-7.3.0-apple/environment-modules-3.2.10-kenvhysdws4zw26tvbrjl37fdyn7inhb/\
          Modules/init/csh
```

You should then be able to load the qmcpack binary into your path by invoking

```
your-laptop> spack load qmcpack+mixed+complex%gcc@7.2.0
```

or more generally

```
your-laptop> spack load qmcpack+<spec>
```

3.8.5 Installing QMCPACK with Spack on Linux

Spack works robustly on the standard flavors of Linux (Ubuntu, CentOS, Ubuntu, etc.) using GCC or Intel compilers.

3.8.6 Installing QMCPACK with Spack on Mac OS X

Spack works on Mac OS X but requires installation of a few packages using Homebrew. You will need to install at minimum the GCC compilers, CMake, and pkg-config. The Intel compiler for Mac on OS X is not well supported by Spack packages and will most likely lead to a compile time failure in one of QMCPACK's dependencies.

3.8.7 Installing QMCPACK with Spack on IBM Blue Gene

This is untested at this time. In principle, it should work as long as each package found in

```
Blue Gene prompt> spack spec qmcpack
```

can be compiled for the compute nodes.

3.8.8 Installing QMCPACK with Spack on Cray Supercomputers

There are a number of issues in the Cray module environment. Spack contributors are working to fix these problems. We will update this section once we have a recipe that works reliably.

3.8.9 Reporting Bugs

Bugs with the QMCPACK Spack package should be filed at the main GitHub Spack repo https://github.com/spack/spack/issues

In the GitHub issue, include `@naromero77` to get the attention of our developer.

3.9 Testing and validation of QMCPACK

We **strongly encourage** running the included tests each time QMCPACK is built. A range of unit and integration tests ensure that the code behaves as expected and that results are consistent with known-good mean-field, quantum chemical, and historical QMC results.

The tests include the following:

- Unit tests: to check fundamental behavior. These should always pass.
- Stochastic integration tests: to check computed results from the Monte Carlo methods. These might fail statistically, but rarely because of the use of three sigma level statistics. These tests are further split into "short" tests, which have just sufficient length to have valid statistics, and "long" tests, to check behavior to higher statistical accuracy.
- Converter tests: to check conversion of trial wavefunctions from codes such as QE and GAMESS to QMCPACK's formats. These should always pass.
- Workflow tests: in the case of QE, we test the entire cycle of DFT calculation, trial wavefunction conversion, and a subsequent VMC run.
- Performance: to help performance monitoring. Only the timing of these runs is relevant.

The test types are differentiated by prefixes in their names, for example, `short-LiH_dimer_ae_vmc_hf_noj_16-1` indicates a short VMC test for the LiH dime.

QMCPACK also includes tests for developmental features and features that are unsupported on certain platforms. To indicate these, tests that are unstable are labeled with the CTest label "unstable." For example, they are unreliable, unsupported, or known to fail from partial implementation or bugs.

When installing QMCPACK you should run at least the unit tests:

```
ctest -R unit
```

These tests take only a few seconds to run. All should pass. A failure here could indicate a major problem with the installation.

A wider range of deterministic integration tests are being developed. The goal is to test much more of QMCPACK than the unit tests do and to do so in a manner that is reproducible across platforms. All of these should eventually pass 100% reliably and quickly. At present, some fail on some platforms and for certain build types.

```
ctest -R deterministic -LE unstable
```

If time allows, the "short" stochastic tests should also be run. The short tests take a few minutes each on a 16-core machine—about 1 hour total depending on the platform. You can run these tests using the following command in the build directory:

```
ctest -R short -LE unstable  # Run the tests with "short" in their name.
                             # Exclude any known unstable tests.
```

The output should be similar to the following:

```
Test project build_gcc
      Start  1: short-LiH_dimer_ae-vmc_hf_noj-16-1
 1/44 Test  #1: short-LiH_dimer_ae-vmc_hf_noj-16-1 ..............  Passed   11.20 sec
      Start  2: short-LiH_dimer_ae-vmc_hf_noj-16-1-kinetic
 2/44 Test  #2: short-LiH_dimer_ae-vmc_hf_noj-16-1-kinetic ......  Passed    0.13 sec
..
42/44 Test #42: short-monoO_1x1x1_pp-vmc_sdj-1-16 ...............  Passed   10.02 sec
      Start 43: short-monoO_1x1x1_pp-vmc_sdj-1-16-totenergy
43/44 Test #43: short-monoO_1x1x1_pp-vmc_sdj-1-16-totenergy .....  Passed    0.08 sec
      Start 44: short-monoO_1x1x1_pp-vmc_sdj-1-16-samples
44/44 Test #44: short-monoO_1x1x1_pp-vmc_sdj-1-16-samples .......  Passed    0.08 sec

100% tests passed, 0 tests failed out of 44

Total Test time (real) = 167.14 sec
```

Note that the number of tests run varies between the standard, complex, and GPU compilations. These tests should pass with three sigma reliability. That is, they should nearly always pass, and when rerunning a failed test it should usually pass. Overly frequent failures suggest a problem that should be addressed before any scientific production.

The full set of tests consist of significantly longer versions of the short tests, as well as tests of the conversion utilities. The runs require several hours each for improved statistics and a much more stringent test of the code. To run all the tests, simply run CTest in the build directory:

```
ctest -LE unstable           # Run all the stable tests. This will take several hours.
```

You can also run verbose tests, which direct the QMCPACK output to the standard output:

```
ctest -V -R short   # Verbose short tests
```

The test system includes specific tests for the complex version of the code.

The input data files for the tests are located in the `tests` directory. The system-level test directories are grouped into `heg`, `molecules`, and `solids`, with particular physical systems under each (for example `molecules/H4_ae` [1]) Under each physical system directory there might be tests for multiple QMC methods or parameter variations. The numerical comparisons and test definitions are in the `CMakeLists.txt` file in each physical system directory.

If *all* the QMC tests fail it is likely that the appropriate mpiexec (or mpirun, aprun, srun, jsrun) is not being called or found. If the QMC runs appear to work but all the other tests fail, it is possible that Python is not working on your system. We suggest checking some of the test console output in `build/Testing/Temporary/LastTest.log` or the output files under `build/tests/`.

Note that because most of the tests are very small, consisting of only a few electrons, the performance is not representative of larger calculations. For example, although the calculations might fit in cache, there will be essentially no vectorization because of the small electron counts. **These tests should therefore not be used for any benchmarking or performance analysis**. Example runs that can be used for testing performance are described in Section 3.9.3.

3.9.1 Deterministic and unit tests

QMCPACK has a set of deterministic tests, predominantly unit tests. All of these tests can be run with the following command (in the build directory):

```
ctest -R deterministic -LE unstable
```

These tests should always pass. Failure could indicate a major problem with the compiler, compiler settings, or a linked library that would give incorrect results.

The output should look similar to the following:

```
Test project qmcpack/build
      Start  1: unit_test_numerics
 1/11 Test  #1: unit_test_numerics ..............   Passed    0.06 sec
      Start  2: unit_test_utilities
 2/11 Test  #2: unit_test_utilities ..............   Passed    0.02 sec
      Start  3: unit_test_einspline
 ...
10/11 Test #10: unit_test_hamiltonian ............   Passed    1.88 sec
      Start 11: unit_test_drivers
11/11 Test #11: unit_test_drivers ................   Passed    0.01 sec

100% tests passed, 0 tests failed out of 11

Label Time Summary:
unit    =   2.20 sec

Total Test time (real) =   2.31 sec
```

Individual unit test executables can be found in `build/tests/bin`. The source for the unit tests is located in the `tests` directory under each directory in `src` (e.g. `src/QMCWavefunctions/tests`).

See Chapter 25 for more details about unit tests.

3.9.2 Integration tests with Quantum Espresso

As described in Section 3.12, it is possible to test entire workflows of trial wavefunction generation, conversion, and eventual QMC calculation. A patched QE must be installed so that the

[1] The suffix "ae" is short for "all-electron," and "pp" is short for "pseudopotential."

pw2qmcpack converter is available.

By adding `-D QE_BIN=your_QE_binary_path` in the CMake command line when building your QMCPACK, tests named with the "qe-" prefix will be included in the test set of your build. You can test the whole pw → pw2qmcpack → qmcpack workflow by

```
ctest -R qe
```

This provides a very solid test of the entire QMC toolchain for plane wave–generated wavefunctions.

3.9.3 Performance tests

Performance tests representative of real research runs are included in the tests/performance directory. They can be used for benchmarking, comparing machine performance, or assessing optimizations. This is in contrast to the majority of the conventional integration tests in which the particle counts are too small to be representative. Care is still needed to remove initialization, I/O, and compute a representative performance measure.

The CTest integration is sufficient to run the benchmarks and measure relative performance from version to version of QMCPACK and to assess proposed code changes. Performance tests are prefixed with "performance." To obtain the highest performance on a particular platform, you must run the benchmarks in a standalone manner and tune thread counts, placement, walker count (etc.). This is essential to fairly compare different machines. Check with the developers if you are unsure of what is a fair change.

For the largest problem sizes, the initialization of spline orbitals might take a large portion of overall runtime. When QMCPACK is run at scale, the initialization is fast because it is fully parallelized. However, the performance runs are most usually run on a single node. Consider running QMCPACK once with `--save_wfs=1` to save the converted spline coefficients to the disk and load them for later runs in the same folder.

The delayed update algorithm in Section 8.2 significantly changes the performance characteristics of QMCPACK. A parameter scan of the maximal number of delays specific to every architecture and problem size is required to achieve the best performance.

NiO performance tests

Follow the instructions in tests/performance/NiO/README to enable and run the NiO tests.

The NiO tests are for bulk supercells of varying size. The QMC runs consist of short blocks of (1) VMC without drift (2) VMC with drift term included, and (3) DMC with constant population. The tests use spline wavefunctions that must be downloaded as described in the README file because of their large size. You will need to set "`-DQMC_DATA=YOUR_DATA_FOLDER -DENABLE_TIMERS=1`" when running CMake as described in the README file.

Two sets of wavefunction are tested: spline orbitals with one- and two-body Jastrow functions and a more complex form with an additional three-body Jastrow function. The Jastrows are the same for each run and are not reoptimized, as might be done for research purposes. Runs in the hundreds of electrons up to low thousands of electrons are representative of research runs performed in 2017. The largest runs target future machines and require very large memory.

3.9.4 Troubleshooting tests

CTest reports briefly pass or fail of tests in printout and also collects all the standard outputs to help investigating how tests fail. If the CTest execution is completed, look at `Testing/Temporary/LastTest.log`. If you manually stop the testing (ctrl+c), look at

Table 3.1: System sizes and names for NiO performance tests. GPU performance tests are named similarly but have different walker counts.

Performance test name	Historical name	Atoms	Electrons	Electrons/spin
performance-NiO-cpu-a32-e384	S8	32	384	192
performance-NiO-cpu-a64-e768	S16	64	768	384
performance-NiO-cpu-a128-e1536	S32	128	1536	768
performance-NiO-cpu-a256-e3072	S64	256	3072	1536
performance-NiO-cpu-a512-e6144	S128	512	6144	3072
performance-NiO-cpu-a1024-e12288	S256	1024	12288	6144

`Testing/Temporary/LastTest.log.tmp`. You can locate the failing tests by searching for the key word "Fail."

3.9.5 Slow testing with OpenMPI

OpenMPI has a default binding policy that makes all the threads run on a single core during testing when there are two or fewer MPI ranks. This significantly increases testing time. If you are authorized to change the default setting, you can just add "hwloc_base_binding_policy=none" in /etc/openmpi/openmpi-mca-params.conf.

3.10 Automated testing of QMCPACK

The QMCPACK developers run automatic tests of QMCPACK on several different computer systems, many on a continuous basis. See the reports at https://cdash.qmcpack.org/CDash/index.php?project=QMCPACK. We currently test the following combinations nightly (workstations) and weekly (supercomputers):

- On a Linux Intel Xeon workstation, combinations of:
  - GCC 8.2.0, Intel2019, Clang6, Clang7, PGI2018
  - No MPI, Intel MPI, and OpenMPI
  - CPU and GPU builds using CUDA 10.0
- On a Linux Intel Knight's Landing workstation:
  - Intel 2017 with Intel MPI and MKL
  - GCC 6.3.1 with Intel MPI and MKL
- On Eos, a Cray XC30 Intel machine:
  - The default Intel programming environment and compiler with Cray MPI and Intel MKL
- On Titan, a Cray XK7 CPU+GPU machine:
  - The GCC programming environment and compiler with Cray MPI and CUDA
  - The GCC programming environment and compiler with Cray MPI
- On Cetus, an IBM Blue Gene Q machine:

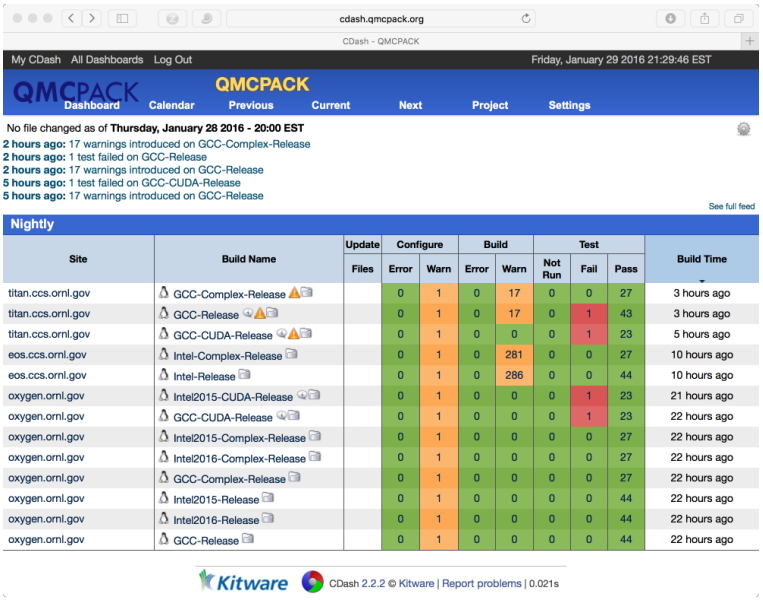


Figure 3.1: Example test results for QMCPACK showing data for a workstation (Intel, GCC, both CPU and GPU builds) and for two ORNL supercomputers. In this example, four errors were found. This dashboard is accessible at https://cdash.qmcpack.org.

– Blue Gene Clang

3.11 Building ppconvert, a pseudopotential format converter

QMCPACK includes a utility—ppconvert—to convert between different pseudopotential formats. Examples include effective core potential formats (in Gaussians), the UPF format used by QE, and the XML format used by QMCPACK itself. The utility also enables the atomic orbitals to be recomputed via a numerical density functional calculation if they need to be reconstructed for use in an electronic structure calculation.

The utility is a stand-alone C++ executable that is not built by default but that is accessible via adding `-DBUILD_PPCONVERT=1` to CMake and then typing `make ppconvert`. A user guide is provided in Section 22.3.3.

3.12 Installing and patching Quantum ESPRESSO

For trial wavefunctions obtained in a plane-wave basis, we mainly support QE. Note that ABINIT and QBox were supported historically and could be reactivated.

QE stores wavefunctions in a nonstandard internal "save" format. To convert these to a conventional HDF5 format file we have developed a converter—pw2qmcpack—which is an add-on to the QE distribution.

To simplify the process of patching QE we have developed a script that will automatically download and patch the source code. The patches are specific to each version. For example, to download and patch QE v6.3:

```
cd external_codes/quantum_espresso
./download_and_patch_qe6.3.sh
```

After running the patch, you must configure QE with the HDF5 capability enabled in either way:

- If your system already has HDF5 installed with Fortran, use the --with-hdf5 configuration option.

```
cd qe-6.3
./configure --with-hdf5=/opt/local   # Specify HDF5 base directory
```

  Check the end of the configure output if HDF5 libraries are found properly. If not, either install a complete library or use the other scheme. If using a parallel HDF5 library, be sure to use the same MPI with QE as used to build the parallel HDF5 library.

  Currently, HDF5 support in QE itself is preliminary. To enable use of pw2qmcpack but use the old non-HDF5 I/O within QE, replace `-D__HDF5` with `-D__HDF5_C` in make.inc.

- If your system has HDF5 with C only, manually edit make.inc by adding `-D__HDF5_C` and `-DH5_USE_16_API` in `DFLAGS` and provide include and library path in `IFLAGS` and `HDF5_LIB`.

The complete process is described in external_codes/quantum_espresso/README.

The tests involving pw.x and pw2qmcpack.x have been integrated into the test suite of QMCPACK. By adding `-D QE_BIN=your_QE_binary_path` in the CMake command line when building your QMCPACK, tests named with the "qe-" prefix will be included in the test set of your build. You can test the whole pw → pw2qmcpack → qmcpack workflow by

```
ctest -R qe
```

See Section 3.9.2 and the testing section for more details.

3.13 How to build the fastest executable version of QMCPACK

To build the fastest version of QMCPACK we recommend the following:

- Use the latest C++ compilers available for your system. Substantial gains have been made optimizing C++ in recent years.
- Use a vendor-optimized BLAS library such as Intel MKL and AMD ACML. Although QMC does not make extensive use of linear algebra, it is used in the VMC wavefunction optimizer to apply the orbital coefficients in local basis calculations and in the Slater determinant update.
- Use a vector math library such as Intel VML. For periodic calculations, the calculation of the structure factor and Ewald potential benefit from vectorized evaluation of sin and cos. Currently we only autodetect Intel VML, as provided with MKL, but support for MASSV and AMD LibM is included via #defines. See, for example, src/Numerics/e2iphi.h. For large supercells, this optimization can gain 10% in performance.

Note that greater speedups of QMC calculations can usually be obtained by carefully choosing the required statistics for each investigation. That is, do not compute smaller error bars than necessary.

3.14 Troubleshooting the installation

Some tips to help troubleshoot installations of QMCPACK:

- First, build QMCPACK on a workstation you control or on any system with a simple and up-to-date set of development tools. You can compare the results of CMake and QMCPACK on this system with any more difficult systems you encounter.
- Use up-to-date development software, particularly a recent CMake.
- Verify that the compilers and libraries you expect are being configured. It is common to have multiple versions installed. The configure system will stop at the first version it finds, which might not be the most recent. If this occurs, directly specify the appropriate directories and files (Section 3.6.3). For example,

```
cmake -DCMAKE_C_COMPILER=/full/path/to/mpicc
  -DCMAKE_CXX_COMPILER=/full/path/to/mpicxx ..
```

- To monitor the compiler and linker settings, use a verbose build, `make VERBOSE=1`. If an individual source file fails to compile you can experiment by hand using the output of the verbose build to reconstruct the full compilation line.

If you still have problems please post to the QMCPACK Google group with full details, or contact a developer.

Chapter 4

Running QMCPACK

QMCPACK requires at least one xml input file, and is invoked via:

```
qmcpack [command line options] <XML input file(s)>
```

4.1 Command line options

QMCPACK offers several command line options that affect how calculations are performed. If the flag is absent, then the corresponding option is disabled.

--dryrun Validate the input file without performing the simulation. This is a good way to ensure that QMCPACK will do what you think it will.

--enable-timers=none|coarse|medium|fine Control the timer granularity when the build option `ENABLE_TIMERS` is enabled.

--help Print version information as well as a list of optional command-line arguments.

--noprint Do not print extra information on Jastrow or pseudopotential. If this flag is not present, QMCPACK will create several `.dat` files that contain information about pseudopotentials (one file per PP) and Jastrow factors (one per Jastrow factor). These file might be useful for visual inspection of the Jastrow, for example.

--save_wfs deprecated Replaced by `save_coefs` XML input tag. Write a `.h5` file containing the real-space B-spline coefficients of the single-particle wavefunctions. See Section 8.3.1 for more information.

--vacuum X Removed, use the "vacuum" input tag described in Section 7.1.

--verbosity=low|high|debug Control the output verbosity. The default low verbosity is concise and, for example, does not include all electron or atomic positions for large systems to reduce output size. Use "high" to see this information and more details of initialization, allocations, QMC method settings, etc.

--version Print version information and optional arguments. Same as **--help**.

4.2 Input files

The input is one or more XML file(s), documented in Chapter 6.

4.3 Output files

QMCPACK generates multiple files, documented in Chapter 11.

4.4 Running in parallel with MPI

QMCPACK is fully parallelized with MPI. When performing an ensemble job, all the MPI ranks are first equally divided into groups that perform individual QMC calculations. Within one calculation, all the walkers are fully distributed across all the MPI ranks in the group. Since MPI requires distributed memory, there must be at least one MPI per node. To maximize the efficiency, more facts should be taken into account. When using MPI+threads on compute nodes with more than one NUMA domain (e.g., AMD Interlagos CPU on Titan or a node with multiple CPU sockets), it is recommended to place as many MPI ranks as the number of NUMA domains if the memory is sufficient (e.g., one MPI task per socket). On clusters with more than one GPU per node (NVIDIA Tesla K80), it is necessary to use the same number of MPI ranks as the number of GPUs per node to let each MPI rank take one GPU.

4.5 Using OpenMP threads

Modern processors integrate multiple identical cores even with hardware threads on a single die to increase the total performance and maintain a reasonable power draw. QMCPACK takes advantage of this compute capability by using threads and the OpenMP programming model as well as threaded linear algebra libraries. By default, QMCPACK is always built with OpenMP enabled. When launching calculations, users should instruct QMCPACK to create the right number of threads per MPI rank by specifying environment variable OMP_NUM_THREADS. Assuming one MPI rank per socket, the number of threads should typically be the number of cores on that socket. Even in the GPU-accelerated version, using threads significantly reduces the time spent on the calculations performed by the CPU.

4.5.1 Nested OpenMP threads

Nested threading is an advanced feature requiring experienced users to finely tune runtime parameters to reach the best performance.

For small-to-medium problem sizes, using one thread per walker or for multiple walkers is most efficient. This is the default in QMCPACK and achieves the shortest time to solution.

For large problems of at least 1,000 electrons, use of nested OpenMP threading can be enabled to reduce the time to solution further, although at some loss of efficiency. In this scheme multiple threads are used in the computations of each walker. This capability is implemented for some of the key computational kernels: the 3D spline orbital evaluation, certain portions of the distance tables, and implicitly the BLAS calls in the determinant update. Use of the batched nonlocal pseudopotential evaluation is also recommended.

Nested threading is enabled by setting `OMP_NUM_THREADS=AA,BB`, `OMP_MAX_ACTIVE_LEVELS=2` and `OMP_NESTED=TRUE` where the additional `BB` is the number of second-level threads. Choosing the thread affinity is critical to the performance. QMCPACK provides a tool qmc-check-affinity (source file src/QMCTools/check-affinity.cpp for details), which might help users investigate the affinity. Knowledge of how the operating system logical CPU cores (/prco/cpuinfo) are bound to the hardware is also needed.

For example, on Blue Gene/Q with a Clang compiler, the best way to fully use the 16 cores each with 4 hardware threads is

```
OMP_NESTED=TRUE
OMP_NUM_THREADS=16,4
MAX_ACTIVE_LEVELS=2
OMP_PLACES=threads
OMP_PROC_BIND=spread,close
```

On Intel Xeon Phi KNL with an Intel compiler, to use 64 cores without using hardware threads:

```
OMP_NESTED=TRUE
OMP_WAIT_POLICY=ACTIVE
OMP_NUM_THREADS=16,4
MAX_ACTIVE_LEVELS=2
OMP_PLACES=cores
OMP_PROC_BIND=spread,close
KMP_HOT_TEAMS_MODE=1
KMP_HOT_TEAMS_MAX_LEVEL=2
```

Most multithreaded BLAS/LAPACK libraries do not spawn threads by default when being called from an OpenMP parallel region. See the explanation in Section 3.6.9. This results in the use of only a single thread in each second-level thread team for BLAS/LAPACK operations. Some vendor libraries like MKL support using multiple threads when being called from an OpenMP parallel region. One way to enable this feature is using environment variables to override the default behavior. However, this forces all the calls to the library to use the same number of threads. As a result, small function calls are penalized with heavy overhead and heavy function calls are slow for not being able to use more threads. Instead, QMCPACK uses the library APIs to turn on nested threading only at selected performance critical calls. In the case of using a serial library, QMCPACK implements nested threading to distribute the workload wherever necessary. Users do not need to control the threading behavior of the library.

4.5.2 Performance considerations

As walkers are the basic units of workload in QMC algorithms, they are loosely coupled and distributed across all the threads. For this reason, the best strategy to run QMCPACK efficiently is to feed enough walkers to the available threads.

In a VMC calculation, the code automatically raises the actual number of walkers per MPI rank to the number of available threads if the user-specified number of walkers is smaller, see "walkers/mpi=XXX" in the VMC output. In a DMC calculation, the target number of walkers should be chosen to be slightly smaller than a multiple of the total number of available threads across all the MPI ranks belongs to this calculation. Since the number of walkers varies from generation to generation, its dynamical value should be slightly smaller or equal to that multiple most of the time.

To achieve better performance, a mixed-precision version (experimental) has been introduced to the CPU code. The mixed-precision CPU code is more aggressive than the GPU version in using single precision (SP) operations. Current implementation uses SP on most calculations, except for matrix inversions and reductions where double precision is required to retain high accuracy. All the constant spline data in wavefunction, pseudopotentials, and Coulomb potentials are initialized in double precision and later stored in single precision. The mixed-precision code is as accurate as the double-precision code up to a certain system size. Cross checking and verification of accuracy is always required but is particularly important above approximately 1,500 electrons. The mixed

precision code is currently tested on solids with real and complex wavefunctions with VMC, VMC using drift, and DMC runs with wavefunction including single Slater determinant and one- and two-body Jastrow factors. Wavefunction optimization is currently not supported.

4.5.3 Memory considerations

When using threads, some memory objects are shared by all the threads. Usually these memory objects are read only when the walkers are evolving, for instance the ionic distance table and wavefunction coefficients. If a wavefunction is represented by B-splines, the whole table is shared by all the threads. It usually takes a large chunk of memory when a large primitive cell was used in the simulation. Its actual size is reported as "MEMORY increase XXX MB BsplineSetReader" in the output file. See details about how to reduce it in Section 8.3.1.

The other memory objects that are distinct for each walker during random walks need to be associated with individual walkers and cannot be shared. This part of memory grows linearly as the number of walkers per MPI rank. Those objects include wavefunction values (Slater determinants) at given electronic configurations and electron-related distance tables (electron-electron distance table). Those matrices dominate the N^2 scaling of the memory usage per walker.

4.6 Running on GPU machines

The GPU version for the NVIDIA CUDA platform is fully incorporated into the main source code. Commonly used functionalities for solid-state and molecular systems using B-spline single-particle orbitals are supported. Use of Gaussian basis sets, three-body Jastrow functions, and many observables are not yet supported. A detailed description of the GPU implementation can be found in Ref. [7].

The current GPU implementation assumes one MPI process per GPU. To use nodes with multiple GPUs, use multiple MPI processes per node. Vectorization is achieved over walkers, that is, all walkers are propagated in parallel. In each GPU kernel, loops over electrons, atomic cores, or orbitals are further vectorized to exploit an additional level of parallelism and to allow coalesced memory access.

4.6.1 Performance considerations

To run with high performance on GPUs it is crucial to perform some benchmarking runs: the optimum configuration is system size, walker count, and GPU model dependent. The GPU implementation vectorizes operations over multiple walkers, so generally the more walkers that are placed on a GPU, the higher the performance that will be obtained. Performance also increases with electron count, up until the memory on the GPU is exhausted. A good strategy is to perform a short series of VMC runs with walker count increasing in multiples of two. For systems with 100s of electrons, typically 128–256 walkers per GPU use a sufficient number of GPU threads to operate the GPU efficiently and to hide memory-access latency. For smaller systems, thousands of walkers might be required. For QMC algorithms where the number of walkers is fixed such as VMC, choosing a walker count the is a multiple of the number of streaming multiprocessors can be most efficient. For variable population DMC runs, this exact match is not possible.

To achieve better performance, the current GPU implementation uses single-precision operations for most of the calculations. Double precision is used in matrix inversions and the Coulomb interaction to retain high accuracy. The mixed-precision GPU code is as accurate as the double-precision CPU code up to a certain system size. Cross checking and verification of accuracy are

encouraged for systems with more than approximately 1,500 electrons. For typical calculations on smaller electron counts, the statistical error bars are much larger then the error introduced by mixed precision.

4.6.2 Memory considerations

In the GPU implementation, each walker has a buffer in the GPU's global memory to store temporary data associated with the wavefunctions. Therefore, the amount of memory available on a GPU limits the number of walkers and eventually the system size that it can process. Additionally, for calculations using B-splines, this data is stored on the GPU in a shared read-only buffer. Often the size of the B-spline data limits the calculations that can be run on the GPU.

If the GPU memory is exhausted, first try reducing the number of walkers per GPU. Coarsening the grids of the B-splines representation (by decreasing the value of the mesh factor in the input file) can also lower the memory usage, at the expense (risk) of obtaining inaccurate results. Proceed with caution if this option has to be considered. It is also possible to distribute the B-spline coefficients table between the host and GPU memory, see option Spline_Size_Limit_MB in Section 8.3.1.

Chapter 5

Units used in QMCPACK

Internally, QMCPACK uses atomic units throughout. Unless stated, all inputs and outputs are also in atomic units. For convenience the analysis tools offer conversions to eV, Ry, Angstrom, Bohr, etc.

Chapter 6

Input file overview

This chapter introduces XML as it is used in QMCPACK's input file. The focus is on the XML file format itself and the general structure of the input file rather than an exhaustive discussion of all keywords and structure elements.

QMCPACK uses XML to represent structured data in its input file. Instead of text blocks like

```
begin project
  id     = vmc
  series = 0
end project

begin vmc
  move     = pbyp
  blocks   = 200
  steps    =  10
  timestep = 0.4
end vmc
```

QMCPACK input looks like

```
   <project id="vmc" series="0">
   </project>

   <qmc method="vmc" move="pbyp">
      <parameter name="blocks"  >  200 </parameter>
      <parameter name="steps"   >   10 </parameter>
      <parameter name="timestep">  0.4 </parameter>
   </qmc>
```

XML elements start with `<element_name>`, end with `</element_name>`, and can be nested within each other to denote substructure (the trial wavefunction is composed of a Slater determinant and a Jastrow factor, which are each further composed of …). `id` and `series` are attributes of the `<project/>` element. XML attributes are generally used to represent simple values, like names, integers, or real values. Similar functionality is also commonly provided by `<parameter/>` elements like those shown above.

The overall structure of the input file reflects different aspects of the QMC simulation: the simulation cell, particles, trial wavefunction, Hamiltonian, and QMC run parameters. A condensed version of the actual input file is shown below:

```
<?xml version="1.0"?>
<simulation>
```

```
  <project id="vmc" series="0">
    ...
  </project>

  <qmcsystem>

    <simulationcell>
      ...
    </simulationcell>

    <particleset name="e">
      ...
    </particleset>

    <particleset name="ion0">
      ...
    </particleset>

    <wavefunction name="psi0" ... >
      ...
      <determinantset>
        <slaterdeterminant>
          ..
        </slaterdeterminant>
      </determinantset>
      <jastrow type="One-Body" ... >
        ...
      </jastrow>
      <jastrow type="Two-Body" ... >
        ...
      </jastrow>
    </wavefunction>

    <hamiltonian name="h0" ... >
      <pairpot type="coulomb" name="ElecElec" ... />
      <pairpot type="coulomb" name="IonIon"   ... />
      <pairpot type="pseudo" name="PseudoPot" ... >
        ...
      </pairpot>
    </hamiltonian>

   </qmcsystem>

   <qmc method="vmc" move="pbyp">
     <parameter name="warmupSteps">   20 </parameter>
     <parameter name="blocks"     >  200 </parameter>
     <parameter name="steps"      >   10 </parameter>
     <parameter name="timestep"   >  0.4 </parameter>
   </qmc>

</simulation>
```

The omitted portions (...) are more fine-grained inputs such as the axes of the simulation cell, the number of up and down electrons, positions of atomic species, external orbital files, starting Jastrow parameters, and external pseudopotential files.

6.1 Project

The <project> tag uses the id and series attributes. The value of id is the first part of the prefix for output file names.

Output file names also contain the series number, starting at the value given by the series tag. After every <qmc> section the series value will increment, giving each section a unique prefix.

For the input file shown previously, the output files will start with vmc.s000, for example vmc.s000.scalar.dat. If there were another <qmc> section in the input file, the corresponding output files would use the prefix vmc.s001.

6.2 Random number initialization

The random number generator state is initialized from the random element using the seed attribute.

```
<random seed="1000"/>
```

If the random element is not present, or the seed value is negative, the seed will be generated from the current time.

In order to initialize the many independent random number generators (one per thread and MPI process), the seed value is used (modulo 1024) as a starting index into a list of prime numbers. Entries in this offset list of prime numbers are then used as the seed for the random generator on each thread and process.

If checkpointing is enabled, the random number state is written to an HDF file at the end of each block (suffix: .random.h5). This file will be read if the mcwalkerset tag is present to perform a restart. For more information, see the checkpoint element in the QMC methods chapter (10) and the section on checkpoint and restart files in 11.6.

Chapter 7

Specifying the system to be simulated

7.1 Specifying the simulation cell

The `simulationcell` block specifies the geometry of the cell, how the boundary conditions should be handled, and how ewald summation should be broken up.

`simulationcell` element				
parent elements:	`qmcsystem`			
child elements:	None			
attribute :				
parameter name	**datatype**	**values**	**default**	**description**
`lattice`	9 floats	any float	Must be specified	Specification of lattice vectors.
`bconds`	string	"p" or "n"	"n n n"	Boundary conditions for each axis.
`vacuum`	float	≥ 1.0	1.0	Vacuum scale.
`LR_dim_cutoff`	float	float	15	Ewald breakup distance.

An example of a `simulationcell` block is given below:

```
<simulationcell>
  <parameter name="lattice">
    3.8       0.0       0.0
    0.0       3.8       0.0
    0.0       0.0       3.8
  </parameter>
  <parameter name="bconds">
     p p p
  </parameter>
  <parameter name="LR_dim_cutoff"> 20 </parameter>
</simulationcell>
```

Here, a cubic cell 3.8 bohr on a side will be used. This simulation will use periodic boundary conditions, and the maximum k vector will be $20/r_{wigner-seitz}$ of the cell.

7.1.1 Lattice

The cell is specified using 3 lattice vectors.

7.1.2 Boundary conditions

QMCPACK offers the capability to use a mixture of open and periodic boundary conditions. The `bconds` parameter expects a single string of three characters separated by spaces, *e.g.* "p p p" for purely periodic boundary conditions. These characters control the behavior of the x, y, and z, axes, respectively. Non periodic directions must be placed after the periodic ones. Examples of valid `bconds` include:

"p p p" Periodic boundary conditions. Corresponds to a 3D crystal.

"p p n" Slab geometry. Corresponds to a 2D crystal.

"p n n" Wire geometry. Corresponds to a 1D crystal.

"n n n" Open boundary conditions. Corresponds to an isolated molecule in a vacuum.

7.1.3 Vacuum

The vacuum option allows adding a vacuum region in slab or wire boundary conditions (`bconds= p p n` or `bconds= p n n`, respectively). The main use is to save memory with spline or plane-wave basis trial wavefunctions, because no basis functions are required inside the vacuum region. For example, a large vacuum region can be added above and below a graphene sheet without having to generate the trial wavefunction in such a large box or to have as many splines as would otherwise be required. Note that the trial wavefunction must still be generated in a large enough box to sufficiently reduce periodic interactions in the underlying electronic structure calculation.

With the vacuum option, the box used for Ewald summation increases along the axis labeled `n` by a factor of `vacuum`. Note that all the particles remain in the original box without altering their positions. i.e. Bond lengths are not changed by this option. The default value is 1, no change to the specified axes.

An example of a `simulationcell` block using `vacuum` is given below. The size of the box along the z-axis increases from 12 to 18 by the vacuum scale of 1.5.

```
<simulationcell>
  <parameter name="lattice">
    3.8       0.0       0.0
    0.0       3.8       0.0
    0.0       0.0      12.0
  </parameter>
  <parameter name="bconds">
     p p n
  </parameter>
  <parameter name="vacuum"> 1.5 </parameter>
  <parameter name="LR_dim_cutoff"> 20 </parameter>
</simulationcell>
```

7.1.4 LR_dim_cutoff

When using periodic boundary conditions direct calculation of the Coulomb energy is not well behaved. As a result, QMCPACK uses an optimized Ewald summation technique to compute the Coulomb interaction.[10]

In the Ewald summation, the energy is broken into short- and long-ranged terms. The short-ranged term is computed directly in real space, while the long-ranged term is computed in reciprocal

space. `LR_dim_cutoff` controls where the short-ranged term ends and the long-ranged term begins. The real-space cutoff, reciprocal-space cutoff, and `LR_dim_cutoff` are related via:

$$texttt{LR}_{dim}_{cutoff} = r_c \times k_c$$

where r_c is the Wigner-Seitz radius, and k_c is the length of the maximum k-vector used in the long-ranged term.

7.2 Specifying the particle set

The `particleset` blocks specify the particles in the QMC simulations: their types, attributes (mass, charge, valence), and positions.

7.2.1 Input specification

`particleset` element				
parent elements:	`simulation`			
child elements:	`group, attrib`			
attribute :				
name	**datatype**	**values**	**default**	**description**
`name`/`id`	text	*any*	e	Name of particle set
`size`^{o}	integer	*any*	0	Number of particles in set
`random`^{o}	text	yes/no	no	Randomize starting positions
`randomsrc`/ `random_source`^{o}	text	`particleset.name`	*none*	Particle set to randomize

`group` element				
parent elements:	`particleset`			
child elements:	`parameter, attrib`			
attribute :				
name	**datatype**	**values**	**default**	**description**
`name`	text	*any*	e	Name of particle set
`size`^{o}	integer	*any*	0	Number of particles in set
`mass`^{o}	real	*any*	1	Mass of particles in set
`unit`^{o}	text	au/amu	au	Units for mass of particles
parameters				
name	**datatype**	**values**	**default**	**description**
`charge`	real	*any*	0	Charge of particles in set
`valence`	real	*any*	0	Valence charge of particles in set
`atomicnumber`	integer	*any*	0	Atomic number of particles in set

attrib element				
parent elements:	particleset,group			
attribute :				
name	**datatype**	**values**	**default**	**description**
name	string	*any*	*none*	Name of attrib
datatype	string	intArray, realArray, posArray, stringArray	*none*	Type of data in attrib
size^o	string	*any*	*none*	Size of data in attrib

7.2.2 Detailed attribute description

particleset required attributes

- name/id
  Unique name for the particle set. Default is "e" for electrons. "i" or "ion0" is typically used for ions.

particleset optional attributes

- size
  Number of particles in set

- random
  Randomize starting positions of particles. Each component of each particle's position is randomized independently in the range of the simulation cell in that component's direction.

- randomsrc/random_source
  Specify source particle set around which to randomize the initial positions of this particle set.

name required attributes

- name/id
  Unique name for the particle set group. Typically, element symbols are used for ions and "u" or "d" for spin-up and spin-down electron groups, respectively.

group optional attributes

- mass
  Mass of particles in set.

- unit
  Units for mass of particles in set (au[$m_e = 1$] or amu[$\frac{1}{12} m_{^{12}\mathrm{C}} = 1$]).

7.2.3 Example use cases

Listing 7.1: particleset elements for ions and electrons randomizing electron start positions.

```
<particleset name="i" size="2">
  <group name="Li">
    <parameter name="charge">3.000000</parameter>
    <parameter name="valence">3.000000</parameter>
    <parameter name="atomicnumber">3.000000</parameter>
  </group>
  <group name="H">
    <parameter name="charge">1.000000</parameter>
    <parameter name="valence">1.000000</parameter>
    <parameter name="atomicnumber">1.000000</parameter>
  </group>
  <attrib name="position" datatype="posArray" condition="1">
  0.0   0.0   0.0
  0.5   0.5   0.5
  </attrib>
  <attrib name="ionid" datatype="stringArray">
     Li H
  </attrib>
</particleset>
<particleset name="e" random="yes" randomsrc="i">
  <group name="u" size="2">
    <parameter name="charge">-1</parameter>
  </group>
  <group name="d" size="2">
    <parameter name="charge">-1</parameter>
  </group>
</particleset>
```

Listing 7.2: particleset elements for ions and electrons specifying electron start positions

```
<particleset name="e">
  <group name="u" size="4">
    <parameter name="charge">-1</parameter>
    <attrib name="position" datatype="posArray">
      2.9151687332e-01 -6.5123272502e-01 -1.2188463918e-01
      5.8423636048e-01  4.2730406357e-01 -4.5964306231e-03
      3.5228575807e-01 -3.5027014639e-01  5.2644808295e-01
     -5.1686250912e-01 -1.6648002292e+00  6.5837023441e-01
    </attrib>
  </group>
  <group name="d" size="4">
    <parameter name="charge">-1</parameter>
    <attrib name="position" datatype="posArray">
      3.1443445436e-01  6.5068682609e-01 -4.0983449009e-02
     -3.8686061749e-01 -9.3744432997e-02 -6.0456005388e-01
      2.4978241724e-02 -3.2862514649e-02 -7.2266047173e-01
     -4.0352404772e-01  1.1927734805e+00  5.5610824921e-01
    </attrib>
  </group>
</particleset>
<particleset name="ion0" size="3">
  <group name="O">
    <parameter name="charge">6</parameter>
    <parameter name="valence">4</parameter>
    <parameter name="atomicnumber">8</parameter>
  </group>
  <group name="H">
    <parameter name="charge">1</parameter>
    <parameter name="valence">1</parameter>
    <parameter name="atomicnumber">1</parameter>
  </group>
  <attrib name="position" datatype="posArray">
    0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
    0.0000000000e+00 -1.4308249289e+00  1.1078707576e+00
    0.0000000000e+00  1.4308249289e+00  1.1078707576e+00
  </attrib>
  <attrib name="ionid" datatype="stringArray">
    O H H
  </attrib>
</particleset>
```

Listing 7.3: particleset elements for ions specifying positions by ion type

```
<particleset name="ion0">
  <group name="O" size="1">
    <parameter name="charge">6</parameter>
    <parameter name="valence">4</parameter>
    <parameter name="atomicnumber">8</parameter>
    <attrib name="position" datatype="posArray">
      0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
    </attrib>
  </group>
  <group name="H" size="2">
    <parameter name="charge">1</parameter>
    <parameter name="valence">1</parameter>
    <parameter name="atomicnumber">1</parameter>
    <attrib name="position" datatype="posArray">
      0.0000000000e+00 -1.4308249289e+00  1.1078707576e+00
      0.0000000000e+00  1.4308249289e+00  1.1078707576e+00
    </attrib>
  </group>
</particleset>
```

Chapter 8

Trial wavefunction specification

8.1 Introduction

This section describes the input blocks associated with the specification of the trial wavefunction in a QMCPACK calculation. These sections are contained within the `<wavefunction> ... </wavefunction>` xml blocks. **Users are expected to rely on converters to generate the input blocks described in this section.** The converters and the workflows are designed such that input blocks require minimum modifications from users. Unless the workflow requires modification of wavefunction blocks (e.g. setting the cutoff in a multi determinant calculation), only expert users should directly alter them.

The trial wavefunction in QMCPACK has a general product form:

$$\Psi_T(\vec{r}) = \prod_k \Theta_k(\vec{r}), \tag{8.1}$$

where each $\Theta_k(\vec{r})$ is a function of the electron coordinates (and possibly ionic coordinates and variational parameters). For problems involving electrons, the overall trial wavefunction must be antisymmetric with respect to electron exchange, so at least one of the functions in the product must be antisymmetric. Notice that, while QMCPACK allows for the construction of arbitrary trial wavefunctions based on the functions implemented in the code (e.g. slater determinants, jastrow functions, etc), the user must make sure that a correct wavefunction is used for the problem at hand. From here on, we assume a standard trial wavefunction for an electronic structure problem,

$$\Psi_T(\vec{r}) = A(\vec{r}) \prod_k J_k(\vec{r}), \tag{8.2}$$

where $A(\vec{r})$ is one of the antisymmetric functions: 1) slater determinant, 2) multi slater determinant, or 3) pfaffian, and J_k is any of the jastrow functions (described in section 8.4). The antisymmetric functions are built from a set of single particle orbitals (`sposet`). QMCPACK implements 4 different types of `sposet`, described in the section below. Each `sposet` is designed for a different type of calculation, so their definition and generation varies accordingly.

8.2 Single determinant wavefunctions

Placing a single determinant for each spin is the most used ansatz for the antisymmetric part of a trial wavefunction. The input xml block for `slaterdeterminant` is give in Listing 8.1. A list of options is given in Table 8.1

slaterdeterminant element				
parent elements:	determinantset			
child elements:	determinant			
attribute :				
name	**datatype**	**values**	**default**	**description**
delay_rank	integer	>0	1	The number of delayed updates.
optimize	text	yes/no	yes	Enable orbital optimization.

Table 8.1: Options for the slaterdeterminant xml-block.

Listing 8.1: slaterdeterminant set XML element.

```
<slaterdeterminant delay_rank="32">
  <determinant id="updet" size="208">
    <occupation mode="ground" spindataset="0">
    </occupation>
  </determinant>
  <determinant id="downdet" size="208">
    <occupation mode="ground" spindataset="0">
    </occupation>
  </determinant>
</slaterdeterminant>
```

Additional information:

- delay_rank. This option enables the delayed updates of Slater matrix inverse when particle-by-particle move is used. By default, delay_rank=1 uses the Fahy's variant [2] of the Sherman-Morrison rank-1 update which is mostly using memory bandwidth bound BLAS-2 calls. With delay_rank>1, the delayed update algorithm [47, 46] turns most of the computation to compute bound BLAS-3 calls. Tuning this parameter is highly recommended to gain the best performance on medium to large problem sizes ($>$ 200 electrons). We have seen up to an order of magnitude speed-up on large problem sizes. When studying the performance of QMCPACK, a scan of this parameter is required and we recommend to start from 32. The best delay_rank giving the maximal speed-up depends the problem size. Usually the larger delay_rank corresponds to a larger problem size. On CPUs, delay_rank must be chosen a multiple of SIMD vector length for good performance of BLAS libraries. The best delay_rank depends on the processor micro architecture. The GPU support is currently under development.

8.3 Single-particle orbitals

8.3.1 Spline basis sets

In this section we describe the use of spline basis sets to expand the `sposet`. Spline basis sets are designed to work seamlessly with plane wave DFT code, e.g. Quantum ESPRESSO as a trial wavefunction generator.

In QMC algorithms, all the SPOs $\{\phi(\vec{r})\}$ need to be updated every time a single electron moves. Evaluating SPOs takes very large portion of computation time. In principle, PW basis set can be used to express SPOs directly in QMC like in DFT. but it introduces an unfavorable scaling due to the fact that the basis set size increases linearly as the system size. For this reason, it is efficient to use a localized basis with compact support and a good transferability from plane wave basis.

In particular, 3D tricubic B-splines provide a basis in which only 64 elements are nonzero at any given point in space [13]. The one-dimensional cubic B-spline is given by,

$$f(x) = \sum_{i'=i-1}^{i+2} b^{i',3}(x)\ p_{i'}, \tag{8.3}$$

where $b^i(x)$ are the piecewise cubic polynomial basis functions and $i = \text{floor}(\Delta^{-1}x)$ is the index of the first grid point $\leq x$. Constructing a tensor product in each Cartesian direction, we can represent a 3D orbital as

$$\phi_n(x,y,z) = \sum_{i'=i-1}^{i+2} b_x^{i',3}(x) \sum_{j'=j-1}^{j+2} b_y^{j',3}(y) \sum_{k'=k-1}^{k+2} b_z^{k',3}(z)\ p_{i',j',k',n}. \tag{8.4}$$

This allows the rapid evaluation of each orbital in constant time. Furthermore, this basis is systematically improvable with a single spacing parameter, so that accuracy is not compromised compared with plane wave basis.

The use of 3D tricubic B-splines greatly improves the computational efficiency. The gain in computation time from plane wave basis set to an equivalent B-spline basis set becomes increasingly large as the system size grows. On the downside, this computational efficiency comes at the expense of increased memory use, which is easily overcome by the large aggregate memory available per node through OpenMP/MPI hybrid QMC.

The input xml block for the spline SPOs is give in Listing 8.2. A list of options is given in Table 8.2. `QMCPACK` has a very useful command line option `--save_wfs` which allows to dump the real space B-spline coefficient table into a h5 file on the disk. When the orbital transformation from k space to B-spline requires more than available amount of scratch memory on the compute nodes, users can perform this step on fat nodes and transfer back the h5 file for QMC calculations.

Listing 8.2: Determinant set XML element.

```
<determinantset type="bspline" source="i" href="pwscf.h5"
                tilematrix="1 1 3 1 2 -1 -2 1 0" twistnum="-1" gpu="yes"
   meshfactor="0.8"
                twist="0  0  0" precision="double">
  <slaterdeterminant>
    <determinant id="updet" size="208">
      <occupation mode="ground" spindataset="0">
      </occupation>
    </determinant>
    <determinant id="downdet" size="208">
      <occupation mode="ground" spindataset="0">
```

determinantset element				
parent elements:	wavefunction			
child elements:	slaterdeterminant			
attribute :				
name	**datatype**	**values**	**default**	**description**
type	text	bspline		Type of sposet.
href	text			Path to hdf5 file from pw2qmcpack.x.
tilematrix	9 integers			Tiling matrix used to expand supercell.
twistnum	integer			Index of the super twist.
twist	3 floats			Super twist.
meshfactor	float	≤ 1.0		Grid spacing ratio.
precision	text	single/double		Precision of spline coefficients.
gpu	text	yes/no		GPU switch.
gpusharing	text	yes/no	no	Share B-spline table across GPUs.
Spline_Size_Limit_MB	integer			Limit B-spline table size on GPU.
check_orb_norm	text	yes/no	yes	Check norms of orbitals from h5 file.
save_coefs	text	yes/no	no	Save the spline coefficients to h5 file.
source	text	*any*	ion0	Particle set with atomic positions.

Table 8.2: Options for the determinantset xml-block associated with B-spline single particle orbital sets.

```
      </occupation>
    </determinant>
  </slaterdeterminant>
</determinantset>
```

Additional information:

- precision. Only effective on CPU version without mixed precision, 'single' is always imposed with mixed precision. Using single precision not only saves memory usage but also speeds up the B-spline evaluation. It is recommended to use single precision since we saw little chance of really compromising the accuracy of calculation.
- meshfactor. It is the ratio of actual grid spacing of B-splines used in QMC calculation with respect to the original one calculated from h5. Smaller meshfactor saves memory usage but reduces accuracy. The effects are similar to reducing plane wave cutoff in DFT calculation. Use with caution!
- twistnum. If positive, it is the index. It is recommended not to take this way since the indexing may show some uncertainty. If negative, the super twist is referred by twist.

- `gpusharing`. If enabled, spline data is shared across multiple GPUs on a given computational node. For example, on a two GPU per node system, each GPU would have half of the orbitals. This enables larger overall spline tables than would fit in the memory of individual GPUs to be utilized, and potentially up to the total GPU memory on a node. To obtain high performance, large electron counts or a high-performing CPU-GPU interconnect is required. In order to use this feature, the following needs to be done:
  - The CUDA Multi-Process Service (MPS) needs to be utilized (e.g. on Summit/SummitDev use "-alloc_flags gpumps" for bsub). If MPS is not detected sharing will be disabled.
  - CUDA_VISIBLE_DEVICES needs to be properly set to control each rank's visible CUDA devices (e.g. on OLCF Summit/SummitDev one needs to create a resource set containing all GPUs with the respective number of ranks with "jsrun –task-per-rs Ngpus -g Ngpus")
- `Spline_Size_Limit_MB`. Allows to distribute the B-spline coefficient table between the host and GPU memory. The compute kernels access host memory via zero-copy. Though the performance penalty introduced by it is significant but allows large calculations to go.

8.3.2 Gaussian basis sets

In this section we describe the use of localized basis sets to expand the `sposet`. The general form of a single particle orbital in this case is given by:

$$\phi_i(\vec{r}) = \sum_k C_{i,k}\ \eta_k(\vec{r}), \tag{8.5}$$

where $\{\eta_k(\vec{r})\}$ is a set of M atom-centered basis functions and $C_{i,k}$ is a coefficient matrix. This `sposet` should be used in calculations of finite systems employing an atom-centered basis set and is typically generated by the *convert4qmc* converter. Examples include calculations of molecules using gaussian basis sets or slater-type basis functions. Initial support for periodic systems is described in Sec. 13. Even though this section is called "Gaussian basis sets" (by far the most common atom-centered basis set), QMCPACK works with any atom-centered basis set built based on either spherical harmonic angular functions or cartesian angular expansions. The radial functions in the basis set can be expanded in either gaussian functions, slater-type functions or numerical radial functions.

In this section we describe the input sections for the atom-centered basis set and the `sposet` for a single slater determinant trial wavefunction. The input sections for multideterminant trial wavefunctions are described in section 8.5. The basic structure for the input block of a single slater determinant is given in Listing 8.3. A list of options for `determinantset` associated with this `sposet` is given in Table 8.3.

Listing 8.3: Basic input block for a single determinant trial wavefunction using a sposet expanded on an atom-centered basis set.

```
<wavefunction id="psi0" target="e">
    <determinantset>
      <basisset>
        ...
      </basisset>
      <slaterdeterminant>
        ...
      </slaterdeterminant>
    </determinantset>
</wavefunction>
```

The definition of the set of atom-centered basis functions is given by the `basisset` block, while the sposet is defined within `slaterdeterminant`. The basisset input block is composed from a collection of `atomicBasisSet` input blocks, one for each atomic species in the simulation where basis functions are centered. The general structure for `basisset` and `atomicBasisSet` are given in Listing 8.4, while the corresponding lists of options are given in Tables 8.4 and 8.5.

determinantset element				
parent elements:	wavefunction			
child elements:	basisset,slaterdeterminant,sposet,multideterminant			
attribute :				
name	**datatype**	**values**	**default**	**description**
name/id	text	*any*	""	Name of determinant set.
type	text	see below	""	Type of sposet.
keyword	text	NMO,GTO,STO	NMO	Type of orbital set generated.
transform	text	yes/no	yes	Transform to numerical radial functions?
source	text	*any*	ion0	Particle set with the position of atom centers.
cuspCorrection	text	yes/no	no	Apply cusp correction scheme to sposet?

Table 8.3: Options for the determinantset xml-block associated with atom-centered single particle orbital sets.

Listing 8.4: Basic input block for basisset.

```
    <basisset name="LCAOBSet">
      <atomicBasisSet name="Gaussian-G2" angular="cartesian" elementType="C"
normalized="no">
        <grid type="log" ri="1.e-6" rf="1.e2" npts="1001"/>
        <basisGroup rid="C00" n="0" l="0" type="Gaussian">
          <radfunc exponent="5.134400000000e-02" contraction="1.399098787100e-02"/>
          ...
        </basisGroup>
        ...
      </atomicBasisSet>
      <atomicBasisSet name="Gaussian-G2" angular="cartesian" type="Gaussian"
elementType="C" normalized="no">
        ...
      </atomicBasisSet>
      ...
    </basisset>
```

basisset element				
parent elements:	determinantset			
child elements:	atomicBasisSet			
attribute :				
name	**datatype**	**values**	**default**	**description**
name/id	text	*any*	""	Name of atom-centered basis set.

Table 8.4: Options for the basisset xml-block associated with atom-centered single particle orbital sets.

atomicBasisSet element				
parent elements:	basisset			
child elements:	grid,basisGroup			
attribute :				
name	**datatype**	**values**	**default**	**description**
name/id	text	*any*	""	Name of atomic basis set.
angular	text	see below	default	Type of angular functions.
expandYlm	text	see below	yes	Expand Ylm shells?
expM	text	see below	yes	Add sign for $(-1)^m$?
elementType/species	text	*any*	e	Atomic species where functions are centered.
normalized	text	yes/no	yes	Are single particle functions normalized?

Table 8.5: Options for the atomicBasisSet xml-block.

basisGroup element				
parent elements:	atomicBasisSet			
child elements:	radfunc			
attribute :				
name	**datatype**	**values**	**default**	**description**
rid/id	text	*any*	""	Name of the basisGroup.
type	text	*any*	""	Type of basisGroup.
n/l/m/s	integer	*any*	0	Quantum numbers of basisGroup.

Table 8.6: Options for the basisGroup xml-block.

Listing 8.5: Basic input block for slaterdeterminant with an atom-centered sposet.

```
<slaterdeterminant>
</slaterdeterminant>
```

element				
parent elements:				
child elements:				
attribute :				
name	**datatype**	**values**	**default**	**description**
name/id	text	*any*	""	Name of determinant set
	text	*any*	""	

Detailed description of attributes:

In the following, we give a more detailed description of all the options presented in the various xml-blocks described in this section. Only non-trivial attributes are described below. Those with

simple yes/no options and whose description above is enough to explain the intended behavior are not included.

`determinantset` attributes:

- `type`
  Type of sposet. For atom-centered based sposets, use type="MolecularOrbital" or type="MO". Other options describe elsewhere in this manual are "spline", "composite", "pw", "heg", "linearopt", etc.

- `keyword/key`
  Type of basis set generated, which doesn't necessarily match the type of the basis set on the input block. The three possible options are: NMO (numerical molecular orbitals), GTO (gaussian-type orbitals), STO (slater-type orbitals). The default option is NMO. By default, QMCPACK will generate numerical orbitals from both GTO and STO types and use cubic or quintic spline interpolation to evaluate the radial functions. This is typically more efficient than evaluating the radial functions in the native basis (gaussians or exponents) and allows for arbitrarily large contractions without any additional cost. To force the use of the native expansion (not recommended), use GTO or STO for each type of input basis set.

- `transform`
  Request (or avoid) a transformation of the radial functions to NMO type. The default and recommended behavior is to transform to numerical radial functions. If `transform` is set to *"yes"*, the option `keyword` is ignored.

- `cuspCorrection`
  Enable (disable) the use of the cusp correction algorithm (CASINO REFERENCE) for a `basisset` built with GTO functions. The algorithm is implemented as described in (CASINO REFERENCE) and only works with transform="yes" and an input GTO basis set. No further input is needed.

`atomicBasisSet` attributes:

- `name/id`
  Name of the basis set. Names should be unique.

- `angular`
  Type of angular functions used in the expansion. In general, two angular basis functions are allowed: "spherical" (for spherical Ylm functions) and "cartesian" (for functions of the type $x^n y^m z^l$).

- `expandYlm`
  Determines whether each basis group is expanded across the corresponding shell of m values (for spherical type) or consistent powers (for cartesian functions). Options:

  - "No": Do not expand angular functions across corresponding angular shell.
  - "Gaussian": Expand according to Gaussian03 format. This function is only compatible with angular="spherical". For a given input (l,m), the resulting order of the angular functions becomes: (1,-1,0) for l=1 and (0,1,-1,2,-2,...,l,-l) for general l.

  - "Natural": Expand angular functions according to (-l,-l+1,...,l-1,l).
  - "Gamess": Expand according to Gamess' format for cartesian functions. Notice that this option is only compatible with angular="cartesian". If angular="cartesian" is used, this option is not necessary.

- `expM`
  Determines whether the sign of the spherical Ylm function associated with m (-1^m) is included in the coefficient matrix or not.

- `elementType/species`
  Name of the species where basis functions are centered. Only one atomicBasisSet block is allowed per species. Additional blocks are ignored. The corresponding species must exist in the `particleset` given as the `source` option to `determinantset`. Basis functions for all the atoms of the corresponding species are included in the basis set, based on the order of atoms in the `particleset`.

`basisGroup` attributes:

- `type`
  Type of input basis radial function. Notice that this refers to the type of radial function in the input xml-block, which might not match the radial function generated internally and used in the calculation (if `transform` is set to "yes"). Also notice that different `basisGroup` blocks within a given `atomicBasisSet` can have different `type`.

- `n/l/m/s`
  Quantum numbers of the basis function. Notice that if `expandYlm` is set to *"yes"* in `atomicBasisSet`, a full shell of basis functions with the appropriate values of *"m"* will be defined for the corresponding value of *"l"*. Otherwise a single basis function will be given for the specific combination of *"(l,m)"*.

`radfunc` attributes for `type`=*"Gaussian"*:

- `TBDoc`

a

`slaterdeterminant` attributes:

- `TBDoc`

8.3.3 Hybrid orbital representation

The hybrid representation of the single particle orbitals combines a localized atomic basis set around atomic cores and B-splines in the interstitial regions to reduce the memory usage while retaining high speed of evaluation and either retaining or increasing overall accuracy. Full details are provided in Ref. [41] and **users of this feature are kindly requested to cite this paper**. In practice, we have seen using meshfactor=0.5 is often possible and achieves huge memory saving. Figure 8.1 illustrates how the regions are assigned. Orbitals within region A are computed as

$$\phi_n^A(\mathbf{r}) = R_{n,l,m}(r)Y_{l,m}(\hat{r})$$

Orbitals in region C are computed as the regular B-spline basis described in subsection 8.3.1 above. The region B interpolates between A and C based on smoothing schemes Consistent(default), SmoothPartial or SmoothAll.

$$\begin{aligned}
\phi_n^B(\mathbf{r}) &= S(r)\phi_n^A(\mathbf{r}) + (1-S(r))\phi_n^C(\mathbf{r}) && \texttt{Consistent} && (8.6)\\
\phi_n^{B'}(\mathbf{r}) &= \nabla\phi_n^B(\mathbf{r}) \\
\phi_n^{B''}(\mathbf{r}) &= \nabla^2\phi_n^B(\mathbf{r}) \\
\phi_n^B(\mathbf{r}) &= S(r)\phi_n^A(\mathbf{r}) + (1-S(r))\phi_n^C(\mathbf{r}) && \texttt{SmoothPartial} && (8.7)\\
\phi_n^{B'}(\mathbf{r}) &= S(r)\phi_n^{A'}(\mathbf{r}) + (1-S(r))\phi_n^{C'}(\mathbf{r}) \\
\phi_n^{B''}(\mathbf{r}) &= \nabla\phi_n^{B'}(\mathbf{r}) \\
\phi_n^B(\mathbf{r}) &= S(r)\phi_n^A(\mathbf{r}) + (1-S(r))\phi_n^C(\mathbf{r}) && \texttt{SmoothAll} && (8.8)\\
\phi_n^{B'}(\mathbf{r}) &= S(r)\phi_n^{A'}(\mathbf{r}) + (1-S(r))\phi_n^{C'}(\mathbf{r}) \\
\phi_n^{B''}(\mathbf{r}) &= S(r)\phi_n^{A''}(\mathbf{r}) + (1-S(r))\phi_n^{C''}(\mathbf{r})
\end{aligned}$$

The available smoothing functions $S(r)$ are LEKS2018(default), coscos and linear.

$$x = \frac{r - r_{\mathrm{A/B}}}{r_{\mathrm{B/C}} - r_{\mathrm{A/B}}} \quad (8.9)$$

$$S(x) = \frac{1}{2} - \frac{1}{2}\tanh\left[2.0\left(x - \frac{1}{2}\right)\right] \quad \texttt{LEKS2018} \quad (8.10)$$

$$S(x) = \frac{1 + \cos\{\pi[1-\cos(\pi x)]/2\}}{2} \quad \texttt{coscos} \quad (8.11)$$

$$S(x) = 1 - x \quad \texttt{linear} \quad (8.12)$$

Smoothing schemes and functions other than the default ones must be tested before use.

To enable hybrid orbital representation, the input XML requires the tag `hybridrep="yes"` shown below. `smoothing_scheme="Consistent"` and `smoothing_function="LEKS2018"` are optional.

Listing 8.6: Hybrid orbital representation input example.

```
<determinantset type="bspline" source="i" href="pwscf.h5"
                tilematrix="1 1 3 1 2 -1 -2 1 0" twistnum="-1" gpu="yes"
                meshfactor="0.8" twist="0  0  0"
                precision="single" hybridrep="yes"
                smoothing_scheme="Consistent" smoothing_function="LEKS2018">
  ...
</determinantset>
```

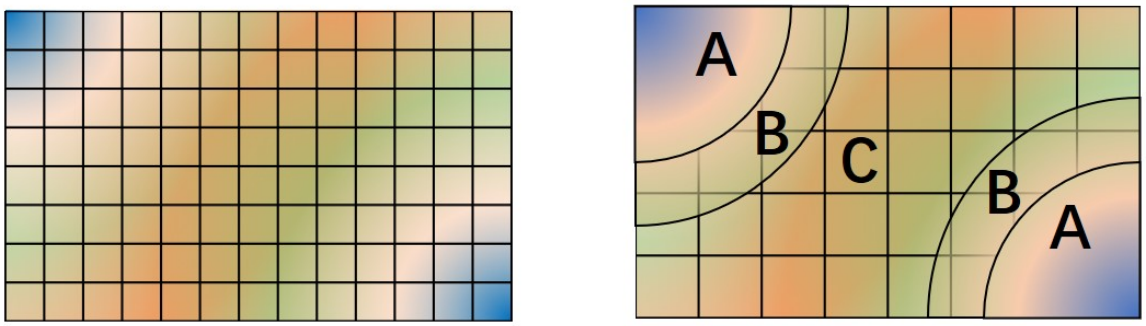


Figure 8.1: Illustration of regular and hybrid orbital representation. Regular B-spline representation (left panel) contains only one region and a sufficiently fine mesh to resolve orbitals near the nucleus. The hybrid orbital representation (right panel) contains near nucleus (A) regions where spherical harmonics and radial functions are used, buffers (B) or interpolation regions, and an interstitial (C) region where a coarse B-spline mesh is utilized.

Second, the information describing the atomic regions is required in the particle set, shown below

Listing 8.7: particleset elements for ions with information needed by hybrid orbital representation.

```
<group name="Ni">
  <parameter name="charge">          18 </parameter>
  <parameter name="valence">         18 </parameter>
  <parameter name="atomicnumber" >   28 </parameter>
  <parameter name="cutoff_radius" > 1.6 </parameter>
  <parameter name="inner_cutoff" >  1.3 </parameter>
  <parameter name="lmax" >            5 </parameter>
  <parameter name="spline_radius" > 1.8 </parameter>
  <parameter name="spline_npoints">  91 </parameter>
</group>
```

The parameters specific to hybrid representation are listed as

`attrib` element				
attribute :				
name	**datatype**	**values**	**default**	**description**
`cutoff_radius`	real	$>= 0.0$	*none*	The cutoff radius for B/C boundary
`lmax`	integer	$>= 0$	*none*	Largetst angular channel
`inner_cutoff`	real	$>= 0.0$	dep.	The cutoff radius for A/B boundary
`spline_radius`	real	> 0.0	dep.	radial function radius used in spline
`spline_npoints`	integer	> 0	dep.	Number of spline knots

- `cutoff_radius` is required for every species. If a species is intended not being covered by atomic regions, setting the value 0.0 will put default values for all the reset parameters. A good value is usually a bit larger than the core radius listed in the pseudopotential file. After a parametric scan, pick the one from the flat energy region with the smallest variance.
- `lmax` is required if `cutoff_radius` > 0.0. The value usually needs to be at least the highest angular momentum plus 2.

- `inner_cutoff` is optional and set as `cutoff_radius` − 0.3 by default which is fine in most cases.
- `spline_radius` and `spline_npoints` are optional. By default, they are calculated based on `cutoff_radius` and a grid distplacement 0.02 bohr. If users prefer inputing them, it is required that `cutoff_radius` <= `spline_radius` − 2 × `spline_radius`/(`spline_npoints` − 1).

In addition, the hybrid orbital representation allows extra optimization to speed up the non-local pseudopotential evaluation using the batched algorithm listed in Sec. 9.2.2.

8.3.4 Plane-wave basis sets

8.3.5 Homogeneous electron gas

The interacting Fermi Liquid has its own special determinantset for filling up a Fermi surface. The shell number can be specified separately for both spin up and spin down. This determines how many electrons to include of each time, only closed shells are currently implemented. The shells are filled according to the rules of a square box, if other lattice vectors are used, the electrons may not fill up a complete shell.

This following example can also be used for Helium simulations too, by specifying the proper pair interaction in the Hamiltonian section.

Listing 8.8: 2D Fermi Liquid example: particle specification

```
<qmcsystem>
<simulationcell name="global">
<parameter name="rs" pol="0" condition="74">6.5</parameter>
<parameter name="bconds">p p p</parameter>
<parameter name="LR_dim_cutoff">15</parameter>
</simulationcell>
<particleset name="e" random="yes">
<group name="u" size="37">
<parameter name="charge">-1</parameter>
<parameter name="mass">1</parameter>
</group>
<group name="d" size="37">
<parameter name="charge">-1</parameter>
<parameter name="mass">1</parameter>
</group>
</particleset>
</qmcsystem>
```

Listing 8.9: 2D Fermi Liquid example (Slater Jastrow wave function)

```
<qmcsystem>
  <wavefunction name="psi0" target="e">
      <determinantset type="electron-gas" shell="7" shell2="7" randomize="true">
  </determinantset>
      <jastrow name="J2" type="Two-Body" function="Bspline" print="no">
        <correlation speciesA="u" speciesB="u" size="8" cusp="0">
          <coefficients id="uu" type="Array" optimize="yes">
        </correlation>
        <correlation speciesA="u" speciesB="d" size="8" cusp="0">
          <coefficients id="ud" type="Array" optimize="yes">
        </correlation>
      </jastrow>
```

8.4 Jastrow Factors

Jastrow factors are among the simplest and most effective ways of including dynamical correlation in the trial many body wavefunction. The resulting many body wavefunction is expressed as the product of an antisymmetric (in the case of Fermions) or symmetric (for Bosons) part and a correlating Jastrow factor like so:

$$\Psi(\vec{R}) = \mathcal{A}(\vec{R}) \exp\left[J(\vec{R})\right] \tag{8.13}$$

In this section we will detail the types and forms of Jastrow factor used in QMCPACK. Note that each type of Jastrow factor needs to be specified using its own individual `jastrow` XML element. For this reason, we have repeated the specification of the `jastrow` tag in each section, with specialization for the options available for that given type of Jastrow.

8.4.1 One-body Jastrow functions

The one-body Jastrow factor is a form that allows for the direct inclusion of correlations between particles that are included in the wavefunction with particles that are not explicitly part of it. The most common example of this are correlations between electrons and ions.

The Jastrow function is specified within a `wavefunction` element and must contain one or more `correlation` elements specifying additional parameters as well as the actual coefficients. Section 8.4.1 gives examples of the typical nesting of `jastrow`, `correlation`, and `coefficient` elements.

Input Specification

Jastrow element				
name	**datatype**	**values**	**defaults**	**description**
name	text		(required)	Unique name for this Jastrow function
type	text	One-body	(required)	Define a one-body function
function	text	Bspline	(required)	BSpline Jastrow
	text	pade2		Pade form
	text	...		...
source	text	name	(required)	Name of attribute of classical particle set
print	text	yes / no	yes	Jastrow factor printed in external file?
elements				
	Correlation			
Contents				
	(None)			

To be more concrete, the one-body Jastrow factors used to describe correlations between electrons and ions take the form below

$$J1 = \sum_{I}^{ion0} \sum_{i}^{e} u_{ab}(|r_i - R_I|) \tag{8.14}$$

where I runs over all of the ions in the calculation, i runs over the electrons and u_{ab} describes the functional form of the correlation between them. Many different forms of u_{ab} are implemented in QMCPACK. We will detail two of the most common ones below.

Spline form

The one-body spline Jastrow function is the most commonly used one-body Jastrow for solids. This form was first described and used in [7]. Here u_{ab} is an interpolating 1D Bspline (tricublc spline on a linear grid) between zero distance and r_{cut}. In 3D periodic systems the default cutoff distance is the Wigner Seitz cell radius. For other periodicities including isolated molecules the r_{cut} must be specified. The cusp can be set. r_i and R_I are most commonly the electron and ion positions, but any particlesets that can provide the needed centers can be used.

Correlation element				
name	**datatype**	**values**	**defaults**	**description**
elementType	text	name	see below	Classical particle target
speciesA	text	name	see below	Classical particle target
speciesB	text	name	see below	Quantum species target
size	integer	> 0	(required)	Number of coefficients
rcut	real	> 0	see below	Distance at which the correlation goes to 0
cusp	real	≥ 0	0	Value for use in Kato cusp condition
spin	text	yes or no	no	Spin dependent jastrow factor
elements				
	Coefficients			
Contents				
	(None)			

Input Specification Additional information:

- `elementType, speciesA, speciesB, spin`. For a spin independent Jastrow factor (spin = "no") elementType should be the name of the group of ions in the classical particleset to which the quantum particles should be correlated. For a spin dependent Jastrow factor (spin = "yes") set speciesA to the group name in the classical particleset and speciesB to the group name in the quantum particleset.

- `rcut`. The cutoff distance for the function in atomic units (bohr). For 3D fully periodic systems this parameter is optional and a default of the Wigner Seitz cell radius is used. Otherwise this parameter is required.

- `cusp`. The one body jastrow factor can be used to make the wavefunction satisfy the electron-ion cusp condition[5]. In this case, the derivative of the jastrow factor as the electron approaches the nucleus will be given by:

$$\left(\frac{\partial J}{\partial r_{iI}}\right)_{r_{iI}=0} = -Z \tag{8.15}$$

  Note that if the antisymmetric part of the wavefunction satisfies the electron-ion cusp condition (for instance by using single particle orbitals that respect the cusp condition) or if a non-divergent pseudopotential is used that the Jastrow should be cuspless at the nucleus and this value should be kept at its default of 0.

Coefficients element				
name	**datatype**	**values**	**defaults**	**description**
id	text		(required)	Unique identifier
type	text	Array	(required)	
optimize	text	yes or no	yes	if no, values are fixed in optimizations
elements				
(None)				
Contents				
(no name)	real array		zeros	Jastrow coefficients

Example use cases Specify a spin-independent function with four parameters. Because rcut is not specified, the default cutoff of the Wigner Seitz cell radius is used; this Jastrow must be used with a 3D periodic system such as a bulk solid. The name of the particleset holding the ionic positions is "i".

```
<jastrow name="J1" type="One-Body" function="Bspline" print="yes" source="i">
 <correlation elementType="C" cusp="0.0" size="4">
   <coefficients id="C" type="Array"> 0  0  0  0  </coefficients>
 </correlation>
</jastrow>
```

Specify a spin-dependent function with seven upspin and seven downspin parameters. The cutoff distance is set to 6 atomic units. Note here that the particleset holding the ions is labeled as ion0 rather than "i" in the other example. Also in this case the ion is Lithium with a coulomb potential, so the cusp condition is satisfied by setting cusp="d".

```
<jastrow name="J1" type="One-Body" function="Bspline" source="ion0" spin="yes">
  <correlation speciesA="Li" speciesB="u" size="7" rcut="6">
    <coefficients id="eLiu" cusp="3.0" type="Array">
    0.0 0.0 0.0 0.0 0.0 0.0 0.0
    </coefficients>
  </correlation>
  <correlation speciesA="C" speciesB="d" size="7" rcut="6">
    <coefficients id="eLid" cusp="3.0" type="Array">
    0.0 0.0 0.0 0.0 0.0 0.0 0.0
    </coefficients>
  </correlation>
</jastrow>
```

Pade form

While the spline Jastrow factor is the most flexible and most commonly used form implemented in QMCPACK , there are times where its flexibility can make it difficult to optimize. As an example, a spline jastrow with a very large cutoff may be difficult to optimize for isolated systems like molecules due to the small number of samples that will be present in the tail of the function. In such cases, a simpler functional form may be advantageous. The second order Pade jastrow factor, given in Eq.8.16 is a good choice in such cases.

$$u_{ab}(r) = \frac{a * r + c * r^2}{1 + b * r} \tag{8.16}$$

Unlike the spline jastrow factor which includes a cutoff, this form has an infinite range and for every particle pair (subject to the minimum image convention) it will be applied. It also is a cuspless jastrow factor, so it should either be used in combination with a single particle basis set that contains the proper cusp or with a smooth pseudopotential.

Correlation element				
name	**datatype**	**values**	**defaults**	**description**
elementType	text	name	see below	Classical particle target
elements				
	Coefficients			
Contents				
	(None)			

parameter element				
name	**datatype**	**values**	**defaults**	**description**
id	string	name	(required)	name for variable
name	string	A or B or C	(required)	see Eq.8.16
optimize	text	yes or no	yes	if no, values are fixed in optimizations
elements				
(None)				
Contents				
(no name)	real	parameter value	(required)	Jastrow coefficients

Input Specification

Example use case Specify a spin independent function with independent jastrow factors for two different species (Li and H). The name of the particleset holding the ionic positions is "i".

```
<jastrow name="J1" function="pade2" type="One-Body" print="yes" source="i">
  <correlation elementType="Li">
    <var id="LiA" name="A">  0.34 </var>
    <var id="LiB" name="B"> 12.78 </var>
    <var id="LiC" name="C">  1.62 </var>
  </correlation>
  <correlation elementType="H"">
```

```
    <var id="HA" name="A">  0.14 </var>
    <var id="HB" name="B"> 6.88 </var>
    <var id="HC" name="C"> 0.237 </var>
  </correlation>
</jastrow>
```

8.4.2 Two-body Jastrow functions

The two-body Jastrow factor is a form that allows for the explicit inclusion of dynamic correlation between two particles included in the wavefunction. It is almost always given in a spin dependent form so as to satisfy the Kato cusp condition between electrons of different spins[5].

The two body Jastrow function is specified within a `wavefunction` element and must contain one or more correlation elements specifying additional parameters as well as the actual coefficients. Section 8.4.2 gives examples of the typical nesting of `jastrow`, `correlation` and `coefficient` elements.

Input Specification

Jastrow element				
name	**datatype**	**values**	**defaults**	**description**
name	text		(required)	Unique name for this Jastrow function
type	text	Two-body	(required)	Define a one-body function
function	text	Bspline	(required)	BSpline Jastrow
print	text	yes / no	yes	Jastrow factor printed in external file?
elements				
Correlation				
Contents				
(None)				

The two-body Jastrow factors used to describe correlations between electrons take the form

$$J2 = \sum_{i}^{e} \sum_{j>i}^{e} u_{ab}(|r_i - r_j|) \tag{8.17}$$

The most commonly used form of two body Jastrow factor supported by the code is a splined Jastrow factor, with many similarities to the one body spline Jastrow.

Spline form

The two-body spline Jastrow function is the most commonly used two-body Jastrow for solids. This form was first described and used in [7]. Here u_{ab} is an interpolating 1D Bspline (tricublc spline on a linear grid) between zero distance and r_{cut}. In 3D periodic systems the default cutoff distance is the Wigner Seitz cell radius. For other periodicities including isolated molecules the r_{cut} must be specified. r_i and r_j are typically electron positions. The cusp condition as r_i approaches r_j is set by the relative spin of the electrons.

Correlation element				
name	**datatype**	**values**	**defaults**	**description**
speciesA	text	u or d	(required)	Quantum species target
speciesB	text	u or d	(required)	Quantum species target
size	integer	> 0	(required)	number of coefficients
rcut	real	> 0	see below	distance at which the correlation goes to 0
spin	text	yes or no	no	spin dependent jastrow factor
elements				
Coefficients				
Contents				
(None)				

Input Specification Additional information:

- `speciesA, speciesB` The scale function u(r) is defined for species pairs uu and ud. There is no need to define ud or dd since uu=dd and ud=du. The cusp condition is computed internally based on the charge of the quantum particles.

Coefficients element				
name	**datatype**	**values**	**defaults**	**description**
id	text		(required)	Unique identifier
type	text	Array	(required)	
optimize	text	yes or no	yes	if no, values are fixed in optimizations
elements				
(None)				
Contents				
(no name)	real array		zeros	Jastrow coefficients

Example use cases Specify a spin-dependent function with 4 parameters for each channel. In this case, the cusp is set at a radius of 4.0 bohr (rather than to the default of the Wigner Seitz cell radius). Also, in this example, the coefficients are set to not be optimized during an optimization step.

```
<jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
  <correlation speciesA="u" speciesB="u" size="8" rcut="4.0">
```

```
    <coefficients id="uu" type="Array" optimize="no"> 0.2309049836 0.1312646071
    0.05464141356 0.01306231516</coefficients>
  </correlation>
  <correlation speciesA="u" speciesB="d" size="8" rcut="4.0">
    <coefficients id="ud" type="Array" optimize="no"> 0.4351561096 0.2377951747
    0.1129144262 0.0356789236</coefficients>
  </correlation>
</jastrow>
```

8.4.3 Long-ranged Jastrow factors

While short-ranged Jastrow factors capture the majority of the benefit for minimizing the total energy and the energy variance, long-ranged Jastrow factors are important to accurately reproduce the short-ranged (long wavelength) behavior of quantities such as the static structure factor, and are therefore essential for modern accurate finite size corrections in periodic systems.

Below two types of long-ranged Jastrow factors are described. The first (the k-space Jastrow) is simply an expansion of the one and/or two body correlation functions in plane waves, with the coefficients comprising the optimizable parameters. The second type have few variational parameters and use the optimized breakup method of Natoli and Ceperley[10] (the Yukawa and Gaskell RPA Jastrows).

Long-ranged Jastrow: k-space Jastrow

The k-space Jastrow introduces explicit long-ranged dependence commensurate with the periodic supercell. This Jastrow is to be used in periodic boundary conditions only.

The input for the k-space Jastrow fuses both one and two-body forms into a single element and so they are discussed together here. The one- and two-body terms in the k-Space Jastrow have the form:

$$J_1 = \sum_{G \neq 0} b_G \rho_G^I \rho_{-G} \tag{8.18}$$

$$J_2 = \sum_{G \neq 0} a_G \rho_G \rho_{-G} \tag{8.19}$$

Here ρ_G is the Fourier transform of the instantaneous electron density:

$$\rho_G = \sum_{n \in electrons} e^{iG \cdot r_n} \tag{8.20}$$

and ρ_G^I has the same form, but for the fixed ions. In both cases the coefficients are restricted to be real, though in general the coefficients for the one-body term need not be. See section 26.8 for more detail.

Input for the k-space Jastrow follows the familar nesting of `jastrow-correlation-coefficients` elements, with attributes unique to the k-space Jastrow at the `correlation` input level.

`jastrow type=kSpace` element				
parent elements:	`wavefunction`			
child elements:	`correlation`			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**kSpace**		Must be kSpace
`name`^{r}	text	*anything*	0	Unique name for Jastrow
`source`^{r}	text	`particleset.name`		Ion particleset name

`correlation` element				
parent elements:	`jastrow type=kSpace`			
child elements:	`coefficients`			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**One-Body**,**Two-Body**		Must be One-Body/Two-Body
`kc`^{r}	real	kc$\geq$ 0	0.0	k-space cutoff in a.u.
`symmetry`^{o}	text	crystal,isotropic,none	crystal	Symmetry of coefficients
`spinDependent`^{o}	boolean	yes,no	no	*No current function*

`coefficients` element				
parent elements:	`correlation`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`id`^{r}	text	*anything*	cG1/cG2	Label for coeffs
`type`^{r}	text	`Array`	0	Must be Array
body text				
	The body text is a list of real values for the parameters.			

Additional information:

- It is normal to provide no coefficients as an initial guess. The number of coefficients will be automatically calculated according to the k-space cutoff + symmetry and set to zero.
- Providing an incorrect number of parameters also results in all parameters being set to zero.
- There is currently no way to turn optimization on/off for the k-space Jastrow. The coefficients are always optimized.
- Spin dependence is currently not implemented for this Jastrow.
- `kc`: Parameters with G vectors magnitudes less than `kc` are included in the Jastrow. If `kc` is zero, it is the same as excluding the k-space term.
- `symmetry=crystal`: Impose crystal symmetry on coefficients according to the structure factor.
- `symmetry=isotropic`: Impose spherical symmetry on coefficients according to G-vector magnitude.
- `symmetry=none`: Impose no symmetry on the coefficients.

Listing 8.10: k-space Jastrow with one- and two-body terms.

```
<jastrow type="kSpace" name="Jk" source="ion0">
   <correlation kc="4.0" type="One-Body" symmetry="cystal">
      <coefficients id="cG1" type="Array">
```

```
      </coefficients>
    </correlation>
    <correlation kc="4.0" type="Two-Body" symmetry="crystal">
      <coefficients id="cG2" type="Array">
      </coefficients>
    </correlation>
  </jastrow>
```

Listing 8.11: k-space Jastrow with one-body term only.

```
  <jastrow type="kSpace" name="Jk" source="ion0">
    <correlation kc="4.0" type="One-Body" symmetry="cystal">
      <coefficients id="cG1" type="Array">
      </coefficients>
    </correlation>
  </jastrow>
```

Listing 8.12: k-space Jastrow with two-body term only.

```
  <jastrow type="kSpace" name="Jk" source="ion0">
    <correlation kc="4.0" type="Two-Body" symmetry="crystal">
      <coefficients id="cG2" type="Array">
      </coefficients>
    </correlation>
  </jastrow>
```

Long-ranged Jastrows: Gaskell RPA and Yukawa forms

NOTE: The Yukawa and RPA Jastrows do not work at present and are currently being revived. Please contact the developers if you are interested in using them.

The exact Jastrow correlation functions contain terms which have a form similar to the Coulomb pair potential. In periodic systems the Coulomb potential is replaced by an Ewald summation of the bare potential over all periodic image cells. This sum is often handled by the optimized breakup method[10] and this same approach is applied to the long-ranged Jastrow factors in QMCPACK.

There are two main long-ranged Jastrow factors of this type implemented in QMCPACK: the Gaskell RPA[39, 40] form and the Yukawa[1] form. Both of these forms were used by Ceperley in early studies of the electron gas[1], but they are also appropriate starting points for general solids.

The Yukawa form is defined in real space. It's long-range form is formally defined as

$$u_Y^{PBC}(r) = \sum_{L \neq 0} \sum_{i<j} u_Y(|r_i - r_j + L|) \tag{8.21}$$

with $u_Y(r)$ given by

$$u_Y(r) = \frac{a}{r}\left(1 - e^{-r/b}\right) \tag{8.22}$$

In QMCPACK a slightly more restricted form is used:

$$u_Y(r) = \frac{r_s}{r}\left(1 - e^{-r/\sqrt{r_s}}\right) \tag{8.23}$$

here "r_s" is understood to be a variational parameter.

The Gaskell RPA form–which contains correct short/long range limits and minimizes the total energy of the electron gas within the RPA–is defined directly in k-space:

$$u_{RPA}(k) = -\frac{1}{2S_0(k)} + \frac{1}{2}\left(\frac{1}{S_0(k)^2} + \frac{4m_e v_k}{\hbar^2 k^2}\right)^{1/2} \tag{8.24}$$

where v_k is the Fourier transform of the Coulomb potential and $S_0(k)$ is the static structure factor of the non-interacting electron gas:

$$S_0(k) = \begin{cases} 1 & k > 2k_F \\ \frac{3k}{4k_F} - \frac{1}{2}\left(\frac{k}{2k_F}\right)^3 & k < 2k_F \end{cases}$$

When written in atomic units, RPA Jastrow implemented in QMCPACK has the form

$$u_{RPA}(k) = \frac{1}{2N_e}\left(-\frac{1}{S_0(k)} + \left(\frac{1}{S_0(k)^2} + \frac{12}{r_s^3 k^4}\right)^{1/2}\right) \tag{8.25}$$

Here "r_s" is again a variational parameter and $k_F \equiv (\frac{9\pi}{4r_s^3})^{1/3}$.

For both the Yukawa and Gaskell RPA Jastrows, the default value for r_s is $r_s = (\frac{3\Omega}{4\pi N_e})^{1/3}$.

`jastrow type=Two-Body function=rpa/yukawa` element				
parent elements:	`wavefunction`			
child elements:	`correlation`			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**Two-Body**		Must be Two-Body
`function`^{r}	text	**rpa/yukawa**		Must be rpa or yukawa
`name`^{r}	text	*anything*	RPA_Jee	Unique name for Jastrow
`longrange`^{o}	boolean	yes/no	yes	Use long-range part
`shortrange`^{o}	boolean	yes/no	yes	Use short-range part
parameters				
`rs`^{o}	rs	$r_s > 0$	$\frac{3\Omega}{4\pi N_e}$	Avg. elec-elec distance
`kc`^{o}	kc	$k_c > 0$	$2\left(\frac{9\pi}{4}\right)^{1/3}\frac{4\pi N_e}{3\Omega}$	K-space cutoff

Listing 8.13: Two body RPA Jastrow with long- and short-ranged parts.

```
<jastrow name=''Jee'' type=''Two-Body'' function=''rpa''>
</jastrow>
```

8.4.4 Three-body Jastrow functions

Explicit three-body correlations can be included in the wavefunction via the three-body Jastrow factor. The three-body electron-electron-ion correlation function ($u_{\sigma\sigma' I}$) currently used in QMCPACK is identical to the one proposed in [3]:

$$\begin{aligned} u_{\sigma\sigma' I}(r_{\sigma I}, r_{\sigma' I}, r_{\sigma\sigma'}) \quad &= \sum_{\ell=0}^{M_{eI}} \sum_{m=0}^{M_{eI}} \sum_{n=0}^{M_{ee}} \gamma_{\ell mn} r_{\sigma I}^{\ell} r_{\sigma' I}^{m} r_{\sigma\sigma'}^{n} \\ &\times \left(r_{\sigma I} - \tfrac{r_c}{2}\right)^3 \Theta\left(r_{\sigma I} - \tfrac{r_c}{2}\right) \\ &\times \left(r_{\sigma' I} - \tfrac{r_c}{2}\right)^3 \Theta\left(r_{\sigma' I} - \tfrac{r_c}{2}\right) \end{aligned} \tag{8.26}$$

Here M_{eI} and M_{ee} are the maximum polynomial orders of the electron-ion and electron-electron distances, respectively, $\{\gamma_{\ell mn}\}$ are the optimizable parameters (modulo constraints), r_c is a cutoff radius, and r_{ab} are the distances between electrons or ions a and b. i.e. The correlation function is only a function of the interparticle distances and not a more complex function of the particle positions, $\mathbf{r}$. As indicated by the Θ functions, correlations are set to zero beyond a distance of $r_c/2$ in either of the electron-ion distances and the largest meaningful electron-electron distance is r_c. This is the highest-order Jastrow correlation function currently implemented.

Today, solid state applications of QMCPACK usually utilize one and two-body B-spline Jastrow functions, with calculations on heavier elements often also using the three-body term described above.

Example use case Here is an example of H2O molecule. After optimizing one and two body Jastrow factors, add the following block in the wavefunction. The coefficients will be filled zero automatically if not given.

```
<jastrow name="J3" type="eeI" function="polynomial" source="ion0" print="yes">
  <correlation ispecies="O" especies="u" isize="3" esize="3" rcut="10">
    <coefficients id="uuO" type="Array" optimize="yes"> </coefficients>
  </correlation>
  <correlation ispecies="O" especies1="u" especies2="d" isize="3" esize="3" rcut="10">
    <coefficients id="udO" type="Array" optimize="yes"> </coefficients>
  </correlation>
  <correlation ispecies="H" especies="u" isize="3" esize="3" rcut="10">
    <coefficients id="uuH" type="Array" optimize="yes"> </coefficients>
  </correlation>
  <correlation ispecies="H" especies1="u" especies2="d" isize="3" esize="3" rcut="10">
    <coefficients id="udH" type="Array" optimize="yes"> </coefficients>
  </correlation>
</jastrow>
```

8.5 Multideterminant wavefunctions

Multiple schemes to generate a multideterminant wavefunction are possible, from CASSF to full CI or selected CI. The QMCPACK converter can convert MCSCF multideterminant wavefunctions from GAMESS[35] and CIPSI[27] wavefunctions from Quantum Package[28] (QP). Full details of how to run a CIPSI calculation and convert the wavefunction for QMCPACK are given in Sec. 14.1.1.

The script `utils/determinants_tools.py` can be used to generate useful information about the multideterminant wavefunction. This script takes, as a required argument, the path of an h5 file corresponding to the wavefunction. Used without optional arguments, it print the number of determinants, the number of CSF's and a histogram of the excitation degree.

```
> determinants_tools.py ./tests/molecules/C2_pp/C2.h5
Summary:
excitation degree 0 count: 1
excitation degree 1 count: 6
excitation degree 2 count: 148
excitation degree 3 count: 27
excitation degree 4 count: 20

n_det 202
```

```
n_csf 104
```

If the `--verbose` argument is used, the script will print each determinant, the associated CSF and the excitation degree relative to the first determinant.

```
> determinants_tools.py -v ./tests/molecules/C2_pp/C2.h5 | head
1
alpha  1111000000000000000000000000000000000000000000000000000000
beta   1111000000000000000000000000000000000000000000000000000000
scf    2222000000000000000000000000000000000000000000000000000000
excitation degree  0

2
alpha  1011100000000000000000000000000000000000000000000000000000
beta   1011100000000000000000000000000000000000000000000000000000
scf    2022200000000000000000000000000000000000000000000000000000
excitation degree  2
```

8.6 Backflow wavefunctions

One can perturb the nodal surface of a single-slater/multi-slater wavefunction through use of a backflow-transformation. Specifically, if we have an antisymmetric function $D(\mathbf{x}_{0\uparrow}, \cdots, \mathbf{x}_{N\uparrow}, \mathbf{x}_{0\downarrow}, \cdots, \mathbf{x}_{N\downarrow})$, and if i_α is the i-th particle of species type α, then the backflow transformation works by making the coordinate transformation $\mathbf{x}_{i_\alpha} \to \mathbf{x}'_{i_\alpha}$, and evaluating D at these new "quasiparticle" coordinates. QMCPACK currently supports quasiparticle transformations given by:

$$\mathbf{x}'_{i_\alpha} = \mathbf{x}_{i_\alpha} + \sum_{\alpha \leq \beta} \sum_{i_\alpha \neq j_\beta} \eta^{\alpha\beta}(|\mathbf{x}_{i_\alpha} - \mathbf{x}_{j_\beta}|)(\mathbf{x}_{i_\alpha} - \mathbf{x}_{j_\beta}) \tag{8.27}$$

Here, $\eta^{\alpha\beta}(|\mathbf{x}_{i_\alpha} - \mathbf{x}_{j_\beta}|)$ is a radially symmetric back flow transformation between species α and β. In QMCPACK, particle i_α is known as the "target" particle and j_β is known as the "source". The main types of transformations are so called one-body terms, which are between an electron and an ion $\eta^{eI}(|\mathbf{x}_{i_e} - \mathbf{x}_{j_I}|)$, and two-body terms. Two body terms are distinguished as those between like and opposite spin electrons: $\eta^{e(\uparrow)e(\uparrow)}(|\mathbf{x}_{i_e(\uparrow)} - \mathbf{x}_{j_e(\uparrow)}|)$ and $\eta^{e(\uparrow)e(\downarrow)}(|\mathbf{x}_{i_e(\uparrow)} - \mathbf{x}_{j_e(\downarrow)}|)$. Henceforth, we will assume that $\eta^{e(\uparrow)e(\uparrow)} = \eta^{e(\downarrow)e(\downarrow)}$.

In the following, we explain how to describe general terms like Eq. 8.27 in a QMCPACK XML file. For specificity, we will consider a particle set consisting of H and He (in that order). This ordering will be important when we build the XML file, so you can find this out either through your specific declaration of <particleset>, by looking at the hdf5 file in the case of plane waves, or by looking at the qmcpack output file in the section labelled "Summary of QMC systems".

8.6.1 Input Specifications

All backflow declarations occur within a single **<backflow> ... </backflow>** block. Backflow transformations occur in **<transformation>** blocks, and have the following input parameters

Just like one and two-body jastrows, parameterization of the backflow transformations are specified within the **<transformation>** blocks by **<correlation>** blocks. Please refer to 8.4.1 for more information.

Transformation element				
name	**datatype**	**values**	**defaults**	**description**
name	text		(required)	Unique name for this Jastrow function
type	text	"e-I"	(required)	Define a one-body backflow transformation.
		"e-e"		Define a two-body backflow transformation.
function	text	Bspline	(required)	B-spline type transformation. (No other types supported)
source	text			"e" if two-body, ion particle set if one-body.

8.6.2 Example Use Case

Having specified the general form, we present a general example of one-body and two-body backflow transformations in a hydrogen-helium mixture. The H and He ions have independent backflow transformations, as do the like and unlike-spin two-body terms. One caveat is in order: ionic backflow transformations must be listed in the order that they appear in the particle set. If in our example, He is listed first, and H is listed second, the following example would be correct. However, switching backflow declaration to H first, then He, will result in an error. Outside of this, declaration of one-body blocks and two-body blocks aren't sensitive to ordering.

```
<backflow>
<!--The One-Body term with independent e-He and e-H terms. IN THAT ORDER -->
<transformation name="eIonB" type="e-I" function="Bspline" source="ion0">
    <correlation cusp="0.0" size="8" type="shortrange" init="no" elementType="He"
    rcut="3.0">
        <coefficients id="eHeC" type="Array" optimize="yes">
            0 0 0 0 0 0 0 0
        </coefficients>
    </correlation>
    <correlation cusp="0.0" size="8" type="shortrange" init="no" elementType="H"
    rcut="3.0">
        <coefficients id="eHC" type="Array" optimize="yes">
            0 0 0 0 0 0 0 0
        </coefficients>
    </correlation>
</transformation>

<!--The Two-Body Term with Like and Unlike Spins -->
<transformation name="eeB" type="e-e" function="Bspline" >
    <correlation cusp="0.0" size="7" type="shortrange" init="no" speciesA="u"
    speciesB="u" rcut="1.2">
        <coefficients id="uuB1" type="Array" optimize="yes">
            0 0 0 0 0 0 0
        </coefficients>
    </correlation>
    <correlation cusp="0.0" size="7" type="shortrange" init="no" speciesA="d"
    speciesB="u" rcut="1.2">
        <coefficients id="udB1" type="Array" optimize="yes">
            0 0 0 0 0 0 0
        </coefficients>
    </correlation>
</transformation>
</backflow>
```

Currently, backflow only works with single-slater determinant wavefunctions. When a backflow

transformation has been declared, it should be placed within the **<determinantset>** block, but outside of the **<slaterdeterminant>** blocks, like so:

```
<determinantset ... >
    <!--basis set declarations go here, if there are any -->

    <backflow>
        <transformation ...>
          <!--Here is where one and two-body terms are defined -->
         </transformation>
     </backflow>

     <slaterdeterminant>
         <!--Usual determinant definitions -->
     </slaterdeterminant>
 </determinantset>
```

8.6.3 Optimization Tips

Backflow is notoriously difficult to optimize–its extremely nonlinear in the variational parameters and also moves the nodal surface around. As such, it is likely that a full Jastrow+Backflow optimization with all parameters initialized to zero may not converge in a reasonable time. If you are experiencing this problem, the following pointers are suggested (in no particular order):

Get a good starting guess for Ψ_T

1. Try optimizing the Jastrow first without backflow.
2. Freeze the jastrow parameters, introduce only the e-e terms in the backflow transformation, and optimize these parameters.
3. Freeze the e-e backflow parameters, and now optimize the e-I terms.
    - If difficulty is encountered here, try optimizing each species independently.
4. Now, unfreeze all jastrow, e-e backflow, and e-I backflow parameters, and reoptimize.

Optimizing Backflow Terms

It is wholly possible that the above prescription might grind to a halt in steps 2 or 3 with the inability to optimize the e-e or e-I backflow transformation independently, especially if it is initialized to zero. To get around this, one suggestion is to build a good starting guess for the e-e or e-I backflow terms iteratively as follows.

1. Start off with a small number of knots initialized to zero. Set r_{cut} to be small (much smaller than an interatomic distance).
2. Optimize the backflow function.
3. If this works, slowly increase r_{cut} and/or the number of knots.
4. Repeat steps 2-3 until there is no noticeable change in energy or variance of Ψ_T.

Tweaking the Optimization Run

The following modifications are worth a try in the optimization block:

- Try setting "useDrift" to "no". This eliminates the use of wavefunction gradients and force biasing in the VMC algorithm. This could be an issue for poorly optimized wavefunctions with pathological gradients.
- Try increasing "exp0" in the optimization block. Larger values of "exp0" cause the search directions to more closely follow those predicted by steepest-descent than those by the linear method.

Note that the new adaptive shift optimizer has not yet been tried with backflow wavefunctions. It should perform better than the older optimizers, but a considered optimization process is still recommended.

8.7 Finite-difference linear response wave functions

The finite-difference linear response wave function (FDLR) is an experimental wave function type, described in detail in Ref.[24]. In this method, the wave function is formed as the linear response of some existing trial wave function in QMCPACK. This derivatives of this linear response are approximated by a simple finite-difference.

Forming a wave function within the linear response space of an existing ansatz can be very powerful. For example, a configuration interaction singles (CIS) wave function can be formed as a linear combination of the first derivatives of a Slater determinant (with respect to its orbital rotation parameters). Thus, in this sense, CIS is the linear response of Hartree–Fock theory.

Forming a CIS wave function as the linear response of an optimizable Slater determinant is where all testing of this wave function has been performed. In theory the implementation is flexible and it can be used with other trial wave functions in QMCPACK, but this has not been tested. The FDLR trial wave function is experimental.

Mathematically, the FDLR wave function has the form:

$$\Psi_{\mathrm{FDLR}}(\mu, \mathbf{X}) = \Psi(\mathbf{X} + \mu) - \Psi(\mathbf{X} - \mu), \tag{8.28}$$

where $\Psi(\mathbf{P})$ is some trial wave function in QMCPACK, and $\mathbf{P}$ are its optimizable parameters. $\mathbf{X}$ are the "base" parameters about which the finite difference is performed (for example, an overall orbital rotation). μ are the "finite difference" parameters, which define the direction of the derivative, and whose magnitude determines the magnitude of the finite-difference. In the limit that the magnitude of **mu** goes to 0, the Ψ_{FDLR} object defined above becomes equivalent to

$$\Psi_{\mathrm{FDLR}}(\mu, \mathbf{X}) = \sum_{pq} \mu_{pq} \frac{\partial \Psi_{\mathrm{det}}(\mathbf{X})}{\partial X_{pq}}, \tag{8.29}$$

which is the desired linear response wave function that we are approximating. In the case that $\Psi(\mathbf{P})$ is a determinant with orbital rotation parameters $\mathbf{P}$, the above is a CIS wave function with CIS expansion coefficients μ and orbital rotation $\mathbf{X}$.

8.7.1 Input Specifications

An FDLR wave function is specified within a `<fdlr> ... </fdlr>` block.

To fully specify an FDLR wave function as above, we require the initial parameters for both $\mathbf{X}$ and μ to be input. This therefore requires two trial wave functions to be provided on input. Each of these is best specified in its own XML file. The names of these two files are provided in an `<include>` tag via `<include wfn_x_href=`` ... '' wfn_d_href=`` ... ''>`. `wfn_x_href` specifies the file that will hold the $\mathbf{X}$ parameters. `wfn_d_href` specifies the file that will hold the μ parameters.

Other options inside the `<include>` tag are `opt_x` and `opt_d`, which specify whether or not $\mathbf{X}$ and μ parameters are optimizable, respectively.

8.7.2 Example Use Case

```
<fdlrwfn name="FDLR">
  <include wfn_x_href="h2.wfn_x.xml" wfn_d_href="h2.wfn_d.xml" opt_x="yes"
    opt_d="yes"/>
</fdlrwfn>
```

with the `h2.wfn_x.xml` file containing one of the wave functions and corresponding set of $\mathbf{X}$ parameters, such as:

```
<?xml version="1.0"?>
<wfn_x>
    <determinantset name="LCAOBSet" type="MolecularOrbital" transform="yes"
    source="ion0">
      <basisset name="LCAOBSet">
        <atomicBasisSet name="Gaussian-G2" angular="cartesian" type="Gaussian"
    elementType="H" normalized="no">
          <grid type="log" ri="1.e-6" rf="1.e2" npts="1001"/>
          <basisGroup rid="H00" n="0" l="0" type="Gaussian">
            <radfunc exponent="1.923840000000e+01" contraction="3.282799101900e-02"/>
            <radfunc exponent="2.898720000000e+00" contraction="2.312039367510e-01"/>
            <radfunc exponent="6.534720000000e-01" contraction="8.172257764360e-01"/>
          </basisGroup>
          <basisGroup rid="H10" n="1" l="0" type="Gaussian">
            <radfunc exponent="1.630642000000e-01" contraction="1.000000000000e+00"/>
          </basisGroup>
        </atomicBasisSet>
      </basisset>

    <slaterdeterminant optimize="yes">
      <determinant id="det_up" sposet="spo-up">
        <opt_vars size="3">
          0.0 0.0 0.0
        </opt_vars>
      </determinant>

      <determinant id="det_down" sposet="spo-dn">
        <opt_vars size="3">
          0.0 0.0 0.0
        </opt_vars>
      </determinant>
    </slaterdeterminant>

      <sposet basisset="LCAOBSet" name="spo-up" size="4" optimize="yes">
```

```
        <occupation mode="ground"/>
        <coefficient size="4" id="updetC">
  2.83630000000000e-01  3.35683000000000e-01  2.83630000000000e-01
    3.35683000000000e-01
  1.66206000000000e-01  1.22367400000000e+00 -1.66206000000000e-01
    -1.22367400000000e+00
  8.68279000000000e-01 -6.95081000000000e-01  8.68279000000000e-01
    -6.95081000000000e-01
 -9.77898000000000e-01  1.19682400000000e+00  9.77898000000000e-01
    -1.19682400000000e+00
</coefficient>
      </sposet>
      <sposet basisset="LCAOBSet" name="spo-dn" size="4" optimize="yes">
        <occupation mode="ground"/>
        <coefficient size="4" id="downdetC">
  2.83630000000000e-01  3.35683000000000e-01  2.83630000000000e-01
    3.35683000000000e-01
  1.66206000000000e-01  1.22367400000000e+00 -1.66206000000000e-01
    -1.22367400000000e+00
  8.68279000000000e-01 -6.95081000000000e-01  8.68279000000000e-01
    -6.95081000000000e-01
 -9.77898000000000e-01  1.19682400000000e+00  9.77898000000000e-01
    -1.19682400000000e+00
</coefficient>
      </sposet>

    </determinantset>
</wfn_x>
```

and similarly for the `h2.wfn_d.xml` file, which will hold the initial μ parameters.

The above is a wave function file for an optimizable determinant wave function for H_2, in a double zeta valence basis set. Thus, the FDLR wave function here would perform CIS on H_2 in a double zeta basis set.

8.8 Gaussian Product Wavefunction

The Gaussian Product wavefunction implements eq. 8.30

$$\Psi(\vec{R}) = \prod_{i=1}^{N} \exp\left[-\frac{(\vec{R}_i - \vec{R}_i^o)^2}{2\sigma_i^2}\right], \tag{8.30}$$

where $\vec{R}_i$ is the position of the i^{th} quantum particle and $\vec{R}_i^o$ is its center. σ_i is the width of the Gaussian orbital around center i.

This variational wavefunction enhances single-particle density at chosen spatial locations with adjustable strengths. It is useful whenever such localization is physically relevant yet not captured by other parts of the trial wavefunction. For example, in an electron-ion simulation of a solid, the ions are localized around their crystal lattice sites. This single-particle localization is not captured by the ion-ion Jastrow. Therefore the addition of this localization term will improve the wavefunction. The simplest use case of this wavefunction is perhaps the quantum harmonic oscillator (please see the "tests/models/sho" folder for examples).

Input Specification

Gaussian Product Wavefunction (ionwf)				
name	**datatype**	**values**	**defaults**	**description**
name	text	ionwf	(required)	Unique name for this wavefunction
width	floats	1.0 -1	(required)	Widths of Gaussian orbitals.
source	text	ion0	(required)	Name of classical particle set.

Additional information:

- `width` There must be one width provided for each quantum particle. If a negative width is given, then its corresponding Gaussian orbital is removed. Negative width is useful if one wants to use Gaussian wavefunction for a subset of the quantum particles.
- `source` The Gaussian centers must be specified in the form of a classical particle set. This classical particle set is likely the ion positions "ion0", hence the name "ionwf". However, one may define arbitrary centers using a different particle set. Please refer to examples in 'tests/models/sho'.

8.8.1 Example Use Case

```
<qmcsystem>
  <simulationcell>
    <parameter name="bconds">
          n n n
    </parameter>
  </simulationcell>
  <particleset name="e">
    <group name="u" size="1">
      <parameter name="mass">5.0</parameter>
      <attrib name="position" datatype="posArray" condition="0">
        0.0001 -0.0001 0.0002
      </attrib>
    </group>
  </particleset>
  <particleset name="ion0" size="1">
    <group name="H">
      <attrib name="position" datatype="posArray" condition="0">
        0 0 0
      </attrib>
    </group>
  </particleset>
  <wavefunction target="e" id="psi0">
    <ionwf name="iwf" source="ion0" width="0.8165"/>
  </wavefunction>
  <hamiltonian name="h0" type="generic" target="e">
    <extpot type="HarmonicExt" mass="5.0" energy="0.3"/>
    <estimator type="latticedeviation" name="latdev"
      target="e"    tgroup="u"
      source="ion0" sgroup="H"/>
  </hamiltonian>
</qmcsystem>
```

Chapter 9

Hamiltonian and Observables

QMCPACK is capable of the simultaneous measurement of the Hamiltonian and many other quantum operators. The Hamiltonian attains a special status among the available operators (also referred to as observables) because it ultimately generates all available information regarding the quantum system. This is evident from an algorithmic standpoint as well since the Hamiltonian (embodied in the the projector) generates the imaginary time dynamics of the walkers in DMC and RMC.

This section covers how the Hamiltonian can be specified, component by component, by the user in the XML format native to QMCPACK . It also covers the input structure of statistical estimators corresponding to quantum observables such as the density, the static structure factor, and forces.

9.1 The Hamiltonian

The many-body Hamiltonian in Hartree units is given by

$$\hat{H} = -\sum_i \frac{1}{2m_i}\nabla_i^2 + \sum_i v^{ext}(r_i) + \sum_{i<j} v^{qq}(r_i, r_j) + \sum_{i\ell} v^{qc}(r_i, r_\ell) + \sum_{\ell<m} v^{cc}(r_\ell, r_m). \tag{9.1}$$

Here, the sums indexed by i/j are over quantum particles, while ℓ/m are reserved for classical particles. Often the quantum particles are electrons and the classical particles are ions, though QMCPACK is not limited in this way. The mass of each quantum particle is denoted m_i, $v^{qq}/v^{qc}/v^{cc}$ are pair potentials between quantum-quantum/quantum-classical/classical-classical particles, and v^{ext} denotes a purely external potential.

QMCPACK is designed modularly so that any potential can be supported with minimal additions to the code base. Potentials currently supported include Coulomb interactions in open and periodic boundary conditions, the modified periodic coulomb (MPC) potential, non-local pseudopotentials, helium pair potentials, and various model potentials such as hard sphere, gaussian, and modified Poschl-Teller.

Reference information and examples for the `<hamiltonian/>` XML element is provided below. Detailed descriptions of the input for individual potentials is given in the sections that follow.

`hamiltonian` element				
parent elements:	`simulation, qmcsystem`			
child elements:	`pairpot extpot estimator constant`(deprecated)			
attributes				
name	**datatype**	**values**	**default**	**description**
`name/id`^{o}	text	*anything*	h0	Unique id for this Hamiltonian instance
`type`^{o}	text		generic	*No current function*
`role`^{o}	text	primary/extra	extra	Designate as primary Hamiltonian or not
`source`^{o}	text	`particleset.name`	i	Identify classical particleset
`target`^{o}	text	`particleset.name`	e	Identify quantum particleset
`default`^{o}	boolean	yes/no	yes	Include kinetic energy term implicitly

Additional information:

- **target:** Must be set to the name of the quantum particeset. The default value is typically sufficient. In normal usage, no other attributes are provided.

Listing 9.1: All electron Hamiltonian XML element.

```
<hamiltonian target="e">
  <pairpot name="ElecElec" type="coulomb" source="e" target="e"/>
  <pairpot name="ElecIon"  type="coulomb" source="i" target="e"/>
  <pairpot name="IonIon"   type="coulomb" source="i" target="i"/>
</hamiltonian>
```

Listing 9.2: Pseudopotential Hamiltonian XML element.

```
<hamiltonian target="e">
  <pairpot name="ElecElec"  type="coulomb" source="e" target="e"/>
  <pairpot name="PseudoPot" type="pseudo"  source="i" wavefunction="psi0"
    format="xml">
    <pseudo elementType="Li" href="Li.xml"/>
    <pseudo elementType="H" href="H.xml"/>
  </pairpot>
  <pairpot name="IonIon"    type="coulomb" source="i" target="i"/>
</hamiltonian>
```

9.2 Pair potentials

Many pair potentials are supported. Though only the most commonly used pair potentials are covered in detail in this section, all currently available potentials are listed briefly below. If a potential you desire is not covered below, or is not present at all, feel free to contact the developers.

`pairpot` factory element				
parent elements:	`hamiltonian`			
type selector:	`type` attribute			
type options:	coulomb		Coulomb/Ewald potential	
	pseudo		Semilocal pseudopotential	
	mpc		Modified Periodic Coulomb interaction/correction	
	cpp		Core polarization potential	
	skpot		*Unknown*	
shared attributes:				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	*See above*	0	Select pairpot type
`name`^{r}	text	*anything*	any	Unique name for this pairpot
`source`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`target`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`units`^{o}	text		hartree	*No current function*

Additional information:

- **type:** Used to select the desired pair potential. Must be selected from the list of type options above.
- **name:** A unique name used to identify this pair potential. Block averaged output data will appear under this name in `scalar.dat` and/or `stat.h5` files.
- **source/target:** These specify the particles involved in a pair interaction. If an interaction is between classical (e.g. ions) and quantum (e.g. electrons), `source/target` should be the name of the classical/quantum particleset.
- Only `coulomb, pseudo, mpc` are described in detail below. The older or less used types (`cpp, skpot`) are not covered.
- Available only if `QMC_CUDA` is not defined: `skpot`.
- Available only if `OHMMS_DIM==3`: `mpc, vhxc, pseudo`.
- Available only if `OHMMS_DIM==3` and `QMC_CUDA` is not defined: `cpp`.

9.2.1 Coulomb potentials

The bare Coulomb potential is used in open boundary conditions:

$$V_c^{open} = \sum_{i<j} \frac{q_i q_j}{|r_i - r_j|} \tag{9.2}$$

When periodic boundary conditions are selected, Ewald summation is used automatically:

$$V_c^{pbc} = \sum_{i<j} \frac{q_i q_j}{|r_i - r_j|} + \frac{1}{2} \sum_{L \neq 0} \sum_{i,j} \frac{q_i q_j}{|r_i - r_j + L|} \tag{9.3}$$

The sum indexed by L is over all non-zero simulation cell lattice vectors. In practice, the Ewald sum is broken into short and long ranged parts in a manner optimized for efficiency (see Ref. [10]) for details.

For information on how to set the boundary conditions, consult Sec. 7.1.

`pairpot type=coulomb` element				
parent elements:	`hamiltonian`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**coulomb**		Must be coulomb
`name/id`^{r}	text	*anything*	ElecElec	Unique name for interaction
`source`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`target`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`pbc`^{o}	boolean	yes/no	yes	Use Ewald summation
`physical`^{o}	boolean	yes/no	yes	Hamiltonian(yes)/observable(no)
`forces`	boolean	yes/no	no	*Deprecated*

Additional information

- **type/source/target** See description for the generic `pairpot` factory element above.
- **name:** Traditional user-specified names for electron-electron, electron-ion, and ion-ion terms are `ElecElec`, `ElecIon`, and `IonIon`, respectively. While any choice can be used, the data analysis tools expect to find columns in `*.scalar.dat` with these names.
- **pbc**: Ewald summation will not be performed if `simulationcell.bconds== n n n`, regardless of the value of `pbc`. Similarly, the `pbc` attribute can only be used to turn off Ewald summation if `simulationcell.bconds!= n n n`. The default value is recommended.
- **physical**: If `physical==yes`, this pair potential is included in the Hamiltonian and will factor into the `LocalEnergy` reported by QMCPACK and also in the DMC branching weight. If `physical==no`, then the pair potential is treated as a passive observable but not as part of the Hamiltonian itself. As such it does not contribute to the outputted `LocalEnergy`. Regardless of the value of `physical` output data will appear in `scalar.dat` in a column headed by `name`.

Listing 9.3: QMCPXML element for Coulomb interaction between electrons.

```
<pairpot name="ElecElec" type="coulomb" source="e" target="e"/>
```

Listing 9.4: QMCPXML element for Coulomb interaction between electrons and ions (all-electron only).

```
<pairpot name="ElecIon"  type="coulomb" source="i" target="e"/>
```

Listing 9.5: QMCPXML element for Coulomb interaction between ions.

```
<pairpot name="IonIon"   type="coulomb" source="i" target="i"/>
```

9.2.2 Pseudopotentials

QMCPACK supports pseudopotentials in semilocal form, which is local in the radial coordinate and non-local in angular coordinates. When all angular momentum channels above a certain threshold (ℓ_{max}) are well approximated by the same potential ($V_{\bar{\ell}} \equiv V_{loc}$), the pseudpotential separates into a fully local channel and an angularly-nonlocal component:

$$V^{PP} = \sum_{ij} \left(V_{\bar{\ell}}(|r_i - \tilde{r}_j|) + \sum_{\ell \neq \bar{\ell}}^{\ell_{max}} \sum_{m=-\ell}^{\ell} |Y_{\ell m}\rangle \left[V_\ell(|r_i - \tilde{r}_j|) - V_{\bar{\ell}}(|r_i - \tilde{r}_j|) \right] \langle Y_{\ell m}| \right) \tag{9.4}$$

Here the electron/ion index is i/j and only one type of ion is shown for simplicity.

Evaluation of the localized pseudopotential energy $\Psi_T^{-1} V^{PP} \Psi_T$ requires additional angular integrals. These integrals are evaluated on a randomly shifted angular grid. The size of this grid is determined by ℓ_{max}. See Ref. [11] for further detail.

QMCPACK uses the FSAtom pseudopotential file format associated with the "Free Software Project for Atomic-scale Simulations" initiated in 2002 (see http://www.tddft.org/fsatom/manifest.php for general information). The FSAtom format uses XML for structured data. Files in this format do not use a specific identifying file extension; they are simply suffixed with ".xml". The tabular data format of CASINO is also supported.

pairpot type=pseudo element				
parent elements:	hamiltonian			
child elements:	pseudo			
attributes				
name	**datatype**	**values**	**default**	**description**
type^r	text	**pseudo**		Must be pseudo
name/id^r	text	*anything*	PseudoPot	*No current function*
source^r	text	particleset.name	i	Ion particleset name
target^r	text	particleset.name	hamiltonian.target	Electron particleset name
pbc^o	boolean	yes/no	yes*	Use Ewald summation
forces	boolean	yes/no	no	*Deprecated*
wavefunction^r	text	wavefunction.name	invalid	Identify wavefunction
format^r	text	xml/table	table	Select file format
algorithm^o	text	batched/default	default	Choose NLPP algorithm

Additional information:

- **type/source/target** See description for the generic pairpot factory element above.
- **name:** Ignored. Instead default names will be present in *scalar.dat output files when pseudopotentials are used. The field LocalECP refers to the local part of the pseudopotential. If non-local channels are present, a NonLocalECP field will be added that contains the non-local energy summed over all angular momentum channels.
- **pbc:** Ewald summation will not be performed if simulationcell.bconds== n n n, regardless of the value of pbc. Similarly, the pbc attribute can only be used to turn off Ewald summation if simulationcell.bconds!= n n n.
- **format:** If format==table, QMCPACK looks for *.psf files containing pseudopotential data in a tabular format. The files must be named after the ionic species provided in particleset (*e.g.* Li.psf and H.psf). If format==xml, additional pseudo child XML elements must be provided (see below). These elements specify individual file names and formats (both the FSAtom XML and CASINO tabular data formats are supported).
- **algorithm** The default algorithm evaluates the ratios of wavefunction components together for each qudarature point and then one point after another. The batched algorithm evaluates the ratios of qudarature points together for each wavefunction component and then one component after another. Internally, it uses VirtualParticleSet for quadrature points. Hybrid orbital representation has an extra optimization enabled when using the batched algorithm.

Listing 9.6: QMCPXML element for pseudopotential electron-ion interaction (psf files).

```
<pairpot name="PseudoPot" type="pseudo"  source="i" wavefunction="psi0"
  format="psf"/>
```

Listing 9.7: QMCPXML element for pseudopotential electron-ion interaction (xml files).

```
<pairpot name="PseudoPot" type="pseudo"  source="i" wavefunction="psi0"
  format="xml">
   <pseudo elementType="Li" href="Li.xml"/>
   <pseudo elementType="H" href="H.xml"/>
</pairpot>
```

Details of `<pseudo/>` input elements are given below. It is possible to include (or construct) a full pseudopotential directly in the input file without providing an external file via `href`. The full XML format for pseudopotentials is not yet covered.

`pseudo` element				
parent elements:	`pairpot type=pseudo`			
child elements:	`header local grid`			
attributes				
name	**datatype**	**values**	**default**	**description**
`elementType/symbol`^{r}	text	`group.name`	none	Identify ionic species
`href`^{r}	text	*filepath*	none	Pseudopotential file path
`format`^{r}	text	xml/casino	xml	Specify file format
`cutoff`^{o}	real			Non-local cutoff radius
`lmax`^{o}	integer			Largest angular momentum
`nrule`^{o}	integer			Integration grid order

Listing 9.8: QMCPXML element for pseudopotential of single ionic species.

```
<pseudo elementType="Li" href="Li.xml"/>
```

9.2.3 Modified periodic Coulomb interaction/correction

The modified periodic Coulomb (MPC) interaction is an alternative to direct Ewald summation. The MPC corrects the exchange correlation hole to more closely match its thermodynamic limit. Because of this, the MPC exhibits smaller finite size errors than the bare Ewald interaction, though a few alternative and competitive finite size correction schemes now exist. The MPC is itself often used just as a finite size correction in postprocessing (set `physical=false` in the input).

`pairpot type=mpc` element				
parent elements:	`hamiltonian`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**mpc**		Must be mpc
`name/id`^{r}	text	*anything*	MPC	Unique name for interaction
`source`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`target`^{r}	text	`particleset.name`	`hamiltonian.target`	Identify interacting particles
`physical`^{o}	boolean	yes/no	no	Hamiltonian(yes)/observable(no)
`cutoff`	real	> 0	30.0	Kinetic energy cutoff

Remarks

- `physical`: Typically set to `no`, meaning the standard Ewald interaction will be used during sampling and MPC will be measured as an observable for finite-size post correction. If `physical` is `yes`, the MPC interaction will be used during sampling. In this case an electron-electron Coulomb `pairpot` element should not be supplied.
- Developer note: Currently the `name` attribute for the mpc interaction is ignored. The name is always reset to `MPC`.

Listing 9.9: Modified periodic coulomb for finite size post-correction.

```
<pairpot type="MPC" name="MPC" source="e" target="e" ecut="60.0" physical="no"/>
```

9.3 General estimators

A broad range of estimators for physical observables are available in QMCPACK . The sections below contain input details for the total number density (`density`), number density resolved by particle spin (`spindensity`), spherically averaged pair correlation function (`gofr`), static structure factor (`sk`), energy density (`energydensity`), one body reduced density matrix (`dm1b`), $S(k)$ based kinetic energy correction (`chiesa`), forward walking (`ForwardWalking`), and force (`Force`) estimators. Other estimators are not yet covered.

When an `<estimator/>` element appears in `<hamiltonian/>`, it is evaluated for all applicable chained QMC runs (*e.g.* VMC→DMC→DMC). Estimators are generally not accumulated during wavefunction optimization sections. If an `<estimator/>` element is instead provided in a particular `<qmc/>` element, that estimator is only evaluated for that specific section (*e.g.* during VMC only).

`estimator` factory element		
parent elements:	`hamiltonian, qmc`	
type selector:	`type` attribute	
type options:	density	Density on a grid
	spindensity	Spin density on a grid
	gofr	Pair correlation function (quantum species)
	sk	Static structure factor
	structurefactor	Species resolved structure factor
	specieskinetic	Species resolved kinetic energy
	latticedeviation	Spatial deviation between two particlesets
	momentum	Momentum distribution
	energydensity	Energy density on uniform or Voronoi grid
	dm1b	One body density matrix in arbitrary basis
	chiesa	Chiesa-Ceperley-Martin-Holzmann kinetic energy correction
	Force	Family of "force" estimators (see 9.5)
	ForwardWalking	Forward walking values for existing estimators
	orbitalimages	Create image files for orbitals, then exit
	flux	Checks sampling of kinetic energy
	localmoment	Atomic spin polarization within cutoff radius
	Pressure	*No current function*

shared attributes:

name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	*See above*	0	Select estimator type
`name`^{r}	text	*anything*	any	Unique name for this estimator

9.3.1 Chiesa-Ceperley-Martin-Holzmann kinetic energy correction

This estimator calculates a finite size correction to the kinetic energy following the formalism laid out in Ref. [12]. The total energy can be corrected for finite size effects by using this estimator in conjunction with the MPC correction.

`estimator type=chiesa` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**chiesa**		Must be chiesa
`name`^{o}	text	*anything*	KEcorr	Always reset to KEcorr
`source`^{o}	text	`particleset.name`	e	Identify quantum particles
`psi`^{o}	text	`wavefunction.name`	psi0	Identify wavefunction

Listing 9.10: "Chiesa" kinetic energy finite size post-correction.

```
<estimator name="KEcorr" type="chiesa" source="e" psi="psi0"/>
```

9.3.2 Density estimator

The particle number density operator is given by

$$\hat{n}_r = \sum_i \delta(r - r_i) \tag{9.5}$$

The `density` estimator accumulates the number density on a uniform histogram grid over the simulation cell. The value obtained for a grid cell c with volume Ω_c is then the average number of particles in that cell:

$$n_c = \int dR |\Psi|^2 \int_{\Omega_c} dr \sum_i \delta(r - r_i) \tag{9.6}$$

`estimator type=density` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**density**		Must be density
`name`^{r}	text	*anything*	any	Unique name for estimator
`delta`^{o}	real array(3)	$0 \leq v_i \leq 1$	0.1 0.1 0.1	Grid cell spacing, unit coords
`x_min`^{o}	real	> 0	0	Grid starting point in x (Bohr)
`x_max`^{o}	real	> 0	|`lattice[0]`|	Grid ending point in x (Bohr)
`y_min`^{o}	real	> 0	0	Grid starting point in y (Bohr)
`y_max`^{o}	real	> 0	|`lattice[1]`|	Grid ending point in y (Bohr)
`z_min`^{o}	real	> 0	0	Grid starting point in z (Bohr)
`z_max`^{o}	real	> 0	|`lattice[2]`|	Grid ending point in z (Bohr)
`potential`^{o}	boolean	yes/no	no	Accumulate local potential, *Deprecated*
`debug`^{o}	boolean	yes/no	no	*No current function*

Additional information:

- `name`: The name provided will be used as a label in the `stat.h5` file for the blocked output data. Post-processing tools expect `name="Density"`.
- `delta`: This sets the histogram grid size used to accumulate the density: `delta="0.1 0.1 0.05"`$\rightarrow 10 \times 10 \times 20$ grid, `delta="0.01 0.01 0.01"`$\rightarrow 100 \times 100 \times 100$ grid. The density grid is written to a `stat.h5` file at the end of each Monte Carlo block. If you request many *blocks* in a `<qmc/>` element, or select a large grid, the resulting `stat.h5` file may be many GB in size.
- `*_min/*_max`: Can be used to select a subset of the simulation cell for the density histogram grid. For example if a (cubic) simulation cell is 20 Bohr on a side, setting `*_min=5.0` and `*_max=15.0` will result in a density histogram grid spanning a $10 \times 10 \times 10$ Bohr cube about the center of the box. Use of `x_min, x_max, y_min, y_max, z_min, z_max` is only appropriate for orthorhombic simulation cells with open boundary conditions.

- When open boundary conditions are used, a `<simulationcell/>` element must be explicitly provided as the first sub-element of `<qmcsystem/>` for the density estimator to work. In this case the molecule should be centered around the middle of the simulation cell ($L/2$) and not the origin (0 since the space within the cell, and hence the density grid, is defined from 0 to L.

Listing 9.11: Density estimator (uniform grid).

```
<estimator name="Density" type="density" delta="0.05 0.05 0.05"/>
```

9.3.3 Spin density estimator

The spin density is similar to the total density described above. In this case, the sum over particles is performed independently for each spin component.

`estimator type=spindensity` element

parent elements: `hamiltonian, qmc`
child elements: *None*
attributes

name	datatype	values	default	description
`type`^{r}	text	**spindensity**		Must be spindensity
`name`^{r}	text	*anything*	any	Unique name for estimator
`report`^{o}	boolean	yes/no	no	Write setup details to std-out

parameters

name	datatype	values	default	description
`grid`^{o}	integer array(3)	$v_i > 0$		Grid cell count
`dr`^{o}	real array(3)	$v_i > 0$		Grid cell spacing (Bohr)
`cell`^{o}	real array(3,3)	*anything*		Volume grid exists in
`corner`^{o}	real array(3)	*anything*		Volume corner location
`center`^{o}	real array(3)	*anything*		Volume center/origin location
`voronoi`^{o}	text	`particleset.name`		*Under development*
`test_moves`^{o}	integer	$>= 0$	0	Test estimator with random moves

Additional information:

- `name`: The name provided will be used as a label in the `stat.h5` file for the blocked output data. Post-processing tools expect `name="SpinDensity"`.
- `grid`: Sets the dimension of the histogram grid. Input like `<parameter name="grid"> 40 40 40 </parameter>` requests a $40 \times 40 \times 40$ grid. The shape of individual grid cells is commensurate with the supercell shape.
- `dr`: Real space dimensions of grid cell edges (Bohr units). Input like `<parameter name="dr"> 0.5 0.5 0.5 </parameter>` in a supercell with axes of length 10 Bohr each (but of arbitrary

shape) will produce a $20 \times 20 \times 20$ grid. The inputted `dr` values are rounded to produce an integer number of grid cells along each supercell axis. Either `grid` or `dr` must be provided, but not both.

- `cell`: When `cell` is provided, a user defined grid volume is used instead of the global supercell. This must be provided if open boundary conditions are used. Additionally, if `cell` is provided, the user must specify where the volume is located in space in addition to its size/shape (`cell`) using either the `corner` or `center` parameters.
- `corner`:The grid volume is defined as $corner + \sum_{d=1}^{3} u_d cell_d$ with $0 < u_d < 1$ ("cell" refers to either the supercell or user provided cell).
- `center`:The grid volume is defined as $center + \sum_{d=1}^{3} u_d cell_d$ with $-1/2 < u_d < 1/2$ ("cell" refers to either the supercell or user provided cell). `corner/center` can be used to shift the grid even if `cell` is not specified. Simultaneous use of `corner` and `center` will cause QMCPACK to abort.

Listing 9.12: Spin density estimator (uniform grid).

```
<estimator type="spindensity" name="SpinDensity" report="yes">
  <parameter name="grid"> 40 40 40 </parameter>
</estimator>
```

Listing 9.13: Spin density estimator (uniform grid centered about origin).

```
<estimator type="spindensity" name="SpinDensity" report="yes">
  <parameter name="grid">
    20 20 20
  </parameter>
  <parameter name="center">
    0.0 0.0 0.0
  </parameter>
  <parameter name="cell">
    10.0  0.0  0.0
     0.0 10.0  0.0
     0.0  0.0 10.0
  </parameter>
</estimator>
```

9.3.4 Pair correlation function, $g(r)$

The functional form of the species resolved radial pair correlation function operator is

$$g_{ss'}(r) = \frac{V}{4\pi r^2 N_s N_{s'}} \sum_{i_s=1}^{N_s} \sum_{j_{s'}=1}^{N_{s'}} \delta(r - |r_{i_s} - r_{j_{s'}}|). \tag{9.7}$$

Here N_s is the number of particles of species s and V is the supercell volume. If $s = s'$, then the sum is restricted so that $i_s \neq j_s$.

In QMCPACK, an estimate of $g_{ss'}(r)$ is obtained as a radial histogram with a set of N_b uniform bins of width δr. This can be expressed analytically as

$$\tilde{g}_{ss'}(r) = \frac{V}{4\pi r^2 N_s N_{s'}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_{s'}} \frac{1}{\delta r} \int_{r-\delta r/2}^{r+\delta r/2} dr' \delta(r' - |r_{si} - r_{s'j}|), \tag{9.8}$$

where the radial coordinate r is restricted to reside at the bin centers, $\delta r/2, 3\delta r/2, 5\delta r/2, \ldots$.

`estimator type=gofr` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**gofr**		Must be gofr
`name`^{o}	text	*anything*	any	*No current function*
`num_bin`^{r}	integer	> 1	20	# of histogram bins
`rmax`^{o}	real	> 0	10	Histogram extent (Bohr)
`dr`^{o}	real	> 0	0.5	*No current function*
`debug`^{o}	boolean	yes/no	no	*No current function*
`target`^{o}	text	`particleset.name`	`hamiltonian.target`	Quantum particles
`source/sources`^{o}	text array	`particleset.name`	`hamiltonian.target`	Classical particles

Additional information:

- **num_bin:** The number of bins in each species pair radial histogram.
- **rmax:** Maximum pair distance included in the histogram. The uniform bin width is $\delta r =$ `rmax/num_bin`. If periodic boundary conditions are used for any dimension of the simulation cell, then the default value of `rmax` is the simulation cell radius instead of 10 Bohr. For open boundary conditions the volume (V) used is 1.0 Bohr^3.
- **source/sources:** If unspecified, only pair correlations between each species of quantum particle will be measured. For each classical particleset specified by `source/sources`, additional pair correlations between each quantum and classical species will be measured. Typically there is only one classical particleset (*e.g.* `source="ion0"`), but there can be several in principle (*e.g.* `sources="ion0 ion1 ion2"`).
- **target:** The default value is the preferred usage (*i.e.* `target` does not need to be provided).
- Data is outputted to the `stat.h5` for each QMC sub-run. Individual histograms are named according to the quantum particleset and index of the pair. For example, if the quantum particleset is named "e" and there are two species (up and down electrons, say), then there will be three sets of histogram data in each `stat.h5` file named `gofr_e_0_0`, `gofr_e_0_1`, and `gofr_e_1_1` for up-up, up-down, and down-down correlations, respectively.

Listing 9.14: Pair correlation function estimator element.

```
<estimator type="gofr" name="gofr" num_bin="200" rmax="3.0" />
```

Listing 9.15: Pair correlation function estimator element with additional electron-ion correlations.

```
<estimator type="gofr" name="gofr" num_bin="200" rmax="3.0" source="ion0" />
```

9.3.5 Static structure factor, $S(k)$

Let $\rho^e_{\mathbf{k}} = \sum_j e^{i\mathbf{k}\cdot\mathbf{r}^e_j}$ be the Fourier space electron density, with $\mathbf{r}^e_j$ being the coordinate of the j-th electron. $\mathbf{k}$ is a wavevector commensurate with the simulation cell. QMCPACK allows the user to accumulate the static electron structure factor $S(\mathbf{k})$ at all commensurate $\mathbf{k}$ such that $|\mathbf{k}| \leq (LR\_DIM\_CUTOFF)r_c$. N^e is the number of electrons, `LR_DIM_CUTOFF` is the optimized breakup parameter, and r_c is the Wigner-Seitz radius. It is defined as follows:

$$S(\mathbf{k}) = \frac{1}{N^e}\langle\rho^e_{-\mathbf{k}}\rho^e_{\mathbf{k}}\rangle \tag{9.9}$$

`estimator type=sk` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	sk		Must be sk
`name`^{r}	text	*anything*	any	Unique name for estimator
`hdf5`^{o}	boolean	yes/no	no	Output to `stat.h5` (yes) or `scalar.dat` (no)

Additional information:

- **name:** Unique name for estimator instance. A data structure of the same name will appear in `stat.h5` output files.
- **hdf5:** If `hdf5==yes` output data for $S(k)$ is directed to the `stat.h5` file (recommended usage). If `hdf5==no`, the data is instead routed to the `scalar.dat` file resulting in many columns of data with headings prefixed by `name` and postfixed by the k-point index (*e.g.* `sk_0 sk_1 …sk_1037 …`).
- This estimator only works in periodic boundary conditions. Its presence in the input file is ignored otherwise.
- This is not a species resolved structure factor. Additionally, for $\mathbf{k}$ vectors commensurate with the unit cell, $S(\mathbf{k})$ will include contributions from the static electronic density, thus meaning it won't accurately measure the electron-electron density response.

Listing 9.16: Static structure factor estimator element.

```
<estimator type="sk" name="sk" hdf5="yes"/>
```

9.3.6 Species kinetic energy

Record species-resolved kinetic energy instead of the total kinetic energy in the `Kinetic` column of scalar.dat. `SpeciesKineticEnergy` is arguable the simplest estimator in QMCPACK. The implementation of this estimator is detailed in `manual/estimator/estimator_implementation.pdf`.

`estimator type=specieskinetic` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	specieskinetic		Must be specieskinetic
`name`^{r}	text	*anything*	any	Unique name for estimator
`hdf5`^{o}	boolean	yes/no	no	Output to `stat.h5` (yes)

Listing 9.17: Species kinetic energy estimator element.

```
<estimator type="specieskinetic" name="skinetic" hdf5="no"/>
```

9.3.7 Lattice deviation estimator

Record deviation of a group of particles in one particle set (target) from a group of particles in another particle set (source).

`estimator type=latticedeviation` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	latticedeviation		Must be latticedeviation
`name`^{r}	text	*anything*	any	Unique name for estimator
`hdf5`^{o}	boolean	yes/no	no	Output to `stat.h5` (yes)
`per_xyz`^{o}	boolean	yes/no	no	Directional-resolved (yes)
`source`^{r}	text	e/ion0/...	no	source particle set
`sgroup`^{r}	text	u/d/...	no	source particle group
`target`^{r}	text	e/ion0/...	no	target particle set
`tgroup`^{r}	text	u/d/...	no	target particle group

Additional information:

- `source`: The "reference" particle set to measure distances from, actual reference points are determined together with `sgroup`.
- `sgroup`: The "reference" particle group to measure distances from.
- `source`: The "target" particle set to measure distances to.

- `sgroup`: The "target" particle group to measure distances to. For example, in Listing 9.18, the distance from the up electron ("u") to the origin of the coordinate system is recorded.
- `per_xyz`: Record direction-resolved distance. In Listing 9.18, the x,y,z coordinates of the up electron will be recorded separately if `per_xyz=yes`.
- `hdf5`: Record particle-resolved distances in the h5 file if `gdf5=yes`..

Listing 9.18: Lattice deviation estimator element.

```
<particleset name="e" random="yes">
  <group name="u" size="1" mass="1.0">
     <parameter name="charge"              >    -1                    </parameter>
     <parameter name="mass"                >    1.0                   </parameter>
  </group>
  <group name="d" size="1" mass="1.0">
     <parameter name="charge"              >    -1                    </parameter>
     <parameter name="mass"                >    1.0                   </parameter>
  </group>
</particleset>

<particleset name="wf_center">
  <group name="origin" size="1">
    <attrib name="position" datatype="posArray" condition="0">
            0.00000000        0.00000000        0.00000000
    </attrib>
  </group>
</particleset>

<estimator type="latticedeviation" name="latdev" hdf5="yes" per_xyz="yes"
  source="wf_center" sgroup="origin" target="e" tgroup="u"/>
```

9.3.8 Energy density estimator

An energy density operator, $\hat{\mathcal{E}}_r$, satisfies

$$\int dr \hat{\mathcal{E}}_r = \hat{H}, \tag{9.10}$$

where the integral is over all space and $\hat{H}$ is the Hamiltonian. In QMCPACK , the energy density is split into kinetic and potential components

$$\hat{\mathcal{E}}_r = \hat{\mathcal{T}}_r + \hat{\mathcal{V}}_r \tag{9.11}$$

with each component given by

$$\hat{\mathcal{T}}_r = \frac{1}{2}\sum_i \delta(r - r_i)\hat{p}_i^2 \tag{9.12}$$

$$\hat{\mathcal{V}}_r = \sum_{i<j} \frac{\delta(r - r_i) + \delta(r - r_j)}{2} \hat{v}^{ee}(r_i, r_j) + \sum_{i\ell} \frac{\delta(r - r_i) + \delta(r - \tilde{r}_\ell)}{2} \hat{v}^{eI}(r_i, \tilde{r}_\ell)$$
$$+ \sum_{\ell<m} \frac{\delta(r - \tilde{r}_\ell) + \delta(r - \tilde{r}_m)}{2} \hat{v}^{II}(\tilde{r}_\ell, \tilde{r}_m).$$

Here r_i and $\tilde{r}_\ell$ represent election and ion positions, respectively, $\hat{p}_i$ is a single electron momentum operator, and $\hat{v}^{ee}(r_i, r_j)$, $\hat{v}^{eI}(r_i, \tilde{r}_\ell)$, $\hat{v}^{II}(\tilde{r}_\ell, \tilde{r}_m)$ are the electron-electron, electron-ion, and ion-ion pair potential operators (including non-local pseudopotentials, if present). This form of the energy density is size consistent, *i.e.* the partially integrated energy density operators of well separated atoms gives the isolated Hamiltonians of the respective atoms. For periodic systems with twist averaged boundary conditions, the energy density is formally correct only for either a set of supercell k-points that correspond to real valued wavefunctions, or a k-point set that has inversion symmetry around a k-point having a real valued wavefunction. For more information about the energy density, see Ref. [8].

In QMCPACK , the energy density can be accumulated on piecewise uniform three dimensional grids in generalized cartesian, cylindrical, or spherical coordinates. The energy density integrated within Voronoi volumes centered on ion positions is also available. The total particle number density is also accumulated on the same grids by the energy density estimator for convenience so that related quantities, such as the regional energy per particle, can be computed easily.

`estimator type=EnergyDensity` element				
parent elements:	`hamiltonian, qmc`			
child elements:	`reference_points, spacegrid`			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**EnergyDensity**		Must be EnergyDensity
`name`^{r}	text	*anything*		Unique name for estimator
`dynamic`^{r}	text	`particleset.name`		Identify electrons
`static`^{o}	text	`particleset.name`		Identify ions

Additional information:

- `name`: Must be unique. A dataset with blocked statistical data for the energy density will appear in the `stat.h5` files labeled as `name`.

Listing 9.19: Energy density estimator accumulated on a 20x10x10 grid over the simulation cell.

```
<estimator type="EnergyDensity" name="EDcell" dynamic="e" static="ion0">
  <spacegrid coord="cartesian">
    <origin p1="zero"/>
    <axis p1="a1" scale=".5" label="x" grid="-1 (.05) 1"/>
    <axis p1="a2" scale=".5" label="y" grid="-1 (.1) 1"/>
    <axis p1="a3" scale=".5" label="z" grid="-1 (.1) 1"/>
  </spacegrid>
</estimator>
```

Listing 9.20: Energy density estimator accumulated within spheres of radius 6.9 Bohr centered on the first and second atoms in the ion0 particleset.

```
<estimator type="EnergyDensity" name="EDatom" dynamic="e" static="ion0">
  <reference_points coord="cartesian">
    r1 1 0 0
    r2 0 1 0
    r3 0 0 1
```

```
    </reference_points>
    <spacegrid coord="spherical">
      <origin p1="ion01"/>
      <axis p1="r1" scale="6.9" label="r"     grid="0 1"/>
      <axis p1="r2" scale="6.9" label="phi"   grid="0 1"/>
      <axis p1="r3" scale="6.9" label="theta" grid="0 1"/>
    </spacegrid>
    <spacegrid coord="spherical">
      <origin p1="ion02"/>
      <axis p1="r1" scale="6.9" label="r"     grid="0 1"/>
      <axis p1="r2" scale="6.9" label="phi"   grid="0 1"/>
      <axis p1="r3" scale="6.9" label="theta" grid="0 1"/>
    </spacegrid>
  </estimator>
```

Listing 9.21: Energy density estimator accumulated within Voronoi polyhedra centered on the ions.

```
  <estimator type="EnergyDensity" name="EDvoronoi" dynamic="e" static="ion0">
    <spacegrid coord="voronoi"/>
  </estimator>
```

The `<reference_points/>` element provides a set of points for later use in specifying the origin and coordinate axes needed to construct a spatial histogramming grid. Several reference points on the surface of the simulation cell (see Table 9.1) as well as the positions of the ions (see the `energydensity.static` attribute) are made available by default. The reference points can be used, for example, to construct a cylindrical grid along a bond with the origin on the bond center.

`reference_points` element				
parent elements:	`estimator type=EnergyDensity`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`coord`^{r}	text	cartesian/cell		Specify coordinate system
body text				
	The body text is a line formatted list of points with labels			

Additional information

- **coord:** If `coord=cartesian`, labeled points are in cartesian (x,y,z) format in units of Bohr. If `coord=cell`, then labeled points are in units of the simulation cell axes.
- **body text:** The list of points provided in the body text are line formatted, with four entries per line (*label coor1 coor2 coor3*). A set of points referenced to the simulation cell are available by default (see table 9.1). If `energydensity.static` is provided, the location of each individual ion is also available (*e.g.* if `energydensity.static=ion0`, then the location of the first atom is available with label ion01, the second with ion02, etc.). All points can be used by label when constructing spatial histogramming grids (see the `spacegrid` element below) used to collect energy densities.

label	point	description
zero	0 0 0	Cell center
a1	a_1	Cell axis 1
a2	a_2	Cell axis 2
a3	a_3	Cell axis 3
f1p	$a_1/2$	Cell face 1+
f1m	-$a_1/2$	Cell face 1-
f2p	$a_2/2$	Cell face 2+
f2m	-$a_2/2$	Cell face 2-
f3p	$a_3/2$	Cell face 3+
f3m	-$a_3/2$	Cell face 3-
cppp	$(a_1 + a_2 + a_3)/2$	Cell corner +,+,+
cppm	$(a_1 + a_2 - a_3)/2$	Cell corner +,+,-
cpmp	$(a_1 - a_2 + a_3)/2$	Cell corner +,-,+
cmpp	$(-a_1 + a_2 + a_3)/2$	Cell corner -,+,+
cpmm	$(a_1 - a_2 - a_3)/2$	Cell corner +,-,-
cmpm	$(-a_1 + a_2 - a_3)/2$	Cell corner -,+,-
cmmp	$(-a_1 - a_2 + a_3)/2$	Cell corner -,-,+
cmmm	$(-a_1 - a_2 - a_3)/2$	Cell corner -,-,-

Table 9.1: Reference points available by default. The vectors a_1, a_2, and a_3 refer to the simulation cell axes. The representation of the cell is centered around zero.

The <spacegrid/> element is used to specify a spatial histogramming grid for the energy density. Grids are constructed based on a set of, potentially non-orthogonal, user provided coordinate axes. The axes are based on information available from reference_points. Voronoi grids are based only on nearest neighbor distances between electrons and ions. Any number of space grids can be provided to a single energy density estimator.

spacegrid element

parent elements: estimator type=EnergyDensity
child elements: origin, axis
attributes

name	datatype	values	default	description
coord^r	text	cartesian cylindrical spherical voronoi		Specify coordinate system

The <origin/> element gives the location of the origin for a non-Voronoi grid.
Additional information:

- **p1/p2/fraction:** The location of the origin is set to p1+fraction*(p2-p1). If only p1 is provided, the origin is at p1.

origin element				
parent elements:	spacegrid			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
p1^r	text	reference_point.label		Select end point
p2^o	text	reference_point.label		Select end point
fraction^o	real		0	Interpolation fraction

The <axis/> element represents a coordinate axis used to construct the, possibly curved, coordinate system for the histogramming grid. Three <axis/> elements must be provided to a non-Voronoi <spacegrid/> element.

axis element				
parent elements:	spacegrid			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
label^r	text	*See below*		Axis/dimension label
grid^r	text		"0 1"	Grid ranges/intervals
p1^r	text	reference_point.label		Select end point
p2^o	text	reference_point.label		Select end point
scale^o	real			Interpolation fraction

Additional information:

- **label:** The allowed set of axis labels depends on the coordinate system (*i.e.* spacegrid.coord). Labels are x/y/z for coord=cartesian, r/phi/z for coord=cylindrical, r/phi/theta for coord=spherical.
- **p1/p2/scale:** The axis vector is set to p1+scale*(p2-p1). If only p1 is provided, the axis vector is p1.
- **grid:** Specifies the histogram grid along the direction specified by label. The allowed grid points fall in the range [-1,1] for label=x/y/z or [0,1] for r/phi/theta. A grid of 10 evenly spaced points between 0 and 1 can be requested equivalently by grid="0 (0.1) 1" or grid="0 (10) 1". Piecewise uniform grids covering portions of the range are supported, *e.g.* grid="-0.7 (10) 0.0 (20) 0.5".
- Note that grid specifies the histogram grid along the (curved) coordinate given by label. The axis specified by p1/p2/scale does not correspond one-to-one with label unless label=x/y/z, but the full set of axes provided define the (sheared) space on top of which the curved (*e.g.* spherical) coordinate system is built.

9.3.9 One body density matrix

The N-body density matrix in DMC is $\hat{\rho}_N = |\Psi_T\rangle\langle\Psi_{FN}|$ (for VMC, substitute Ψ_T for Ψ_{FN}). The one body reduced density matrix (1RDM) is obtained by tracing out all particle coordinates but

one:

$$\hat{n}_1 = \sum_n Tr_{R_n} |\Psi_T\rangle\langle\Psi_{FN}| \tag{9.13}$$

In the formula above, the sum is over all electron indices and $Tr_{R_n}(*) \equiv \int dR_n \langle R_n| * |R_n\rangle$ with $R_n = [r_1, ..., r_{n-1}, r_{n+1}, ..., r_N]$. When the sum is restricted over spin up or down electrons, one obtains a density matrix for each spin species. The 1RDM computed by QMCPACK is partitioned in this way.

In real space, the matrix elements of the 1RDM are

$$n_1(r, r') = \langle r|\hat{n}_1|r'\rangle = \sum_n \int dR_n \Psi_T(r, R_n)\Psi^*_{FN}(r', R_n) \tag{9.14}$$

A more efficient and compact representation of the 1RDM is obtained by expanding in the single particle orbitals obtained from a Hartree-Fock or DFT calculation, $\{\phi_i\}$:

$$\begin{aligned} n_1(i,j) &= \langle\phi_i|\hat{n}_1|\phi_j\rangle \\ &= \int dR \Psi^*_{FN}(R)\Psi_T(R) \sum_n \int dr'_n \frac{\Psi_T(r'_n, R_n)}{\Psi_T(r_n, R_n)} \phi_i(r'_n)^* \phi_j(r_n) \end{aligned} \tag{9.15}$$

The integration over r' in Eq. 9.15 is inefficient when one is also interested in obtaining matrices involving energetic quantities, such as the energy density matrix of Ref. [9] or the related (and more well known) Generalized Fock matrix. For this reason, an approximation is introduced as follows:

$$n_1(i,j) \approx \int dR \Psi_{FN}(R)^* \Psi_T(R) \sum_n \int dr'_n \frac{\Psi_T(r'_n, R_n)^*}{\Psi_T(r_n, R_n)^*} \phi_i(r_n)^* \phi_j(r'_n) \tag{9.16}$$

For VMC, FN-DMC, FP-DMC, and RN-DMC the formula above represents an exact sampling of the 1RDM corresponding to $\hat{\rho}^\dagger_N$ (see appendix A of Ref. [9] for more detail).

`estimator type=dm1b` element				
parent elements:	`hamiltonian, qmc`			
child elements:	*none*			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**dm1b**		Must be dm1b
`name`^{r}	text	*anything*		Unique name for estimator
parameters				
name	**datatype**	**values**	**default**	**description**
`basis`^{r}	text array	sposet.name(s)		Orbital basis
`integrator`^{o}	text	uniform_grid uniform density	uniform_grid	Integration method
`evaluator`^{o}	text	loop/matrix	loop	Evaluation method
`scale`^{o}	real	$0 < scale < 1$	1.0	Scale integration cell
`center`^{o}	real array(3)	*any point*		Center of cell
`points`^{o}	integer	> 0	10	Grid points in each dim
`samples`^{o}	integer	> 0	10	MC samples
`warmup`^{o}	integer	> 0	30	MC warmup
`timestep`^{o}	real	> 0	0.5	MC time step
`use_drift`^{o}	boolean	yes/no	no	Use drift in VMC
`check_overlap`^{o}	boolean	yes/no	no	Print overlap matrix
`check_derivatives`^{o}	boolean	yes/no	no	Check density derivatives
`acceptance_ratio`^{o}	boolean	yes/no	no	Print accept ratio
`rstats`^{o}	boolean	yes/no	no	Print spatial stats
`normalized`^{o}	boolean	yes/no	yes	`basis` comes norm'ed
`volume_normed`^{o}	boolean	yes/no	yes	`basis` norm is volume
`energy_matrix`^{o}	boolean	yes/no	no	Energy density matrix

Additional information

- `name`: Density matrix results appear in `stat.h5` files labeled according to `name`.
- `basis`: List of `sposet.name`'s. The total set of orbitals contained in all `sposet`'s comprises the basis (subspace) the one body density matrix is projected onto. This set of orbitals generally includes many virtual orbitals that are not occupied in a single reference Slater determinant.
- `integrator`: This selects the method used to perform the additional single particle integration. Options are `uniform_grid` (uniform grid of points over the cell), `uniform` (uniform random sampling over the cell), and `density` (Metropolis sampling of approximate density: $\sum_{b\in\texttt{basis}}|\phi_b|^2$, not well tested, please check results carefully!). Depending on the integrator selected, different subsets of the other input parameters are active.
- `evaluator`: Select for-loop or matrix multiply implementations. Matrix is preferred for speed. Both implementations should give the same results, but please check as this has not been

exhaustively tested.

- `scale:` Resize the simulation cell by scale for use as an integration volume (active for `integrator=uniform/uniform_grid`).
- `center:` Translate the integration volume to center at this point (active for `integrator=uniform/uniform_gri` If `center` is not provided, the scaled simulation cell is used as is.
- `points:` The number of grid points in each dimension for `integrator=uniform_grid`. For example, `points=10` results in a uniform 10x10x10 grid over the cell.
- `samples:` Sets the number of Monte Carlo samples collected each step (active for `integrator=uniform/density`).
- `warmup:` Number of warmup Metropolis steps at the start of the run, prior to data collection (active for `integrator=density`).
- `timestep:` Drift-diffusion timestep used in Metropolis sampling (active for `integrator=density`).
- `use_drift:` Enable drift in Metropolis sampling (active for `integrator=density`).
- `check_overlap:` Print the overlap matrix (computed via simple Riemann sums) to the log and then abort. Note that subsequent analysis based on the 1RDM is simplest if the input orbitals are orthogonal.
- `check_derivatives:` Print analytic and numerical derivatives of the approximate (sampled) density for several sample points, then abort.
- `acceptance_ratio:` Print the acceptance ratio of the density sampling to the log each step.
- `rstats:` Print statistical information about the spatial motion of the sampled points to the log each step.
- `normalized:` Declare whether the inputted orbitals are normalized or not. If `normalized=no`, direct Riemann integration over a 200x200x200 grid will be used to compute the normalizations prior to use.
- `volume_normed:` Declare whether the inputted orbitals are normalized to the cell volume (default) or not (a norm of 1.0 is assumed in this case). Currently, B-spline orbitals coming from Quantum Espresso and HEG planewave orbitals native to QMCPACK are known to be volume normalized.
- `energy_matrix:` Also accumulate the one body reduced energy density matrix and write it to `stat.h5`. This matrix is not covered in any detail here; the interested reader is referred to Ref. [9].

Listing 9.22: One body density matrix with uniform grid integration.

```
<estimator type="dm1b" name="DensityMatrices">
  <parameter name="basis"        >  spo_u spo_uv  </parameter>
  <parameter name="evaluator"    >  matrix        </parameter>
  <parameter name="integrator"   >  uniform_grid  </parameter>
  <parameter name="points"       >  4             </parameter>
```

```
    <parameter name="scale"        >  1.0          </parameter>
    <parameter name="center"       >  0 0 0        </parameter>
  </estimator>
```

Listing 9.23: One body density matrix with uniform sampling.

```
  <estimator type="dm1b" name="DensityMatrices">
    <parameter name="basis"        >  spo_u spo_uv  </parameter>
    <parameter name="evaluator"    >  matrix        </parameter>
    <parameter name="integrator"   >  uniform       </parameter>
    <parameter name="samples"      >  64            </parameter>
    <parameter name="scale"        >  1.0           </parameter>
    <parameter name="center"       >  0 0 0         </parameter>
  </estimator>
```

Listing 9.24: One body density matrix with density sampling.

```
  <estimator type="dm1b" name="DensityMatrices">
    <parameter name="basis"        >  spo_u spo_uv  </parameter>
    <parameter name="evaluator"    >  matrix        </parameter>
    <parameter name="integrator"   >  density       </parameter>
    <parameter name="samples"      >  64            </parameter>
    <parameter name="timestep"     >  0.5           </parameter>
    <parameter name="use_drift"    >  no            </parameter>
  </estimator>
```

Listing 9.25: Example sposet initialization for density matrix use. Occupied and virtual orbital sets are created separately, then joined (`basis="spo_u spo_uv"`).

```
  <sposet_builder type="bspline" href="../dft/pwscf_output/pwscf.pwscf.h5"
    tilematrix="1 0 0 0 1 0 0 0 1" twistnum="0" meshfactor="1.0" gpu="no"
    precision="single">
    <sposet type="bspline" name="spo_u"  group="0" size="4"/>
    <sposet type="bspline" name="spo_d"  group="0" size="2"/>
    <sposet type="bspline" name="spo_uv" group="0" index_min="4" index_max="10"/>
  </sposet_builder>
```

Listing 9.26: Example sposet initialization for density matrix use. Density matrix orbital basis created separately (`basis="dm_basis"`).

```
  <sposet_builder type="bspline" href="../dft/pwscf_output/pwscf.pwscf.h5"
    tilematrix="1 0 0 0 1 0 0 0 1" twistnum="0" meshfactor="1.0" gpu="no"
    precision="single">
    <sposet type="bspline" name="spo_u"  group="0" size="4"/>
    <sposet type="bspline" name="spo_d"  group="0" size="2"/>
    <sposet type="bspline" name="dm_basis" size="50" spindataset="0"/>
  </sposet_builder>
```

9.4 Forward Walking Estimators

Forward walking is a method by which one can sample the pure fixed-node distribution $\langle\Phi_0|\Phi_0\rangle$. Specifically, one multiplies each walker's DMC mixed estimate for the observable $\mathcal{O}$, $\frac{\mathcal{O}(\mathbf{R})\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}$, by

the weighting factor $\frac{\Phi_0(\mathbf{R})}{\Psi_T(\mathbf{R})}$. As it turns out, this weighting factor for any walker $\mathbf{R}$ is proportional to the total number of descendants the walker will have after a sufficiently long projection time β.

To forward walk on an observable, one declares a generic forward walking estimator within a `<hamiltonian>` block, and then specifies the observables to forward walk on and forward walking parameters. Here is a summary.

`estimator type=ForwardWalking` element				
parent elements:	`hamiltonian, qmc`			
child elements:	`Observable`			
attributes				
name	**datatype**	**values**	**default**	**description**
`type`^{r}	text	**ForwardWalking**		Must be "ForwardWalking"
`name`^{r}	text	*anything*	any	Unique name for estimator

`Observable` element				
parent elements:	`estimator, hamiltonian, qmc`			
child elements:	*None*			
attributes				
name	**datatype**	**values**	**default**	**description**
`name`^{r}	text	*anything*	any	Registered name of existing estimator on which to forward walk.
`max`^{r}	integer	> 0		The maximum projection time in steps (`max`$= \beta/\tau$).
`frequency`^{r}	text	≥ 1		Dump data only for every `frequency`-th to `scalar.dat` file

Additional information:

- **Cost**: Due to having to store histories of observables up to `max` time-steps, one should multiply the memory cost of storing the non-forward walked observables variables by `max`. Not an issue for things like the potential energy, but can be prohibitive for observables like density, forces, etc.
- **Naming Convention**: Forward walked observables are automatically named `FWE_name_i`, where `i` is the forward walked expectation value at time step `i`, and `name` is whatever name appears in the `<Observable>` block. This is also how it will appear in the `scalar.dat` file.

In the following example case, QMCPACK forward walks on the potential energy for 300 time steps, and dumps the forward walked value at every time step.

Listing 9.27: Forward walking estimator element.

```
<estimator name="fw" type="ForwardWalking">
    <Observable name="LocalPotential" max="300" frequency="1"/>
     <!--- Additional Observable blocks go here -->
 </estimator>
```

9.5 "Force" estimators

QMCPACK supports force estimation by use of the Chiesa-Ceperly-Zhang (CCZ) estimator. Currently, open and periodic boundary conditions are supported, but for all-electron calculations only.

Without loss of generality, the CCZ estimator for the z-component of the force on an ion centered at the origin is given by the following expression:

$$F_z = -Z \sum_{i=1}^{N_e} \frac{z_i}{r_i^3} [\theta(r_i - \mathcal{R}) + \theta(\mathcal{R} - r_i) \sum_{\ell=1}^{M} c_\ell r_i^\ell] \tag{9.17}$$

Z is the ionic charge, M is the degree of the smoothing polynomial, $\mathcal{R}$ is a real-space cutoff of the sphere within which the bare-force estimator is smoothed, and c_ℓ are predetermined coefficients. These coefficients are chosen to minimize the weighted mean square error between the bare force estimate and the s-wave filtered estimator. Specifically,

$$\chi^2 = \int_0^{\mathcal{R}} dr\, r^m \, [f_z(r) - \tilde{f}_z(r)]^2 \tag{9.18}$$

Here, m is the weighting exponent, $f_z(r)$ is the unfiltered radial force density for the z force component, and $\tilde{f}_z(r)$ smoothed polynomial function for the same force density. The reader is invited to refer to the original paper for a more thorough explanation of the methodology, but with the notation in hand, QMCPACK takes the following parameters.

`estimator type=Force` element				
parent elements:	`hamiltonian, qmc`			
child elements:	`parameter`			
attributes				
name	**datatype**	**values**	**default**	**description**
`mode`^{o}	text	*See above*	bare	Select estimator type
`type`^{r}	text	Force		Must be "Force"
`name`^{o}	text	*anything*	ForceBase	Unique name for this estimator
`pbc`^{o}	boolean	yes/no	yes	Using periodic BC's or not
`addionion`^{o}	boolean	yes/no	no	Add the ion-ion force contribution to output force estimate.
parameters				
name	**datatype**	**values**	**default**	**description**
`rcut`^{o}	real	> 0	1.0	Real space cutoff $\mathcal{R}$ in bohr.
`nbasis`^{o}	integer	> 0	2	Degree of smoothing polynomial M
`weightexp`^{o}	integer	> 0	2	χ^2 weighting exponent m.

Additional information:

- **Naming Convention**: The unique identifier `name` is appended with `name_X_Y` in the `scalar.dat` file, where `X` is the ion ID number, and `Y` is the component ID (an integer with x=0, y=1, z=2). All force components for all ions are computed and dumped to the `scalar.dat` file.
- **Miscellaneous**: Usually, the default choice of `weightexp` is sufficient. Different combinations of `rcut` and `nbasis` should be tested though to minimize variance and bias. There is of course a tradeoff, with larger `nbasis` and smaller `rcut` leading to smaller biases and larger variances.

The following is an example use case.

```
<estimator name="myforce" type="Force" mode="cep" addionion="yes">
    <parameter name="rcut">0.1</parameter>
    <parameter name="nbasis">4</parameter>
    <parameter name="weightexp">2</parameter>
</estimator>
```

Chapter 10

Quantum Monte Carlo Methods

qmc factory element				
parent elements:	simulation, loop			
type selector:	method attribute			
type options:	vmc	Variational Monte Carlo		
	linear	Wavefunction optimization with linear method		
	dmc	Diffusion Monte Carlo		
	rmc	Reptation Monte Carlo		
shared attributes:				
name	**datatype**	**values**	**default**	**description**
method	text	listed above	invalid	QMC driver
move	text	pbyp, alle	pbyp	method used to move electrons
gpu	text	yes, no	dep.	use the GPU
trace	text		no	???
checkpoint	integer	-1, 0, n	-1	checkpoint frequency
record	integer	n	0	save configuration every n steps
target	text			???
completed	text			???
append	text	yes, no	yes	???

Additional information:

- move. There are two ways implemented to move electrons. The more used method is the particle-by-particle move. In this method, only one electron is moved for acceptance or rejection. The other method is the all-electron move, namely all the electrons are moved once for testing acceptance or rejection.
- gpu. When the executable is compiled with CUDA, the target computing device can be chosen by this switch. With a regular CPU only compilation, this option is not effective.
- checkpoint. Enable or disable checkpointing and specify the frequency of output. Possible values are:

  -1 No checkpoint (default setting).

  0 Dump after the completion of a qmc section.

n Dump after every n blocks. Also dump at the end of the run.

The particle configurations will be written to a `.config.h5` file.

Listing 10.1: The following is an example of running a simulation that can be restarted .

```
<qmc method="dmc" move="pbyp"  checkpoint="0">
  <parameter name="timestep">         0.004  </parameter>
  <parameter name="blocks">           100   </parameter>
  <parameter name="steps">            400    </parameter>
</qmc>
```

The checkpoint flag instructs qmcpack to output walker configurations. This also works in Variational Monte Carlo. This will output an h5 file with the name "projectid"."run-number".config.h5. Check that this file exists before attempting a restart.

To continue a run, specify the `mcwalkerset` element before your VMC/DMC block:

Listing 10.2: Restart (read walkers from previous run)

```
<mcwalkerset fileroot="BH.s002" version="0 6" collected="yes"/>
 <qmc method="dmc" move="pbyp"  checkpoint="0">
   <parameter name="timestep">         0.004  </parameter>
   <parameter name="blocks">           100   </parameter>
   <parameter name="steps">            400    </parameter>
 </qmc>
```

BH is the project id and s002 is the calculation number to read in the walkers from the previous run.

In the project id section, make sure that the series number is different than any existing one to avoid overwriting it.

10.1 Variational Monte Carlo

Additional information:

- `walkers`. The initial default number of walkers is one per OpenMP thread or per MPI task if threading is disabled, with a minimum of one per thread. One walker per thread will be created in the event fewer walkers than threads are requested.
- `blocks`. This parameter is universal for all the QMC methods. The Monte Carlo processes are divided into a number of blocks each containing a number of steps. At the end of each block, all the statistics accumulated in the block is dumped in to files, e.g. scalar.dat. Typically each block should have sufficient number of steps that the I/O at the end of each block is negligible compared to the computational cost. Each block should not take so long that it is difficult to monitor progress. There should be a sufficient number of blocks to perform statistical analysis.
- `warmupsteps`. Warm-up steps are steps used only for equilibration. Property measurements are not performed during warm-up steps.
- `steps`. The number of energy and other property measurements to perform per block.

vmc method				
parameters				
name	**datatype**	**values**	**default**	**description**
walkers	integer	> 0	dep.	number of walkers per MPI task
blocks	integer	≥ 0	1	number of blocks
steps	integer	≥ 0	1	number of steps per block
warmupsteps	integer	≥ 0	0	number of steps for warming up
substeps	integer	≥ 0	1	number of substeps per step
usedrift	text	yes, no	no	use the algorithm with drift
timestep	real	> 0	0.1	time step for each electron move
samples	integer	≥ 0	0	number of walker samples for DMC/optimization
stepsbetweensamples	integer	> 0	1	period of sample accumulation
samplesperthread	integer	≥ 0	0	number of samples per thread
storeconfigs	integer	all values	0	store configurations o
blocks_between_recompute	integer	≥ 0	dep.	wavefunction recompute frequency

- substeps. For each substep, each of the electrons is attempted to be moved only once by either particle-by-particle or all-electron move. Because the local energy is only evaluated at each full step and not substep, substeps are computationally cheaper and can be used to reduce correlation between property measurements at lower cost.
- usedrift. The VMC is implemented in two algorithms with or without drift. In the no-drift algorithm, the move of each electron is proposed with a Gaussian distribution. The standard deviation is chosen as the timestep input. In the drift algorithm, electrons are moved by langevin dynamics.
- timestep. The meaning of timestep depends on whether the drift is used or not. In general, larger timesteps reduce the time correlation but might also reduce the acceptance ratio, reducing overall statistical efficiency. For VMC, typically the acceptance ratio should be close to 50% for an efficient simulation.
- samples. Seperate from conventional energy and other property measurements, samples refers to storing whole electron configurations in memory ("walker samples") as would be needed by subsequent wavefunction optimization or DMC steps. **A standard VMC run to measure the energy does not need samples to be set.**

$$\text{samples} = \frac{\text{blocks} \cdot \text{steps} \cdot \text{walkers}}{\text{stepsbetweensamples}} \cdot \text{number of MPI tasks}$$

- samplesperthread. This is an alternative way to set the target amount of samples, and can

be useful when preparing a stored population for a subsequent DMC calculation.

$$\text{samplesperthread} = \frac{\text{blocks} \cdot \text{steps}}{\text{stepsbetweensamples}}$$

- `stepsbetweensamples`. Due to the fact that samples generated by consecutive steps are correlated, having stepsbetweensamples larger than 1 can be used to reduces that correlation. In practice, using larger substeps is cheaper than using stepsbetweensamples to decorrelate samples.
- `storeconfigs`. If storeconfigs is set to a non-zero value, then electron configurations during the VMC run will be saved to files.
- `blocks_between_recompute`. Recompute the accuracy critical determinant part of the wavefunction from scratch. =1 by default when using mixed precision. =0 (no recompute) by default when not using mixed precision. Recomputing introduces a performance penalty dependent on system size.

An example VMC section for a simple VMC run:

```
<qmc method="vmc" move="pbyp">
  <estimator name="LocalEnergy" hdf5="no"/>
  <parameter name="walkers">    256 </parameter>
  <parameter name="warmupSteps">  100 </parameter>
  <parameter name="substeps">  5 </parameter>
  <parameter name="blocks">  20 </parameter>
  <parameter name="steps">  100 </parameter>
  <parameter name="timestep">  1.0 </parameter>
  <parameter name="usedrift">   yes </parameter>
</qmc>
```

Here we set 256 walkers per MPI, have a brief initial equilibration of 100 steps, and then 20 blocks of 100 steps with 5 substeps each.

The following is an example of VMC section storing configurations (walker samples) for optimization.

```
<qmc method="vmc" move="pbyp" gpu="yes">
  <estimator name="LocalEnergy" hdf5="no"/>
  <parameter name="walkers">    256 </parameter>
  <parameter name="samples">    2867200 </parameter>
  <parameter name="stepsbetweensamples">    1 </parameter>
  <parameter name="substeps">  5 </parameter>
  <parameter name="warmupSteps">  5 </parameter>
  <parameter name="blocks">  70 </parameter>
  <parameter name="timestep">  1.0 </parameter>
  <parameter name="usedrift">   no </parameter>
</qmc>
```

10.2 Wavefunction Optimization

Optimizing wavefunction is critical in all kinds of real-space quantum Monte Carlo calculations because it significantly improves both the accuracy and efficiency of computation. However, it is very difficult to directly adopt deterministic minimization approaches due to the stochastic nature of evaluating quantities with Monte Carlo. Thanks to the algorithmic breakthrough during the first

decade of this century and the tremendous computer power available, it becomes feasible to optimize tens of thousands of parameters in a wavefunction for a solid or molecule. QMCPACK has multiple optimizers implemented based on the state-of-the-art linear method. We are continually improving our optimizers for the robustness and friendliness and trying to provide a single solution. Due to the large variation of wavefunction types carrying distinct characteristics, using several optimizer may be needed in some cases. It is highly suggested to read the recommendation from the experts maintaining these optimizers.

A typical optimization block looks like the following. It starts with method="linear" and contains three blocks of parameters.

```
<loop max="10">
 <qmc method="linear" move="pbyp" gpu="yes">
   <!-- Specify the VMC options -->
   <parameter name="walkers">              256 </parameter>
   <parameter name="samples">          2867200 </parameter>
   <parameter name="stepsbetweensamples">    1 </parameter>
   <parameter name="substeps">               5 </parameter>
   <parameter name="warmupSteps">            5 </parameter>
   <parameter name="blocks">                70 </parameter>
   <parameter name="timestep">             1.0 </parameter>
   <parameter name="usedrift">              no </parameter>
   <estimator name="LocalEnergy" hdf5="no"/>
   ...
   <!-- Specify the correlated sampling options and define the cost function -->
   <parameter name="minwalkers">            0.3 </parameter>
        <cost name="energy">               0.95 </cost>
        <cost name="unreweightedvariance"> 0.00 </cost>
        <cost name="reweightedvariance">   0.05 </cost>
   ...
   <!-- Specify the optimizer options -->
   <parameter name="MinMethod">    OneShiftOnly </parameter>
   ...
 </qmc>
</loop>
```

- loop is helpful to execute identical optimization blocks repeatedly.
- The first part is highly identical to a regular VMC block.
- The second part is to specify the correlated sampling options and define the cost function.
- The last part is used to specify the options of different optimizers. They can be very distinct from one optimizer to another.

10.2.1 VMC run for the optimization

The VMC calculation for the wavefunction optimization has a strict requirement that `samples` or `samplesperthread` must be specified because of the optimizer needs for the stored samples. The input parameters of this part are identical to the VMC method.

Recommendations:

- Run the inclusive VMC calculation correctly and efficiently, because the this part takes significant amount of time in optimization. For example, make sure the derived steps per block is 1 and use larger substeps to control the correlation between samples.

- A reasonable starting wavefunction is necessary. A lot of optimization fails because of a bad wavefunction starting point. The sign of a bad initial wavefunction includes but not limited to very long equilibration time, low acceptance ratio and huge variance. The first thing to do after a failed optimization is to check the information provided by the VMC calculation via *.scalar.dat files.

10.2.2 Correlated sampling and Cost function

After generating the samples with VMC, the derivatives of the wavefunction with respect to the parameters are computed for proposing a new set of parameters by optimizers. And later, a correlated sampling calculation is performed to quickly evaluate values of the cost function on the old set of parameters and the new set for the further decisions. The input parameters are listed in the following table.

`linear` method				
parameters				
name	**datatype**	**values**	**default**	**description**
`nonlocalpp`	text	yes, no	no	include non-local PP energy in the cost function
`minwalkers`	real	0–1	0.3	lower bound of the effective weight
`maxWeight`	real	> 1	1e6	Maximum weight allowed in reweighting

Additional information:

- `maxWeight`. The default should be good.
- `nonlocalpp`. Non-local PP contribution to the local energy depends on the wavefunction. When a new set of parameter is proposed, this contribution needs to be updated if the cost function consists of local energy. Fortunately, non-local contribution is chosen small when making a PP for small locality error. We can ignore its change and avoid the expensive computational cost. GPU code has a implementation issue that large amount of memory is consumed with this option.
- `minwalkers`. A CRITICAL parameter. When the ratio of effective samples to actual number of samples in a reweighting step goes lower than `minwalkers`, the proposed set of parameters is invalid.

The cost function consists of three components: energy, unreweighted variance and reweighted variance.

```
<cost name="energy">                0.95 </cost>
<cost name="unreweightedvariance">  0.00 </cost>
<cost name="reweightedvariance">    0.05 </cost>
```

10.2.3 Optimizers

QMCPACK implements a few optimizers having different preference aiming for different priorities. They can be switched among 'OneShiftOnly' (default), 'adaptive' and 'quartic' (old) by the following line in the optimization block.

<parameter name="MinMethod"> THE METHOD YOU LIKE </parameter>

OneShiftOnly

The OneShiftOnly optimizer targets a fast optimization by moving parameters more aggressively. It works with OpenMP and GPU and can be considered for large systems. This method relies on the effective weight of correlated sampling rather than the cost function value to justify a new set of parameters. If effective weight is larger than `minwalkers`, the new set is taken no matter the cost function value decreases or not. If a proposed set is rejected, the standard output prints the measured ratio of effective samples to the total number of samples and adjustment on `minwalkers` can be made if needed.

`linear` method				
parameters				
name	**datatype**	**values**	**default**	**description**
`shift_i`	real	> 0	0.01	Direct stabilizer added to the Hamiltonian matrix
`shift_s`	real	> 0	1.00	Initial stabilizer based on the overlap matrix

Additional information:

- `shift_i`. This is the direct term added to the diagonal of the Hamiltonian matrix. More stable but slower optimization with a large value.
- `shift_s`. This is the initial value of the stabilizer based on the overlap matrix added to the Hamiltonian matrix. More stable but slower optimization with a large value. The used value is auto-adjusted by the optimizer.

Recommendations:

- Default `shift_i`, `shift_s` should be fine.
- For hard cases, increasing `shift_i` (factor of 5 or 10) can significantly stabilize the optimization by reducing the pace towards the optimal parameter set.
- If the VMC energy of the last optimization iterations grows significantly, increase `minwalkers` closer to 1 and make the optimization stable.
- If the first iterations of optimization are rejected on a reasonable initial wavefunction, lower the `minwalkers` value based on the measured value printed in the standard output to accept the move.

It is recommended to use this optimizer in two sections with a very small `minwalkers` in the first and a large value in the second like the following. In the very beginning, parameters are far away form optimal values and large changes are proposed by the optimizer. Having a small `minwalkers` allows accepting these changes much easier. When the energy gradually converges, we can have a large `minwalkers` to avoid risky parameter sets.

```
<loop max="6">
 <qmc method="linear" move="pbyp" gpu="yes">
   <!-- Specify the VMC options -->
   <parameter name="walkers">                1 </parameter>
   <parameter name="samples">            10000 </parameter>
   <parameter name="stepsbetweensamples">    1 </parameter>
   <parameter name="substeps">               5 </parameter>
```

```
    <parameter name="warmupSteps">            5 </parameter>
    <parameter name="blocks">                25 </parameter>
    <parameter name="timestep">             1.0 </parameter>
    <parameter name="usedrift">              no </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <!-- Specify the optimizer options -->
    <parameter name="MinMethod">    OneShiftOnly </parameter>
    <parameter name="minwalkers">           1e-4 </parameter>
  </qmc>
</loop>
<loop max="12">
  <qmc method="linear" move="pbyp" gpu="yes">
    <!-- Specify the VMC options -->
    <parameter name="walkers">               1 </parameter>
    <parameter name="samples">           20000 </parameter>
    <parameter name="stepsbetweensamples">    1 </parameter>
    <parameter name="substeps">               5 </parameter>
    <parameter name="warmupSteps">            2 </parameter>
    <parameter name="blocks">                50 </parameter>
    <parameter name="timestep">             1.0 </parameter>
    <parameter name="usedrift">              no </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <!-- Specify the optimizer options -->
    <parameter name="MinMethod">    OneShiftOnly </parameter>
    <parameter name="minwalkers">            0.5 </parameter>
  </qmc>
</loop>
```

For each optimization step, you will see

```
The new set of parameters is valid. Updating the trial wave function!
```

or

```
The new set of parameters is not valid. Revert to the old set!
```

Occasional rejection is fine. Frequent rejection indicates potential problems and users should inspect the VMC calculation or change optimization strategy. To track the progress of optimization, using command "qmca -q ev *.scalar.dat" to look at the VMC energy and variance for each optimization step.

adaptive

The default setting of the adaptive optimizer is to construct the linear method Hamiltonian and overlap matrices explicitly and add different shifts to the Hamiltonian matrix as "stabilizers". The generalized eigenvalue problem is solved for each shift to obtain updates to the wave function parameters. Then a correlated sampling is performed for each shift's updated wave function and the initial trial wave function using the middle shift's updated wave function as the guiding function. The cost function for these wave functions is compared, and the update corresponding to the best cost function is selected. In the next iteration, the median magnitude of the stabilizers is set to that that generated the best update in the current iteration, thus adapting the magnitude of the stabilizers automatically.

When the trial wave function contains more than ten thousand parameters, constructing and storing the linear method matrices may become a memory bottleneck. To avoid explicit construction of these matrices, the adaptive optimizer implements the block linear method (BLM) approach. [15] The BLM tries to find an approximate solution $\vec{c}_{opt}$ to the standard LM generalized eigenvalue

problem by dividing the variable space into a number of blocks and making intelligent estimates for which directions within those blocks will be most important for constructing $\vec{c}_{opt}$. which is then obtained by solving a smaller, more memory-efficient eigenproblem in the basis of these supposedly important block-wise directions.

`linear` method

parameters

name	datatype	values	default	description
`max_relative_change`	real	> 0	10.0	Allowed change in cost function
`max_param_change`	real	> 0	0.3	Allowed change in wave function parameter
`shift_i`	real	> 0	0.01	Initial diagonal stabilizer added to the Hamiltonian matrix
`shift_s`	real	> 0	1.00	Initial overlap-based stabilizer added to the Hamiltonian matrix
`chase_lowest`	text	yes, no	yes	Chase the lowest eigenvector in iterative solver
`chase_closest`	text	yes, no	no	Chase the eigenvector closest to initial guess
`block_lm`	text	yes, no	no	Use block linear method
`nblocks`	integer	> 0		# of blocks in BLM
`nolds`	integer	> 0		# of old update vectors used in BLM
`nkept`	integer	> 0		# of eigenvectors to keep per block in BLM

Additional information:

- `shift_i`. This is the initial coefficient used to scale the diagonal stabilizer. More stable but slower optimization is expected with a large value. The adaptive method will automatically adjust this value after each linear method iteration.
- `shift_s`. This is the initial coefficient used to scale the overlap-based stabilizer. More stable but slower optimization is expected with a large value. The adaptive method will automatically adjust this value after each linear method iteration.
- `nblocks`. This is the number of blocks used in block LM. The amount of memory required to store LM matrices decreases with increased number of blocks. But the error introduced by BLM would increase with number of blocks.
- `nolds`. In BLM, the inter-block correlation is accounted for by including a small number of wave function update vectors outside the block. Larger `nolds` would include more inter-block correlation and more accurate results, but also higher memory requirements.
- `nkept`. This is the number of update directions retained from each block in the BLM. If all directions are retained in each block, then the BLM becomes equivalent to the standard LM. Retaining 5 or fewer directions per block is often sufficient.

Recommendations:

- Default `shift_i`, `shift_s` should be fine.

- When there are fewer than about 5,000 variables being optimized, the traditional LM is preferred as it has a lower overhead than the BLM when the number of variables is small.
- Initial experience with the BLM suggests that a few hundred blocks and a handful of `nolds` and `nkept` often provide a good balance between memory use and accuracy. In general, using fewer blocks should be more accurate but will require more memory.

```
<loop max="15">
 <qmc method="linear" move="pbyp">
   <!-- Specify the VMC options -->
   <parameter name="walkers">                1 </parameter>
   <parameter name="samples">            20000 </parameter>
   <parameter name="stepsbetweensamples">    1 </parameter>
   <parameter name="substeps">               5 </parameter>
   <parameter name="warmupSteps">            5 </parameter>
   <parameter name="blocks">                50 </parameter>
   <parameter name="timestep">             1.0 </parameter>
   <parameter name="usedrift">              no </parameter>
   <estimator name="LocalEnergy" hdf5="no"/>
   <!-- Specify the correlated sampling options and define the cost function -->
        <cost name="energy">               1.00 </cost>
        <cost name="unreweightedvariance"> 0.00 </cost>
        <cost name="reweightedvariance">   0.00 </cost>
   <!-- Specify the optimizer options -->
   <parameter name="MinMethod">adaptive</parameter>
   <parameter name="max_relative_cost_change">10.0</parameter>
   <parameter name="shift_i"> 1.00 </parameter>
   <parameter name="shift_s"> 1.00 </parameter>
   <parameter name="max_param_change"> 0.3 </parameter>
   <parameter name="chase_lowest"> yes </parameter>
   <parameter name="chase_closest"> yes </parameter>
   <parameter name="block_lm"> no </parameter>
   <!-- Specify the BLM specific options if needed
     <parameter name="nblocks"> 100 </parameter>
     <parameter name="nolds"> 5 </parameter>
     <parameter name="nkept"> 3 </parameter>
   -->
 </qmc>
</loop>
```

The adaptive optimizer is also able to optimize individual excited states directly. [14] In this case, it tries to minimize the following function:

$$\Omega[\Psi] = \frac{\langle \Psi | \omega - H | \Psi \rangle}{\left\langle \Psi | (\omega - H)^2 | \Psi \right\rangle}$$

The global minimum of this function corresponds to the state whose energy lies immediately above the shift parameter ω in the energy spectrum. For example, if ω were placed in between the ground state energy and the first excited state energy and the wave function ansatz was capable of a good description for the first excited state, then the wave function would be optimized for the first excited state. It is important to note that, if the ansatz is not capable of a good description of the excited state in question, the optimization may converge to a different state, as is known to occur in some circumstances for traditional ground state optimizations. Note also that the ground state can be targeted by this method by choosing ω to be below the ground state energy, although we should

stress that this is not the same thing as a traditional ground state optimization and will in general give a slightly different wave function. Excited state targeting requires two additional parameters, as shown in this table.

Excited State Targeting				
parameters				
name	**datatype**	**values**	**default**	**description**
`targetExcited`	text	yes, no	no	Whether to use the excited state targeting optimization
`omega`	real	real numbers	none	Energy shift used to target different excited states

Excited state recommendations:

- Due to the finite variance in any approximate wave function, it is recommended to set $\omega = \omega_0 - \sigma$, where ω_0 is placed just below the energy of the targeted state and σ^2 is the energy variance.
- In order to obtain an unbiased excitation energy, one should optimize the ground state with the excited state variational principle as well by setting `omega` below the ground state energy. Note that using the ground state variational principle for the ground state and the excited state variational principle for the excited state creates a bias in favor of the ground state.

quartic

This is an older optimizer method retained for compatibility. We recommend starting with the newest OneShiftOnly or adaptive optimizers. The quartic optimizer fits a quartic polynomial to 7 values of the cost function obtained using reweighting along chosen direction and determines the optimal move. This optimizer is very robust but a bit conservative to accept new steps especially when large parameters changes are proposed.

`linear` method				
parameters				
name	**datatype**	**values**	**default**	**description**
`bigchange`	real	> 0	50.0	Largest parameter change allowed
`alloweddifference`	real	> 0	1e-4	Allowed increased in energy
`exp0`	real	any value	-16.0	Initial value for stabilizer
`stabilizerscale`	real	> 0	2.0	Increase in value of exp0 between iterations
`nstabilizers`	integer	> 0	3	Number of stabilizers to try
`max_its`	integer	> 0	1	Number of inner loops with same samples

Additional information:

- `exp0`. It is the initial value for stabilizer (shift to diagonal of H). The actual value of stabilizer is 10^{exp0}.

Recommendations:

- For hard cases (e.g. simultaneous optimization of long MSD and 3-Body J), set exp0 to 0 and do a single inner iteration (max its=1) per sample of configurations.

```
<!-- Specify the optimizer options -->
<parameter name="MinMethod">quartic</parameter>
<parameter name="exp0">-6</parameter>
<parameter name="alloweddifference"> 1.0e-4 </parameter>
<parameter name="nstabilizers"> 1 </parameter>
<parameter name="bigchange">15.0</parameter>
```

10.2.4 General recommendations

Here are a few recommendations to make wavefunction optimization easier.

- All electron wavefunctions are typically more difficult to optimize than pseudopotential ones due to the importance of the wavefunction near the nucleus.
- Two body Jastrow contributes the largest portion of correlation energy from bare Slater determinants. For this reason, the recommended order for optimizing wavefunction components is two-body, one-body, three-body Jastrow factors and MSD coefficients.
- For two-body spline Jastrows, always start from a reasonable one. The lack of physically-motivated constraints in the functional form at large distances can cause slow convergence if starting from zero.
- One-body spline Jastrow from old calculations can be a good starting point.
- Three-body polynomial Jastrow can start from zero. It is beneficial to first optimize one-body and two-body Jastrow factors without adding three-body terms in the calculation and then add the three-body Jastrow and optimize all the three components together.

10.3 Diffusion Monte Carlo

Additional information:

- `targetwalkers`. A DMC run can be considered a restart run or a new run. A restart run is considered to be any method block beyond the first one, such as when a DMC method block that follows a VMC block. Alternatively, if the user reads in configurations from disk it is also considered a restart run. In the case of a restart run, the DMC driver will use the configurations from the previous run, and this variable will not be used. For a new run, if the number of walkers is less than the number of threads, then the number of walkers will be set equal to the number of threads.

- `blocks`. Number of blocks run during an DMC method block. A block consists of a number of DMC steps (steps), after which all the statistics accumulated in the block are written to disk.

- `steps`. Number of diffusion Monte Carlo steps in a block.

- `warmupsteps`. Warm-up steps are steps at the beginning of a DMC run in which the instantaneous average energy is used to update the trial energy. During regular steps, E_{ref} is used.

- `timestep`. The timestep determines the accuracy of the imaginary time propagator. Generally, multiple time steps are used to extrapolate to the infinite time step limit. A good range of timesteps in which to perform time step extrapolation will typically have a minimum of 99% acceptance probability for each step.

- `checkproperties`. When using particle by particle driver, this variable specifies how often to reset all the variables kept in the buffer.

- `maxcpusecs`. The default is 100 hours. Once the specified time has elapsed, the program will finalize the simulation even if not all blocks are completed.

- `energyUpdateInterval`. The default is to update the trial energy at every step. Otherwise the trial energy is updated every `energyUpdateInterval` steps.

$$E_{\text{trial}} = \text{refEnergy} + \text{feedback} \cdot (\ln \text{targetWalkers} - \ln N)$$

  where N is the current population.

- `refEnergy`. The default reference energy is taken from the VMC run that precedes the DMC run. This value is updated to the current mean whenever branching happens.

- `feedback`. Variable used to determine how strong to react to population fluctuations when doing population control. See the equation in energyUpdateInterval for more details.

- `useBareTau`. The same time step is used whether a move is rejected to not. The default is to use an effective time step when a move is rejected.

- `warmupByReconfiguration`. Warmup DMC is done with a fixed population

- `sigmaBound`. Determine the branch cutoff to limit wild weights based on the sigma and sigmaBound

- `killnode`. When running fixed-node, if a walker attempts to cross a node, the move will normally be rejected. If killnode = "yes", then walkers are destroyed when they cross a node.
- `reconfiguration`. If reconfiguration is "yes", then run with a fixed walker population using the reconfiguration technique.
- `branchInterval`. Number of steps between branching. The total number of DMC steps in a block will be BranchInterval*Steps.
- `substeps`. Same as BranchInterval.
- `nonlocalmoves`. Evaluate pseudopotentials using one of the nonlocal move algorithms such as T-moves.
  - no(default): imposes the locality approximation.
  - yes/v0: implements the algorithm in the 2006 Casula paper [25]
  - v1: implements the v1 algorithm in the 2010 Casula paper [26].
  - v2: is **not implemented** and **skipped** due to the existence of the v2 algorithm in the 2010 Casula paper [26].
  - v3: (Experimental) implements an algorithm similar to v1 but is much faster. v1 computes the transition probability before each single electron T-move selection due to the acceptance of previous T-moves. v3 mostly reuses the transition probability computed during the evaluation of non-local pseudopotentials for the local energy, namely before accepting any T-moves, and only recomputes the transition probability of the electrons within the same pseudopotential region of any electrons touched by T-moves. This is an approximation to v1, and results in a slightly different time step error, but significantly reduces the computational cost. v1 and v3 agrees at zero time step. This faster algorithm will be the topic of a paper currently in preparation.

  The v1 and v3 algorithms are size-consistent and important advances over the previous v0 non-size-consistent algorithm. Investigating the importance of size consistency is highly recommended.
- `scaleweight`. Scaling weight per Umrigar/Nightengale. CUDA only.
- `MaxAge`. Set the weight of a walker to min(currentweight,0.5) after a walker has not moved for MaxAge steps. Needed if persistent walkers appear during the course of a run.
- `MaxCopy`. When determining the number of copies of a walker to branch, set the number of copies equal to min(Multiplicity,MaxCopy).
- `fastgrad`. Calculates gradients with either the fast version or the full-ratio version.
- `maxDisplSq`. When running a DMC calculation with particle by particle, this sets the maximum displacement allowed for a single particle move. All distance displacements larger than the max is rejected. If initialized to a negative value, it becomes equal to Lattice(LR/rc).
- `sigmaBound`. Determine the branch cutoff to limit wild weights based on the sigma and sigmaBound
- `storeconfigs`. If storeconfigs is set to a non-zero value, then electron configurations during the DMC run will be saved. This option is disabled for the OpenMP version of DMC.

- `blocks_between_recompute`. See details in VMC section 10.1.
- `drift_modifier`. Drift modification schemes. 'UNR' corresponds to eq. (35) in [19]. The parameter 'a' can be modified by setting `drift_modifier_UNR_a` (default 1.0).

Listing 10.3: The following is an example of a very simple DMC section.

```
<qmc method="dmc" move="pbyp" target="e">
  <parameter name="blocks">100</parameter>
  <parameter name="steps">400</parameter>
  <parameter name="timestep">0.010</parameter>
  <parameter name="warmupsteps">100</parameter>
</qmc>
```

The time step should be adjusted for each problem individually. Please refer to the theory section on diffusion Monte Carlo.

Listing 10.4: The following is an example of running a simulation that can be restarted .

```
<qmc method="dmc" move="pbyp"  checkpoint="0">
  <parameter name="timestep">         0.004  </parameter>
  <parameter name="blocks">           100   </parameter>
  <parameter name="steps">            400    </parameter>
</qmc>
```

The checkpoint flag instructs qmcpack to output walker configurations. This also works in Variational Monte Carlo. This will output an h5 file with the name "projectid"."run-number".config.h5. Check that this file exists before attempting a restart. To read in this file for a continuation run, specify the following:

Listing 10.5: Restart (read walkers from previous run)

```
<mcwalkerset fileroot="BH.s002" version="0 6" collected="yes"/>
```

BH is the project id and s002 is the calculation number to read in the walkers from the previous run.

Combining VMC and DMC in a single run (and wave function optimization can be combined in this way too) is the standard way in which QMCPACK is typical run. There is no need to run two separate jobs, as method sections can be stacked, and walkers are transferred between them.

Listing 10.6: Combined VMC and DMC run

```
<qmc method="vmc" move="pbyp" target="e">
  <parameter name="blocks">100</parameter>
  <parameter name="steps">4000</parameter>
  <parameter name="warmupsteps">100</parameter>
  <parameter name="samples">1920</parameter>
  <parameter name="walkers">1</parameter>
  <parameter name="timestep">0.5</parameter>
</qmc>
<qmc method="dmc" move="pbyp" target="e">
  <parameter name="blocks">100</parameter>
  <parameter name="steps">400</parameter>
  <parameter name="timestep">0.010</parameter>
```

```
    <parameter name="warmupsteps">100</parameter>
  </qmc>
  <qmc method="dmc" move="pbyp" target="e">
    <parameter name="warmupsteps">500</parameter>
    <parameter name="blocks">50</parameter>
    <parameter name="steps">100</parameter>
    <parameter name="timestep">0.005</parameter>
  </qmc>
```

10.4 Reptation Monte Carlo

Like diffusion monte carlo, reptation monte carlo (RMC) is a projector based method, allowing us the ability to sample the fixed-node wavefunciton. However, by exploiting the path-integral formulation of Schrödinger's equation, the RMC algorithm can offer some advantages over traditional DMC, such as sampling both the mixed and pure fixed-node distributions in polynomial time, as well as not having population fluctuations and biases. The current implementation does not work with T-moves.

There are two adjustable parameters that affect the quality of the RMC projection: imaginary projection time β of the sampling path (commonly called a "reptile"), and the Trotter time step τ. β must be chosen to be large enough such that $e^{-\beta\hat{H}}|\Psi_T\rangle \approx |\Phi_0\rangle$ for mixed observables, and $e^{-\frac{\beta}{2}\hat{H}}|\Psi_T\rangle \approx |\Phi_0\rangle$ for pure observables. The reptile is discretized into $M = \beta/\tau$ beads at the cost of an $\mathcal{O}(\tau)$ time-step error for observables arising from the Trotter-Suzuki breakup of the short-time propagator.

The following table lists some of the more practical

Additional information:

Because of the sampling differences between DMC ensembles of walkers and RMC reptiles, the RMC block should contain the following estimator declaration to ensure correct sampling: `<estimator name="RMC" hdf5="no">`.

- `beta` or `beads`? One can specify one or the other, and from the Trotter time-step, the code will construct an appropriately sized reptile. If both are given, `beta` overrides `beads`.
- **Mixed vs. Pure observables?** Configurations sampled by the endpoints of the reptile are distributed according to the mixed distribution $f(\mathbf{R}) = \Psi_T(\mathbf{R})\Phi_0(\mathbf{R})$. Any observable that is computable within DMC and is dumped to the scalar.dat file will likewise be found in the scalar.dat file generated by RMC, except there will be an appended `_m` to alert the user that the observable was computed on the mixed distribution. For pure observables, care must be taken in the interpretation. If the observable is diagonal in the position basis (in layman's terms, if it is entirely computable from a single electron configuration $\mathbf{R}$, like the potential energy), and if the observable does not have an explicit dependence on the trial wavefunction (for example, the local energy has an explicit dependence on the trial wavefunction from the kinetic energy term), then pure estimates will be correctly computed. These observables will be found in either the scalar.dat file, where they will be appended with a `_p` suffix, or in the stat.h5 file. No mixed estimators will be dumped to the h5 file.
- **Sampling**. For pure estimators, one should check the traces of both pure and mixed estimates. Ergodicity is a known problem in RMC. Because we use the bounce algorithm, it is possible for the reptile to bounce back and forth without changing the electron coordinates of the central beads. This might not easily show up with mixed estimators, since these are

accumulated at constantly regrown ends, but pure estimates are accumulated on these central beads, and so can exhibit strong autocorrelations in pure estimate traces.

- **Propagator**: Our implementation of RMC uses Moroni's DMC link action (symmetrized), with Umrigar's scaled drift near nodes. In this regard, the propagator is identical to the one QMCPACK uses in DMC.
- **Sampling**: We use Ceperley's bounce algorithm. MaxAge is used in case the reptile gets stuck, at which point the code forces move acceptance, stops accumulating statistics, and requilibrates the reptile. Very rarely will this be required. For move proposals, we use particle-by-particle VMC a total of N_e times to generate a new all-electron configuration, at which point the action is computed and the move is either accepted or rejected.

`dmc` method

parameters

name	datatype	values	default	description
`targetwalkers`	integer	> 0	dep.	overall total number of walkers
`blocks`	integer	≥ 0	1	number of blocks
`steps`	integer	≥ 0	1	number of steps per block
`warmupsteps`	integer	≥ 0	0	number of steps for warming up
`timestep`	real	> 0	0.1	time step for each electron move
`checkproperties`	integer	≥ 0	100	number of steps between walker updates
`maxcpusecs`	real	≥ 0	3.6e5	maximum allowed walltime in seconds
`energyUpdateInterval`	integer	≥ 0	0	trial energy update interval
`refEnergy`	AU	all values	dep.	reference energy
`feedback`	double	≥ 0	1.0	population feedback on the trial energy
`useBareTau`	option	yes,no	0	do not use effective time step
`warmupByReconfiguration`	option	yes,no	0	warm up with a fixed population
`sigmaBound`	double	≥ 0	10	parameter to cutoff large weights
`killnode`	string	yes/other	no	kill or reject walkers that cross nodes
`reconfiguration`	string	yes/pure/other	no	fixed population technique
`branchInterval`	integer	≥ 0	1	branching interval
`substeps`	integer	≥ 0	1	branching interval
`nonlocalmoves`	string	yes,no,v0,v1,v3	no	run with T-moves
`scaleweight`	string	yes/other	yes	scale weights (CUDA only)
`MaxAge`	double	≥ 0	10	kill persistent walkers
`MaxCopy`	double	≥ 0	2	limit population growth
`maxDisplSq`	real	all values	-1	maximum particle move
`storeconfigs`	integer	all values	0	store configurations
`use_nonblocking`	string	yes/no	yes	using non-blocking send/recv
`blocks_between_recompute`	integer	≥ 0	dep.	wavefunction recompute frequency
`drift_modifier`	string	UNR	UNR	drift modification scheme

`rmc` method				
parameters				
name	**datatype**	**values**	**default**	**description**
`beta`	real	> 0	dep.	reptile projection time β
`timestep`	real	> 0	0.1	Trotter time step τ for each electron move
`beads`	int	> 0	1	Number of reptile beads $M = \beta/\tau$
`blocks`	integer	≥ 0	1	number of blocks
`steps`	integer	≥ 0	1	number of steps per block
`vmcpresteps`	integer	≥ 0	0	propagates reptile using VMC for given number of steps
`warmupsteps`	integer	≥ 0	0	number of steps for warming up
`MaxAge`	integer	≥ 0	0	force accept for stuck reptile if age exceeds MaxAge.

Chapter 11

Output overview

QMCPACK writes several output files which report information about the simulation (e.g. the physical properties such as the energy), as well as information about the computational aspects of the simulation, checkpoints, and restarts. The types of output files generated depend on the details of a calculation. The list below is not meant to be exhaustive, but rather to highlight some salient features of the more common filetypes. Further detail can be found in the description of the estimator one is interested in computing.

11.1 The .scalar.dat file

The most important output file is the `.scalar.dat` file. This file contains the output of block averaged properties of the system such as the local energy and other estimators. Each line corresponds to an average over $N_{walkers} * N_{steps}$ samples. By default, the quantities reported in the `.scalar.dat` file include:

LocalEnergy The local energy.

LocalEnergy_sq The local energy squared.

LocalPotential The local potential energy.

Kinetic The kinetic energy.

ElecElec The electron-electron potential energy.

IonIon The ion-ion potential energy.

LocalECP The energy due to the pseudopotential/effective core potential.

NonLocalECP The non-local energy due to the pseudopotential/effective core potential.

MPC The modified periodic coulomb potential energy.

BlockWeight The number of MC samples in the block.

BlockCPU The number of seconds to compute the block.

AcceptRatio The acceptance ratio.

QMCPACK includes a python utility, `qmca`, which can be used to process these files. Details and examples are given in chapter 12.

11.2 The .opt.xml file

This file is generated after a VMC wave function optimization, and contains the part of the input file which lists the optimized optimized jastrow factors. Conveniently, this file is already formatted such it can easily be incorporated into a DMC input file.

11.3 The .qmc.xml file

This file contains information about the computational aspects of the simulation, for example, which parts of the code are being executed when. This file is only generated in an ensemble run in which qmcpack runs multiple input files.

11.4 The .dmc.dat file

This file contains information similar to the `.scalar.dat` file, but also includes extra information about the details of a DMC calculation. For example, information about the walker population.

Index The block number.

LocalEnergy The local energy.

Variance The variance.

Weight The number of samples in the block.

NumOfWalkers The number of walkers times the number of steps.

AvgSentWalkers The average number of walkers sent. During a DMC simulation walkers may be created or destroyed. At every step, QMCPACK will do some load balancing to ensure that the walkers are evenly distributed across nodes.

TrialEnergy The trial energy. See 10.3 for an explanation of the trial energy.

DiffEff The diffusion efficiency.

LivingFraction The fraction of the walker population from the previous step that survived to the current step.

11.5 The .bandinfo.dat file

This file contains information from the trial wavefunction about the band structure of the system, including the available k-points. This can be helpful in constructing trial wavefunctions.

11.6 Checkpoint and restart files

11.6.1 The .cont.xml file

This file enables continuation of the run. It is mostly a copy of the input XML file with the series number incremented, and the `mcwalkerset` element added to read the walkers from a config file. The `.cont.xml` file is always created, but other files it depends on are only present if checkpointing is enabled.

11.6.2 The .config.h5 file

This file contains stored walker configurations.

11.6.3 The .random.h5 file

This file contains the state of the random number generator to allow restarts. (Older versions used an XML file with a suffix of `.random.xml`).

Chapter 12

Analyzing QMCPACK data

12.1 Using the qmca tool to obtain total energies and related quantities

The `qmca` tool is the primary means of analyzing scalar valued data generated by QMCPACK. Output files that contain scalar valued data are `*.scalar.dat` and `*.dmc.dat` (see chapter 11 for a detailed description of these files). Quantities that are available for analysis in `*.scalar.dat` files include the local energy and its variance, the kinetic energy, the potential energy and its components, the acceptance ratio, and the average cpu time spent per block, among others. The `*.dmc.dat` files provide information regarding the DMC walker population in addition to the local energy.

Basic capabilities of `qmca` include calculating mean values and associated error bars, processing multiple files at once in batched fashion, performing twist averaging, plotting mean values by series, and plotting traces (per block or step) of the underlying data. These capabilities are explained with accompanying examples in the following subsections.

To use `qmca`, installations of Python and NumPy must be present on the local machine. For graphical plotting, the matplotlib module must also be available.

An overview of all supported input flags to `qmca` can be obtained by typing "`qmca`" at the command line with no other inputs (also try "`qmca -x`" for a short list of examples):

```
>qmca
  no files provided, please see help info below

  Usage: qmca [options] [file(s)]

  Options:
    --version             show program's version number and exit
    -v, --verbose         Print detailed information (default=False).
    -q QUANTITIES, --quantities=QUANTITIES
                          Quantity or list of quantities to analyze.  See names
                          and abbreviations below (default=all).
    -u UNITS, --units=UNITS
                          Desired energy units.  Can be Ha (Hartree), Ry
                          (Rydberg), eV (electron volts), kJ_mol (k.
                          joule/mole), K (Kelvin), J (Joules) (default=Ha).
    -e EQUILIBRATION, --equilibration=EQUILIBRATION
                          Equilibration length in blocks (default=auto).
    -a, --average         Average over files in each series (default=False).
    -w WEIGHTS, --weights=WEIGHTS
```

```
                          List of weights for averaging (default=None).
    -b, --reblock         (pending) Use reblocking to calculate statistics
                          (default=False).
    -p, --plot            Plot quantities vs. series (default=False).
    -t, --trace           Plot a trace of quantities (default=False).
    -h, --histogram       (pending) Plot a histogram of quantities
                          (default=False).
    -o, --overlay         Overlay plots (default=False).
    --legend=LEGEND       Placement of legend.  None for no legend, outside for
                          outside legend (default=upper right).
    --noautocorr          Do not calculate autocorrelation. Warning: error bars
                          are no longer valid! (default=False).
    --noac                Alias for --noautocorr (default=False).
    --sac                 Show autocorrelation of sample data (default=False).
    --sv                  Show variance of sample data (default=False).
    -i, --image           (pending) Save image files (default=False).
    -r, --report          (pending) Write a report (default=False).
    -s, --show_options    Print user provided options (default=False).
    -x, --examples        Print examples and exit (default=False).
    --help                Print help information and exit (default=False).
    -d DESIRED_ERROR, --desired_error=DESIRED_ERROR
                          Show number of samples needed for desired error bar
                          (default=none).
    -n PARTICLE_NUMBER, --enlarge_system=PARTICLE_NUMBER
                          Show number of samples needed to maintain error bar on
                          larger system: desired particle number first, current
                          particle number second (default=none)
```

12.1.1 Obtaining a statistically correct mean and error bar

A rough guess at the mean and error bar of the local energy can be obtained in the following way with `qmca`:

```
>qmca -q e qmc.s000.scalar.dat
qmc  series 0  LocalEnergy           =  -45.876150 +/- 0.017688
```

In this case the VMC energy of an 8 atom cell of diamond is estimated to be $-45.876(2)$ Hartrees. This rough guess should not be used for production-level or publication quality estimates.

To obtain production-level results, the underlying data should first inspected visually to ensure that all data included in the averaging can be attributed to a distribution sharing the same mean. The first steps of essentially any Monte Carlo calculation (the "equilibration phase") do not belong to the equilibrium distribution and should be excluded from estimates of the mean and its error bar.

We can plot a data trace ("`-t`") of the local energy in the following way:

```
>qmca -t -q e -e 0 qmc.s000.scalar.dat
```

The "`-e 0`" part indicates that we do not want any data to be excluded from the calculation of averages initially. The resulting plot is shown in Fig. 12.1. The unphysical equilibration period is visible on the left side of the plot.

Most of the data fluctuates around a well defined mean (consistent variations around a flat line). This property is important to verify by plotting the trace for each QMC run.

If we exclude none of the equilibration data points, we get an erroneous estimate of $-45.870(2)$ Ha for the local energy:

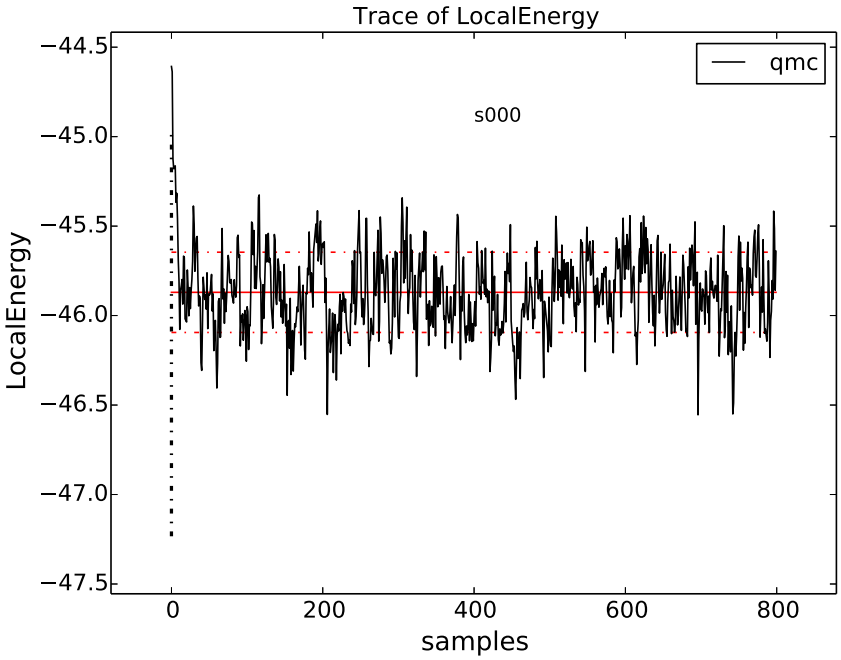


Figure 12.1: Trace of the VMC local energy for an 8 atom cell of diamond generated with `qmca`. The x-axis ("samples") refers to the VMC block index in this case.

```
>qmca -q e -e 0 qmc.s000.scalar.dat
qmc  series 0  LocalEnergy           =  -45.870071 +/- 0.018072
```

The equilibration period is typically estimated by eye, though one should check a few conservative values to ensure that the mean remains unaffected. In this dataset, the equilibration appears to have been reached after 100 samples or so. After excluding the first 100 VMC blocks from the analysis we get:

```
>qmca -q e -e 100 qmc.s000.scalar.dat
qmc  series 0  LocalEnergy           =  -45.877363 +/- 0.017432
```

This estimate (−45.877(2) Ha) differs significantly from the −45.870(2) Ha figure obtained from the full set of data, but it agrees with the rough estimate of −45.876(2) Hartrees obtained with the abbreviated command ("`qmca -q e qmc.s000.scalar.dat`"). This is because `qmca` makes a heuristic guess at the equilibration period and got it reasonably correct in this case. There are many cases where the heuristic guess fails and it should not be relied on for quality results.

We have so far obtained a statistically correct mean. To obtain a statistically correct error bar it is best to include ∼100 or more statistically independent samples. An estimate of the number of independent samples can be obtained by considering the autocorrelation time, which is essentially a measure of the number of samples that must be traversed before an uncorrelated/independent sample is reached. We can get an estimate of the autocorrelation time in the following way:

```
>qmca -q e -e 100 qmc.s000.scalar.dat --sac
qmc  series 0  LocalEnergy           =  -45.877363 +/- 0.017432    4.8
```

The flag "`--sac`" stands for (s)how (a)uto(c)orrelation. In this case the autocorrelation estimate is $4.8 \approx 5$ samples. Since the total run contained 800 samples and we have excluded 100 of them, we can estimate the number of independent samples as $(800 - 100)/5 = 140$. In this case, the error bar is expected to be estimated reasonably well.

Please keep in mind that the error bar represents the expected range of the mean with a certainty of only $\sim 70\%$, i.e. it is a one sigma error bar. The actual mean value will lie outside the range indicated by the error bar in one out of every three runs and in a set of 20 runs one value can be expected to deviate from its estimate by twice the error bar.

12.1.2 Judging wavefunction optimization

Wavefunction optimization is a highly non-linear and sometimes sensitive process. As such, there is a risk that systematic errors encountered at this stage of the QMC process can be propagated into subsequent (expensive) DMC runs unless they are guarded against with vigilance.

In this section we again consider an 8 atom cell of diamond, but now in the context of Jastrow optimization (one- and two-body terms). In optimization runs it is often preferable to use a large number of `warmupsteps` (~ 100) so that equilibration bias does not propagate into the optimization process. We can check that the added warmup has had its intended effect by again checking the local energy trace:

```
>qmca -t -q e *scalar*
```

The resulting plot can be found in Fig. 12.2. In this case sufficient `warmupsteps` were used to exit the equilibration period before samples were collected and we can proceed without using the "`-e`" option with `qmca`.

After inspecting the trace, we should inspect the text output from `qmca`, now including the total energy and its variance:

```
>qmca -q ev opt*scalar.dat
                        LocalEnergy               Variance           ratio
opt  series 0  -44.823616 +/- 0.007430   7.054219 +/- 0.041998   0.1574
opt  series 1  -45.877643 +/- 0.003329   1.095362 +/- 0.041154   0.0239
opt  series 2  -45.883191 +/- 0.004149   1.077942 +/- 0.021555   0.0235
opt  series 3  -45.877524 +/- 0.003094   1.074047 +/- 0.010491   0.0234
opt  series 4  -45.886062 +/- 0.003750   1.061707 +/- 0.014459   0.0231
opt  series 5  -45.877668 +/- 0.003475   1.091585 +/- 0.021637   0.0238
opt  series 6  -45.877109 +/- 0.003586   1.069205 +/- 0.009387   0.0233
opt  series 7  -45.882563 +/- 0.004324   1.058771 +/- 0.008651   0.0231
```

The flags "`-q ev`" requested the energy (`e`) and the variance (`v`). For this combination of quantities, a third column ("`ratio`") is printed containing the ratio of the variance and the absolute value of the local energy. The variance/energy ratio is an intensive quantity and is useful to inspect regardless of the system under study. Successful optimization of molecules and solids of any size generally result in comparable values for the variance/energy ratio.

The first line of the output ("`series 0`") corresponds to the local energy and variance of the system without a Jastrow factor (all Jastrow coefficients were initialized to zero in this case), reflecting the quality of the orbitals alone. For pseudopotential systems, a variance/energy ratio > 0.20 Ha generally indicates there is a problem with the input orbitals that needs to be resolved prior to performing wavefunction optimization.

The subsequent lines correspond to energies and variances of intermediate parameterizations of the trial wavefunction during the optimization process. The output line containing "`opt series 1`", for example, corresponds to the trial wavefunction parameterized during the "`series 0`" step

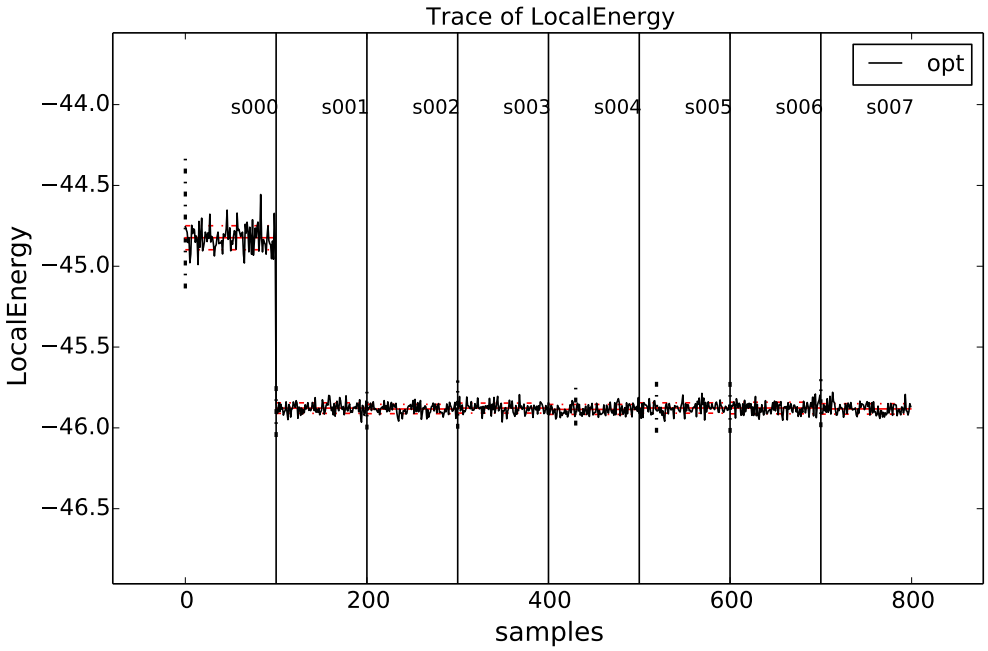


Figure 12.2: Trace of the local energy during one- and two-body Jastrow optimization for an 8 atom cell of diamond generated with `qmca`. Data for each optimization cycle (QMCPACK series) is separated by a vertical black line.

(the parameters of this wavefunction would be found in an output file matching `*s000*opt.xml`). The first thing to check about the resulting optimization is again the variance/energy ratio. For pseudopotential systems, a variance/energy ratio < 0.03 Ha is consistent with a trial wavefunction of production quality, and values of 0.01 Ha are rarely obtainable for standard Slater-Jastrow wavefunctions. By this metric, all parameterizations obtained for optimizations performed in series 0-6 are of comparable quality (note that the quality of the wavefunction obtained during optimization series 7 is effectively unknown).

A good way to further discriminate among the parameterizations is to plot the energy and variance as a function of series with `qmca`:

```
>qmca -p -q ev opt*scalar.dat
```

The "`-p`" option results in plots of means plus error bars vs. series for all requested quantities. The resulting plots for the local energy and variance are shown in Fig. 12.3. In this case the resulting energies and variances are statistically indistinguishable for all optimization cycles.

A good way to choose the optimal wavefunction for use in DMC is to select the one with lowest statistically significant energy within the set of optimized wavefunctions with reasonable variance (*e.g.* among those with variance/energy ratio < 0.03 Ha). For pseudopotential calculations, minimizing according to the total energy is recommended to reduce locality errors in DMC.

12.1.3 Judging diffusion Monte Carlo runs

Judging the quality of the DMC projection process requires more care than this needed in VMC. In order to reduce bias, a small timestep is required in the approximate projector but this also leads

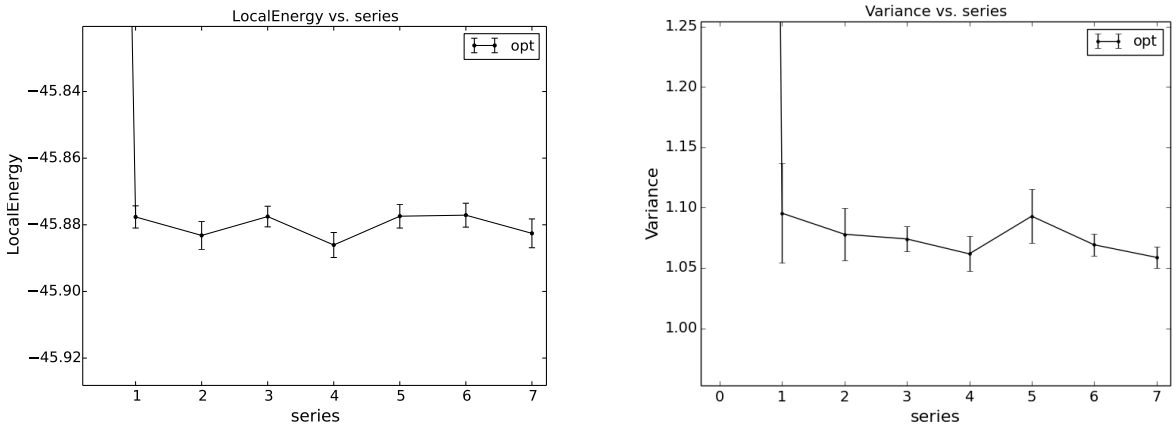


Figure 12.3: Energy and variance vs. optimization series for an 8 atom cell of diamond as plotted by `qmca`.

to slow equilibration and long autocorrelation times. Systematic errors in the projection process can also arise from statistical fluctuations due to pseudopotentials or from trial wavefunctions with larger than necessary variance.

To illustrate the problems that can arise with respect to slow equilibration and long autocorrelation times, we consider the 8 atom diamond system with VMC (200 blocks of 160 steps) followed by DMC (400 blocks of 5 steps) with a small timestep (0.002 Ha^{-1}). A good first step in assessing the quality of any DMC run is to plot the trace of the local energy:

```
>qmca -t -q e -e 0 *scalar*
```

The resulting trace plot is shown in Fig. 12.4. As always, the DMC local energy decreases exponentially away from the VMC value but in this case it takes a long time to do so. At least half of the DMC run is inefficiently consumed by equilibration. If we are not careful to inspect and remove the transient, the estimated DMC energy will be strongly biased by the transient as shown by the horizontal red line (estimated mean) in the figure. The autocorrelation time is also large ($\sim$ 12 blocks):

```
>qmca -q e -e 200 --sac *s001.scalar*
qmc  series 1  LocalEnergy           =  -46.045720 +/- 0.004813   11.6
```

Of the included 200 blocks, fewer than 20 contribute to the estimated error bar, indicating that we cannot trust the reported error bar. This can also be demonstrated directly from the data. If we halve the number of samples included to 100, we would expect from Gaussian statistics that the error bar would grow by a factor of $\sqrt{2}$, but instead we get

```
>qmca -q e -e 300 *s001.scalar*
qmc  series 1  LocalEnergy           =  -46.048537 +/- 0.009280
```

which erroneously shows an estimated increase in the error bar by a factor of about two. Overall this run is simply too short to gain meaningful information.

Consider the case where we are interested in the cohesive energy of diamond and, after having performed a timestep study of the cohesive energy, we have found that the energy difference between bulk diamond and atomic carbon converges to our required accuracy with a larger timestep of 0.01 Ha^{-1}. In a production setting, a small cell could be used to determine the appropriate timestep

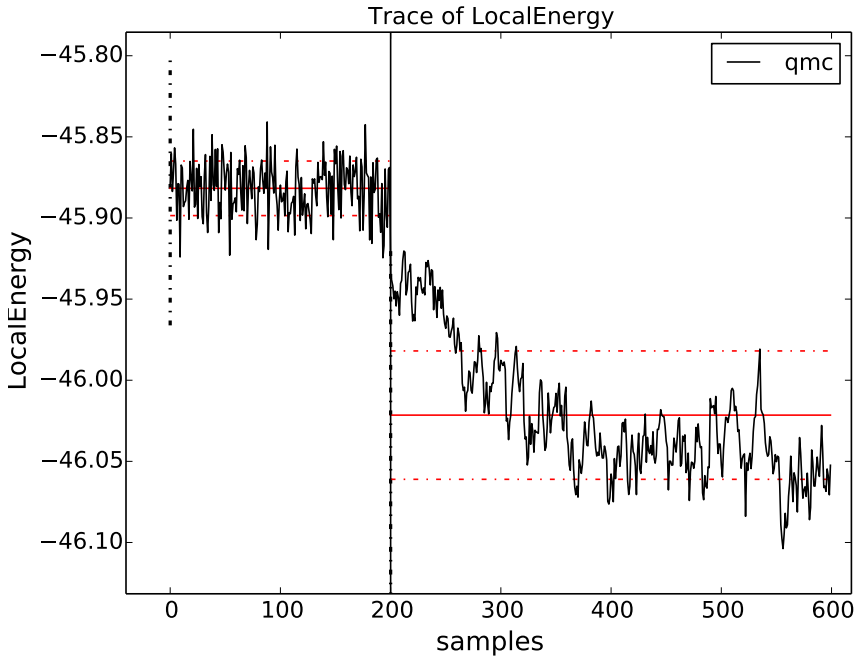


Figure 12.4: Trace of the local energy for VMC followed by DMC with a small timestep (0.002 Ha^{-1}) for an 8 atom cell of diamond generated with `qmca`.

while a larger cell would subsequently be used to obtain a converged cohesive energy, though for purposes of demonstration we still proceed with the 8 atom cell here. The new timestep of 0.01 Ha^{-1} will result in a shorter autocorrelation time than the smaller timestep used previously, but we would like to shorten the equilibration time further still. This can be achieved by using a larger timestep (say 0.02 Ha^{-1}) in a short intermediate DMC run used to walk down the transient. The rapidly achieved equilibrium with the 0.02 Ha^{-1} timestep projector will be much nearer to the 0.01 Ha^{-1} timestep one we seek than the original VMC equilibrium, and so we can expect a shortened secondary equilibration time in the production 0.01 Ha^{-1} timestep run. Note that this procedure is fully general, even if one has to deal with an even shorter timestep–*e.g.* 0.002 Ha^{-1}–for a particular problem.

We now rerun the prior example but with an intermediate DMC calculation using 40 blocks of 5 steps with a timestep of 0.02 Ha^{-1} followed by a production DMC calculation using 400 blocks of 10 steps with a timestep of 0.01 Ha^{-1}. We again plot the local energy trace using `qmca`

```
>qmca -t -q e -e 0 *scalar*
```

with the result shown in Fig. 12.5. The projection transient has been effectively contained in the short DMC run with a larger timestep. As expected, the production run contains only a short equilibration period. Removing the first 20 blocks as a precaution, we obtain an estimate of the total energy in VMC and DMC:

```
>qmca -q ev -e 20 --sac qmc.*.scalar.dat
                           LocalEnergy              Variance           ratio
qmc  series 0  -45.881042 +/- 0.001283    1.0   1.076726 +/- 0.007013    1.0   0.0235
qmc  series 1  -46.040814 +/- 0.005046    3.9   1.011303 +/- 0.016807    1.1   0.0220
qmc  series 2  -46.032960 +/- 0.002077    5.2   1.014940 +/- 0.002547    1.0   0.0220
```

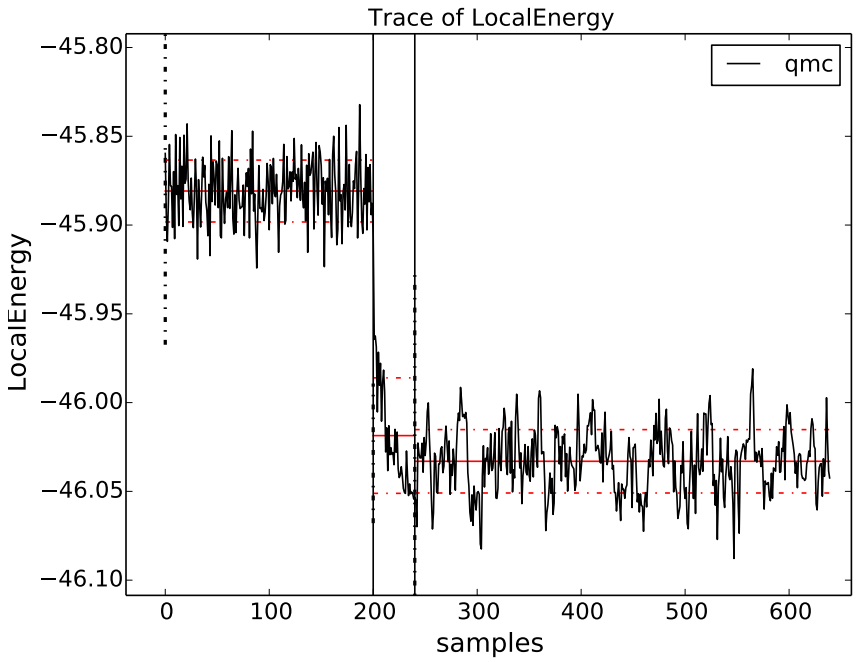


Figure 12.5: Trace of the local energy for VMC followed by a short intermediate DMC with a large timestep (0.02 Ha^{-1}) and finally a production DMC run with a timestep of 0.01 Ha^{-1}. Calculations were performed in an 8 atom cell of diamond.

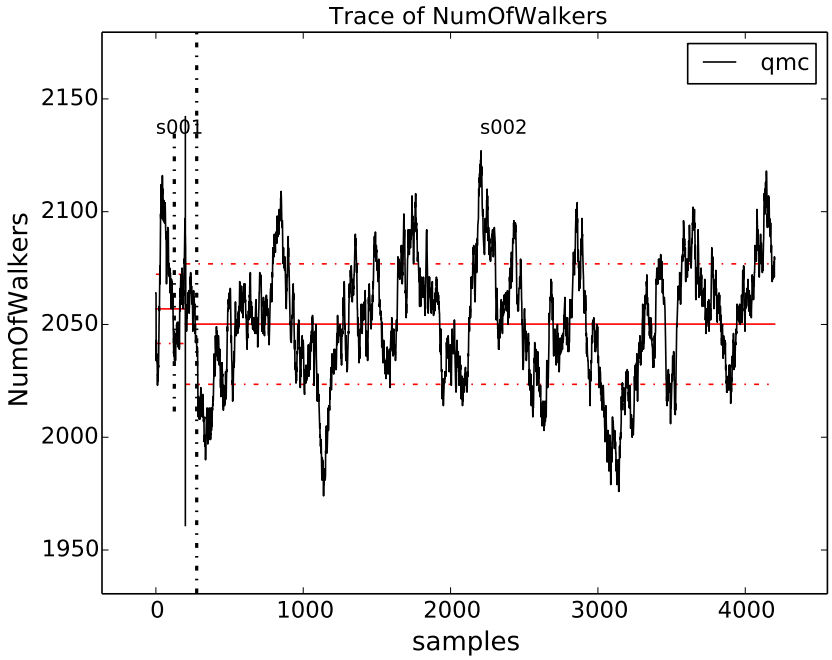


Figure 12.6: Trace of the DMC walker population for an 8 atom cell of diamond obtained with `qmca`.

Notice that the variance energy ratio in DMC (0.220 Ha) is similar to, but slightly smaller than, what is obtained with VMC (0.235 Ha). If the DMC variance/energy ratio is ever significantly larger than in VMC, this is cause to be concerned about the correctness of the DMC run. Also notice the estimated autocorrelation time ($\sim$ 5 blocks). This leaves us with an estimated $\sim$ 76 independent samples, though we should recall that the autocorrelation time is also a statistical estimate which can be improved with more data. We can gain a better estimate of the autocorrelation time by using the `*.dmc.dat` files which contain output data resolved per step rather than per block (there are 10$\times$ more steps than blocks in this example case):

```
>qmca -q ev -e 200 --sac qmc.s002.dmc.dat
                          LocalEnergy              Variance           ratio
qmc  series 2  -46.032909 +/- 0.002068   31.2   1.015781 +/- 0.002536    1.4   0.0221
```

This results in an estimated autocorrelation time of $\sim$ 31 steps, or $\sim$ 3 blocks, indicating that we actually have $\sim$ 122 independent samples which should be sufficient to obtain a trustworthy error bar. Our final DMC total energy is estimated to be $-46.0329(2)$ Ha.

Another simulation property that should be explicitly monitored is the behavior of the DMC walker population. Data regarding the walker population is contained in the `*.dmc.dat` files. In Fig. 12.6 we show the trace of the DMC walker population for the current run:

```
>qmca -t -q nw *dmc.dat
qmc  series 1  NumOfWalkers          =  2056.905405 +/- 8.775527
qmc  series 2  NumOfWalkers          =  2050.164160 +/- 4.954850
```

Following a DMC run the walker population should be checked for two qualities: 1) that the population is sufficiently large (a number $>$ 2000 is generally sufficient to reduce population control

bias) and 2) that the population fluctuates benignly around its intended target value. In this case the target walker count (provided in the input file) was 2048 and we can confirm from the plot that the population is simply fluctuating around this value. Also from the text output we have a dynamic population estimate of 2050(5) walkers. Rapid population reductions or increases–population explosions–are indicative of problems with a run. These issues sometimes result from using a considerably poor wavefunction (see comments regarding variance/energy ratio above and in the preceding subsections). QMCPACK has internal guards in place that prevent the population from exceeding certain maximum and minimum bounds, so in particularly faulty runs one might see the population "stabilize" to a constant value much larger or smaller than the target. In these cases the cause(s) for the divergent population behavior need to be investigated and resolved before proceeding further.

12.1.4 Obtaining other quantities

A number of other scalar valued quantities are available with `qmca`. To obtain text output for all quantities available, simply exclude the "`-q`" option used in the prior examples. Below is example output for a DMC calculation of the 8 atom diamond system from the `scalar.dat` file:

```
>qmca -e 20 qmc.s002.scalar.dat
qmc  series 2
  LocalEnergy           =          -46.0330 +/-           0.0021
  Variance              =            1.0149 +/-           0.0025
  Kinetic               =            33.851 +/-            0.019
  LocalPotential        =           -79.884 +/-            0.020
  ElecElec              =          -11.4483 +/-           0.0083
  LocalECP              =           -22.615 +/-            0.029
  NonLocalECP           =            5.2815 +/-           0.0079
  IonIon                =            -51.10 +/-             0.00
  LocalEnergy_sq        =           2120.05 +/-             0.19
  BlockWeight           =          20514.27 +/-            48.38
  BlockCPU              =            1.4890 +/-           0.0038
  AcceptRatio           =         0.9963954 +/-        0.0000055
  Efficiency            =             71.88 +/-             0.00
  TotalTime             =            565.80 +/-             0.00
  TotalSamples          =           7795421 +/-                0
```

Similarly, for the `dmc.dat` file we get

```
>qmca -e 20 qmc.s002.dmc.dat
qmc  series 2
  LocalEnergy           =          -46.0329 +/-           0.0020
  Variance              =            1.0162 +/-           0.0025
  TotalSamples          =           8201275 +/-                0
  TrialEnergy           =          -46.0343 +/-           0.0023
  DiffEff               =         0.9939150 +/-        0.0000088
  Weight                =            2050.23 +/-             4.82
  NumOfWalkers          =               2050 +/-                5
  LivingFraction        =           0.996427 +/-          0.000021
  AvgSentWalkers        =             0.2625 +/-           0.0011
```

Any subset of desired quantities can be obtained by using the "`-q`" option with either the full names of the quantities listed above

```
>qmca -q 'LocalEnergy Kinetic LocalPotential' -e 20 qmc.s002.scalar.dat
qmc  series 2
  LocalEnergy           =          -46.0330 +/-           0.0021
```

```
  Kinetic               =            33.851 +/-           0.019
  LocalPotential        =           -79.884 +/-           0.020
```

or with their corresponding abbreviations

```
>qmca -q ekp -e 20 qmc.s002.scalar.dat
qmc  series 2
  LocalEnergy           =          -46.0330 +/-           0.0021
  Kinetic               =            33.851 +/-           0.019
  LocalPotential        =           -79.884 +/-           0.020
```

Abbreviations for each quantity can be found by typing `qmca` at the command line with no other input. A current list is provided below:

```
  Abbreviations and full names for quantities:
    ar              = AcceptRatio
    bc              = BlockCPU
    bw              = BlockWeight
    ce              = CorrectedEnergy
    de              = DiffEff
    e               = LocalEnergy
    ee              = ElecElec
    eff             = Efficiency
    ii              = IonIon
    k               = Kinetic
    kc              = KEcorr
    l               = LocalECP
    le2             = LocalEnergy_sq
    mpc             = MPC
    n               = NonLocalECP
    nw              = NumOfWalkers
    p               = LocalPotential
    sw              = AvgSentWalkers
    te              = TrialEnergy
    ts              = TotalSamples
    tt              = TotalTime
    v               = Variance
    w               = Weight
```

Please see the output overview for `scalar.dat` (Sec. 11.1) and `dmc.dat` (Sec. 11.4) for more information about these quantities. The data analysis aspects for these quantities is essentially the same as for the local energy as covered in the preceding subsections. Quantities that do not belong to an equilibrium distribution (*e.g.* `BlockCPU`) are somewhat different, though they still exhibit statistical fluctuations.

12.1.5 Processing multiple files

Batch file processing is a common use case for `qmca`. If we consider an "equation of state" calculation involving the 8 atom diamond cell we have used so far, we might be interested in the total energy for the various supercell volumes along the trajectory from compression to expansion. After checking the traces ("`qmca -t -q e scale_*/vmc/*scalar*`") to settle on a sensible equilibration cutoff as discussed in the preceding subsections we can obtain the total energies all at once:

```
>qmca -q ev -e 40 scale_*/vmc/*scalar*
                           LocalEnergy               Variance           ratio
scale_0.80/vmc/qmc  series 0 -44.670984 +/- 0.006051  2.542384 +/- 0.019902  0.0569
scale_0.82/vmc/qmc  series 0 -44.982818 +/- 0.005757  2.413011 +/- 0.022626  0.0536
```

```
scale_0.84/vmc/qmc  series 0 -45.228257 +/- 0.005374  2.258577 +/- 0.019322  0.0499
scale_0.86/vmc/qmc  series 0 -45.415842 +/- 0.005532  2.204980 +/- 0.052978  0.0486
scale_0.88/vmc/qmc  series 0 -45.570215 +/- 0.004651  2.061374 +/- 0.014359  0.0452
scale_0.90/vmc/qmc  series 0 -45.683684 +/- 0.005009  1.988539 +/- 0.018267  0.0435
scale_0.92/vmc/qmc  series 0 -45.751359 +/- 0.004928  1.913282 +/- 0.013998  0.0418
scale_0.94/vmc/qmc  series 0 -45.791622 +/- 0.005026  1.843704 +/- 0.014460  0.0403
scale_0.96/vmc/qmc  series 0 -45.809256 +/- 0.005053  1.829103 +/- 0.014536  0.0399
scale_0.98/vmc/qmc  series 0 -45.806235 +/- 0.004963  1.775391 +/- 0.015199  0.0388
scale_1.00/vmc/qmc  series 0 -45.783481 +/- 0.005293  1.726869 +/- 0.012001  0.0377
scale_1.02/vmc/qmc  series 0 -45.741655 +/- 0.005627  1.681776 +/- 0.011496  0.0368
scale_1.04/vmc/qmc  series 0 -45.685101 +/- 0.005353  1.682608 +/- 0.015423  0.0368
scale_1.06/vmc/qmc  series 0 -45.615164 +/- 0.005978  1.652155 +/- 0.010945  0.0362
scale_1.08/vmc/qmc  series 0 -45.543037 +/- 0.005191  1.646375 +/- 0.013446  0.0361
scale_1.10/vmc/qmc  series 0 -45.450976 +/- 0.004794  1.707649 +/- 0.048186  0.0376
scale_1.12/vmc/qmc  series 0 -45.371851 +/- 0.005103  1.686997 +/- 0.035920  0.0372
scale_1.14/vmc/qmc  series 0 -45.265490 +/- 0.005311  1.631614 +/- 0.012381  0.0360
scale_1.16/vmc/qmc  series 0 -45.161961 +/- 0.004868  1.656586 +/- 0.014788  0.0367
scale_1.18/vmc/qmc  series 0 -45.062579 +/- 0.005971  1.671998 +/- 0.019942  0.0371
scale_1.20/vmc/qmc  series 0 -44.960477 +/- 0.004888  1.651864 +/- 0.009756  0.0367
```

In this case, we are using a Jastrow factor optimized only at the equilibrium geometry ("`scale_1.00`") but with radial cutoffs restricted to the Wigner-Seitz radius of the most compressed supercell ("`scale_0.80`") to avoid introducing wavefunction cusps at the cell boundary (QMCPACK would have aborted with a warning in this case, had we tried). It is clear that this restricted Jastrow factor is not an optimal choice as it yields variance/energy ratios between 0.036 and 0.057 Ha. This issue is largely a result of our undersized (8 atom) supercell and larger cells should always be used in real production calculations.

Batch processing is also possible for multiple quantities. If multiple quantities are requested, an additional line is inserted to separate results from different runs:

```
>qmca -q 'e bc eff' -e 40 scale_*/vmc/*scalar*
scale_0.80/vmc/qmc  series 0
  LocalEnergy           =          -44.6710 +/-           0.0061
  BlockCPU              =           0.02986 +/-          0.00038
  Efficiency            =          38104.00 +/-             0.00

scale_0.82/vmc/qmc  series 0
  LocalEnergy           =          -44.9828 +/-           0.0058
  BlockCPU              =           0.02826 +/-          0.00013
  Efficiency            =          44483.91 +/-             0.00

scale_0.84/vmc/qmc  series 0
  LocalEnergy           =          -45.2283 +/-           0.0054
  BlockCPU              =           0.02747 +/-          0.00030
  Efficiency            =          52525.12 +/-             0.00

scale_0.86/vmc/qmc  series 0
  LocalEnergy           =          -45.4158 +/-           0.0055
  BlockCPU              =           0.02679 +/-          0.00013
  Efficiency            =          50811.55 +/-             0.00

scale_0.88/vmc/qmc  series 0
  LocalEnergy           =          -45.5702 +/-           0.0047
  BlockCPU              =           0.02598 +/-          0.00015
  Efficiency            =          74148.79 +/-             0.00

scale_0.90/vmc/qmc  series 0
```

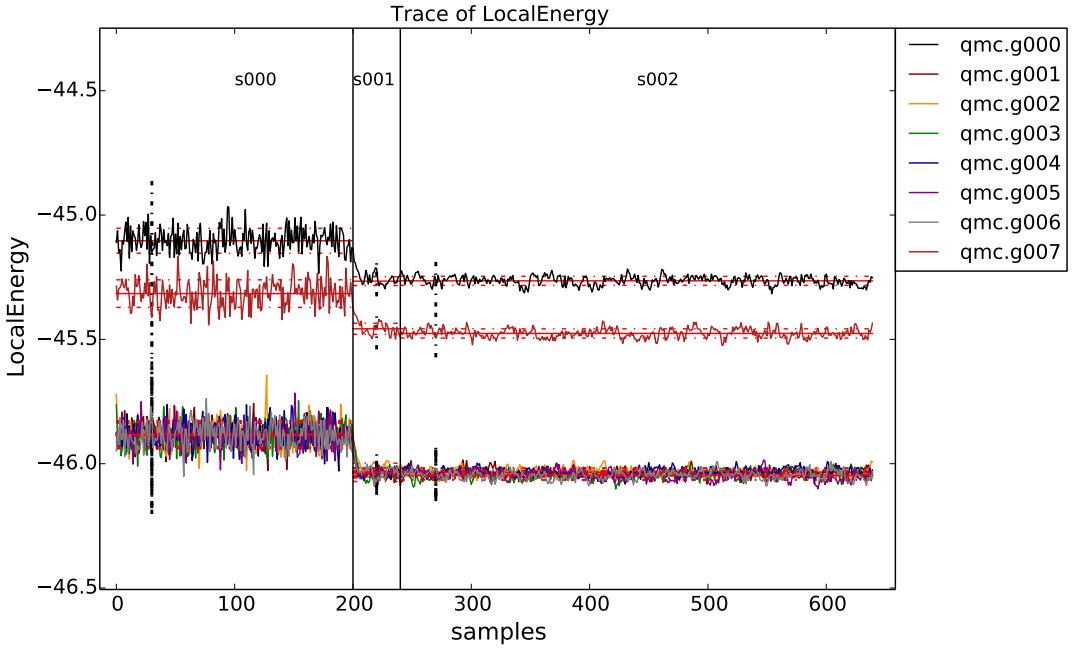


Figure 12.7: Overlapped energy traces from VMC to DMC for an 8 supercell of diamond obtained with `qmca`. Data for each twist appears in a different color.

```
  LocalEnergy            =          -45.6837 +/-           0.0050
  BlockCPU               =           0.02527 +/-          0.00011
  Efficiency             =          65714.98 +/-             0.00

...
```

12.1.6 Twist averaging

Twist averaging can be performed straightforwardly for any output quantity listed in Sec. 12.1.4 with `qmca`. We illustrate these capabilities by repeating the 8 atom diamond DMC runs performed in Sec. 12.1.3 at eight real valued supercell twist angles (a $2 \times 2 \times 2$ Monkhorst-Pack grid centered at the Γ-point). Data traces for each twist can be overlapped on the same plot:

```
>qmca -to -q e -e '30 20 30' *scalar* --legend outside
```

The "`-o`" option requests the plots be overlapped; eight separate plots would be generated otherwise. The equilibration input "`-e '30 20 30'`" cuts out from the analyzed data the first 30 blocks for series 0 (VMC), 20 blocks for series 1 (intermediate DMC), and 30 blocks for series 2 (production DMC). The resulting plot is shown in Fig. 12.7

Twist averaging is performed by providing the "`-a`" option. If provided on its own, uniform weights are applied to each twist angle. To obtain a trace plot with twist averaging enforced, use a command similar to the following:

```
>qmca -a -t -q e -e '30 20 30' *scalar*
```

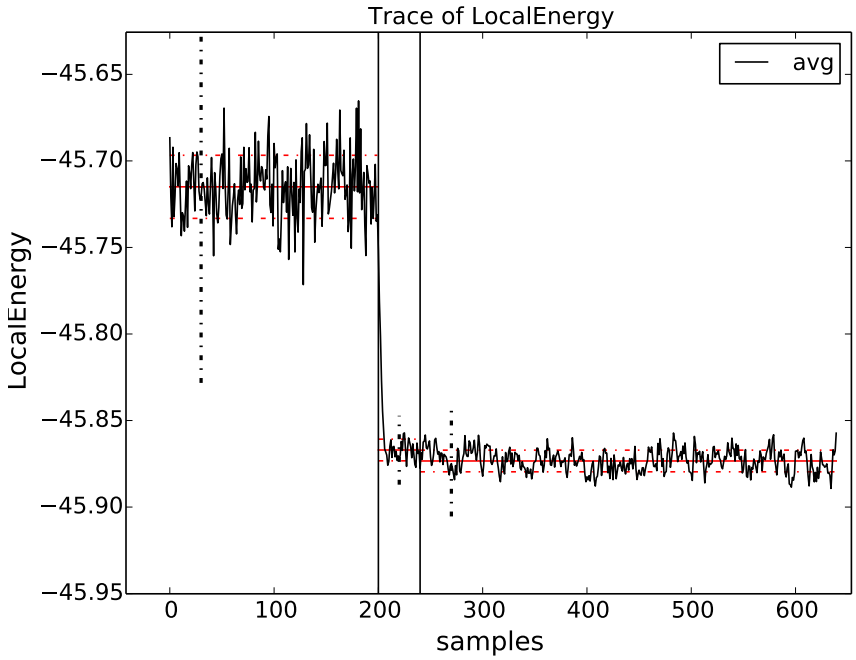


Figure 12.8: Twist averaged energy trace from VMC to DMC for an 8 supercell of diamond obtained with `qmca`.

The resulting plot is shown in Fig. 12.8. As can be seen from the trace plot, the chosen equilibration lengths are appropriate and we proceed to obtain the twist averaged total energy from the `scalar.dat` files

```
>qmca -a -q ev -e 30 --sac *s002.scalar*
                           LocalEnergy              Variance           ratio
avg  series 2  -45.873369 +/- 0.000753    5.3   1.028751 +/- 0.001056    1.3   0.0224
```

and also from the `dmc.dat` files

```
>qmca -a -q ev -e 300 --sac *s002.dmc*
                           LocalEnergy              Variance           ratio
avg  series 2  -45.873371 +/- 0.000741   30.5   1.028843 +/- 0.000972    1.6   0.0224
```

yielding a twist averaged total energy of −45.8733(8) Ha.

As can be seen from the Fig. 12.7, some of the twist angles are degenerate. This is seen more clearly in the text output:

```
>qmca -q ev -e 30 *s002.scalar*
                           LocalEnergy              Variance           ratio
qmc.g000  series 2  -45.264510 +/- 0.001942   1.057065 +/- 0.002318   0.0234
qmc.g001  series 2  -46.035511 +/- 0.001806   1.015992 +/- 0.002836   0.0221
qmc.g002  series 2  -46.035410 +/- 0.001538   1.015039 +/- 0.002661   0.0220
qmc.g003  series 2  -46.047285 +/- 0.001898   1.018219 +/- 0.002588   0.0221
qmc.g004  series 2  -46.034225 +/- 0.002539   1.013420 +/- 0.002835   0.0220
qmc.g005  series 2  -46.046731 +/- 0.002963   1.018337 +/- 0.004109   0.0221
qmc.g006  series 2  -46.047133 +/- 0.001958   1.021483 +/- 0.003082   0.0222
qmc.g007  series 2  -45.476146 +/- 0.002065   1.070456 +/- 0.003133   0.0235
```

The degenerate twists grouped by set are $\{0\}$, $\{1, 2, 4\}$, $\{3, 5, 6\}$, $\{7\}$.

Alternatively, the run could have been performed at *only* the four unique (irreducible) twist angles. We will emulate this situation by analyzing data for twists 0, 1, 3, and 7 only. In a production setting with irreducibly weighted twists, run would be performed on these twists alone; we reuse the uniform twist data for illustration purposes only.

We can use `qmca` to perform twist averaging with different weights applied to each twist

```
>qmca -a -w '1 3 3 1' -q ev -e 30 *g000*2*sc* *g001*2*sc* *g003*2*sc* *g007*2*sc*
                           LocalEnergy               Variance           ratio
avg  series 2  -45.873631 +/- 0.001044   1.028769 +/- 0.001520   0.0224
```

yielding a total energy value of $-45.874(1)$ Ha, in agreement with the uniform weighted twist average performed above.

The decision of whether or not to perform irreducible weighted twist averaging should be made on the basis of efficiency. The relative efficiency of irreducible vs. uniform weighted twist averaging depends on the irreducible weights and the ratio of the lengths of the available sampling and equilibration periods. A formula for the relative efficiency of these two cases is derived and discussed in more detail in Appendix A.

12.1.7 Setting output units

Estimates outputted by `qmca` are in Hartree units by default. The output units for energetic quantities can be changed by using the "`-u`" option.

Energy in Hartrees:

```
>qmca -q e -u Ha -e 20 qmc.s002.scalar.dat
qmc  series 2  LocalEnergy           =  -46.032960 +/- 0.002077
```

Energy in electron volts:

```
>qmca -q e -u eV -e 20 qmc.s002.scalar.dat
qmc  series 2  LocalEnergy           =  -1252.620565 +/- 0.056521
```

Energy in Rydbergs:

```
>qmca -q e -u rydberg -e 20 qmc.s002.scalar.dat
qmc  series 2  LocalEnergy           =  -92.065919 +/- 0.004154
```

Energy in kilojoules per mole:

```
>qmca -q e -u kj_mol -e 20 qmc.s002.scalar.dat
qmc  series 2  LocalEnergy           =  -120859.512998 +/- 5.453431
```

12.1.8 Speeding up trace plotting

When working with many files or files with many entries, `qmca` may take a long time to produce plots. The time delay is actually due to the autocorrelation time estimate used to calculate error bars. The calculation time for the autocorrelation scales as $\mathcal{O}(M^2)$, with M being the number of statistical samples. If you are only interested in plotting traces and not in the estimated error bars, the autocorrelation time estimation can be turned off with the "`--noac`" option:

```
>qmca -t -q e -e 20 --noac qmc.s002.scalar.dat
```

Please note that the resulting error bars printed to the console will be underestimated and are not meaningful. Do *not* use "`--noac`" in conjunction with the "`-p`" plotting option as these plots are of no use without meaningful error bars.

12.1.9 Short usage examples

Plotting a trace of the local energy:

```
>qmca -t -q e *scalar*
```

Applying an equilibration cutoff to VMC data (series 0):

```
>qmca -q e -e 30 *s000.scalar*
```

Applying the same equilibration cutoff to VMC and DMC data (series 0, 1, 2):

```
>qmca -q e -e 20 *scalar*
```

Applying different equilibration cutoffs to VMC and DMC data (series 0, 1, 2):

```
>qmca -q e -e '30 20 40' *scalar*
```

Obtaining the energy, variance, and variance/energy ratio for all series:

```
>qmca -q ev -e 30 *scalar*
```

Overlaying plots of mean + error bar for energy and variance for separate two- and three- body Jastrow optimization runs:

```
>qmca -po -q ev ./optJ2/*scalar* ./optJ3/*scalar*
```

Obtaining the acceptance ratio:

```
>qmca -q ar -e 30 *scalar*
```

Obtaining the average DMC walker population:

```
>qmca -q nw -e 400 *s002.dmc.dat
```

Obtaining the Monte Carlo efficiency:

```
>qmca -q eff -e 30 *scalar*
```

Obtaining the total wallclock time per series:

```
>qmca -q tt -e 0 *scalar*
```

Obtaining the average wallclock time spent per block:

```
>qmca -q bc -e 0 *scalar*
```

Obtaining a subset of desired quantities:

```
>qmca -q 'e v ar eff' -e 30 *scalar*
```

Obtaining all available quantities:

```
>qmca -e 30 *scalar*
```

Obtaining the twist averaged total energy with uniform weights:

```
>qmca -a -q e -e 40 *g*s002.scalar.dat
```

Obtaining the twist averaged total energy with specific weights:

```
>qmca -a -w '1 3 3 1' -q e -e 40 *g*s002.scalar.dat
```

Obtaining the local, kinetic, and potential energies in eV:

```
>qmca -q ekp -e 30 -u eV *scalar*
```

12.1.10 Production quality checklist

1. Inspect the trace plots ("`-t`" option) for any oddities in the data. Typical behavior is a short equilibration period followed by benign fluctuations around a clear mean value. There should not be any large spikes in the data. This applies to *all* runs (VMC, optimization, DMC, etc.).

2. Remove all equilibration steps ("`-e`" option) from the data by inspecting the trace plot.

3. Check the quality of the orbitals (standalone Jastrow-less VMC or sometimes the first `scalar` file produced during optimization) by inspecting the variance/energy ratio "`qmca -q ev *scalar*`". For pseudopotential systems without a Jastrow, the variance/energy ratio should not exceed 0.2 Ha, otherwise there is a problem with the orbitals.

4. Check the quality of the optimized Jastrow factor by inspecting the variance/energy ratio. For pseudopotential systems with a Jastrow, the variance/energy ratio should not exceed 0.04 Ha for pseudopotential systems. A good Jastrow is indicated by a variance/energy ratio in the range $0.01 - 0.03$ Ha. A value less than 0.01 Ha is difficult to achieve.

5. Confirm that the optimization has converged by plotting the energy and variance vs. optimization series ("`qmca -p -q ev *scalar*`"). Do not assume that optimization has converged in only a few cycles. Use at least 10 cycles of with around 100,000 samples unless you already have experience with the system in question.

6. Optimize Jastrow factors according to energy minimization to reduce locality errors arising from the use of non-local pseudopotentials in DMC. A good approach is to optimize with a few cycles of variance minimization followed by several cycles of energy minimization.

7. Occasionally try optimizing with more samples and/or cycles to see if improved results are obtained.

8. If using a B-spline representation of the orbitals, converge the VMC energy and variance with respect to the mesh size (controlled via meshfactor). This is best done in the presence of any Jastrow factor to reduce noise. Consider using the hybrid LMTO representation of the orbitals as this can reduce both the VMC/DMC variance and DMC timestep error in addition to saving memory.

9. Check the variance/energy ratio of all production VMC and DMC calculations. In all cases the DMC ratio should be slightly less than the VMC one and both should abide the guidelines above, *i.e.* the ratio should be less than 0.04 Ha for pseudopotential systems. The production ratio should also be consistent with what is observed during wavefunction optimization.

10. Be aware of population control bias in DMC. Run with a population of ~ 2000 or greater. Occasionally repeat a run using a larger population to explicitly confirm that population control bias is small.

11. Check the stability of the DMC walker population by plotting the trace of the population size ("`qmca -t -q nw *dmc.dat`"). Verify that the average walker population is consistent with the requested value provided in the input.

12. In DMC, perform a timestep study to either 1) obtain extrapolated results, or 2) obtain a timestep for future production where an energy difference shows convergence (*e.g.* a band gap or defect formation energy). For pseudopotential systems, converged timesteps for many

systems are in the range $0.002 - 0.01$ Ha^{-1}, but the actual converged timestep must be explicitly checked.

13. In periodic systems, converge the total energy with respect to the size of the twist/k-point grid. Results for smaller systems can easily be transferred to larger ones (*e.g.* a 2x2x2 twist grid in a 2x2x2 tiled cell is equivalent to a 1x1x1 twist grid in a 4x4x4 tiled cell).

14. In periodic systems, perform finite size extrapolation including two body corrections (needed for cohesive energy/phase stability studies) unless it can be shown that finite size effects cancel for the energy difference in question (*e.g.* some defect formation energies).

12.2 Using the qmc-fit tool for statistical timestep extrapolation and curve fitting

The `qmc-fit` tool is used to provide statistical estimates of curve fitting parameters based on QMCPACK data. While `qmc-fit` will eventually support many types of fitted curves (*e.g.* Morse potential binding curves, various equation of state fitting curves, etc.), it is currently limited to estimating fitting parameters related to timestep extrapolation.

12.2.1 The jack-knife statistical technique

The `qmc-fit` tool obtains estimates of fitting parameter means and associated error bars via the "jack-knife" technique. The jack-knife method is a powerful and general tool to obtain meaningful error bars for any quantity that is related in a non-linear fashion to an underlying set of statistical data. For this reason, we give a brief overview of the jack-knife technique before proceeding with usage instructions for the `qmc-fit` tool.

Consider N statistical variables $\{x_n\}_{n=1}^{N}$ that have been outputted by one or more simulation runs. If we have M samples of each of the N variables, then the mean values of each these variables can be estimated in the standard way, i.e. $\bar{x}_n \approx \frac{1}{M}\sum_{m=1}^{M} x_{nm}$.

Suppose we are interested in P statistical quantities $\{y_p\}_{p=1}^{P}$ that are related to the original N variables by a known multidimensional function F:

$$\begin{aligned} y_1, y_2, \ldots, y_P = F(x_1, x_2, \ldots, x_N) \quad \text{or} \\ \vec{y} = F(\vec{x}) \end{aligned} \tag{12.1}$$

The relationship implied by F is completely general. For example the $\{x_n\}$ might be elements of a matrix with $\{y_p\}$ being the eigenvalues, or F might be a fitting procedure for N energies at different timesteps with P fitting parameters. An approximate guess at the mean value of $\vec{y}$ can be obtained by evaluating F at the mean value of $\vec{x}$ (i.e. $F(\bar{x}_1 \ldots \bar{x}_N)$), but with this approach we have no way to estimate the statistical error bar of any $\bar{y}_p$.

In the jack-knife procedure, the statistical variability intrinsic to the underlying data $\{x_n\}$ is used to obtain estimates of the mean and error bar of $\{y_p\}$. We first construct a new set of x statistical data by taking the average over all samples but one:

$$\tilde{x}_{nm} = \frac{1}{N-1}(N\bar{x}_n - x_{nm}) \qquad m \in [1, M] \tag{12.2}$$

The result is a distribution of approximate x mean values. These are used to construct a distribution of approximate means for y:

$$\tilde{y}_{1m}, \ldots, \tilde{y}_{Pm} = F(\tilde{x}_{1m}, \ldots, \tilde{x}_{Nm}) \qquad m \in [1, M] \tag{12.3}$$

Estimates for the mean and error bar of the quantities of interest can finally be obtained using the formulas below:

$$\bar{y}_p = \frac{1}{M}\sum_{m=1}^{M} \tilde{y}_{pm} \tag{12.4}$$

$$\sigma_{y_p} = \sqrt{\frac{M-1}{M}\left(\sum_{m=1}^{M} \tilde{y}_{pm}^2 - M\bar{y}_p^2\right)} \tag{12.5}$$

12.2.2 Performing timestep extrapolation

In this section, we use a 32 atom supercell of MnO as an example system for timestep extrapolation. Data for this system has been collected in DMC using the following sequence of timesteps: 0.04, 0.02, 0.01, 0.005, 0.0025, 0.00125 Ha^{-1}. For a typical production pseudopotential study, timesteps in the range $0.02 - 0.002$ Ha^{-1} are usually sufficient and it is recommended to increase the number of steps/blocks by a factor of two when the timestep is halved. In order to perform accurate statistical fitting, we must first understand the equilibration and autocorrelation properties of the inputted local energy data. After plotting the local energy traces (`qmca -t -q e -e 0 ./qmc*/*scalar*`) it is clear that an equilibration period of 30 blocks is reasonable. Approximate autocorrelation lengths are also obtained with `qmca`:

```
>qmca -e 30 -q e --sac ./qmc*/qmc.g000.s002.scalar.dat
./qmc_tm_0.00125/qmc.g000 series 2 LocalEnergy = -3848.234513 +/- 0.055754  1.7
./qmc_tm_0.00250/qmc.g000 series 2 LocalEnergy = -3848.237614 +/- 0.055432  2.2
./qmc_tm_0.00500/qmc.g000 series 2 LocalEnergy = -3848.349741 +/- 0.069729  2.8
./qmc_tm_0.01000/qmc.g000 series 2 LocalEnergy = -3848.274596 +/- 0.126407  3.9
./qmc_tm_0.02000/qmc.g000 series 2 LocalEnergy = -3848.539017 +/- 0.075740  2.4
./qmc_tm_0.04000/qmc.g000 series 2 LocalEnergy = -3848.976424 +/- 0.075305  1.8
```

The autocorrelation must be removed from the data prior to jack-knifing and so we will reblock the data by a factor of 4.

The `qmc-fit` tool can be used in the following way to obtain a linear timestep fit of the data:

```
>qmc-fit ts -e 30 -b 4 -s 2 -t '0.00125 0.0025 0.005 0.01 0.02 0.04' ./qmc*/*scalar*
fit function  : linear
fitted formula: (-3848.193 +/- 0.037) + (-18.95 +/- 1.95)*t
intercept     : -3848.193 +/- 0.037  Ha
```

The input arguments are as follows: `ts` indicates we are performing a timestep fit, "`-e 30`" is the equilibration period removed from each set of scalar data, "`-b 4`" indicates the data will be reblocked by a factor of 4 (*e.g.* a file containing 400 entries will be block averaged into a new set of 100 prior to jack-knife fitting), "`-s 2`" indicates that the timestep data begins with series 2 (scalar files matching `*s000*` or `*s001*` are to be excluded), and "`-t '0.00125 0.0025 0.005 0.01 0.02 0.04' `" provides a list of timestep values corresponding to the inputted scalar files. The "`-e`" and "`-b`" options can receive a list of file-specific values (same format as "`-t`") if desired. As can be seen from the text output, the parameters for the linear fit are printed with error bars obtained with jack-knife resampling and the zero timestep "intercept" is $-3848.19(4)$ Ha. In addition to text output, the command above will result in a plot of the fit with the zero timestep value shown as a red dot, as shown in the left panel of Fig. 12.9.

Different fitting functions are supported via the "`-f`" option. Currently supported options include `linear` ($a + bt$), `quadratic` ($a + bt + ct^2$), and `sqrt` ($a + b\sqrt{t} + ct$). Results for a quadratic fit are shown below as well as in the right panel of Fig. 12.9.

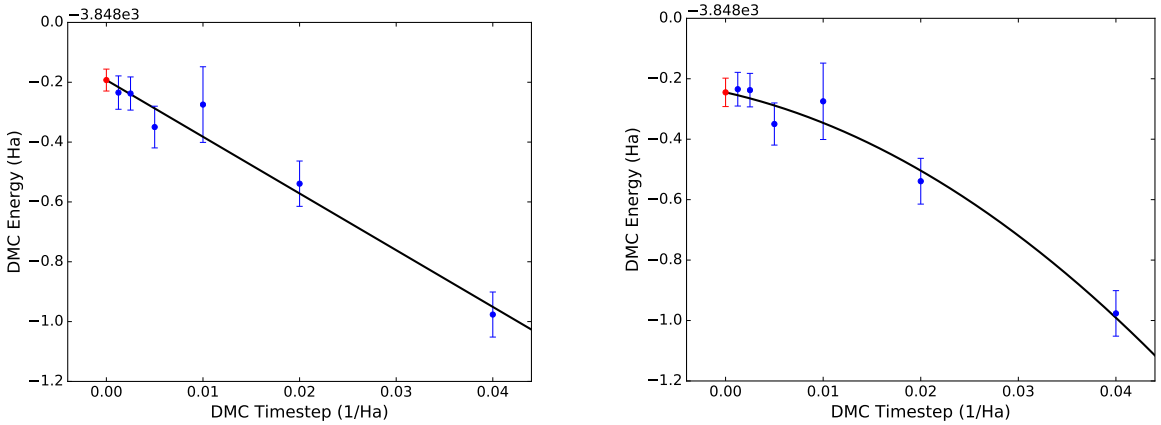


Figure 12.9: Linear (left) and quadratic (right) timestep fits to DMC data for a 32 atom supercell of MnO obtained with `qmc-fit`. Zero timestep estimates are indicated by the red data point on the left side of either panel.

```
>qmc-fit ts -f quadratic -e30 -b4 -s2 -t '0.00125 0.0025 0.005 0.01 0.02 0.04'
    ./qmc*/*scalar*
fit function  : quadratic
fitted formula: (-3848.245 +/- 0.047) + (-7.25 +/- 8.33)*t + (-285.00 +/- 202.39)*t^2
intercept     : -3848.245 +/- 0.047  Ha
```

In this case we find a zero timestep estimate of $-3848.25(5)$ Ha^{-1}. A timestep of 0.04 Ha^{-1} might be on the large side to include in timestep extrapolation and it is likely to have an outsize influence in the case of linear extrapolation. Upon excluding this point, linear extrapolation yields a zero timestep value of $-3848.22(4)$ Ha^{-1}. It should be noted that quadratic extrapolation can result in intrinsically larger uncertainty in the extrapolated value. For example, when the 0.04 Ha^{-1} point is excluded the uncertainty grows by 50% and we obtain an estimated value of $-3848.28(7)$ instead.

12.3 Densities and spin-densities

TBD.

Chapter 13

Periodic LCAO for solids

13.1 Introduction

QMCPACK implements linear combination of atomic orbitals (LCAO) and Gaussian basis sets in periodic boundary conditions. This method uses orders of magnitude less memory than the real space spline wavefunction. While the spline scheme enables very fast evaluation of the wavefunction, it may require too much on-node memory for a large complex cell. The periodic Gaussian evaluation provides a fallback that will definitely fit in available memory, but at significantly increased computational expense. Well designed Gaussian basis sets should be used to accurately represent the wavefunction, typically including both diffuse and high angular momentum functions.

The initial implementation is limited to the Γ-point using trial wavefunctions generated by PySCF[36], but other codes such as Crystal can be interfaced on request.

LCAO schemes use physical considerations to construct a highly efficient basis set compared to plane waves. Typically only a few tens of basis functions per atom are required compared to thousands of plane-waves. Many forms of LCAO schemes exist and are being implemented in QMCPACK. The details of the already implemented methods will be described in the following section of the manual.

Gaussian Trial Orbitals (GTOs): The Gaussian basis functions follow a radial-angular decomposition

$$\phi(\mathbf{r}) = R_l(r)Y_{lm}(\theta,\phi) \tag{13.1}$$

where $Y_{lm}(\theta,\phi)$ is a spherical harmonic, l and m are the angular momentum and its z component, and r,θ,ϕ are spherical coordinates. In practice they are atom centered and the l expansion typically includes 1-3 additional channels compared to the formally occupied states of the atom. e.g. 4-6 for a nickel atom with occupied s, p, and d electron shells.

The evaluation of GTOs within PBC differs slightly from evaluating GTOs in Open Boundary Conditions (OBC). The orbitals are evaluated at a distance r in the primitive cell (similar to OBC) and then the contributions of the periodic images are added by evaluating the orbital at a distance $r+T$ where T is a translation of the cell lattice vector. This requires loops over the periodic images until the contributions are orbitals Φ. In the current implementation, the number of periodic images is an input parameter named *PBCimages*, which takes three integers corresponding to the number of periodic images along the supercell axes (X, Y and Z axes for a cubic cell). By default these parameters are set to *PBCimages= 5 5 5* but they **require manual convergence checks**. Convergence checks can be performed by checking the total energy convergence with respect to *PBCimages*, similar to checks performed for plane wave cutoff energy and b-spline grids. Use of diffuse Gaussians may require these parameters to be increased, while sharply localized Gaussians

may permit a decrease. The cost of evaluating the wavefunction increases sharply as *PBCimages* is increased. This input parameter will be replaced by a tolerance factor and numerical screening in future.

13.2 Generating and using periodic gaussian trial wavefunctions using PySCF

Similar to any QMC calculation, using periodic GTOs requires the generation of a periodic trial wavefunction. QMCPACK is currently interfaced to PySCF which is a multipurpose electronic structure written mainly in Python with key numerical functionality implemented via optimized C and C++ libraries[36]. Such a wavefunction can be generated following the example for a 2x1x1 supercell, below. Note that the current implementation and examples cover only the use of k-points where symmetry allows real coefficients to be used. This allows calculation at Γ) and, e.g., some high symmetry k-points at the Brillouin zone edges. More general k-points requiring complex coefficients will be supported in future releases.

Listing 13.1: Example PySCF input for single k-point calculation for a 2x1x1 Carbon supercell.

```
#!/usr/bin/env python

import numpy
from pyscf.pbc import gto, scf, dft
from mpi4pyscf.pbc import df
from pyscf.pbc.tools.pbc import super_cell

nmp = [2, 1, 1]

cell = gto.Cell()

cell.a = '''
         3.37316115       3.37316115       0.00000000
         0.00000000       3.37316115       3.37316115
         3.37316115       0.00000000       3.37316115'''
cell.atom = '''
   C        0.00000000       0.00000000       0.00000000
   C        1.686580575      1.686580575      1.686580575
            '''
cell.basis='bfd-vtz'
cell.ecp = 'bfd'

cell.unit='B'
cell.drop_exponent=0.1

cell.verbose = 5
cell.build()


supcell = super_cell(cell, nmp)
mydf = df.FFTDF(supcell)
mydf.auxbasis = 'weigend'

mf = dft.RKS(supcell)
mf.xc = 'lda'
```

```
mf.exxdiv = 'ewald'
mf.with_df = mydf

e_scf=mf.kernel()


print 'e_scf',e_scf

kpts=[]
title="C_Diamond"
from PyscfToQmcpack import savetoqmcpack
savetoqmcpack(supcell,mf,title=title,kpts=kpts)
```

Note that the last 4 lines of the file

```
kpts=[]
title="C_Diamond"
from PyscfToQmcpack import savetoqmcpack
savetoqmcpack(supcell,mf,title=title,kpts=kpts)
```

contains an empty list of k-points (since this is a gamma point calculation) and a title. The title variable will be the name of the HDF5 file where all the data needed by QMCPACK will be stored. The function *savetoqmcpack* will be called at the end of the calculation and will generate the HDF5 similarly to the non-periodic PySCF calculation in section 22.3.1 (convert4qmc). The function is distributed with QMCPACK and located in the qmcpack/src/QMCTools directory under the name *PyscfToQmcpack.py*. In order for the script to work, you need to specify the path to the file in your PYTHONPATH such as

```
export PYTHONPATH=QMCPACK_PATH/src/QMCTools:$PYTHONPATH
```

When using multiple k-points, it is necessary to expand the kpoints into the equivalent supercell, adjust for the phase factor in the coefficient's value due to the translation by the lattice vector and order the molecular coefficients from each k-point according to their occupation. These operations are all automated in the *savetoqmcpack()* function.

The following example corresponds to the same carbon system (2x1x1) however, in this case, we use a primitive simulation cell and a 2x1x1 kpoint mesh.

Listing 13.2: Example PySCF input for single k-point calculation for a 2x1x1 Carbon supercell.

```
#!/usr/bin/env python

import numpy
from pyscf.pbc import gto, scf, dft,df
kmesh = [2, 1, 1]

cell = gto.Cell()
cell.a = '''
         3.37316115       3.37316115       0.00000000
         0.00000000       3.37316115       3.37316115
         3.37316115       0.00000000       3.37316115'''
cell.atom = '''
   C        0.00000000       0.00000000       0.00000000
   C        1.686580575      1.686580575      1.686580575
            '''
```

```
cell.basis='bfd-vtz'
cell.ecp = 'bfd'

cell.unit='B'
cell.drop_exponent=0.1

cell.verbose = 5

cell.build()

kpts = cell.make_kpts(kmesh)
kpts -= kpts[0]

mydf = df.GDF(cell,kpts)
mydf.auxbasis = 'weigend'
mf = scf.KRHF(supcell,kpts).density_fit()

mf.exxdiv = 'ewald'
mf.with_df = mydf
e_scf=mf.kernel()


title="C_Diamond-211"

from PyscfToQmcpack import savetoqmcpack
savetoqmcpack(supcell,mf,title=title,kpts=kpts,kmesh=kmesh)
```

Note the difference between the 2 input files where:

```
kmesh=[2,1,1]   #k-point mesh


kpts = cell.make_kpts(kmesh)
kpts -= kpts[0]
```

Will generate k-points centered around the Γ-point and will insure the molecular coefficients are real.

```
mf = scf.KRHF(supcell,kpts).density_fit()
```

The Computational algorithm chosen in PySCF is *KRHF* instead of *RHF*.

Finally, to generate the HDF5 file needed by QMCPACK we call the *savetoqmcpack* function

```
from PyscfToQmcpack import savetoqmcpack
savetoqmcpack(supcell,mf,title=title,kpts=kpts,kmesh=kmesh)
```

In this call, we simply specify the k-point mesh used in order to force the converter to generate the desired cell. Note that if the parameter *kmesh* is omitted, the converter will still try to "guess" it.

In order to generate QMCPACK input files, you will need to run for both cases *convert4qmc* exactly as specified in section 22.3.1;

```
convert4qmc -pyscf C_Diamond.h5
```

This tool can be used with any option described in convert4qmc. Since the HDF5 contains all the information needed, there is no need to specify any other specific tag for periodicity.A supercell at Γ-point or using multiple k-points will work without further modification..

Running convert4qmv will generate 3 input files;

Listing 13.3: CDiamond.structure.xml. This file contains the geometry of the system.

```
<?xml version="1.0"?>
<qmcsystem>
  <simulationcell>
    <parameter name="lattice">
  6.74632230000000e+00  6.74632230000000e+00  0.00000000000000e+00
  0.00000000000000e+00  3.37316115000000e+00  3.37316115000000e+00
  3.37316115000000e+00  0.00000000000000e+00  3.37316115000000e+00
</parameter>
    <parameter name="bconds">p p p</parameter>
    <parameter name="LR_dim_cutoff">15</parameter>
  </simulationcell>
  <particleset name="ion0" size="4">
    <group name="C">
      <parameter name="charge">4</parameter>
      <parameter name="valence">4</parameter>
      <parameter name="atomicnumber">6</parameter>
    </group>
    <attrib name="position" datatype="posArray">
  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
  1.6865805750e+00  1.6865805750e+00  1.6865805750e+00
  3.3731611500e+00  3.3731611500e+00  0.0000000000e+00
  5.0597417250e+00  5.0597417250e+00  1.6865805750e+00
</attrib>
    <attrib name="ionid" datatype="stringArray">
 C C C C
</attrib>
  </particleset>
  <particleset name="e" random="yes" randomsrc="ion0">
    <group name="u" size="8">
      <parameter name="charge">-1</parameter>
    </group>
    <group name="d" size="8">
      <parameter name="charge">-1</parameter>
    </group>
  </particleset>
</qmcsystem>
```

As one can see, that for both examples the 2 atom primitive cell has been expanded to contain 4 atoms in a 2x1x1 carbon cell.

Listing 13.4: CDiamond.wfj-Twist0.xml. This file contains the trial wavefunction.

```
<?xml version="1.0"?>
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="MolecularOrbital" name="LCAOBSet" source="ion0"
    transform="yes" twist="0  0  0" href="C_Diamond.h5" PBCimages="5  5  5">
      <slaterdeterminant>
        <determinant id="updet" size="8">
          <occupation mode="ground"/>
```

```
          <coefficient size="116" spindataset="0"/>
        </determinant>
        <determinant id="downdet" size="8">
          <occupation mode="ground"/>
          <coefficient size="116" spindataset="0"/>
        </determinant>
      </slaterdeterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation size="10" speciesA="u" speciesB="u">
        <coefficients id="uu" type="Array"> 0 0 0 0 0 0 0 0 0 0</coefficients>
      </correlation>
      <correlation size="10" speciesA="u" speciesB="d">
        <coefficients id="ud" type="Array"> 0 0 0 0 0 0 0 0 0 0</coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" source="ion0" print="yes">
      <correlation size="10" cusp="0" elementType="C">
        <coefficients id="eC" type="Array"> 0 0 0 0 0 0 0 0 0 0</coefficients>
      </correlation>
    </jastrow>
  </wavefunction>
</qmcsystem>
```

This files contains information related to the trial wavefunction. It is identical to the input file from an Open Boundary Conditions calculation to the exception of the following tags:

*.wfj.xml specific tags

tag	tag type	default	description
twist	3 doubles	Gamma (0 0 0)	coordinate of the twist to compute
href	string	default	name of the HDF5 file generated by PySCF and used for convert4qmc
PBCimages	3 Integer	5 5 5	Number of periodic images to evaluate the orbitals

Other files containing QMC methods (such as optimization, VMC and DMC blocks) will be generated and will behave in a similar fashion regardless of the type of SPO in the trial wavefunction.

Chapter 14

Selected Configuration Interaction

A direct path towards improving the accuracy of a QMC calculation is through a better trial wavefunction. While using a multireference wavefunction can be straightforward in theory, in actual practice methods such as CASSCF are not always intuitive and often require being an expert in either the method or the code generating the wavefunction. An alternative is to use a Selected Configuration of Interaction method (selected CI) such as CIPSI (Configuration Interaction using a Perturbative Selection done Iteratively). This provides a direct route to systematically improving the wavefunction.

14.1 Theoretical Background

The principle behind selected CI is rather simple and was first published in 1955 by R.K. Nesbet[30]. The first calculations on atoms were performed by Diner, Malrieu and Claverie[29] in 1967, and it become computationally viable for larger molecules in 2013 by Caffarel *et al.*[27]

As described by Caffarel et al in Ref. [27], multi-determinantal expansions of the ground-state wavefunction Ψ_T are written as a linear combination of Slater determinants

$$\sum_k c_k \sum_q d_{k,q} D_{k,q\uparrow}(r^\uparrow) D_{k,q\downarrow}(r^\downarrow) \tag{14.1}$$

where each determinant corresponds to a given occupation by the N_α and N_β electrons of $N = N_\alpha + N_\beta$ orbitals among a set of M spin-orbitals $\{\phi_1, ., \phi_M\}$ (restricted case). When no symmetries are considered, the maximum number of such determinants is

$$\left(\begin{array}{c} M \\ N_\alpha \end{array} \right) . \left(\begin{array}{c} M \\ N_\beta \end{array} \right) . \tag{14.2}$$

a number that grows factorially with M and N. The best representation of the exact wavefunction in the determinantal basis is the full configuration interaction (FCI) wave function written as

$$|\Psi_0\rangle = \sum_i c_i |D_i\rangle \tag{14.3}$$

where c_i are the ground-state coefficients obtained by diagonalizing the matrix, $H_{ij} = \langle D_i|H|D_j\rangle$, within the full orthonormalized set $\langle D_i||D_j\rangle = \delta_{ij}$, of determinants $|D_i\rangle$. CIPSI provides a convenient method to build up to this full wavefunction with a single criteria.

A CIPSI wavefunction is built iteratively starting from a reference wavefunction, usually Hartree-Fock or CASSCF, by adding all single and double excitations and then iteratively selecting

relevant determinants according to some criteria. Detailed iterative steps can be found in the reference by Caffarel *et al.* and references within[27, 34, 31, 33] but are summarized below:

- Step 1: Define a reference wavefunction:

$$|\Psi\rangle = \sum_{i \in D} c_i |i\rangle \quad E_{var} = \frac{\langle \Psi | \hat{H} | \Psi \rangle}{\langle \Psi | | \Psi \rangle} \tag{14.4}$$

- Step 2: Generate external determinants $|\alpha\rangle$:
  New determinants are added by generating all single and double excitations from determinants $i \in D$ such as:

$$\langle \Psi_0^{(n)} | H | D_{i_c} \rangle \neq 0 \tag{14.5}$$

- Step 3: Evaluate second order perturbative contribution to each determinant $|\alpha\rangle$:

$$\Delta E = \frac{\langle \Psi | \hat{H} | \alpha \rangle \langle \alpha | \hat{H} | \Psi \rangle}{E_{var} - \langle \alpha | \hat{H} | \alpha \rangle} \tag{14.6}$$

- Step 4: Select the determinants with the largest contributions and add them to the Hamiltonian.
- Step 5: Diagonalize the Hamiltonian within the new added determinants and update the wavefunction and the the value of E_{var}.
- Step 6: Iterate until reaching convergence.

Repeating this process leads to a multi-reference trial wavefunction of high quality that can be used in QMC.

$$\Psi_T(r) = e^{J(r)} \sum_k c_k \sum_q d_{k,q} D_{k,q\uparrow}(r^\uparrow) D_{k,q\downarrow}(r^\downarrow) \tag{14.7}$$

The linear coefficients c_k are then optimized with the presence of the Jastrow function.

It is important to note that:

- When all determinants $|\alpha\rangle$ are selected, the full configuration interaction result is obtained.
- CIPSI can be seen as a deterministic counter part of FCIQMC.
- In practice, any wavefunction method can be made multireference with CIPSI. For instance, a multireference Coupled Cluster (MRCC) with CIPSI is implemented in QP.[32]
- At any time, with CIPSI selection, $E_{PT_2} = \sum_\alpha \Delta E_\alpha$ estimates the distance to the FCI solution.

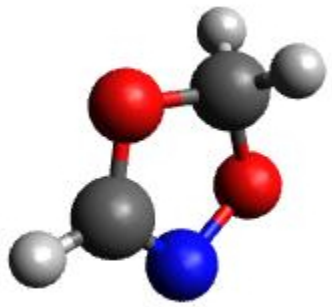


Figure 14.1: $C_2O_2H_3N$ molecule.

14.1.1 CIPSI wavefunction interface

The CIPSI method is implemented in the *Quantum Package* (QP) code[28] developed by the Caffarel group. Once the trial wavefunction is generated, QP is able to produce output readable by the QMCPACK converter as described in section 22.3.1.
QP can be installed with multiple plugins for different levels of theory in quantum chemistry. When installing the "QMC" plugin, QP can save the wavefunction in a format readable by the QMCPACK converter.

In the following we use the $C_2O_2H_3N$ molecule (Fig 14.1) as an example of how to run a multireference calculation with CIPSI as a trial wavefunction for QMCPACK . The choice of this molecule is motivated by its multireference nature. While the molecule remains small enough for CCSD(T) calculations with aug-cc-pVTZ basis set, the D1 diagnostic shows a very high value for $C_2O_2H_3N$, suggesting a multireference character. Therefore, an accurate reference for the system is not available and it becomes difficult to trust the quality of a single-determinant wavefunction, even when using the DFT-B3LYP exchange and correlation functional. Therefore, in the following, we show an example of how to systematically improve the nodal surface by increasing the number of determinants in the trial wavefunction.

The following steps show how to run from Hartree-Fock to selected CI using QP, convert the wavefunction to a QMCPACK trial wavefunction and finally how to analyze the result.

- Step 1: Generate the QP input file:
  QP takes for input an XYZ file containing the geometry of the molecule such as:

```
8
C2O2H3N
C       1.067070    -0.370798    0.020324
C      -1.115770    -0.239135    0.081860
O      -0.537581     1.047619   -0.091020
N       0.879629     0.882518    0.046830
H      -1.525096    -0.354103    1.092299
H      -1.868807    -0.416543   -0.683862
H       2.035229    -0.841662    0.053363
O      -0.025736    -1.160835   -0.084319
```

The input file is generated through the following command line:

```
qp_create_ezfio_from_xyz C2O2H3N.xyz -b cc-pvtz
```

This means that we will be simulating the molecule in all-electrons within the cc-pVTZ basis set. Other options are of course possible such as using ECPs, different spin multiplicities etc. For more details, to the Quantum Package tutorial https://github.com/LCPQ/quantum_package/wiki/Tutorial.
A directory called *C2O2H3N.ezfio* is created and contains all the relevant data to run the SCF Hartree-Fock calculation. Note that due to the large size of molecular orbitals (220), it is preferable to run QP in parallel. QP parallelization is based on a Master/Slave process allowing a master node to manage the work load between multiple MPI processes through the LibZMQ library. In practice one submits the run to one master node, then submits as many nodes as necessary to speed up the calculations. If a "slave" node dies before the end of its task, the master node will resubmit the workload to another available node. If more nodes are added at any time during the simulation, the master node will use them to reduce the time to solution.

- Step 2: Running Hartree-Fock:
  To save the integrals on disk and avoid recomputing them later, edit the ezfio directory with the following command:

```
qp_edit C2O2H3N.ezfio
```

This will generate a temporary file showing all the contents of the simulation and opens an editor to allow modification of their values. Look for *disk_access_ao_integrals* and modify its value from *None* to *Write*

To run a simulation with QP, use the binary *qp_run* with the desired level of theory, in this case Hartree-Fock (SCF).

```
mpirun -np 1 qp_run SCF C2O2H3N.ezfio &> C2O2H3N-SCF.out
```

If run in serial, the evaluation of the integrals and the Hamiltonian diagonalization would take a substantial amount of computer time. It is recommended to add a few more *slave-nodes* to help speed up the calculation.

```
mpirun -np 20 qp_run -slave qp_ao_ints C2O2H3N.ezfio &> C2O2H3N-SCF-Slave.out
```

The total Hartree-Fock energy of the system in cc-pVTZ is $E_{HF} = -283.0992$Ha.

- Step 2: Freeze Core electrons:
  In order to avoid making excitation from the core electrons, freeze the core electrons and only do the excitations from the valence electrons.

```
qp_set_frozen_core.py C2O2H3N.ezfio
```

This will will automatically freeze the orbitals from 1 to 5, and leave the remaining active.

- Step 3: Atomic orbitals to Molecular Orbital transformation
  This step, transforming the atomic orbitals to molecular orbitals, is the most costly, especially given that its implementation in Quantum Package is serial. It is recommended to do it in a separate run and on one node.

```
qp_run four_idx_transform C2O2H3N.ezfio
```

The MO integrals are now saved on disk and unless the orbitals are changed, they will not be recomputed.

- Step 4: CIPSI
  At this point the wavefunction is ready for the selected CI. By default QP has 2 convergence criteria. The number of determinants (set by default to 1M) or the value of PT2 (set by default to 1.10^{-4}Ha). For this molecule, the total number of determinants in the FCI space is $2.07e+88$ determinants. While his number is completely out of range of what is possible to compute, we will set the limit of determinants in QP to 5M determinants and see if the nodal surface of the wavefunction is converged enough for the DMC. It is important to remember at this point that the main value of CIPSI compared to other selected CI method, is that the value of PT2 is evaluated directly at each step giving a good estimated of the error to the FCI energy. This allows us to conclude that when the E+PT2 energy is converged, the nodal surface is probably also converged.
  Similar to the SCF runs, FCI runs have to be submitted in parallel with a *Master/Slave* process:

```
mpirun -np 1 qp_run fci_zmq C2O2H3N.ezfio &> C2O2H3N-FCI.out
mpirun -np 199 qp_run -slave selection_davidson_slave C2O2H3N.ezfio\\
&> C2O2H3N-FCI-Slave.out
```

- Step 5 (optional): Natural orbitals
  While this step is optional, it is important to note that using natural orbitals instead of Hartree Fock orbitals will always improve the quality of the wavefunction and improve the quality of the nodal surface by reducing the number of needed determinants for the same accuracy. When a full convergence to the FCI limit is attainable, this step will not lead to any change in the energy but will only reduce the total number of determinants. However, if a full convergence is not possible, this step can increase significantly the accuracy of the calculation at the same number of determinants.

```
qp_run save_natorb C2O2H3N.ezfio
```

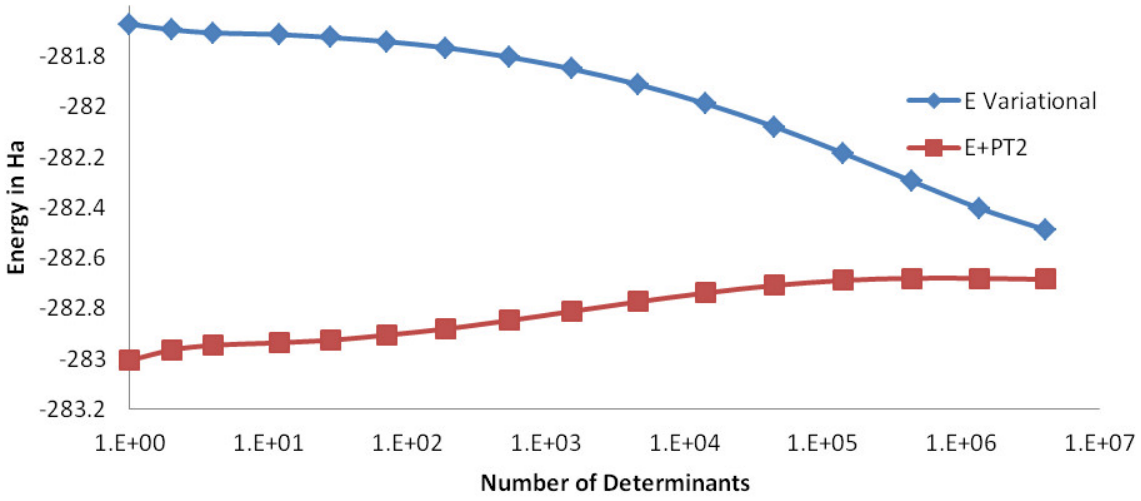


Figure 14.2: Evolution of the variational energy and the Energy + PT2 as a function of the number of determinants for the $C_2O_2H_3N$ molecule.

Table 14.1: Energies of $C_2O_2H_3N$ using orbitals from Hartree-Fock, natural orbitals, 0.4M and 4M determinants

Method	N_det	Energy
Hartree-Fock	1	-281.6729
Natural Orbitals	1	-281.6735
E_Variational	438,753	-282.2951
E_Variational	4,068,271	-282.4882
E+PT2	438,753	-282.6809
E+PT2	4,068,271	-282.6805

At this point, the orbitals are modified, a new AO→MO transformation is required and Steps 3 and 4 need to be run again.

- Step 6: Analysis of the CIPSI results
  Figure 14.2 shows the evolution of the variational energy and the energy corrected with PT2 as a function of the number of determinants up to 4M determinants. While it is clear that the raw variational energy is far from being converged, the Energy + PT2 appears converged around 0.4M determinants.

- Step 7: Truncation of the number of determinants.
  While using all the 4M determinants from CIPSI always guarantees that all important determinants are kept in the wavefunction, practically, such a large number of determinants would make any QMC calculation prohibitively expensive, as the cost of evaluating a determinant in DMC grows as $\sqrt{N_{det}}$ where N_{det} is the number of determinants in the trial wavefunction. To truncate the number of determinants, we follow the method described by

Scemama *et. al* [31] where the wavefunction is truncated by removing independently spin-up and spin-down determinants whose contribution to the norm of the wavefunction is below a user-defined threshold, ϵ. For this step, we choose to truncate the determinants whose coefficients are below, 1.10^{-3}, 1.10^{-4}, 1.10^{-5} and 1.10^{-6}, translating to 239, 44539, 541380 and 908128 determinants, respectively.

To truncate the determinants in QP, edit the ezfio file as follows:

```
qp_edit C2O2H3N.ezfio
```

then look for *ci_threshold* and modify the value according to the desired threshold. Use the following run to truncate the determinants:

```
qp_run truncate_wf_spin C2O2H3N.ezfio
```

- Step 7: Save the wavefunction for QMCPACK
  The wavefunction in QP is now ready to be converted to QMCPACK format. Save the wavefunction into QMCPACK format and then convert the wavefunction using the *convert4qmc* tool

```
qp_run save_for_qmcpack C2O2H3N.ezfio &> C2O2H3N.dump
convert4qmc -QP C2O2H3N.dump -addCusp -production
```

Since we are running all-electron calculations, orbitals in QMC need to be corrected for the electron-nuclearcusp condition.. This is done by adding the option *-addCusp* to *convert4qmc*, which adds a tag forcing QMCPACK to run the correction or read them from a file if pre-computed. When running multiple DMC runs with different truncation thresholds, only the number of determinants is varied and the orbitals remain unchanged from one calculation to another and the cusp correction needs only be run once.

- Step 7: Running QMCPACK
  At this point, running a multideterminant DMC becomes identical to running a regular DMC with QMCPACK ; After correcting the orbitals for the cusp, optimize the Jastrow functions and then run the DMC. It is however important to mention a few remarks;

(1) QMCPACK allows reoptimization of the coefficients of the determinants during the Jastrow optimization step. While this has proven to lower the energy significantly when the number of determinants is below 10k, a large number of determinants from CIPSI is often too large to optimize conveniently. Keeping the coefficients of the determinants from CIPSI unoptimized is an alternative strategy.

(2) The large determinant expansion and the Jastrows are both trying to recover the missing correlations from the system. When optimizing the Jastrows, we recommend to first optimize J1 and J2 without the J3, and then with the added J3. Trying to initially optimze J1, J2 and J3 at the same time may lead to numerical instabilities.

(3) The parameters of the Jastrow function will need to be optimized for each truncation scheme and usually cannot be reused efficiently from one truncation scheme to another.

Table 14.2: DMC Energies and CIPSI(E+PT2) of $C_2O_2H_3N$ in function of the number of determinants in the trial wavefunction.

N_det	DMC	CISPI
1	-283.0696 (6)	-283.0063
239	-283.0730 (9)	-282.9063
44,539	-283.078 (1)	-282.7339
541,380	-283.088 (1)	-282.6772
908,128	-283.089 (1)	-282.6775

- Step 8: Analyzing DMC results from QMCPACK

  From Table 14.2, we can see that increasing the number of determinants from 0.5M to almost 1M determinant keeps the energy within error bars and does not improve the quality of the nodal surface. We can conclude that the DMC energy is converged at 0.54M determinants. It is important to note that this number of determinants also corresponds to the convergence of E+PT2 in CIPSI calculations, confirming for this case that the convergence of the nodal surface can follow the convergence of E+PT2 instead of the more difficult variational energy.

As mentioned in previous sections, DMC is variational relative to the exact nodal surface. A nodal surface is "better" if it lowers the DMC energy. To assess the quality of the nodal surface from CIPSI, we compare these DMC results to other single determinant calculations from multiple nodal surfaces and theories. Figure 14.3 shows the energy of the $C_2O_2H_3N$ molecule as a function of different single-determinant (SD) trial wavefunctions with an aug-cc-pVTZ basis set, including Hartree-Fock (HF), DFT-PBE and hybrid functionals B3LYP and PBE0. The last 4 points in the plot show the systematic improvement of the nodal surface as a function of the number of determinants.

When the DMC-CIPSI energy is converged with respect to the number of determinants, its nodal surface is still lower than the best SD-DMC (B3LYP) by 6(1)mHa. When compared to CCSD(T) with the same basis set, $E_{CCSD(T)}$ is 4mHa higher than DMC-CIPSI and 2mHa lower than DMC-B3LYP. While 6 (1) mHa can seem very small, it is however important to remember that CCSD(T) cannot describe correctly multireference systems and therefore it is impossible to assess the correctness of the SD-DMC result, making CIPSI-DMC calculations an ideal benchmark tool for multireference systems.

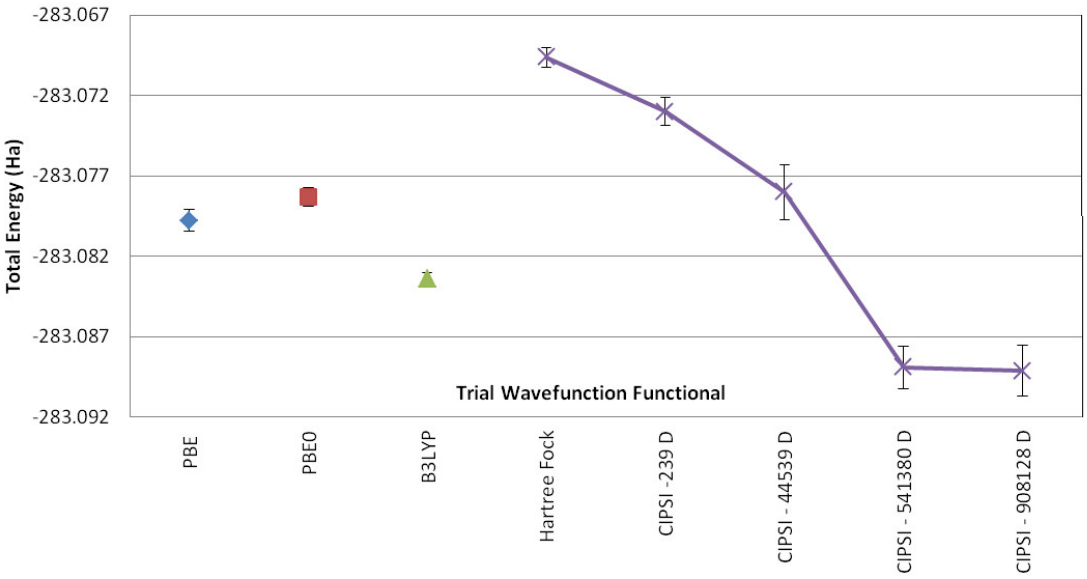


Figure 14.3: DMC energy of $C_2O_2H_3N$ molecule as a function of different single determinant trial wavefunctions with aug-ccp-VTZ basis set using nodal surfaces from Hartree-Fock (HF), DFT-PBE and DFT with hybrid functionals PBE0 and P3LYP. The CIPSI trial wavefunction contains as indicated 239, 44539, 514380 and 908128 determinants (D).

Chapter 15

Auxiliary-Field Quantum Monte Carlo

The Auxiliary-Field Quantum Monte Carlo (AFQMC) method is an orbital-space formulation of the imaginary-time propagation algorithm. We refer the reader to one of the review articles on the method [23, 21, 22], for a detailed description of the algorithm. It uses the Hubbard-Stratonovich transformation to express the imaginary-time propagator, which is inherently a 2-body operator, as an integral over 1-body propagators which can be efficiently applied to an arbitrary Slater determinant. This transformation allows us to represent the interacting many-body system as an average over a non-interacting system (e.g. Slater determinants) in a time-dependent fluctuating external field (the Auxiliary fields). The walkers in this case represent non-orthogonal Slater determinants, whose time average represent the desired quantum state. QMCPACK currently implements the phaseless AFQMC algorithm of Zhang and Krakauer [21], where a trial wave-function is used to project the simulation to the real axis, controlling the fermionic sign problem at the expense of a bias. This approximation is similar in spirit to the fixed-node approximation in real-space DMC, but applied in the Hilbert space where the AFQMC random walk occurs.

15.1 Theoretical Background

... Coming Soon ...

15.2 Input

The input for an AFQMC calculation is fundamentally different to the input for other real-space algorithms in QMCPACK. The main source of input comes from the Hamiltonian matrix elements in an appropriate single particle basis. This must be evaluated by an external code and saved in a format that QMCPACK can read. More details about file formats are found below. The input file has six basic xml-blocks: `AFQMCInfo`, `Hamiltonian`, `Wavefunction`, `WalkerSet`, `Propagator`, and `execute`. The first five define input structures required for various types of calculations. The `execute` block represents actual calculations and takes as input the other blocks. Non-execution blocks are parsed first, followed by a second pass where execution blocks are parsed (and executed) in order. Listing 13.1 shows an example of a minimal input file for an AFQMC calculation. Table 15.1 shows a brief description of the most important parameters in the calculation. All xml sections contain a "name" argument used to identify the resulting object within QMCPACK. For example,

in the example, multiple Hamiltonian objects with different names can be defined. The one actually used in the calculation is the one passed to "execute" as ham.

Listing 15.1: Sample input file for AFQMC.

```
<?xml version="1.0"?>
<simulation method="afqmc">
  <project id="Carbon" series="0"/>

  <AFQMCInfo name="info0">
    <parameter name="NMO">32</parameter>
    <parameter name="NAEA">16</parameter>
    <parameter name="NAEB">16</parameter>
  </AFQMCInfo>

  <Hamiltonian name="ham0" info="info0">
    <parameter name="filename">../fcidump.h5</parameter>
  </Hamiltonian>

  <Wavefunction name="wfn0" type="MSD" info="info0">
    <parameter name="filetype">ascii</parameter>
    <parameter name="filename">wfn.dat</parameter>
  </Wavefunction>

  <WalkerSet name="wset0">
    <parameter name="walker_type">closed</parameter>
  </WalkerSet>

  <Propagator name="prop0" info="info0">
  </Propagator>

  <execute wset="wset0" ham="ham0" wfn="wfn0" prop="prop0" info="info0">
    <parameter name="timestep">0.005</parameter>
    <parameter name="blocks">10000</parameter>
    <parameter name="nWalkers">20</parameter>
  </execute>

</simulation>
```

Below is a list of all input sections for AFQMC calculations, along with a detailed explanation of accepted parameters. Since the code is under active development, the list of parameters and their interpretation can change in the future.

`AFQMCInfo`: input block that defines basic information about the calculation. It is passed to all other input blocks to propagate the basic information: `<AFQMCInfo name="info0">`

- **NMO**. Number of molecular orbitals, i.e., number of states in the single particle basis.
- **NAEA**. Number of Active Electrons-Alpha, i.e., number of spin-up electrons.
- **NAEB**. Number of Active Electrons-Beta, i.e., number of spin-down electrons.

`Hamiltonian`: controls the object that reads, stores and manages the hamiltonian. `<Hamiltonian name="ham0" type="SparseGeneral" info="info0">`

- **filename**. Name of file with the `Hamiltonian`. This is a required parameter.

- **cutoff_1bar**. Cutoff applied to integrals during reading. Any term in the hamiltonian smaller than this value is set to zero. (For filetype="hdf5", the cutoff is only applied to the 2-electron integrals). Default: 1e-8

- **cutoff_decomposition**. Cutoff used to stop the iterative cycle in the generation of the Cholesky decomposition of the 2-electron integrals. The generation of Cholesky vectors is stopped when the maximum error in the diagonal reaches this value. In case of an eigenvalue factorization, this becomes the cutoff applied to the eigenvalues. Only eigenvalues above this value are kept. Default: 1e-6

- **nblocks**. This parameter controls the distribution of the 2-electron integrals among processors. In the default behavior (nblocks=1), all nodes contain the entire list of integrals. If nblocks > 1, the of nodes in the calculation will be split in nblocks groups. Each node in a given group contains the same subset of integrals and subsequently operates on this subset during any further operation that requires the hamiltonian. The maximum number of groups is NMO. Currently only works for filetype="hdf5" and the file must contain integrals. Not yet implemented for input hamiltonians in the form of Cholesky vectors or for ASCII input. Coming soon! Default: No distribution

- **printEig**. If "yes", prints additional information during the Cholesky decomposition. Default: no

- **fix_2eint**. If this is set to "yes", orbital pairs that are found not to be positive definite are ignored in the generation of the Cholesky factorization. This is necessary if the 2-electron integrals are not positive definite due to round-off errors in their generation. Default: no

`Wavefunction`: controls the object that manages the trial wave-functions. This block expects a list of xml-blocks defining actual trial-wave functions for various roles. `<Wavefunction name="wfn0" type="MSD/PHMSD" info="info0">`

- **filename**. Name of file with wave-function information.

- **cutoff**. cutoff applied to the terms in the calculation of the local energy. Only terms in the hamiltonian above this cutoff are included in the evaluation of the energy. Default: 1e-6

- **nnodes**. Defines the parallelization of the local energy evaluation and the distribution of the `Hamiltonian` matrix (not to be confused with the list of 2-electron integrals managed by `Hamiltonian`. These are not the same.) If nnodes > 1, the nodes in the simulation are split into groups of nnodes, each group works collectively in the evaluation of the local energy of their walkers. This helps distribute the effort involved in the evaluation of the local energy among the nodes in the group, but also distributes the memory associated with the wave-function among the nodes in the group. Default: No distribution

- **ndet**. Number of determinants to read from file. Default: Read all determinants.

- **cutoff**. For sparse hamiltoniants, this defines the cutoff applied to the half-rotated 2-electron integrals. Default: 0.0

- **nbatch**. This turns on(>=1)/off(==0) batched calculation of density matrices and overlaps. -1 means all the walkers in the batch. Default: 0 (CPU) / -1 (GPU)

- **nbatch_qr**. This turns on(>=1)/off(==0) batched QR calculation. -1 means all the walkers in the batch. Default: 0 (CPU) / -1 (GPU)

`WalkerSet`: Controls the object that handles the set of walkers. `<WalkerSet name="wset0">`

- **walker_type**. Type of walker set: closed or collinear. Default: collinear
- **pop_control**. Population control algorithm. Options: "simple": Uses a simple branching scheme with a fluctuating population. Walkers with weight above max_weight are split into multiple walkers of weight reset_weight. Walkers with weight below min_weight are killed with probability (weight/min_weight); "pair": Fixed-population branching algorithm, based on QWalk's branching algorithm. Pairs of walkers with weight above/below max_weight/min_weight are combined into 2 walkers with weights equal to $(w_1+w_2)/2$. The probability of replicating walker w1 (larger weight) occurs with probability $w_1/(w_1+w_2)$, otherwise walker w2 (lower weight) is replicated; "comb": Fixed-population branching algorithm based on the Comb method. Will be available in the next release. Default: "pair"
- **min_weight**. Weight at which walkers are possibly killed (with probability weight/min_weight). Default: 0.05
- **max_weight**. Weight at which walkers are replicated. Default: 4.0
- **reset_weight**. Weight to which replicated walkers are reset to. Default: 1.0

`Propagator`: Controls the object that manages the propagators. `<Propagator name="prop0" info="info0">`

- **cutoff**. Cutoff applied to Cholesky vectors. Elements of the Cholesky vectors below this value are set to zero. Only meaningful with sparse hamiltonians. Default: 1e-6
- **substractMF**. If "yes", apply mean-field subtraction based on the ImpSamp trial wave-function. Must set to "no" to turn it off. Default: yes
- **vbias_bound**. Upper bound applied to the vias potential. Components of the vias potential above this value are truncated there. The bound is currently applied to $\sqrt{\tau}v_{bias}$, so a larger value must be used as either the time-step or the fluctuations increase (e.g. from running a larger system or using a poor trial wave-function). Default: 3.0
- **apply_constrain**. If "yes", apply the phaseless constrain to the walker propagation. Currently, setting this to "no" produces unknown behavior, since free propagation algorithm has not been tested. Default: yes
- **hybrid**. If "yes", use hybrid propagation algorithm. This propagation scheme doesn't use the local energy during propagation, leading to significant speed ups when its evaluation cost is high. The local energy of the ImpSamp trial wave-function is never evaluated. To obtain energy estimates in this case, you must define an Estimator xml-block with the `Wavefunction` block. The local energy of this trial wave-function is evaluated and printed. It is possible to use a previously defined trial wave-function in the Estimator block, just set its "name" argument to the name of a previously defined wave-function. In this case, the same object is used for both roles. Default: no
- **nnodes**. Controls the parallel propagation algorithm. If nnodes > 1, the nodes in the simulation are split into groups of nnodes nodes, each group working collectively to propagate their walkers. Default: 1 (Serial algorithm)

- **nbatch**. This turns on(>=1)/off(==0) batched calculation of density matrices and overlaps. -1 means all the walkers in the batch. Default: 0 (CPU) / -1 (GPU)

- **nbatch_qr**. This turns on(>=1)/off(==0) batched QR calculation. -1 means all the walkers in the batch. Default: 0 (CPU) / -1 (GPU)

`execute`: Defines an execution region. `<execute wset="wset0" ham="ham0" wfn="wfn0" prop="prop0" info="info0">`

- **nWalkers**. initial number of walkers per core group (see ncores). This sets the number of walkers for a given gorup of "ncores" on a node, the total number of walkers in the simulation depends on the total number of nodes and on the total number of cores on a node in the following way: $\#walkers_total = nWalkers * \#nodes * \#cores_total/ncores$.
  Default: 5

- **timestep**. time step in 1/a.u.
  Default: 0.01

- **blocks**. number of blocks. Slow operations occur once per block, e.g. write to file, slow observables, checkpoints, etc.
  Default: 100

- **step**. number of steps within a block. Operations that occur at the step level include: load balance, orthogonalization, branching, etc.
  Default: 1

- **substep**. number of substeps within a step. Only walker propagation occurs in a substep.
  Default: 1

- **ortho**. number of steps between orthogonalization. Default: 1

- **ncores**. Number of nodes in a task group. This number defines the number of cores on a node that share the parallel work associated with a distributed task. This number is used in the `Wavefunction` and `Propagator` task groups. The walker sets are shares by the ncores on a given node in the task group.

- **checkpoint**. Number of blocks between checkpoint files are generated. If a value smaller than 1 is given, no file is generated. If **hdf_write_file** is not set, a default name is used. **Default: 0**

- **hdf_write_file**. If set (and checkpoint>0), a checkpoint file with this name will be written.

- **hdf_read_file**. If set, the simulation will be restarted from the given file.

Within the `Estimators` xml block has an argument **name**: the type of estimator we want to measure. Currently available estimators include: "basic", "energy", "mixed_one_rdm", and "back_propagation".

The basic estimator has the following optional parameters:

- **timers**. print timing information. Default: true

The back_propagation estimator has the following parameters:

- **ortho**. Number of back-propagation steps between orthogonalization. Default: 10
- **nsteps**. Maximum number of back-propagation steps. Default: 10
- **naverages**. Number of back propagation calculations to perform. The number of steps will be chosed equally distributed in the range 0,nsteps. Default: 1
- **block_size**. Number of blocks to use in the internal average of the back propagated estimator. This is used to block data and reduce the size of the output. Default: 1
- **nskip**. Number of blocks to skip at the start of the calculation for equilibration purposes. Default: 0

15.3 File formats

... Coming Soon ...

15.4 Advice/Useful Information

AFQMC calculations are computationally expensive and require some care in order to obtain reasonable performance. Below is a growing list of useful advice for new users followed by a sample input for a large calculation.

- Generate Cholesky-decomposed integrals with external codes instead of the 2-electron integrals directly. The generation of the Cholesky factorization is faster and consumes less memory.
- Use the hybrid algorithm for walker propagation. Set steps/substeps to adequate values to reduce the number of energy evaluations. This is essential when using large multi-determinant expansions.
- Adjust cutoffs in the wave-function and propagator bloxks until desired accuracy is reached. The cost of the calculation will depend on these cutoffs.
- Adjust ncores/nWalkers to obtain better efficiency. Larger nWalkers will lead to more efficient linear algebra operations, but will increase the time per step. Larger ncores will reduce the time per step, but will reduce efficiency due to inefficiencies in the parallel implementation. For large calculations, values between 6-12 for both quantities should be reasonable, depending on architecture.

Listing 15.2: Example of sections of an AFQMC input file for a large calculation.

```
...

  <Hamiltonian name="ham0" type="SparseGeneral" info="info0">
    <parameter name="filename">fcidump.h5</parameter>
    <parameter name="cutoff_1bar">1e-6</parameter>
    <parameter name="cutoff_decomposition">1e-5</parameter>
  </Hamiltonian>

  <Wavefunction name="wfn0" type="MSD" info="info0">
```

```
    <parameter name="filetype">ascii</parameter>
    <parameter name="filename">wfn.dat</parameter>
  </Wavefunction>

  <WalkerSet name="wset0">
    <parameter name="walker_type">closed</parameter>
  </WalkerSet>

  <Propagator name="prop0" info="info0">
    <parameter name="hybrid">yes</parameter>
  </Propagator>

  <execute wset="wset0" ham="ham0" wfn="wfn0" prop="prop0" info="info0">
    <parameter name="ncores">8</parameter>
    <parameter name="timestep">0.01</parameter>
    <parameter name="blocks">10000</parameter>
    <parameter name="steps">10</parameter>
    <parameter name="substeps">5</parameter>
    <parameter name="nWalkers">8</parameter>
    <parameter name="ortho">5</parameter>
  </execute>
```

15.5 Using PySCF to generate integrals for AFQMC

PySCF (https://github.com/sunqm/pyscf) is a collection of electronic structure programs powered by Python. It is the recommended program for the generation of input for AFQMC calculations in QMCPACK. We refer the reader to the documentation of the code (http://sunqm.github.io/pyscf/) for a detailed description of the features and the functionality of the code. While the notes below are not meant to replace a detailed study of the PySCF documentation, these notes describe useful knowledge and tips in the use of pyscf for the generation of input for QMCPACK.

For molecular systems or periodic calculations at the Gamma point, PySCF provides a routine that generates the integral file in Molpro's FCIDUMP format, which contains all the information needed to run AFQMC with a single determinant trial wave-function. Below is an example using this routine to generate the FCIDUMP file for an 8-atom unit cell of carbon in the diamond structure with HF orbitals. For a detailed description, see PySCF's documentation.

Listing 15.3: Simple example showing how to generate FCIDUMP files with PySCF

```
import numpy
from pyscf.tools import fcidump
from pyscf.pbc import gto, scf, tools

cell = gto.Cell()
cell.a = '''
  3.5668 0 0
  0 3.5668 0
  0 0 3.5668'''
cell.atom = '''
  C 0. 0. 0.
  C 0.8917 0.8917 0.8917
  C 1.7834 1.7834 0.
  C 2.6751 2.6751 0.8917
  C 1.7834 0. 1.7834
  C 2.6751 0.8917 2.6751
  C 0. 1.7834 1.7834
```

```
  C 0.8917 2.6751 2.6751'''
cell.basis = 'gth-szv'
cell.pseudo = 'gth-pade'
cell.gs = [10]*3 # for testing purposes, must be increased for converged results
cell.verbose = 4
cell.build()

mf = scf.RHF(cell)
ehf = mf.kernel()
print("HF energy (per unit cell) = %.17g" % ehf)

c = mf.mo_coeff
h1e = reduce(numpy.dot, (c.T, mf.get_hcore(), c))
eri = mf.with_df.ao2mo(c,compact=True)

# nuclear energy + electronic ewald
e0 = cell.energy_nuc() + tools.pbc.madelung(cell, numpy.zeros(3))*cell.nelectron * -.5
fcidump.from_integrals('fcidump.dat', h1e, eri, c.shape[1],cell.nelectron, ms=0,
    tol=1e-8, nuc=e0)
```

afqmc method				
parameters in AFQMCInfo				
name	**datatype**	**values**	**default**	**description**
NMO	integer	≥ 0	no	number of molecular orbitals
NAEA	integer	≥ 0	no	number of active electrons of spin up
NAEB	integer	≥ 0	no	number of active electrons of spin down
parameters in Hamiltonian				
info	argument			name of AFQMCInfo block
filename	string		no	name of file with the hamiltonian
filetype	string	hdf5	yes	native HDF5-based format of QMCPACK
parameters in Wavefunction				
info	argument			name of AFQMCInfo block
type	argument	MSD	no	linear combination of (assumed non-orthogonal) Slater determinants
		PHMSD		CI-type multi-determinant wave function
filetype	string	ascii	no	ASCII data file type
		hdf5		HDF5 data file type
parameters in WalkerSet				
walker_type	string	collinear	yes	Request a collinear walker set.
		closed	no	Request a closed shell (doubly-occupied) walker set.
parameters in Propagator				
type	argument	afqmc	afqmc	type of propagator
info	argument			name of AFQMCInfo block
hybrid	string	yes	yes	Use hybrid propagation algorithm.
		no		Use local energy based propagation algorithm.
parameters in execute				
wset	argument			
ham	argument			
wfn	argument			
prop	argument			
info	argument			name of AFQMCInfo block
nWalkers	integer	≥ 0	5	initial number of walkers per task group
timestep	real	> 0	0.01	time step in 1/a.u.
blocks	integer	≥ 0	100	number of blocks
step	integer	> 0	1	number of steps within a block
substep	integer	> 0	1	number of substeps within a step
ortho	integer	> 0	1	number of steps between walker orthogonalization.

Table 15.1: Input options for AFQMC in QMCPACK

Chapter 16

Examples

WARNING: THESE EXAMPLES ARE NOT CONVERGED! YOU MUST CONVERGE PARAMETERS (SIMULATION CELL SIZE, JASTROW PARAMETER NUMBER/CUTOFF, TWIST NUMBER, DMC TIME STEP, DFT PLANE WAVE CUTOFF, DFT K-POINT MESH, ETC.) FOR REAL CALCUATIONS!

The following examples should run in serial on a modern workstation in a few hours.

16.1 Using QMCPACK directly

In `examples/molecules`, there are the following examples. Each directory also contains a `README` file with more details.

Directory	Description
`H2O`	H2O molecule from GAMESS orbitals
`He`	Helium atom with simple wavefunctions

16.2 Using Nexus

For more information about Nexus, see the User Guide in `nexus/documentation`.

For Python to find the Nexus library, the PYTHONPATH environment variable should be set to `<QMCPACK source>/nexus/library`. For these examples to work properly, the executables for Quantum ESPRESSO and QMCPACK either need to be on the path, or the paths in the script should be adjusted.

These examples can be found under the `nexus/examples/qmcpack` directory.

Directory	Description
`diamond`	Bulk diamond with VMC
`graphene`	Graphene sheet with DMC
`c20`	C20 cage molecule
`oxygen_dimer`	Binding curve for O_2 molecule
`H2O`	H_2O molecule with Quantum ESPRESSO orbitals
`LiH`	LiH crystal with Quantum ESPRESSO orbitals

Chapter 17

Lab 1: Monte Carlo Statistical Analysis

17.1 Topics covered in this Lab

This lab focuses on the basics of analyzing data from Monte Carlo (MC) calculations. In this lab, participants will use data from VMC calculations of a simple one-electron system with an analytically soluble system (the ground state of the hydrogen atom) to understand how to interpret a MC situation. Most of these analyses will also carry over to diffusion Monte Carlo (DMC) simulations. Topics covered include:

- averaging Monte Carlo variables
- the statisical error bar of mean values
- effects of autocorrelation and variance on the error bar
- the relationship between Monte Carlo timestep and autocorrelation
- the use of blocking to reduce autocorrelation
- the significance of the acceptance ratio
- the significance of the sample size
- how to determine whether a Monte Carlo run was successful
- the relationship between wavefunction quality and variance
- gauging the efficiency of Monte Carlo runs
- the cost of scaling up to larger system sizes

17.2 Lab directories and files

```
labs/lab1_qmc_statistics/
│
├── atom                              - H atom VMC calculation
│   ├── H.s000.scalar.dat                - H atom VMC data
```

```
│   └── H.xml                          - H atom VMC input file
│
├── autocorrelation                   - varying autocorrelation
│   ├── H.dat                          - data for gnuplot
│   ├── H.plt                          - gnuplot for time step vs. E_L, tau_c
│   ├── H.s000.scalar.dat              - H atom VMC data: time step = 10
│   ├── H.s001.scalar.dat              - H atom VMC data: time step =  5
│   ├── H.s002.scalar.dat              - H atom VMC data: time step =  2
│   ├── H.s003.scalar.dat              - H atom VMC data: time step =  1
│   ├── H.s004.scalar.dat              - H atom VMC data: time step =  0.5
│   ├── H.s005.scalar.dat              - H atom VMC data: time step =  0.2
│   ├── H.s006.scalar.dat              - H atom VMC data: time step =  0.1
│   ├── H.s007.scalar.dat              - H atom VMC data: time step =  0.05
│   ├── H.s008.scalar.dat              - H atom VMC data: time step =  0.02
│   ├── H.s009.scalar.dat              - H atom VMC data: time step =  0.01
│   ├── H.s010.scalar.dat              - H atom VMC data: time step =  0.005
│   ├── H.s011.scalar.dat              - H atom VMC data: time step =  0.002
│   ├── H.s012.scalar.dat              - H atom VMC data: time step =  0.001
│   ├── H.s013.scalar.dat              - H atom VMC data: time step =  0.0005
│   ├── H.s014.scalar.dat              - H atom VMC data: time step =  0.0002
│   ├── H.s015.scalar.dat              - H atom VMC data: time step =  0.0001
│   └── H.xml                          - H atom VMC input file
│
├── average                           - Python scripts for average/std. dev.
│   ├── average.py                     - average five E_L from H atom VMC
│   ├── stddev2.py                     - standard deviation using (E_L)^2
│   └── stddev.py                      - standard deviation around the mean
│
├── basis                             - varying basis set for orbitals
│   ├── H__exact.s000.scalar.dat       - H atom VMC data using STO basis
│   ├── H_STO-2G.s000.scalar.dat       - H atom VMC data using STO-2G basis
│   ├── H_STO-3G.s000.scalar.dat       - H atom VMC data using STO-3G basis
│   └── H_STO-6G.s000.scalar.dat       - H atom VMC data using STO-6G basis
│
├── blocking                          - varying block/step ratio
│   ├── H.dat                          - data for gnuplot
│   ├── H.plt                          - gnuplot for N_block vs. E, tau_c
│   ├── H.s000.scalar.dat              - H atom VMC data 50000:1 blocks:steps
│   ├── H.s001.scalar.dat              - "  "    "    "  25000:2 blocks:steps
│   ├── H.s002.scalar.dat              - "  "    "    "  12500:4 blocks:steps
│   ├── H.s003.scalar.dat              - "  "    "    "  6250: 8 blocks:steps
│   ├── H.s004.scalar.dat              - "  "    "    "  3125:16 blocks:steps
│   ├── H.s005.scalar.dat              - "  "    "    "  2500:20 blocks:steps
│   ├── H.s006.scalar.dat              - "  "    "    "  1250:40 blocks:steps
│   ├── H.s007.scalar.dat              - "  "    "    "  1000:50 blocks:steps
│   ├── H.s008.scalar.dat              - "  "    "    "  500:100 blocks:steps
│   ├── H.s009.scalar.dat              - "  "    "    "  250:200 blocks:steps
│   ├── H.s010.scalar.dat              - "  "    "    "  125:400 blocks:steps
│   ├── H.s011.scalar.dat              - "  "    "    "  100:500 blocks:steps
│   ├── H.s012.scalar.dat              - "  "    "    "  50:1000 blocks:steps
│   ├── H.s013.scalar.dat              - "  "    "    "  40:1250 blocks:steps
│   ├── H.s014.scalar.dat              - "  "    "    "  20:2500 blocks:steps
```

```
│   ├── H.s015.scalar.dat                  - "  "    "    "  10:5000 blocks:steps
│   └── H.xml                             - H atom VMC input file
│
├── blocks                              -  varying total number of blocks
│   ├── H.dat                             - data for gnuplot
│   ├── H.plt                             - gnuplot for N_block vs. E
│   ├── H.s000.scalar.dat                 - H atom VMC data    500 blocks
│   ├── H.s001.scalar.dat                 - "  "    "    "    2000 blocks
│   ├── H.s002.scalar.dat                 - "  "    "    "    8000 blocks
│   ├── H.s003.scalar.dat                 - "  "    "    "   32000 blocks
│   ├── H.s004.scalar.dat                 - "  "    "    "  128000 blocks
│   └── H.xml                             - H atom VMC input file
│
├── dimer                           - comparing no and simple Jastrow factor
│   ├── H2_STO___no_jastrow.s000.scalar.dat - H dimer VMC data without Jastrow
│   └── H2_STO_with_jastrow.s000.scalar.dat - H dimer VMC data with Jastrow
│
├──  docs                              - documentation
│   ├──  Lab_1_MC_Analysis.pdf            - this document
│   └──  Lab_1_Slides.pdf                 - slides presented in the lab
│
├── nodes                               - varying number of computing nodes
│   ├──  H.dat                            - data for gnuplot
│   ├──  H.plt                            - gnuplot for N_node vs. E
│   ├──  H.s000.scalar.dat                - H atom VMC data with  32 nodes
│   ├──  H.s001.scalar.dat                - H atom VMC data with 128 nodes
│   └──  H.s002.scalar.dat                - H atom VMC data with 512 nodes
│
├── problematic                         - problematic VMC run
│   └──  H.s000.scalar.dat                - H atom VMC data with a problem
│
└── size                                 - scaling with number of particles
    ├──  01________H.s000.scalar.dat       - H atom VMC data
    ├──  02_______H2.s000.scalar.dat       - H dimer "   "
    ├──  06________C.s000.scalar.dat       - C atom  "   "
    ├──  10______CH4.s000.scalar.dat       - methane "   "
    ├──  12_______C2.s000.scalar.dat       - C dimer "   "
    ├──  16_____C2H4.s000.scalar.dat       - ethene  "
    ├──  18___CH4CH4.s000.scalar.dat       - methane dimer VMC data
    ├──  32_C2H4C2H4.s000.scalar.dat       - ethene dimer   "   "
    ├──  nelectron_tcpu.dat                - data for gnuplot
    └──  Nelectron_tCPU.plt                - gnuplot for N_elec vs. t_CPU
```

17.3 Atomic units

QMCPACK operates in Hartree atomic units to reduce the number of factors in the Schrödinger equation. Thus, the unit of length is the bohr (5.291772×10^{-11} m = 0.529177 Å); the unit of energy is the hartree (4.359744×10^{-18} J = 27.211385 eV). The energy of the ground state of the hydrogen atom in these units is -0.5 hartrees.

17.4 Reviewing statistics

We will practice taking the average (mean) and standard deviation of some Monte Carlo data by hand to review the basic definitions.

Enter Python's command line by typing **python** [**Enter**]. You will see a prompt "»>".

The mean of a data set is given by:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{17.1}$$

To calculate the average of five local energies from a MC calculation of the ground state of an electron in the hydrogen atom, input (truncate at the thousandths place if you cannot copy and paste; script versions are also available in the `average` directory):

```
(
(-0.45298911858) +
(-0.45481953564) +
(-0.48066105923) +
(-0.47316713469) +
(-0.46204733302)
)/5.
```

Then, press [**Enter**] to get:

```
>>> ((-0.45298911858) + (-0.45481953564) + (-0.48066105923) +
(-0.47316713469) + (-0.4620473302))/5.
-0.46473683566800006
```

To understand the significance of the mean, we also need the standard deviation around the mean of the data (also called the error bar), given by:

$$\sigma = \sqrt{\frac{1}{N(N-1)}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{17.2}$$

To calculate the standard deviation around the mean (-0.464736835668) of these five data points, put in:

```
( (1./(5.*(5.-1.))) * (
(-0.45298911858-(-0.464736835668))**2 + \\
(-0.45481953564-(-0.464736835668))**2 +
(-0.48066105923-(-0.464736835668))**2 +
(-0.47316713469-(-0.464736835668))**2 +
(-0.46204733302-(-0.464736835668))**2 )
)**0.5
```

Then, press [**Enter**] to get:

```
>>> ( (1./(5.*(5.-1.))) * ( (-0.45298911858-(-0.464736835668))**2 +
(-0.45481953564-(-0.464736835668))**2 + (-0.48066105923-(-0.464736835668))**2 +
(-0.47316713469-(-0.464736835668))**2 + (-0.46204733302-(-0.464736835668))**2
) )**0.5
0.0053303187464332066
```

Thus, we might report this data as having a value -0.465 +/- 0.005 hartrees. This calculation of the standard deviation assumes that the average for this data is fixed, but we may continually add Monte Carlo samples to the data so it is better to use an estimate of the error bar that does not rely on the overall average. Such an estimate is given by:

$$\tilde{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}[(x^2)_i - (x_i)^2]} \tag{17.3}$$

To calculate the standard deviation with this formula, input the following, which includes the square of the local energy calculated with each corresponding local energy:

```
( (1./(5.-1.)) * (
(0.60984565298-(-0.45298911858)**2) + \\
(0.61641291630-(-0.45481953564)**2) +
(1.35860151160-(-0.48066105923)**2) + \\
(0.78720769003-(-0.47316713469)**2) +
(0.56393677687-(-0.46204733302)**2) )
)**0.5
```

and press [**Enter**] to get:

```
>>> ((1./(5.-1.))*((0.60984565298-(-0.45298911858)**2)+
(0.61641291630-(-0.45481953564)**2)+(1.35860151160-(-0.48066105923)**2)+
(0.78720769003-(-0.47316713469)**2)+(0.56393677687-(-0.46204733302)**2))
)**0.5
0.84491636672906634
```

This much larger standard deviation, acknowledging that the mean of this small data set is not the average in the limit of infinite sampling more accurately, reports the value of the local energy as -0.5 +/- 0.8 hartrees.

Type **quit()** and press [**Enter**] to exit the Python command line.

17.5 Inspecting Monte Carlo data

QMCPACK outputs data from MC calculations into files ending in scalar.dat. Several quantities are calculated and written for each block of Monte Carlo steps in successive columns to the right of the step index.

Change directories to `atom`, and open the file ending in scalar.dat with a text editor (e.g., **vi *.scalar.dat** or **emacs *.scalar.dat**. If possible, adjust the terminal so that lines do not wrap. The data will begin as follows (broken into three groups to fit on this page):

```
#   index    LocalEnergy         LocalEnergy_sq      LocalPotential     ...
          0   -4.5298911858e-01    6.0984565298e-01   -1.1708693521e+00
          1   -4.5481953564e-01    6.1641291630e-01   -1.1863425644e+00
          2   -4.8066105923e-01    1.3586015116e+00   -1.1766446209e+00
          3   -4.7316713469e-01    7.8720769003e-01   -1.1799481122e+00
          4   -4.6204733302e-01    5.6393677687e-01   -1.1619244081e+00
          5   -4.4313854290e-01    6.0831516179e-01   -1.2064503041e+00
          6   -4.5064926960e-01    5.9891422196e-01   -1.1521370176e+00
          7   -4.5687452611e-01    5.8139614676e-01   -1.1423627617e+00
          8   -4.5018503739e-01    8.4147849706e-01   -1.1842075439e+00
          9   -4.3862013841e-01    5.5477715836e-01   -1.2080979177e+00
```

The first line begins with a #, indicating that this line does not contain MC data but rather the labels of the columns. After a blank line, the remaining lines consist of the MC data. The first column, labeled index, is an integer indicating which block of MC data is on that line. The second column contains the quantity usually of greatest interest from the simulation, the local energy. Since this simulation did not use the exact ground state wave function, it does not produce -0.5 hartrees as the local energy although the value lies within about 10%. The value of the local energy fluctuates from block to block and the closer the trial wave function is to the ground state, the smaller these fluctuations will be. The next column contains an important ingredient in estimating the error in the MC average–the square of the local energy–found by evaluating the square of the Hamiltonian.

```
...   Kinetic             Coulomb             BlockWeight        ...
       7.1788023352e-01   -1.1708693521e+00    1.2800000000e+04
       7.3152302871e-01   -1.1863425644e+00    1.2800000000e+04
       6.9598356165e-01   -1.1766446209e+00    1.2800000000e+04
       7.0678097751e-01   -1.1799481122e+00    1.2800000000e+04
       6.9987707508e-01   -1.1619244081e+00    1.2800000000e+04
       7.6331176120e-01   -1.2064503041e+00    1.2800000000e+04
       7.0148774798e-01   -1.1521370176e+00    1.2800000000e+04
       6.8548823555e-01   -1.1423627617e+00    1.2800000000e+04
       7.3402250655e-01   -1.1842075439e+00    1.2800000000e+04
       7.6947777925e-01   -1.2080979177e+00    1.2800000000e+04
```

The fourth column from the left consists of the values of the local potential energy. In this simulation, it is identical to the Coulomb potential (contained in the sixth column) because the one electron in the simulation has only the potential energy coming from its interaction with the nucleus. In many-electron simulations, the local potential energy contains contributions from the electron-electron Coulomb interactions and the nuclear potential or pseudopotential. The fifth column contains the local kinetic energy value for each MC block, obtained from the Laplacian of the wave function. The sixth column shows the local Coulomb interaction energy. The seventh column displays the weight each line of data has in the average (the weights are identical in this simulation).

```
...    BlockCPU            AcceptRatio
       6.0178991748e-03    9.8515625000e-01
       5.8323097461e-03    9.8562500000e-01
       5.8213412744e-03    9.8531250000e-01
       5.8330412549e-03    9.8828125000e-01
       5.8108362256e-03    9.8625000000e-01
       5.8254170264e-03    9.8625000000e-01
       5.8314813086e-03    9.8679687500e-01
       5.8258469971e-03    9.8726562500e-01
       5.8158433545e-03    9.8468750000e-01
       5.7959401123e-03    9.8539062500e-01
```

The eighth column shows the CPU time (in seconds) to calculate the data in that line. The ninth column from the left contains the acceptance ratio (1 being full acceptance) for Monte Carlo steps in that line's data. Other than the block weight, all quantities vary from line to line.

Exit the text editor ([**Esc**] **:q!** [**Enter**] in vi, [**Ctrl**]**-x** [**Ctrl**]**-c** in emacs).

17.6 Averaging quantities in the MC data

QMCPACK includes the qmca Python tool to average quantities in the scalar.dat file (and also the dmc.dat file of DMC simulations). Without any flags, qmca will output the average of each column

with a quantity in the scalar.dat file as follows.

Execute qmca by **qmca *.scalar.dat**, which for this data outputs:

```
H  series 0
LocalEnergy           =          -0.45446 +/-          0.00057
Variance              =             0.529 +/-            0.018
Kinetic               =            0.7366 +/-           0.0020
LocalPotential        =           -1.1910 +/-           0.0016
Coulomb               =           -1.1910 +/-           0.0016
LocalEnergy_sq        =             0.736 +/-            0.018
BlockWeight           =    12800.00000000 +/-       0.00000000
BlockCPU              =        0.00582002 +/-       0.00000067
AcceptRatio           =          0.985508 +/-         0.000048
Efficiency            =        0.00000000 +/-       0.00000000
```

After one blank, qmca prints the title of the subsequent data, gleaned from the data file name. In this case, H.s000.scalar.dat became "H series 0". Everything before the first ".s" will be interpreted as the title, and the number between ".s" and the next "." will be interpreted as the series number.

The first column under the title is the name of each quantity qmca averaged. The column to the right of the equal signs contains the average for the quantity of that line, and the column to the right of the plus-slash-minus is the statistical error bar on the quantity. All quantities calculated from MC simulations have and must be reported with a statistical error bar!

Two new quantities not present in the scalar.dat file are computed by qmca from the data– variance and efficiency. We will look at these later in this lab.

To view only one value, **qmca** takes the **-q (quantity)** flag. For example, the output of **qmca -q LocalEnergy *.scalar.dat** in this directory produces a single line of output:

```
H  series 0  LocalEnergy = -0.454460 +/- 0.000568
```

Type **qmca –help** to see the list of all quantities and their abbreviations.

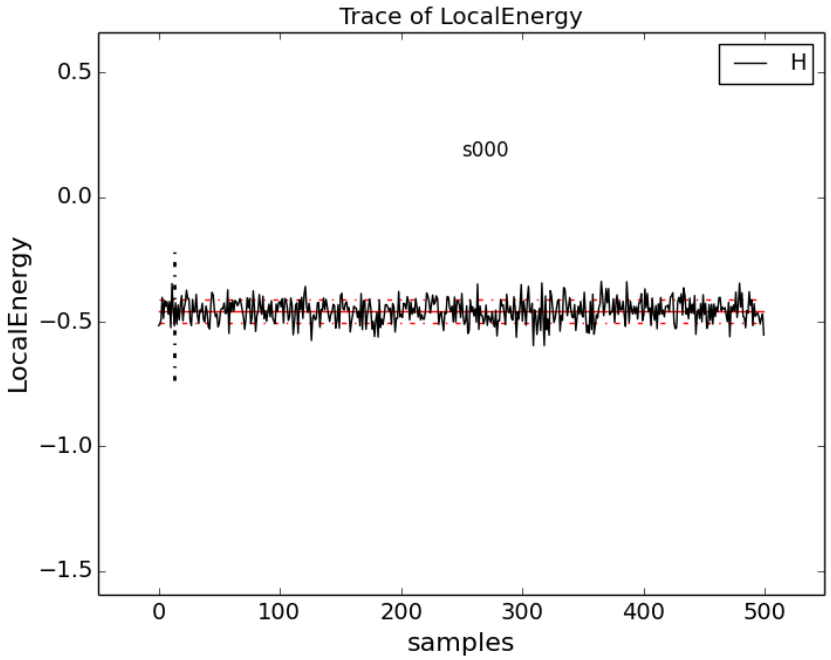
17.7 Evaluating MC simulation quality

There are several aspects of a MC simulation to consider in deciding how well it went. Besides the deviation of the average from an expected value (if there is one), the stability of the simulation in its sampling, the autocorrelation between MC steps, the value of the acceptance ratio (accepted steps over total proposed steps), and the variance in the local energy all indicate the quality of a MC simulation. We will look at these one by one.

17.7.1 Tracing MC quantities

Visualizing the evolution of MC quantities over the course of the simulation by a *trace* offers a quick picture of whether the random walk had expected behavior. qmca plots traces with the -t flag.

Type **qmca -q e -t H.s000.scalar.dat**, which produces a graph of the trace of the local energy:

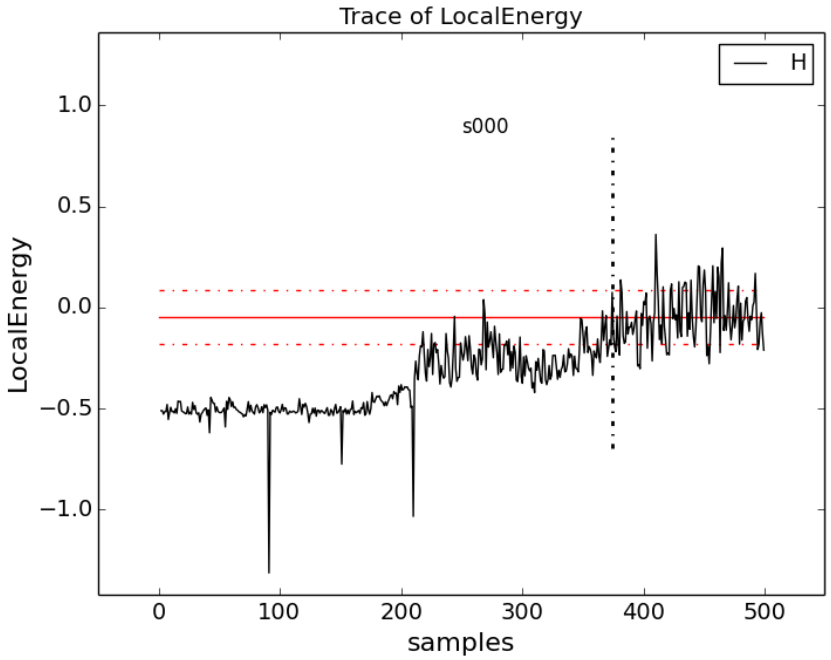


The solid black line connects the values of the local energy at each MC block (labeled "samples"). The average value is marked with a horizontal, solid red line. One standard deviation above and below the average are marked with horizontal, dashed red lines.

The trace of this run is largely centered around the average with no large-scale oscillations or major shifts, indicating a good quality MC run.

Try tracing the kinetic and potential energies, seeing that their behavior is comparable to the total local energy.

Change to directory `problematic` and type **qmca -q e -t H.s000.scalar.dat** to produce this graph:



Here, the local energy samples cluster around the expected -0.5 hartrees for the first 150 samples or so and then begin to oscillate more wildly and increase erratically toward 0, indicating a poor quality MC run.

Again, trace the kinetic and potential energies in this run and see how their behavior compares to the total local energy.

17.7.2 Blocking away autocorrelation

Autocorrelation occurs when a given MC step biases subsequent MC steps, leading to samples that are not statistically independent. We must take this autocorrelation into account in order to obtain accurate statistics. qmca outputs autocorrelation when given the --sac flag.

Change to directory `autocorrelation` and type **qmca -q e --sac H.s000.scalar.dat**.

```
H  series 0  LocalEnergy = -0.454982 +/- 0.000430    1.0
```

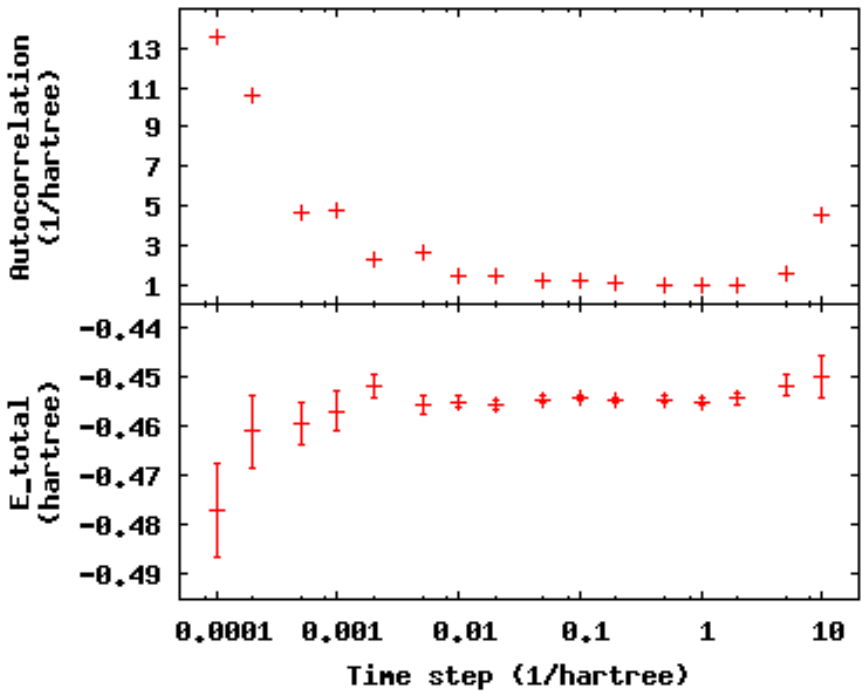
The value after the error bar on the quantity is the autocorrelation (1.0 in this case).

Proposing too small a step in configuration space, the MC *time step*, can lead to autocorrelation since the new samples will be in the neighborhood of previous samples. Type **grep timestep H.xml** to see the varying time step values in this QMCPACK input file (H.xml):

```
<parameter name="timestep">10</parameter>
<parameter name="timestep">5</parameter>
<parameter name="timestep">2</parameter>
<parameter name="timestep">1</parameter>
<parameter name="timestep">0.5</parameter>
<parameter name="timestep">0.2</parameter>
<parameter name="timestep">0.1</parameter>
<parameter name="timestep">0.05</parameter>
<parameter name="timestep">0.02</parameter>
<parameter name="timestep">0.01</parameter>
```

```
<parameter name="timestep">0.005</parameter>
<parameter name="timestep">0.002</parameter>
<parameter name="timestep">0.001</parameter>
<parameter name="timestep">0.0005</parameter>
<parameter name="timestep">0.0002</parameter>
<parameter name="timestep">0.0001</parameter>
```

Generally, as the time step decreases, the autocorrelation will increase (caveat: very large time steps will also have increasing autocorrelation). To see this, type **qmca -q e --sac *.scalar.dat** to see the energies and autocorrelation times, then plot with gnuplot by inputting **gnuplot H.plt**:

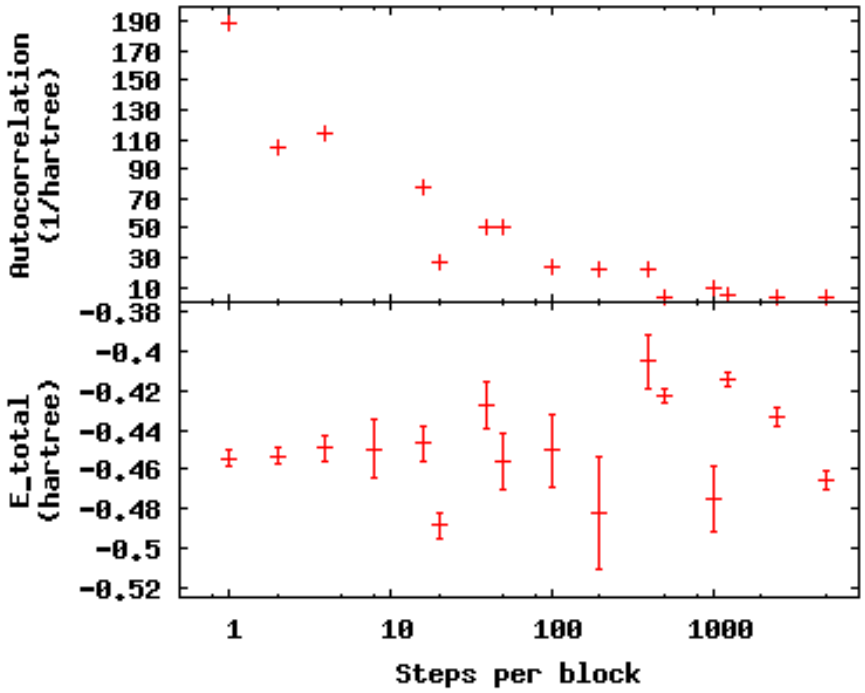


The error bar also increases with the autocorrelation.

Press **q** [**Enter**] to quit gnuplot.

To get around the bias of autocorrelation, we group the MC steps into blocks, take the average of the data in the steps of each block, and then finally average the averages in all the blocks. QMCPACK outputs the block averages as each line in the scalar.dat file. (For DMC simulations, in addition to the scalar.dat, QMCPACK outputs the quantities at each step to the dmc.dat file, which permits reblocking the data differently from the specification in the input file.)

Change directories to `blocking`. Here we look at the time step of the last data set in the `autocorrelation` directory. Verify this by typing **grep timestep H.xml** to see that all values are set to 0.001. Now to see how we will vary the blocking, type **grep -A1 blocks H.xml**. The parameter "steps" indicates the number of steps per block, and the parameter "blocks" gives the number of blocks. For this comparison, the total number of MC steps (equal to the product of "steps" and "blocks") is fixed at 50000. Now check the effect of blocking on autocorrelation–type **qmca -q e --sac *scalar.dat** to see the data and **gnuplot H.plt** to visualize the data:



The greatest number of steps per block produces the smallest autocorrelation time. The larger number of blocks over which to average at small step-per-block number masks the corresponding increase in error bar with increasing autocorrelation.

Press **q** [**Enter**] to quit gnuplot.

17.7.3 Balancing autocorrelation and acceptance ratio

Adjusting the time step value also affects the ratio of accepted steps to proposed steps. Stepping nearby in configuration space implies that the probability distribution is similar and thus more likely to result in an accepted move. Keeping the acceptance ratio high means the algorithm is efficiently exploring configuration space and not sticking at particular configurations. Return to the `autocorrelation` directory. Refresh your memory on the time steps in this set of simulations by **grep timestep H.xml**. Then, type **qmca -q ar *scalar.dat** to see the acceptance ratio as it varies with decreasing time step:

```
H  series 0  AcceptRatio = 0.047646 +/- 0.000206
H  series 1  AcceptRatio = 0.125361 +/- 0.000308
H  series 2  AcceptRatio = 0.328590 +/- 0.000340
H  series 3  AcceptRatio = 0.535708 +/- 0.000313
H  series 4  AcceptRatio = 0.732537 +/- 0.000234
H  series 5  AcceptRatio = 0.903498 +/- 0.000156
H  series 6  AcceptRatio = 0.961506 +/- 0.000083
H  series 7  AcceptRatio = 0.985499 +/- 0.000051
H  series 8  AcceptRatio = 0.996251 +/- 0.000025
H  series 9  AcceptRatio = 0.998638 +/- 0.000014
H  series 10  AcceptRatio = 0.999515 +/- 0.000009
H  series 11  AcceptRatio = 0.999884 +/- 0.000004
H  series 12  AcceptRatio = 0.999958 +/- 0.000003
H  series 13  AcceptRatio = 0.999986 +/- 0.000002
H  series 14  AcceptRatio = 0.999995 +/- 0.000001
```

```
H  series 15  AcceptRatio = 0.999999 +/- 0.000000
```

By series 8 (time step = 0.02), the acceptance ratio is in excess of 99%.

Considering the increase in autocorrelation and subsequent increase in error bar as time step decreases, it is important to choose a time step that trades off appropriately between acceptance ratio and autocorrelation. In this example, a time step of 0.02 occupies a spot where acceptance ratio is high (99.6%), and autocorrelation is not appreciably larger than the minimum value (1.4 vs. 1.0).

17.7.4 Considering variance

Besides autocorrelation, the dominant contributor to the error bar is the *variance* in the local energy. The variance measures the fluctuations around the average local energy, and, as the fluctuations go to zero, the wave function reaches an exact eigenstate of the Hamiltonian. qmca calculates this from the local energy and local energy squared columns of the scalar.dat.

Type **qmca -q v H.s009.scalar.dat** to calculate the variance on the run with time step balancing autocorrelation and acceptance ratio:

```
H  series 9  Variance = 0.513570 +/- 0.010589
```

Just as the total energy doesn't tell us much by itself, neither does the variance. However, comparing the ratio of the variance to the energy indicates how the magnitude of the fluctuations compares to the energy itself. Type **qmca -q ev H.s009.scalar.dat** to calculate the energy and variance on the run side by side with the ratio:

```
                     LocalEnergy               Variance        ratio
H  series 0  -0.454460 +/- 0.000568   0.529496 +/- 0.018445   1.1651
```

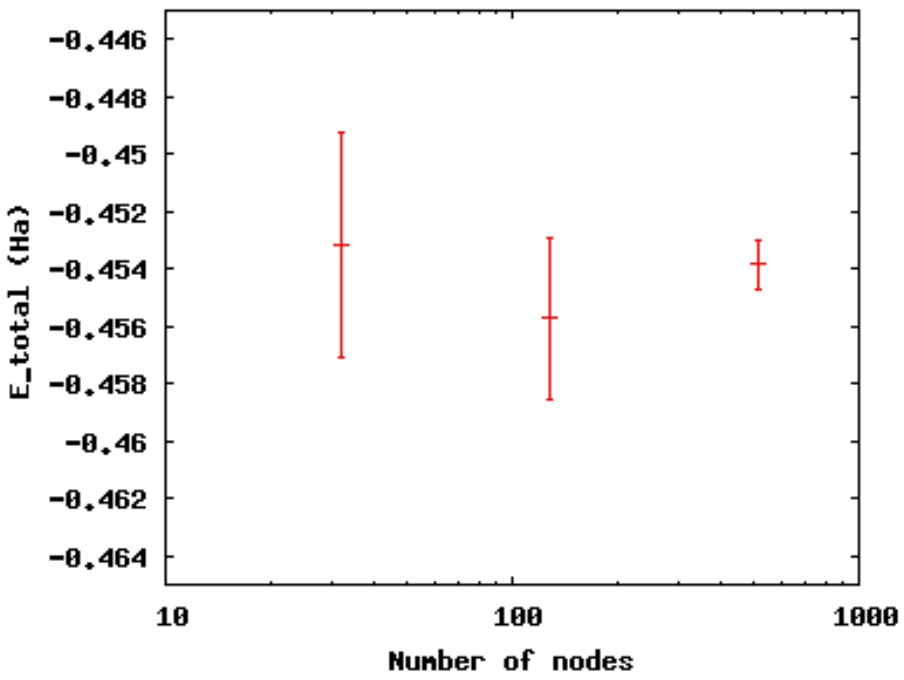
1.1651 is a very high ratio indicating the square of the fluctuations is on average larger than the value itself. In the next section, we will approach ways to improve the variance that subsequent labs will build upon.

17.8 Reducing statistical error bars

17.8.1 Increasing MC sampling

Increasing the number of MC samples in a data set reduces the error bar as the inverse of the square root of the number of samples. There are two ways to increase the number of MC samples in a simulation: running more samples in parallel and increasing the number of blocks (with fixed number of steps per block, this increases the total number of MC steps).

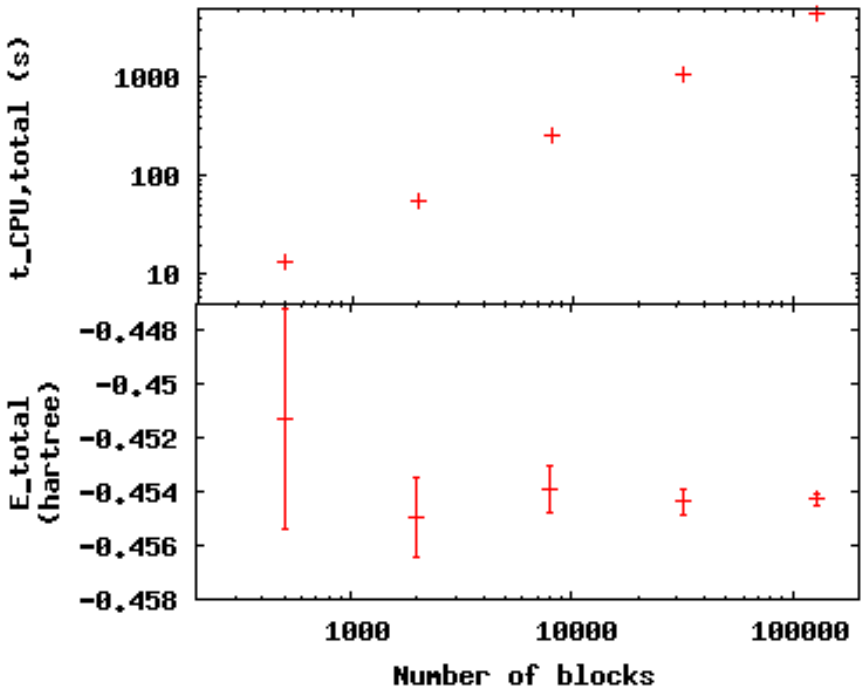
To see the effect of the running more samples in parallel, change to the directory `nodes`. The series here increases the number of nodes by factors of four from 32 to 128 to 512. Type **qmca -q ev *scalar.dat** and note the change in the error bar on the local energy as the number of nodes. Visualize this with **gnuplot H.plt**:



Increasing the number of blocks, unlike running in parallel, increases the total CPU time of the simulation.

Press **q** [**Enter**] to quit gnuplot.

To see the effect of increasing the block number, change to the directory `blocks`. To see how we will vary the number of blocks, type **grep -A1 blocks H.xml**. The number of steps remains fixed, thus increasing the total number of samples. Visualize the tradeoff by inputting **gnuplot H.plt**:



Press **q** [**Enter**] to quit gnuplot.

17.8.2 Improving the basis set

In all of the above examples, we are using the sum of two gaussian functions (STO-2G) to approximate what should be a simple decaying exponential (STO = Slater-type orbital) for the wave function of the ground state of the hydrogen atom. The sum of multiple copies of a function varying each copy's width and amplitude with coefficients is called a *basis set*. As we add gaussians to the basis set, the approximation improves, the variance goes toward zero and the energy goes to -0.5 hartrees. In nearly every other case, the exact function is unknown, and we add basis functions until the total energy does not change within some threshold.

Change to the directory `basis` and look at the total energy and variance as we change the wave function by typing **qmca -q ev H_*:**

```
                        LocalEnergy              Variance           ratio
H_STO-2G  series 0  -0.454460 +/- 0.000568   0.529496 +/- 0.018445   1.1651
H_STO-3G  series 0  -0.465386 +/- 0.000502   0.410491 +/- 0.010051   0.8820
H_STO-6G  series 0  -0.471332 +/- 0.000491   0.213919 +/- 0.012954   0.4539
H_exact   series 0  -0.500000 +/- 0.000000   0.000000 +/- 0.000000   -0.0000
```

qmca also puts out the ratio of the variance to the local energy in a column to the right of the variance error bar. A typical high quality value for this ratio is lower than 0.1 or so–none of these few-gaussian wave functions satisfy that rule of thumb.

Use qmca to plot the trace of the local energy, kinetic energy, and potential energy of H_exact– the total energy is constantly -0.5 hartree even though the kinetic and potential energies fluctuate from configuration to configuration.

17.8.3 Adding a Jastrow factor

Another route to reducing the variance is the introduction of a Jastrow factor to account for electron-electron correlation (not the statistical autocorrelation of Monte Carlo steps but the physical avoidance that electrons have of one another). To do this, we will switch to the hydrogen dimer with the exact ground state wave function of the atom (STO basis)–this will not be exact for the dimer. The ground state energy of the hydrogen dimer is -1.174 hartrees.

Change directories to `dimer` and put in **qmca -q ev *scalar.dat** to see the result of adding a simple, one-parameter Jastrow to the STO basis for the hydrogen dimer at experimental bond length:

```
                            LocalEnergy               Variance
H2_STO___no_jastrow  series 0  -0.876548 +/- 0.005313   0.473526 +/- 0.014910
H2_STO_with_jastrow  series 0  -0.912763 +/- 0.004470   0.279651 +/- 0.016405
```

The energy reduces by 0.044 +/- 0.006 hartrees and the variance by 0.19 +/- 0.02. This is still 20% above the ground state energy, and subsequent labs will cover how to improve on this with improved forms of the wave function that capture more of the physics.

17.9 Scaling to larger numbers of electrons

17.9.1 Calculating the efficiency

The inverse of the product of CPU time and the variance measures the *efficiency* of an MC calculation. Use qmca to calculate efficiency by typing **qmca -q eff *scalar.dat** to see the efficiency of these two H_2 calculations:

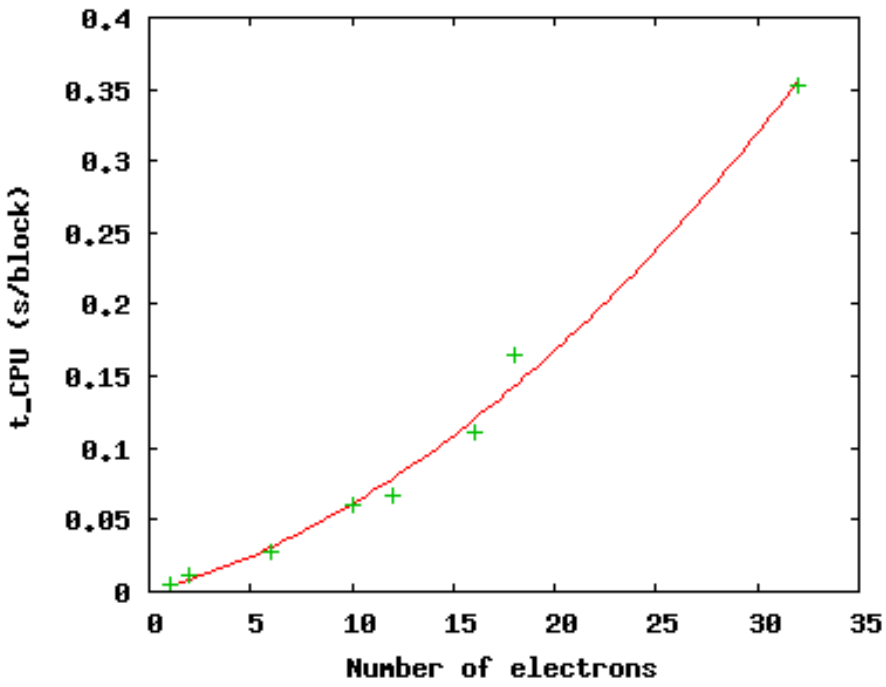
```
H2_STO___no_jastrow  series 0  Efficiency = 16698.725453 +/- 0.000000
H2_STO_with_jastrow  series 0  Efficiency = 52912.365609 +/- 0.000000
```

The Jastrow factor increased the efficiency in these calculations by a factor of three, largely through the reduction in variance (check the average block CPU time to verify this claim).

17.9.2 Scaling up

To see how MC scales with increasing particle number, change directories to `size`. Here are the data from runs of increasing number of electrons for H, H_2, C, CH_4, C_2, C_2H_4, $(CH_4)_2$, and $(C_2H_4)_2$ using the STO-6G basis set for the orbitals of the Slater determinant. The file names begin with the number of electrons simulated for those data.

Use **qmca -q bc *scalar.dat** to see that the CPU time per block increases with number of electrons in the simulation, then plot the total CPU time of the simulation by **gnuplot Nelectron_tCPU.plt**:



The green pluses represent the CPU time per block at each electron number. The red line is a quadratic fit to those data. For a fixed basis set size, we expect the time to scale quadratically up to 1000s of electrons, at which point a cubic scaling term may become dominant. Knowing the scaling allows you to roughly project the calculation time for a larger number of electrons.

Press **q** [**Enter**] to quit gnuplot.

This isn't the whole story, however. The variance of the energy also increases with a fixed basis set as the number of particles increases at a faster rate than the energy decreases. To see this, type **qmca -q ev *scalar.dat**:

```
                          LocalEnergy              Variance
01________H  series 0  -0.471352 +/- 0.000493     0.213020 +/- 0.012950
02_______H2  series 0  -0.898875 +/- 0.000998     0.545717 +/- 0.009980
06________C  series 0  -37.608586 +/- 0.020453   184.322000 +/- 45.481193
10______CH4  series 0  -38.821513 +/- 0.022740   169.797871 +/- 24.765674
12_______C2  series 0  -72.302390 +/- 0.037691   491.416711 +/- 106.090103
16_____C2H4  series 0  -75.488701 +/- 0.042919   404.218115 +/- 60.196642
18___CH4CH4  series 0  -58.459857 +/- 0.039309   498.579645 +/- 92.480126
32_C2H4C2H4  series 0  -91.567283 +/- 0.048392   632.114026 +/- 69.637760
```

The increase in variance is not uniform, but the general trend is upward with a fixed wave function form and basis set. Subsequent labs will address how to improve the wave function in order to keep the variance manageable.

Chapter 18

Lab 2: QMC Basics

18.1 Topics covered in this Lab

This lab focuses on the basics of performing quality QMC calculations. As an example participants test an oxygen pseudopotential within DMC by calculating atomic and dimer properties, a common step prior to production runs. Topics covered include:

- converting pseudopotentials into QMCPACK's FSATOM format
- generating orbitals with Quantum ESPRESSO
- converting orbitals into QMCPACK's ESHDF format with pw2qmcpack
- optimizing Jastrow factors with QMCPACK
- removing DMC timestep error via extrapolation
- automating QMC workflows with Nexus
- testing pseudopotentials for accuracy

18.2 Lab outline

1. download and conversion of oxygen atom pseudopotential
2. DMC timestep study of the neutral oxygen atom
   (a) DFT orbital generation with Quantum ESPRESSO
   (b) orbital conversion with `pw2qmcpack.x`
   (c) optimization of Jastrow correlation factor with QMCPACK
   (d) DMC run with multiple timesteps
3. DMC timestep study of the first ionization potential of oxygen
   (a) repetition of a-d above for ionized oxygen atom
4. automated DMC calculations of the oxygen dimer binding curve

18.3 Lab directories and files

```
%
labs/lab2_qmc_basics/
|
├── oxygen_atom           - oxygen atom calculations
|   ├── O.q0.dft.in          - Quantum ESPRESSO input for DFT run
|   ├── O.q0.p2q.in          - pw2qmcpack.x input for orbital conversion run
|   ├── O.q0.opt.in.xml      - QMCPACK input for Jastrow optimization run
|   ├── O.q0.dmc.in.xml      - QMCPACK input file for neutral O DMC
|   ├── ip_conv.py           - tool to fit oxygen IP vs timestep
|   └── reference            - directory w/ completed runs
|
├── oxygen_dimer          - oxygen dimer calculations
|   ├── dimer_fit.py         - tool to fit dimer binding curve
|   ├── O_dimer.py           - automation script for dimer calculations
|   ├── pseudopotentials     - directory for pseudopotentials
|   └── reference            - directory w/ completed runs
|
└── your_system           - performing calculations for an arbitrary system (yours)
    ├── example.py           - example nexus file for periodic diamond
    ├── pseudopotentials     - directory containing C pseudopotentials
    └── reference            - directory w/ completed runs
```

18.4 Obtaining and converting a pseudopotential for oxygen

First enter the `oxygen_atom` directory:

```
cd labs/lab2_qmc_basics/oxygen_atom/
```

Throughout the rest of the lab, locations will be specified with respect to `labs/lab2_qmc_basics` (e.g. `oxygen_atom`).

We will use a potential from the Burkatzki-Filippi-Dolg pseudopotential database. Although the full database is available in QMCPACK distribution (`trunk/pseudopotentials/BFD/`), we use a BFD pseudopotential to illustrate the process of converting and testing an external potential for use with QMCPACK. To obtain the pseudopotential, go to http://www.burkatzki.com/pseudos/index.2.html and click on the "Select Pseudopotential" button. Next click on oxygen in the periodic table. Click on the empty circle next to "V5Z" (a large gaussian basis set) and click on "Next". Select the Gamess format and click on "Retrive Potential". Helpful information about the pseudopotential will be displayed. The desired portion is at the bottom (the last 7 lines). Copy this text into the editor of your choice (e.g. `emacs` or `vi`) and save it as `O.BFD.gamess` (be sure to include a newline at the end of the file). To transform the pseudopotential into the FSATOM XML format used by QMCPACK, use the `ppconvert` tool:

```
ppconvert --gamess_pot O.BFD.gamess --s_ref "1s(2)2p(4)" \
 --p_ref "1s(2)2p(4)" --d_ref "1s(2)2p(4)" --xml O.BFD.xml
```

Observe the notation used to describe the reference valence configuration for this helium-core PP: `1s(2)2p(4)`. The `ppconvert` tool uses the following convention for the valence states: the first s state is labeled `1s` (`1s`, `2s`, `3s`, …), the first p state is labeled `2p` (`2p`, `3p`, …), the first d state is labeled `3d` (`3d`, `4d`, …). Copy the resulting xml file into the `oxygen_atom` directory.

Note: the command to convert the PP into QM Espresso's UPF format is similar (both formats are required):

```
ppconvert --gamess_pot O.BFD.gamess --s_ref "1s(2)2p(4)" \
 --p_ref "1s(2)2p(4)" --d_ref "1s(2)2p(4)" --log_grid --upf O.BFD.upf
```

For reference, the text of `O.BFD.gamess` should be:

```
O-QMC GEN 2 1
3
6.00000000 1 9.29793903
55.78763416 3 8.86492204
-38.81978498 2 8.62925665
1
38.41914135 2 8.71924452
```

The full QMCPACK pseudopotential is also included in `oxygen_atom/reference/O.BFD.*`.

18.5 DFT with Quantum ESPRESSO to obtain the orbital part of the wavefunction

With the pseudopotential in hand, the next step toward a QMC calculation is to obtain the Fermionic part of the wavefunction, in this case a single Slater determinant constructed from DFT-LDA orbitals for a neutral oxygen atom. If you had trouble with the pseudopotential conversion step, pre-converted pseudopotential files are located in the `oxygen_atom/reference` directory.

Quantum ESPRESSO input for the DFT-LDA ground state of the neutral oxygen atom can be found in `O.q0.dft.in` and also listing 18.1 below. Setting `wf_collect=.true.` instructs Quantum Espresso to write the orbitals to disk at the end of the run. Option `wf_collect=.true.` may be a potential problem in large simulations, it is recommended to avoid it and use the converter pw2qmcpack in parallel, see details in Sec. 22.3.2. Note that the plane-wave energy cutoff has been set to a reasonable value of 300 Ry here (`ecutwfc=300`). This value depends on the pseudopotentials used, and in general should be selected by running DFT→(orbital conversion)→VMC with increasing energy cutoffs until the lowest VMC total energy and variance is reached.

Listing 18.1: Quantum ESPRESSO input file for the neutral oxygen atom (`O.q0.dft.in`)

```
&CONTROL
   calculation       = 'scf'
   restart_mode      = 'from_scratch'
   prefix            = 'O.q0'
   outdir            = './'
   pseudo_dir        = './'
   disk_io           = 'low'
   wf_collect        = .true.
/

&SYSTEM
   celldm(1)         = 1.0
   ibrav             = 0
   nat               = 1
   ntyp              = 1
   nspin             = 2
```

```
   tot_charge        = 0
   tot_magnetization = 2
   input_dft         = 'lda'
   ecutwfc           = 300
   ecutrho           = 1200
   nosym             = .true.
   occupations       = 'smearing'
   smearing          = 'fermi-dirac'
   degauss           = 0.0001
/

&ELECTRONS
   diagonalization   = 'david'
   mixing_mode       = 'plain'
   mixing_beta       = 0.7
   conv_thr          = 1e-08
   electron_maxstep  = 1000
/


ATOMIC_SPECIES
   O  15.999 O.BFD.upf

ATOMIC_POSITIONS alat
   O     9.44863067       9.44863161       9.44863255

K_POINTS automatic
   1 1 1  0 0 0

CELL_PARAMETERS cubic
        18.89726133       0.00000000       0.00000000
         0.00000000      18.89726133       0.00000000
         0.00000000       0.00000000      18.89726133
```

Run Quantum ESPRESSO by typing

```
mpirun -np 4 pw.x -input O.q0.dft.in >&O.q0.dft.out&
```

The DFT run should take a few minutes to complete. If desired, you can track the progress of the DFT run by typing "`tail -f O.q0.dft.out` ". Once finished, you should check the LDA total energy in `O.q0.dft.out`by typing "`grep '! ' O.q0.dft.out`". The result should be close to

```
!    total energy              =     -31.57553905 Ry
```

The orbitals have been written in a format native to Quantum ESPRESSO in the `O.q0.save` directory. We will convert them into the ESHDF format expected by QMCPACK by using the `pw2qmcpack.x` tool. The input for `pw2qmcpack.x` can be found in the file `O.q0.p2q.in` and also in listing 18.2 below.

Listing 18.2: `pw2qmcpack.x` input file for orbital conversion (`O.q0.p2q.in`)

```
&inputpp
   prefix      = 'O.q0'
   outdir      = './'
   write_psir  = .false.
/
```

Perform the orbital conversion now by typing the following:

```
mpirun -np 1 pw2qmcpack.x<O.q0.p2q.in>&O.q0.p2q.out&
```

Upon completion of the run, a new file should be present containing the orbitals for QMCPACK: `O.q0.pwscf.h5`. Template XML files for particle (`O.q0.ptcl.xml`) and wavefunction (`O.q0.wfs.xml`) inputs to QMCPACK should also be present.

18.6 Optimization with QMCPACK to obtain the correlated part of the wavefunction

The wavefunction we have obtained to this point corresponds to a non-interacting Hamiltonian. Once the Coulomb pair potential is switched on between particles, it is known analytically that the exact wavefunction has cusps whenever two particles meet spatially and in general the electrons become correlated. This is represented in the wavefunction by introducing a Jastrow factor containing at least pair correlations

$$\Psi_{Slater-Jastrow} = e^{-J}\Psi_{Slater} \tag{18.1}$$

$$J = \sum_{\sigma\sigma'}\sum_{i<j} u_2^{\sigma\sigma'}(|r_i - r_j|) + \sum_{\sigma}\sum_{iI} u_1^{\sigma I}(|r_i - r_I|) \tag{18.2}$$

Here σ is a spin variable while r_i and r_I represent electron and ion coordinates, respectively. The introduction of J into the wavefunction is similar to F12 methods in quantum chemistry, though it has been present in essentially all QMC studies since the first applications the method (circa 1965).

How are the functions $u_2^{\sigma\sigma'}$ and u_1^{σ} obtained? Generally, they are approximated by analytical functions with several unknown parameters that are determined by minimizing the energy or variance directly within VMC. This is effective because the energy and variance reach a global minimum only for the true ground state wavefunction (Energy $= E \equiv \langle\Psi|\hat{H}|\Psi\rangle$, Variance $= V \equiv \langle\Psi|(\hat{H} - E)^2|\Psi\rangle$). For this exercise, we will focus on minimizing the variance.

First, we need to update the template particle and wavefunction information in `O.q0.ptcl.xml` and `O.q0.wfs.xml`. We want to simulate the O atom in open boundary conditions (the default is periodic). To do this open `O.q0.ptcl.xml` with your favorite text editor (e.g. `emacs` or `vi`) and replace

```
<parameter name="bconds">
   p p p
</parameter>
<parameter name="LR_dim_cutoff">
   15
</parameter>
```

with

```
<parameter name="bconds">
   n n n
</parameter>
```

Next we will select Jastrow factors appropriate for an atom. In open boundary conditions, the B-spline Jastrow correlation functions should cut off to zero at some distance away from the atom. Open `O.q0.wfs.xml` and add the following cutoffs (`rcut` in Bohr radii) to the correlation factors:

```
...
<correlation speciesA="u" speciesB="u" size="8" rcut="10.0">
...
<correlation speciesA="u" speciesB="d" size="8" rcut="10.0">
...
<correlation elementType="O" size="8" rcut="5.0">
...
```

These terms correspond to $u_2^{\uparrow\uparrow}/u_2^{\downarrow\downarrow}$, $u_2^{\uparrow\downarrow}$, and $u_1^{\uparrow O}/u_1^{\downarrow O}$, respectively. In each case, the correlation function (u_*) is represented by piecewise continuous cubic B-splines. Each correlation function has eight parameters which are just the values of u on a uniformly spaced grid up to `rcut`. Initially the parameters (`coefficients`) are set to zero:

```
<correlation speciesA="u" speciesB="u" size="8" rcut="10.0">
  <coefficients id="uu" type="Array">
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  </coefficients>
</correlation>
```

Finally, we need to assemble particle, wavefunction, and pseudopotential information into the main QMCPACK input file (`O.q0.opt.in.xml`) and specify inputs for the Jastrow optimization process. Open `O.q0.opt.in.xml` and write in the location of the particle, wavefunction, and pseudopotential files ("`<!-- ... -->`" are comments):

```
...
<!-- include simulationcell and particle information from pw2qmcpqack -->
<include href="O.q0.ptcl.xml"/>
...
<!-- include wavefunction information from pw2qmcpqack -->
<include href="O.q0.wfs.xml"/>
...
<!-- O pseudopotential read from "O.BFD.xml" -->
<pseudo elementType="O" href="O.BFD.xml"/>
...
```

The relevant portion of the input describing the linear optimization process is

```
<loop max="MAX">
  <qmc method="linear" move="pbyp" checkpoint="-1">
    <cost name="energy"              >  ECOST    </cost>
    <cost name="unreweightedvariance">  UVCOST   </cost>
    <cost name="reweightedvariance"  >  RVCOST   </cost>
    <parameter name="timestep"       >  TS       </parameter>
    <parameter name="samples"        >  SAMPLES  </parameter>
    <parameter name="warmupSteps"    >  50       </parameter>
    <parameter name="blocks"         >  200      </parameter>
    <parameter name="subSteps"       >  1        </parameter>
    <parameter name="nonlocalpp"     >  yes      </parameter>
    <parameter name="useBuffer"      >  yes      </parameter>
    ...
  </qmc>
</loop>
```

An explanation of each input variable can be found below. The remaining variables control specialized internal details of the linear optimization algorithm. The meaning of these inputs is beyond the scope of this lab and reasonable results are often obtained keeping these values fixed.

energy Fraction of trial energy in the cost function.

unreweightedvariance Fraction of unreweighted trial variance in the cost function. Neglecting the weights can be more robust.

reweightedvariance Fraction of trial variance (including the full weights) in the cost function.

timestep Timestep of the VMC random walk, determines spatial distance moved by each electron during MC steps. Should be chosen such that the acceptance ratio of MC moves is around 50% (30-70% is often acceptable). Reasonable values are often between 0.2 and 0.6 Ha^{-1}.

samples Total number of MC samples collected for optimization, determines statistical error bar of cost function. Often efficient to start with a modest number of samples (50k) and then increase as needed. More samples may be required if the wavefunction contains a large number of variational parameters. MUST be be a multiple of the number of threads/cores .

warmupSteps Number of MC steps discarded as a warmup or equilibration period of the random walk. If this is too small, it will bias the optimization procedure.

blocks Number of average energy values written to output files. Should be greater than 200 for meaningful statistical analysis of output data (*e.g.* via `qmca`).

subSteps Number of MC steps in between energy evaluations. Each energy evaluation is expensive so taking a few steps to decorrelate between measurements can be more efficient. Will be less efficient with many substeps.

nonlocalpp,useBuffer If `nonlocalpp="no"`, then nonlocal part of the pseudopotential is not included when computing the cost function. If `useBuffer="yes"`, then temporary data is stored to speed up nonlocal pseudopotential evaluation at the expense of memory consumption.

loop max Number of times to repeat the optimization. Using the resulting wavefunction from the previous optimization in the next one improves the results. Typical choices range between 8 and 16.

The cost function defines the quantity to be minimized during optimization. The three components of the cost function, energy, unreweighted variance, and reweighted variance should sum to one. Dedicating 100% of the cost function to unreweighted variance is often a good choice. Another common choice is to try 90/10 or 80/20 mixtures of reweighted variance and energy. Using 100% energy minimization is desirable for reducing DMC pseudopotential localization errors, but the optimization process is less stable and should only be attempted after performing several cycles of e.g. variance minimization first (the entire `loop` section can be duplicated with a different cost function each time).

Replace `MAX`, `EVCOST`, `UVCOST`, `RVCOST`, `TS`, and `SAMPLES` in the `loop` with appropriate starting values in the `O.q0.opt.in.xml` input file. Perform the optimization run by typing

```
mpirun -np 4 qmcpack O.q0.opt.in.xml >&O.q0.opt.out&
```

The run should only take a few minutes for reasonable values of loop `max` and `samples`.

Log file output will appear in `O.q0.opt.out`. The beginning of each linear optimization will be marked with text similar to

```
=========================================================
  Start QMCFixedSampleLinearOptimize
  File Root O.q0.opt.s011 append = no
=========================================================
```

At the end of each optimization section the change in cost function, new values for the Jastrow parameters, and elapsed wallclock time are reported:

```
 OldCost: 7.0598901869e-01 NewCost: 7.0592576381e-01 Delta Cost:-6.3254886314e-05
...
  <optVariables href="O.q0.opt.s011.opt.xml">
uu_0 6.9392504232e-01 1 1  ON 0
uu_1 4.9690781460e-01 1 1  ON 1
uu_2 4.0934542375e-01 1 1  ON 2
uu_3 3.7875640157e-01 1 1  ON 3
uu_4 3.7308380014e-01 1 1  ON 4
uu_5 3.5419786809e-01 1 1  ON 5
uu_6 4.3139019377e-01 1 1  ON 6
uu_7 1.9344371667e-01 1 1  ON 7
ud_0 3.9219009713e-01 1 1  ON 8
ud_1 1.2352664647e-01 1 1  ON 9
ud_2 4.4048945133e-02 1 1  ON 10
ud_3 2.1415676741e-02 1 1  ON 11
ud_4 1.5201803731e-02 1 1  ON 12
ud_5 2.3708169445e-02 1 1  ON 13
ud_6 3.4279064930e-02 1 1  ON 14
ud_7 4.3334583596e-02 1 1  ON 15
eO_0 -7.8490123937e-01 1 1  ON 16
eO_1 -6.6726618338e-01 1 1  ON 17
eO_2 -4.8753453838e-01 1 1  ON 18
eO_3 -3.0913993774e-01 1 1  ON 19
eO_4 -1.7901872177e-01 1 1  ON 20
eO_5 -8.6199000697e-02 1 1  ON 21
eO_6 -4.0601160841e-02 1 1  ON 22
eO_7 -4.1358075061e-03 1 1  ON 23
  </optVariables>
...
  QMC Execution time = 2.8218972974e+01 secs
```

The cost function should decrease during each linear optimization (`Delta cost < 0`). Try "`grep OldCost *opt.out`". You should see something like this:

```
 OldCost: 1.2655186572e+00 NewCost: 7.2443875597e-01 Delta Cost:-5.4107990118e-01
 OldCost: 7.2229830632e-01 NewCost: 6.9833678217e-01 Delta Cost:-2.3961524143e-02
 OldCost: 8.0649629434e-01 NewCost: 8.0551871147e-01 Delta Cost:-9.7758287036e-04
 OldCost: 6.6821241388e-01 NewCost: 6.6797703487e-01 Delta Cost:-2.3537901148e-04
 OldCost: 7.0106275099e-01 NewCost: 7.0078055426e-01 Delta Cost:-2.8219672877e-04
 OldCost: 6.9538522411e-01 NewCost: 6.9419186712e-01 Delta Cost:-1.1933569922e-03
 OldCost: 6.7709626744e-01 NewCost: 6.7501251165e-01 Delta Cost:-2.0837557922e-03
 OldCost: 6.6659923822e-01 NewCost: 6.6651737755e-01 Delta Cost:-8.1860671682e-05
 OldCost: 7.7828995609e-01 NewCost: 7.7735482525e-01 Delta Cost:-9.3513083900e-04
 OldCost: 7.2717974404e-01 NewCost: 7.2715201115e-01 Delta Cost:-2.7732880747e-05
 OldCost: 6.9400639873e-01 NewCost: 6.9257183689e-01 Delta Cost:-1.4345618444e-03
 OldCost: 7.0598901869e-01 NewCost: 7.0592576381e-01 Delta Cost:-6.3254886314e-05
```

Blocked averages of energy data, including the kinetic energy and components of the potential energy, are written to `scalar.dat` files. The first is named "`O.q0.opt.s000.scalar.dat`", with a series number of zero (`s000`). In the end there will be `MAX` of them, one for each series.

When the job has finished, use the `qmca` tool to assess the effectiveness of the optimization process. To look at just the total energy and the variance, type "`qmca -q ev O.q0.opt*scalar*`". This will print the energy, variance, and the variance/energy ratio in Hartree units:

```
                          LocalEnergy              Variance           ratio
O.q0.opt  series 0  -15.739585 +/- 0.007656   0.887412 +/- 0.010728   0.0564
O.q0.opt  series 1  -15.848347 +/- 0.004089   0.318490 +/- 0.006404   0.0201
O.q0.opt  series 2  -15.867494 +/- 0.004831   0.292309 +/- 0.007786   0.0184
O.q0.opt  series 3  -15.871508 +/- 0.003025   0.275364 +/- 0.006045   0.0173
O.q0.opt  series 4  -15.865512 +/- 0.002997   0.278056 +/- 0.006523   0.0175
O.q0.opt  series 5  -15.864967 +/- 0.002733   0.278065 +/- 0.004413   0.0175
O.q0.opt  series 6  -15.869644 +/- 0.002949   0.273497 +/- 0.006141   0.0172
O.q0.opt  series 7  -15.868397 +/- 0.003838   0.285451 +/- 0.007570   0.0180
...
```

Plots of the data can also be obtained with the "`-p`" option ("`qmca -p -q ev O.q0.opt*scalar*`").

Identify which optimization series is the "best" according to your cost function. It is likely that multiple series are similar in quality. Note the `opt.xml` file corresponding to this series. This file contains the final value of the optimized Jastrow parameters to be used in the DMC calculations of the next section of the lab.

Questions and Exercises

1. What is the acceptance ratio of your optimization runs? (use "`qmca -q ar O.q0.opt*scalar*`") Do you expect the Monte Carlo sampling to be efficient?

2. How do you know when the optimization process has converged?

3. (optional) Optimization is sometimes sensitive to initial guesses of the parameters. If you have time, try varying the initial parameters, including the cutoff radius (`rcut`) of the Jastrow factors (remember to change `id` in the `<project/>` element). Do you arrive at a similar set of final Jastrow parameters? What is the lowest variance you are able to achieve?

18.7 DMC timestep extrapolation I: neutral O atom

The diffusion Monte Carlo (DMC) algorithm contains two biases in addition to the fixed node and pseudopotential approximations that are important to control: timestep and population control bias. In this section we will focus on estimating and removing timestep bias from DMC calculations. The essential fact to remember is that the bias vanishes as the timestep goes to zero while the needed computer time increases inversely with the timestep.

In the same directory you used to perform wavefunction optimization (`oxygen_atom`) you will find a sample DMC input file for the neutral oxygen atom named `O.q0.dmc.in.xml`. Open this file in a text editor and note the differences from the optimization case. Wavefunction information is no longer included from `pw2qmcpack`, but instead should come from the optimization run:

```
<!-- OPT_XML is from optimization, e.g. O.q0.opt.s008.opt.xml -->
<include href="OPT_XML"/>
```

Replace "`OPT_XML`" with the `opt.xml` file corresponding to the best Jastrow parameters you found in the last section (this is a file name similar to `O.q0.opt.s008.opt.xml`).

The QMC calculation section at the bottom is also different. The linear optimization blocks have been replaced with XML describing a VMC run followed by DMC. The input keywords are described below.

timestep Timestep of the VMC/DMC random walk. In VMC choose a timestep corresponding to an acceptance ratio of about 50%. In DMC the acceptance ratio is often above 99%.

warmupSteps Number of MC steps discarded as a warmup or equilibration period of the random walk.

steps Number of MC steps per block. Physical quantities, such as the total energy, are averaged over walkers and steps.

blocks Number of blocks. This is also the number of average energy values written to output files. Should be greater than 200 for meaningful statistical analysis of output data (*e.g.* via `qmca`). The total number of MC steps each walker takes is `blocks`×`steps`.

samples VMC only. This is the number of walkers used in subsequent DMC runs. Each DMC walker is initialized with electron positions sampled from the VMC random walk.

nonlocalmoves DMC only. If yes/no, use the locality approximation/T-moves for non-local pseudopotentials. T-moves generally improve the stability of the algorithm and restore the variational principle for small systems (T-moves version 1).

The purpose of the VMC run is to provide initial electron positions for each DMC walker. Setting *textttwalkers*] = 1 in the VMC block ensures there will be only one VMC walker per execution thread. There will be a total of 4 VMC walkers in this case (see **`0.q0.dmc.qsub.in`**). We want the electron positions used to initialize the DMC walkers to be decorrelated from one another. A VMC walker will often decorrelate from its current position after propagating for a few Ha^{-1} in imaginary time (in general this is system dependent). This leads to a rough rule of thumb for choosing `blocks` and `steps` for the VMC run (`VWALKERS` = 4 here):

$$\texttt{VBLOCKS} \times \texttt{VSTEPS} \geq \frac{\texttt{DWALKERS}}{\texttt{VWALKERS}} \frac{5\ \mathrm{Ha}^{-1}}{\texttt{VTIMESTEP}} \tag{18.3}$$

Fill in the VMC XML block with appropriate values for these parameters. There should be more than one DMC walker per thread and enough walkers in total to avoid population control bias. The general rule of thumb is to have more than ~ 2000 walkers, although the dependence of the total energy on population size should be explicitly checked from time to time.

To study timestep bias, we will perform a sequence of DMC runs over a range of timesteps (0.1 Ha^{-1} is too large and timesteps below 0.002 Ha^{-1} are probably too small). A common approach is to select a fairly large timestep to begin with and then decrease the timestep by a factor of two in each subsequent DMC run. The total amount of imaginary time the walker population propagates should be the same for each run. A simple way to accomplish this is to choose input parameters in the following way

$$\begin{aligned} \texttt{timestep}_n &= \texttt{timestep}_{n-1}/2 \\ \texttt{warmupSteps}_n &= \texttt{warmupSteps}_{n-1} \times 2 \\ \texttt{blocks}_n &= \texttt{blocks}_{n-1} \\ \texttt{steps}_n &= \texttt{steps}_{n-1} \times 2 \end{aligned} \tag{18.4}$$

Each DMC run will require about twice as much computer time as the one preceding it. Note that the number of blocks is kept fixed for uniform statistical analysis. `blocks` $\times$ `steps` $\times$ `timestep` $\sim$ 60 Ha^{-1} is sufficient for this system.

Choose an initial DMC timestep and create a sequence of N timesteps according to 18.4. Make N copies of the DMC XML block in the input file

```
<qmc method="dmc" move="pbyp">
   <parameter name="warmupSteps"        >    DWARMUP        </parameter>
   <parameter name="blocks"             >    DBLOCKS        </parameter>
   <parameter name="steps"              >    DSTEPS         </parameter>
   <parameter name="timestep"           >    DTIMESTEP      </parameter>
   <parameter name="nonlocalmoves"      >    yes            </parameter>
</qmc>
```

Fill in `DWARMUP`, `DBLOCKS`, `DSTEPS`, and `DTIMESTEP` for each DMC run according to 18.4. Start the DMC timestep extrapolation run by typing:

```
mpirun -np 4 qmcpack O.q0.dmc.in.xml >&O.q0.dmc.out&
```

The run should take only a few minutes to complete.

QMCPACK will create files prefixed with `O.q0.dmc`. The log file is `O.q0.dmc.out`. As before, block averaged data is written to `scalar.dat` files. In addition, DMC runs produce `dmc.dat` files which contain energy data averaged only over the walker population (one line per DMC step). The `dmc.dat` files also provide a record of the walker population at each step.

Use the `PlotTstepConv.pl` to obtain a linear fit to the timestep data (type "`PlotTstepConv.pl O.q0.dmc.in.xml 40`"). You should see a plot similar to fig. 18.1. The tail end of the text output displays the parameters for the linear fit. The "`a`" parameter is the total energy extrapolated to zero timestep in Hartree units.

```
...
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = -15.8925         +/- 0.0007442    (0.004683%)
b               = -0.0457479       +/- 0.0422       (92.24%)
...
```

Questions and Exercises

1. What is the $\tau \to 0$ extrapolated value for the total energy?

2. What is the maximum timestep you should use if you want to calculate the total energy to an accuracy of 0.05 eV? For convenience, 1 Ha = 27.2113846 eV.

3. What is the acceptance ratio for this (bias$<$ 0.05 eV) run? Does it follow the rule of thumb for sensible DMC (acceptance ratio $>$ 99%) ?

4. Check the fluctuations in the walker population (`qmca -t -q nw O.q0.dmc*dmc.dat --noac`). Does the population seem to be stable?

5. (Optional) Study population control bias for the oxygen atom. Select a few population sizes . Copy `O.q0.dmc.in.xml` to a new file and remove all but one DMC run (select a single

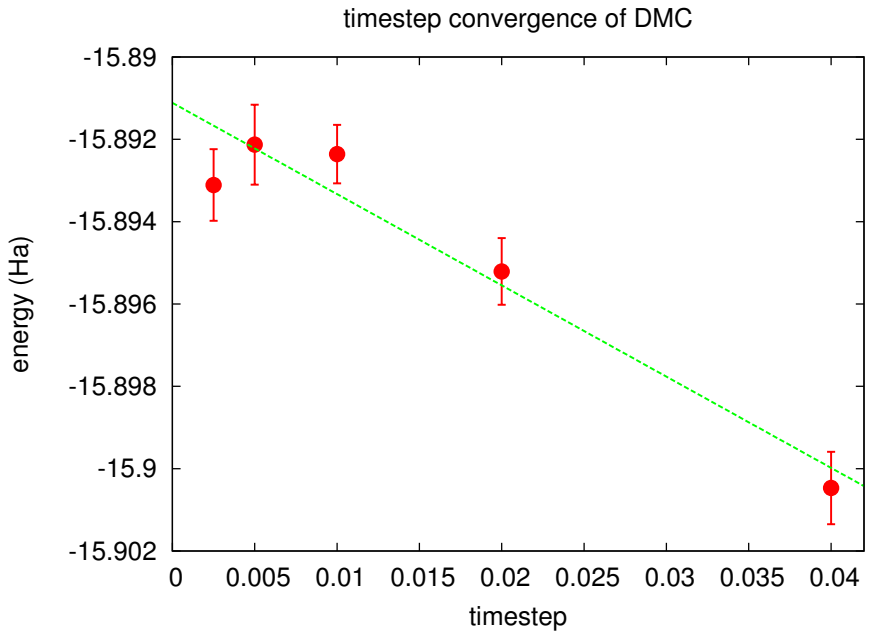


Figure 18.1: Linear fit to DMC timestep data from `PlotTstepConv.pl`.

timestep). Make one copy of the new file for each population, set "`samples`", and choose a unique `id` in `<project/>`. Use `qmca` to study the dependence of the DMC total energy on the walker population. How large is the bias compared to timestep error? What bias is incurred by following the "rule of thumb" of a couple thousand walkers? Will population control bias generally be an issue for production runs on modern parallel machines?

18.8 DMC timestep extrapolation II: O atom ionization potential

In this section, we will repeat the calculations of the prior two sections (optimization, timestep extrapolation) for the +1 charge state of the oxygen atom. Comparing the resulting 1st ionization potential (IP) with experimental data will complete our first test of the BFD oxygen pseudopotential. In actual practice, higher IP's could also be tested prior to performing production runs.

Obtaining the timestep extrapolated DMC total energy for ionized oxygen should take much less (human) time than for the neutral case. For convenience, the necessary steps are briefly summarized below.

1. Obtain DFT orbitals with Quantum ESPRESSO
   (a) Copy the DFT input (`O.q0.dft.in`) to `O.q1.dft.in`
   (b) Edit `O.q1.dft.in` to match the +1 charge state of the oxygen atom

```
...
prefix            = 'O.q1'
...
tot_charge        = 1
tot_magnetization = 3
...
```

(c) Perform the DFT run: `mpirun -np 4 pw.x -input O.q1.dft.in >&O.q1.dft.out&`

2. Convert the orbitals to ESHDF format

   (a) Copy the pw2qmcpack input (`O.q0.p2q.in`) to `O.q1.p2q.in`

   (b) Edit `O.q1.p2q.in` to match the file prefix used in DFT

```
...
prefix = 'O.q1'
...
```

   (c) Perform the orbital conversion run: `mpirun -np 1 pw2qmcpack.x<O.q1.p2q.in>&O.q1.p2q.out&`

3. Optimize the Jastrow factor with QMCPACK

   (a) Copy the optimization input (`O.q0.opt.in.xml`) to `O.q1.opt.in.xml`

   (b) Edit `O.q1.opt.in.xml` to match the file prefix used in DFT

```
...
<project id="O.q1.opt" series="0">
...
<include href="O.q1.ptcl.xml"/>
...
<include href="O.q1.wfs.xml"/>
...
```

   (c) Edit the particle XML file (O.q1.ptcl.xml) to have open boundary conditions

```
<parameter name="bconds">
  n n n
</parameter>
```

   (d) Add cutoffs to the Jastrow factors in the wavefunction XML file (O.q1.wfs.xml)

```
...
<correlation speciesA="u" speciesB="u" size="8" rcut="10.0">
...
<correlation speciesA="u" speciesB="d" size="8" rcut="10.0">
...
<correlation elementType="O" size="8" rcut="5.0">
...
```

   (e) Perform the Jastrow optimization run: `mpirun -np 4 qmcpack O.q1.opt.in.xml >&O.q1.opt.out&`

   (f) Identify the optimal set of parameters with `qmca` (`[your opt.xml]`).

4. DMC timestep study with QMCPACK

   (a) Copy the DMC input (`O.q0.dmc.in.xml`) to `O.q1.dmc.in.xml`

   (b) Edit `O.q1.dmc.in.xml` to use the DFT prefix and the optimal Jastrow

```
...
<project id="O.q1.dmc" series="0">
...
<include href="O.q1.ptcl.xml"/>
...
<include href="[your opt.xml]"/>
...
```

(c) Perform the DMC run: `mpirun -np 4 qmcpack O.q1.dmc.in.xml >&O.q1.dmc.out&`

(d) Obtain the DMC total energy extrapolated to zero timestep with `PlotTstepConv.pl`.

The process listed above, which excludes additional steps for orbital generation and conversion, can become tedious to perform by hand in production settings where many calculations are often required. For this reason automation tools are introduced for calculations involving the oxygen dimer in section 18.10 of the lab.

Questions and Exercises

1. What is the $\tau \to 0$ extrapolated DMC value for the 1st ionization potential of oxygen?
2. How does the extrapolated value compare to the experimental IP? Go to http://physics.nist.gov/PhysRefData/ASD/ionEnergy.html and enter "`O I`" in the box labeled "`Spectra`" and click on the "`Retrieve Data`" button.
3. What can we conclude about the accuracy of the pseudopotential? What factors complicate this assessment?
4. Explore the sensitivity of the IP to the choice of timestep. Type "`./ip_conv.py`" to view three timestep extrapolation plots: two for the $q = 0, 1$ total energies and one for the IP. Is the IP more, less, or similarly sensitive to timestep than the total energy?
5. What is the maximum timestep you should use if you want to calculate the ionization potential to an accuracy of 0.05 eV? What factor of cpu time is saved by assessing timestep convergence on the IP (a total energy difference) vs. a single total energy?
6. Are the acceptance ratio and population fluctuations reasonable for the $q = 1$ calculations?

18.9 DMC workflow automation with Nexus

Production QMC projects are often composed of many similar workflows. The simplest of these is a single DMC calculation involving four different compute jobs:

1. Orbital generation via Quantum ESPRESSO or GAMESS.
2. Conversion of orbital data via `pw2qmcpack.x` or `convert4qmc`.
3. Optimization of Jastrow factors via QMCPACK.
4. DMC calculation via QMCPACK.

Simulation workflows quickly become more complex with increasing costs in terms of human time for the researcher. Automation tools can decrease both human time and error if used well.

The set of automation tools we will be using is known as Nexus [16], which is distributed with QMCPACK. Nexus is capable of generating input files, submitting and monitoring compute jobs, passing data between simulations (such as relaxed structures, orbital files, optimized Jastrow parameters, etc.), and data analysis. The user interface to Nexus is through a set of functions defined in the Python programming language. User scripts that execute simple workflows resemble input files and do not require programming experience. More complex workflows require only basic programming constructs (*e.g.* for loops and if statements). Nexus input files/scripts should be easier to navigate than QMCPACK input files and more efficient than submitting all the jobs by hand.

Nexus is driven by simple user-defined scripts that resemble keyword-driven input files. An example Nexus input file that performs a single VMC calculation (with pre-generated orbitals) is shown below. Take a moment to read it over and especially note the comments (prefixed with "#") explaining most of the contents. If the input syntax is unclear you may want to consult portions of appendix 18.12, which gives a condensed summary of Python constructs. An additional example and details about the inner workings of Nexus can be found in the reference publication [16].

```
#! /usr/bin/env python

# import Nexus functions
from nexus import settings,job,get_machine,run_project
from nexus import generate_physical_system
from nexus import generate_qmcpack,vmc

settings(                               # Nexus settings
    pseudo_dir    = './pseudopotentials', # location of PP files
    runs          = '',                   # root directory for simulations
    results       = '',                   # root directory for simulation results
    status_only   = 0,                    # show simulation status, then exit
    generate_only = 0,                    # generate input files, then exit
    sleep         = 3,                    # seconds between checks on sim. progress
    machine       = 'ws4',                # workstation with 4 cores
    )

qmcjob = job(                           # specify job parameters
    cores   = 4,                          # use 4 MPI tasks
    threads = 1,                          # 1 OpenMP thread per node
    app     = 'qmcpack'                   # use QMCPACK executable (assumed in PATH)
    )

qmc_calcs = [                           # list QMC calculation methods
    vmc(                                  #   VMC
        walkers     =   1,                #     1 walker
        warmupsteps =  50,                #    50 MC steps for warmup
        blocks      = 200,                #   200 blocks
        steps       =  10,                #    10 steps per block
        timestep    =  .4                 #   0.4 1/Ha timestep
        )]

dimer = generate_physical_system(       # make a dimer system
    type       = 'dimer',                 # system type is dimer
    dimer      = ('O','O'),               # dimer is two oxygen atoms
    separation = 1.2074,                  # separated by 1.2074 Angstrom
```

```
    Lbox       = 15.0,                    # simulation box is 15 Angstrom
    units      = 'A',                     # Angstrom is dist. unit
    net_spin   = 2,                       # nup-ndown is 2
    O          = 6                        # pseudo-oxygen has 6 valence el.
    )

qmc = generate_qmcpack(                 # make a qmcpack simulation
    identifier   = 'example',             # prefix files with 'example'
    path         = 'scale_1.0',           # run in ./scale_1.0 directory
    system       = dimer,                 # run the dimer system
    job          = qmcjob,                # set job parameters
    input_type   = 'basic',               # basic qmcpack inputs given below
    pseudos      = ['O.BFD.xml'],         # list of PP's to use
    orbitals_h5  = 'O2.pwscf.h5',         # file with orbitals from DFT
    bconds       = 'nnn',                 # open boundary conditions
    jastrows     = [],                    # no jastrow factors
    calculations = qmc_calcs              # QMC calculations to perform
    )

run_project(qmc)                        # write input file and submit job
```

18.10 Automated binding curve of the oxygen dimer

In this section we will use Nexus to calculate the DMC total energy of the oxygen dimer over a series of bond lengths. The equilibrium bond length and binding energy of the dimer will be determined by performing a polynomial fit to the data (Morse potential fits should be preferred in production tests). Comparing these values with corresponding experimental data provides a second test of the BFD pseudopotential for oxygen.

Enter the `oxygen_dimer` directory. Copy your BFD pseudopotential from the atom runs into `oxygen_dimer/pseudopotentials` (be sure to move both files: `.upf` and `.xml`). Open `O_dimer.py` with a text editor. The overall format is similar to the example file shown in the last section. The main difference is that a full workflow of runs (DFT orbital generation, orbital conversion, optimization and DMC) are being performed rather than a single VMC run.

As in the example in the last section, the oxygen dimer is generated with the `generate_physical_system` function:

```
dimer = generate_physical_system(
    type       = 'dimer',
    dimer      = ('O','O'),
    separation = 1.2074*scale,
    Lbox       = 10.0,
    units      = 'A',
    net_spin   = 2,
    O          = 6
    )
```

Similar syntax can be used to generate crystal structures or to specify systems with arbitrary atomic configurations and simulation cells. Notice that a "`scale`" variable has been introduced to stretch or compress the dimer.

Next, objects representing a Quantum ESPRESSO (PWSCF) run and subsequent orbital conversion step are constructed with respective `generate_*` functions:

```
dft = generate_pwscf(
    identifier   = 'dft',
```

```
    ...
    input_dft    = 'lda',
    ...
    )
sims.append(dft)

# describe orbital conversion run
p2q = generate_pw2qmcpack(
    identifier   = 'p2q',
    ...
    dependencies = (dft,'orbitals'),
    )
sims.append(p2q)
```

Note the `dependencies` keyword. This keyword is used to construct workflows out of otherwise separate runs. In this case, the dependency indicates that the orbital conversion run must wait for the DFT to finish prior to starting.

Objects representing QMCPACK simulations are then constructed with the `generate_qmcpack` function:

```
opt = generate_qmcpack(
    identifier   = 'opt',
    ...
    jastrows     = [('J1','bspline',8,5.0),
                    ('J2','bspline',8,10.0)],
    calculations = [
        loop(max=12,
             qmc=linear(
                energy               = 0.0,
                unreweightedvariance = 1.0,
                reweightedvariance   = 0.0,
                timestep             = 0.3,
                samples              = 61440,
                warmupsteps          = 50,
                blocks               = 200,
                substeps             = 1,
                nonlocalpp           = True,
                usebuffer            = True,
                walkers              = 1,
                minwalkers           = 0.5,
                maxweight            = 1e9,
                usedrift             = False,
                minmethod            = 'quartic',
                beta                 = 0.025,
                exp0                 = -16,
                bigchange            = 15.0,
                alloweddifference    = 1e-4,
                stepsize             = 0.2,
                stabilizerscale      = 1.0,
                nstabilizers         = 3,
                )
             )
        ],
    dependencies = (p2q,'orbitals'),
    )
sims.append(opt)

qmc = generate_qmcpack(
```

```
    identifier   = 'qmc',
    ...
    jastrows     = [],
    calculations = [
        vmc(
            walkers     =   1,
            warmupsteps =  30,
            blocks      =  20,
            steps       =  10,
            substeps    =   2,
            timestep    =  .4,
            samples     = 2048
            ),
        dmc(
            warmupsteps   = 100,
            blocks        = 400,
            steps         =  32,
            timestep      = 0.01,
            nonlocalmoves = True,
            )
        ],
    dependencies = [(p2q,'orbitals'),(opt,'jastrow')],
    )
sims.append(qmc)
```

Shared details such as the run directory, job, pseudopotentials, and orbital file have been omitted (`...`). The "`opt`" run will optimize a 1-body B-spline Jastrow with 8 knots having a cutoff of 5.0 Bohr and a B-spline Jastrow (for up-up and up-down correlations) with 8 knots and cutoffs of 10.0 Bohr. The Jastrow list for the DMC run is empty and the usage of `dependencies` above indicates that the DMC run depends on the optimization run for the Jastrow factor. Nexus will submit the "`opt`" run first and upon completion it will scan the output, select the optimal set of parameters, pass the Jastrow information to the "`qmc`" run and then submit the DMC job. Independent job workflows are submitted in parallel when permitted . No input files are written or job submissions made until the "`run_project`" function is reached:

```
run_project(sims)
```

All of the simulations objects have been collected into a list (`sims`) for submission.

As written, `O_dimer.py` will only perform calculations at the equilibrium separation distance of 1.2074 Angstrom, since the list of scaling factors (representing stretching or compressing the dimer) only contains one value (`scales = [1.00]`). Modify the file now to perform DMC calculations across a range of separation distances with each DMC run using the Jastrow factor optimized at the equilibrium separation distance. Specifically, you will want to change the list of scaling factors to include both compression (`scale<1.0`) and stretch (`scale>1.0`):

```
scales = [1.00,0.90,0.95,1.05,1.10]
```

Note that "`1.00`" is left in front because we are going to optimize the Jastrow factor first at the equilibrium separation and reuse this Jastrow factor for all other separation distances. This procedure is used because it can reduce variations in localization errors (due to pseudopotentials in DMC) along the binding curve.

Change the "`status_only`" parameter in the "`settings`" function to `1` and type "`./O_dimer.py`" at the command line. This will print the status of all simulations:

```
Project starting
```

```
  checking for file collisions
  loading cascade images
    cascade 0 checking in
    cascade 10 checking in
    cascade 4 checking in
    cascade 13 checking in
    cascade 7 checking in
  checking cascade dependencies
    all simulation dependencies satisfied

  cascade status
    setup, sent_files, submitted, finished, got_output, analyzed
    000000  dft     ./scale_1.0
    000000  p2q     ./scale_1.0
    000000  opt     ./scale_1.0
    000000  qmc     ./scale_1.0
    000000  dft     ./scale_0.9
    000000  p2q     ./scale_0.9
    000000  qmc     ./scale_0.9
    000000  dft     ./scale_0.95
    000000  p2q     ./scale_0.95
    000000  qmc     ./scale_0.95
    000000  dft     ./scale_1.05
    000000  p2q     ./scale_1.05
    000000  qmc     ./scale_1.05
    000000  dft     ./scale_1.1
    000000  p2q     ./scale_1.1
    000000  qmc     ./scale_1.1
    setup, sent_files, submitted, finished, got_output, analyzed
```

In this case, five simulation "cascades" (workflows) have been identified, each one starting and ending with "`dft`" and "`qmc`" runs, respectively. The six status flags (`setup`, `sent_files`, `submitted`, `finished`, `got_output`, `analyzed`) each show `0`, indicating that no work has been done yet.

Now change "`status_only`" back to `0`, set "`generate_only`" to `1`, and run `0_dimer.py` again. This will perform a dry-run of all simulations. The dry-run should finish in about 20 seconds:

```
Project starting
  checking for file collisions
  loading cascade images
    cascade 0 checking in
    cascade 10 checking in
    cascade 4 checking in
    cascade 13 checking in
    cascade 7 checking in
  checking cascade dependencies
    all simulation dependencies satisfied

  starting runs:
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  poll 0  memory 91.03 MB
    Entering ./scale_1.0 0
      writing input files  0 dft
    Entering ./scale_1.0 0
      sending required files  0 dft
      submitting job  0 dft
  ...
  poll 1  memory 91.10 MB
  ...
```

```
  Entering ./scale_1.0 0
    Would have executed:
      export OMP_NUM_THREADS=1
      mpirun -np 4 pw.x -input dft.in

poll 2  memory 91.10 MB
  Entering ./scale_1.0 0
    copying results  0 dft
  Entering ./scale_1.0 0
    analyzing  0 dft
...
poll 3  memory 91.10 MB
  Entering ./scale_1.0 1
    writing input files  1 p2q
  Entering ./scale_1.0 1
    sending required files  1 p2q
    submitting job  1 p2q
...
  Entering ./scale_1.0 1
    Would have executed:
      export OMP_NUM_THREADS=1
      mpirun -np 1 pw2qmcpack.x<p2q.in

poll 4  memory 91.10 MB
  Entering ./scale_1.0 1
    copying results  1 p2q
  Entering ./scale_1.0 1
    analyzing  1 p2q
...
poll 5  memory 91.10 MB
  Entering ./scale_1.0 2
    writing input files  2 opt
  Entering ./scale_1.0 2
    sending required files  2 opt
    submitting job  2 opt
...
  Entering ./scale_1.0 2
    Would have executed:
      export OMP_NUM_THREADS=1
      mpirun -np 4 qmcpack opt.in.xml

poll 6  memory 91.16 MB
  Entering ./scale_1.0 2
    copying results  2 opt
  Entering ./scale_1.0 2
    analyzing  2 opt
...
poll 7  memory 93.00 MB
  Entering ./scale_1.0 3
    writing input files  3 qmc
  Entering ./scale_1.0 3
    sending required files  3 qmc
    submitting job  3 qmc
...
  Entering ./scale_1.0 3
    Would have executed:
      export OMP_NUM_THREADS=1
      mpirun -np 4 qmcpack qmc.in.xml
...
```

```
  poll 17  memory 93.06 MB
Project finished
```

Nexus polls the simulation status every 3 seconds and sleeps in between. The "scale_*" directories should now contain several files:

```
scale_1.0
   dft.in
   O.BFD.upf
   O.BFD.xml
   opt.in.xml
   p2q.in
   pwscf_output
   qmc.in.xml
   sim_dft/
       analyzer.p
       input.p
       sim.p
   sim_opt/
       analyzer.p
       input.p
       sim.p
   sim_p2q/
       analyzer.p
       input.p
       sim.p
   sim_qmc/
       analyzer.p
       input.p
       sim.p
```

Take a minute to inspect the generated input (`dft.in`, `p2q.in`, `opt.in.xml`, `qmc.in.xml`) files. The pseudopotential files (`O.BFD.upf` and `O.BFD.xml`) have been copied into each local directory. Four additional directories have been created: `sim_dft`, `sim_p2q`, `sim_opt` and `sim_qmc`. The `sim.p` files in each directory contain the current status of each simulation. If you run `O_dimer.py` again, it should not attempt to rerun any of the simulations:

```
Project starting
  checking for file collisions
  loading cascade images
    cascade 0 checking in
    cascade 10 checking in
    cascade 4 checking in
    cascade 13 checking in
    cascade 7 checking in
  checking cascade dependencies
    all simulation dependencies satisfied

  starting runs:
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  poll 0  memory 64.25 MB
Project finished
```

This way one can continue to add to the `O_dimer.py` file (*e.g.* adding more separation distances) without worrying about duplicate job submissions.

Let's actually submit the jobs in the dimer workflow now. Reset the state of the simulations by removing the `sim.p` files ("`rm ./scale*/sim*/sim.p`"), set "`generate_only`" to `0`, and rerun `O_dimer.py`. It should take about 20 minutes for all the jobs to complete. You may wish to open

another terminal to monitor the progress of the individual jobs while the current terminal runs `O_dimer.py` in the foreground. You can begin the first exercise below once the optimization job completes.

Questions and Exercises

1. Evaluate the quality of the optimization at `scale=1.0` using the `qmca` tool. Did the optimization succeed? How does the variance compare with the neutral oxygen atom? Is the wavefunction of similar quality to the atomic case?

2. Evaluate the traces of the local energy and the DMC walker population for each separation distance with the `qmca` tool. Are there any anomalies in the runs? Is the acceptance ratio reasonable? Is the wavefunction of similar quality across all separation distances?

3. Use the `dimer_fit.py` tool located in `oxygen_dimer` to fit the oxygen dimer binding curve. To get the binding energy of the dimer, we will need the DMC energy of the atom. Before performing the fit, answer: What DMC timestep should be used for the oxygen atom results? The tool accepts three arguments ("`./dimer_fit.py P N E Eerr`"), `P` is the prefix of the DMC input files (should be "`qmc`" at this point), `N` is the order of the fit (use 2 to start), `E` and `Eerr` are your DMC total energy and error bar, respectively for the oxygen atom (in eV). A plot of the dimer data will be displayed and text output will show the DMC equilibrium bond length and binding energy as well as experimental values. How accurately does your fit to the DMC data reproduce the experimental values? What factors affect the accuracy of your results?

4. Refit your data with a fourth-order polynomial. How do your predictions change with a fourth-order fit? Is a fourth-order fit appropriate for the available data?

5. Add new "`scale`" values to the list in `O_dimer.py` that interpolate between the original set (e.g. expand to `[1.00,0.90,0.925,0.95,0.975,1.025,1.05,1.075,1.10]`). Perform the DMC calculations and redo the fits. How accurately does your fit to the DMC data reproduce the experimental values? Should this pseudopotential be used in production calculations?

6. (Optional) Perform optimization runs at the extremal separation distances corresponding to `scale=[0.90,1.10]`. Are the individually optimized wavefunctions of significantly better quality than the one imported from `scale=1.00`? Why? What form of Jastrow factor might give an even better improvement?

18.11 (Optional) Running your system with QMCPACK

This section covers a fairly simple route to get started on QMC calculations of an arbitrary system of interest using the Nexus workflow management system to setup input files and optionally perform the runs. The example provided in this section uses QM Espresso (PWSCF) to generate the orbitals forming the Slater determinant part of the trial wavefunction. PWSCF is a natural choice for solid

state systems and it can be used for surface/slab and molecular systems as well, albeit at the price of describing additional vacuum space with plane waves.

To start out with, you will need pseudopotentials (PP's) for each element in your system in both the UPF (PWSCF) and FSATOM/XML (QMCPACK) formats. A good place to start is the Burkatzki-Filippi-Dolg (BFD) pseudopotential database (http://www.burkatzki.com/pseudos/index.2.html), which we have already used in our study of the oxygen atom. The database does not contain PP's for the 4th and 5th row transition metals or any of the lanthanides or actinides. If you need a PP that is not in the BFD database, you may need to generate and test one manually (*e.g.* with OPIUM, http://opium.sourceforge.net/). Otherwise, use `ppconvert` as outlined in section 18.4 to obtain PP's in the formats used by PWSCF and QMCPACK. Enter the `your_system` lab directory and place the converted PP's in `your_system/pseudopotentials`.

Before performing production calculations (more than just the initial setup in this section) be sure to converge the plane wave energy cutoff in PWSCF as these PP's can be rather hard, sometimes requiring cutoffs in excess of 300 Ry. Depending on the system under study, the amount of memory required to represent the orbitals (QMCPACK uses 3D B-splines) becomes prohibitive and one may be forced to search for softer PP's.

Beyond pseudopotentials, all that is required to get started are the atomic positions and the dimensions/shape of the simulation cell. The Nexus file `example.py` illustrates how to setup PWSCF and QMCPACK input files by providing minimal information regarding the physical system (an 8-atom cubic cell of diamond in the example). Most of the contents should be familiar from your experience with the automated calculations of the oxygen dimer binding curve in section 18.10 (if you've skipped ahead you may want to skim that section for relevant information). The most important change is the expanded description of the physical system:

```
# details of your physical system (diamond conventional cell below)
my_project_name = 'diamond_vmc'   # directory to perform runs
my_dft_pps      = ['C.BFD.upf']   # pwscf pseudopotentials
my_qmc_pps      = ['C.BFD.xml']   # qmcpack pseudopotentials

#  generate your system
#    units      :  'A'/'B' for Angstrom/Bohr
#    axes       :  simulation cell axes in cartesian coordinates (a1,a2,a3)
#    elem       :  list of atoms in the system
#    pos        :  corresponding atomic positions in cartesian coordinates
#    kgrid      :  Monkhorst-Pack grid
#    kshift     :  Monkhorst-Pack shift (between 0 and 0.5)
#    net_charge :  system charge in units of e
#    net_spin   :  # of up spins - # of down spins
#    C = 4      :  (pseudo) carbon has 4 valence electrons
my_system = generate_physical_system(
    units      = 'A',
    axes       = [[ 3.57000000e+00, 0.00000000e+00, 0.00000000e+00],
                  [ 0.00000000e+00, 3.57000000e+00, 0.00000000e+00],
                  [ 0.00000000e+00, 0.00000000e+00, 3.57000000e+00]],
    elem       = ['C','C','C','C','C','C','C','C'],
    pos        = [[ 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
                  [ 8.92500000e-01, 8.92500000e-01, 8.92500000e-01],
                  [ 0.00000000e+00, 1.78500000e+00, 1.78500000e+00],
                  [ 8.92500000e-01, 2.67750000e+00, 2.67750000e+00],
                  [ 1.78500000e+00, 0.00000000e+00, 1.78500000e+00],
                  [ 2.67750000e+00, 8.92500000e-01, 2.67750000e+00],
```

```
                   [ 1.78500000e+00, 1.78500000e+00, 0.00000000e+00],
                   [ 2.67750000e+00, 2.67750000e+00, 8.92500000e-01]],
    kgrid      = (1,1,1),
    kshift     = (0,0,0),
    net_charge = 0,
    net_spin   = 0,
    C          = 4       # one line like this for each atomic species
    )

my_bconds       = 'ppp'  #  ppp/nnn for periodic/open BC's in QMC
                         #  if nnn, center atoms about (a1+a2+a3)/2
```

If you have a system you would like to try with QMC, make a copy of `example.py` and fill in the relevant information about the pseudopotentials, simulation cell axes, and atomic species/positions. Otherwise, you can proceed with `example.py` as it is.

Set "`generate_only`" to `1` and type "`./example.py`" or similar to generate the input files. All files will be written to "`./diamond_vmc`" ("`./[my_project_name]`" if you have changed "`my_project_name`" in the file). The input files for PWSCF, pw2qmcpack, and QMCPACK are `scf.in`, `pw2qmcpack.in`, and `vmc.in.xml`, respectively. Take some time to inspect the generated input files. If you have questions about the file contents, or run into issues with the generation process, feel free to consult with a lab instructor.

If desired, you can submit the runs directly with `example.py`. To do this, first reset the Nexus simulation record by typing "`rm ./diamond_vmc/sim*/sim.p`" or similar and set "`generate_only`" back to `0`. Next rerun `example.py` (you may want to redirect the text output).

Alternatively the runs can be submitted by hand:

```
mpirun -np 4 pw.x<scf.in>&scf.out&

(wait until JOB DONE appears in scf.out)

mpirun -np 1 pw2qmcpack.x<p2q.in>&p2q.out&
```

Once the conversion process has finished the orbitals should be located in the file `diamond_vmc/pwscf_output/pwscf.pwscf.h5`. Open `diamond_vmc/vmc.in.xml` and replace "`MISSING.h5`" with "`./pwscf_output/pwscf.pwscf.h5`". Next submit the VMC run:

```
mpirun -np 4 qmcpack vmc.in.xml>&vmc.out&
```

Note: If your system is large, the above process may not complete within the time frame of this lab. Working with a stripped down (but relevant) example is a good idea for exploratory runs.

Once the runs have finished, you may want to begin exploring Jastrow optimization and DMC for your system. Example calculations are provided at the end of `example.py` in the commented out text.

18.12 Appendix A: Basic Python constructs

Basic Python data types (`int`, `float`, `str`, `tuple`, `list`, `array`, `dict`, `obj`) and programming constructs (**`if`** statements, **`for`** loops, functions w/ keyword arguments) are briefly overviewed below. All examples can be executed interactively in Python. To do this, type "`python`" at the command line and paste any of the shaded text below at the "`>>>`" prompt. For more information about effective use of Python, consult the detailed online documentation: https://docs.python.org/2/.

Intrinsic types: `int, float, str`

```
#this is a comment
i=5                     # integer
f=3.6                   # float
s='quantum/monte/carlo' # string
n=None                  # represents "nothing"

f+=1.4                  # add-assign (-,*,/ also): 5.0
2**3                    # raise to a power: 8
str(i)                  # int to string: '5'
s+'/simulations'        # joining strings: 'quantum/monte/carlo/simulations'
'i={0}'.format(i)       # format string: 'i=5'
```

Container types: `tuple, list, array, dict, obj`

```
from numpy import array  # get array from numpy module
from generic import obj  # get obj from Nexus' generic module

t=('A',42,56,123.0)     # tuple

l=['B',3.14,196]        # list

a=array([1,2,3])        # array

d={'a':5,'b':6}         # dict

o=obj(a=5,b=6)          # obj

                        # printing
print t                 #  ('A', 42, 56, 123.0)
print l                 #  ['B', 3.1400000000000001, 196]
print a                 #  [1 2 3]
print d                 #  {'a': 5, 'b': 6}
print o                 #    a               = 5
                        #    b               = 6

len(t),len(l),len(a),len(d),len(o) #number of elements: (4, 3, 3, 2, 2)

t[0],l[0],a[0],d['a'],o.a  #element access: ('A', 'B', 1, 5, 5)

s = array([0,1,2,3,4])  # slices: works for tuple, list, array
s[:]                    #   array([0, 1, 2, 3, 4])
s[2:]                   #   array([2, 3, 4])
s[:2]                   #   array([0, 1])
s[1:4]                  #   array([1, 2, 3])
s[0:5:2]                #   array([0, 2, 4])

                        # list operations
l2 = list(l)            #   make independent copy
l.append(4)             #   add new element: ['B', 3.14, 196, 4]
l+[5,6,7]               #   addition: ['B', 3.14, 196, 4, 5, 6, 7]
3*[0,1]                 #   multiplication:  [0, 1, 0, 1, 0, 1]

b=array([5,6,7])        # array operations
a2 = a.copy()           #   make independent copy
a+b                     #   addition: array([ 6, 8, 10])
```

```
a+3                     #   addition: array([ 4, 5, 6])
a*b                     #   multiplication: array([ 5, 12, 21])
3*a                     #   multiplication: array([3, 6, 9])

                        # dict/obj operations
d2 = d.copy()           #   make independent copy
d['c'] = 7              #   add/assign element
d.keys()                #   get element names: ['a', 'c', 'b']
d.values()              #   get element values: [5, 7, 6]

                        # obj-specific operations
o.c = 7                 #   add/assign element
o.set(c=7,d=8)          #   add/assign multiple elements
```

An important feature of Python to be aware of is that assignment is most often by reference, *i.e.* new values are not always created. This point is illustrated below with an `obj` instance, but it also holds for `list`, `array`, `dict`, and others.

```
>>> o = obj(a=5,b=6)
>>>
>>> p=o
>>>
>>> p.a=7
>>>
>>> print o
  a               = 7
  b               = 6

>>> q=o.copy()
>>>
>>> q.a=9
>>>
>>> print o
  a               = 7
  b               = 6
```

Here `p` is just another name for `o`, while `q` is a fully independent copy of it.

Conditional Statements: `if/elif/else`

```
a = 5
if a is None:
    print 'a is None'
elif a==4:
    print 'a is 4'
elif a<=6 and a>2:
    print 'a is in the range (2,6]'
elif a<-1 or a>26:
    print 'a is not in the range [-1,26]'
elif a!=10:
    print 'a is not 10'
else:
    print 'a is 10'
#end if
```

The "*`#end if`*" is not part of Python syntax, but you will see text like this throughout Nexus for clear encapsulation.

Iteration: `for`

```
from generic import obj

l = [1,2,3]
m = [4,5,6]
s = 0
for i in range(len(l)):  # loop over list indices
    s += l[i] + m[i]
#end for

print s                  # s is 21

s = 0
for v in l:              # loop over list elements
    s += v
#end for

print s                  # s is 6

o = obj(a=5,b=6)
s = 0
for v in o:              # loop over obj elements
    s += v
#end for

print s                  # s is 11

d = {'a':5,'b':4}
for n,v in o.iteritems():# loop over name/value pairs in obj
    d[n] += v
#end for

print d                  # d is {'a': 10, 'b': 10}
```

Functions: `def`, argument syntax

```
def f(a,b,c=5):          # basic function, c has a default value
    print a,b,c
#end def f

f(1,b=2)                 # prints: 1 2 5


def f(*args,**kwargs):   # general function, returns nothing
    print args           #     args: tuple of positional arguments
    print kwargs         #   kwargs: dict of keyword arguments
#end def f

f('s',(1,2),a=3,b='t')   # 2 pos., 2 kw. args, prints:
                         #   ('s', (1, 2))
                         #   {'a': 3, 'b': 't'}

l = [0,1,2]
f(*l,a=6)                # pos. args from list, 1 kw. arg, prints:
                         #   (0, 1, 2)
                         #   {'a': 6}
```

```
o = obj(a=5,b=6)
f(*l,**o)                # pos./kw. args from list/obj, prints:
                         #   (0, 1, 2)
                         #   {'a': 5, 'b': 6}

f(                       # indented kw. args, prints
    blocks   = 200,      #   ()
    steps    = 10,       #   {'steps': 10, 'blocks': 200, 'timestep': 0.01}
    timestep = 0.01
    )

o = obj(                 # obj w/ indented kw. args
    blocks   = 100,
    steps    =  5,
    timestep = 0.02
    )

f(**o)                   # kw. args from obj, prints:
                         #   ()
                         #   {'timestep': 0.02, 'blocks': 100, 'steps': 5}
```

Chapter 19

Lab 3: Advanced Molecular Calculations

19.1 Topics covered in this Lab

This lab covers molecular QMC calculations with wavefunctions of increasing sophistication. All of the trial wavefunctions are initially generated with the GAMESS code. Topics covered include:

- Generating single determinant trial wavefunctions with GAMESS (HF and DFT)
- Generating multi-determinant trial wavefunctions with GAMESS (CISD, CASCI, SOCI)
- Optimizing wavefunctions (Jastrow factors and CSF coefficients) with QMC
- DMC timestep and walker population convergence studies
- Systematic progressions of Jastrow factors in VMC
- Systematic convergence of DMC energies with multi-determinant wavefunctions
- Influence of orbitals basis choice on DMC energy

19.2 Lab directories and files

```
labs/lab3_advanced_molecules/exercises
|
├── ex1_first-run-hartree-fock    - basic work flow from Hatree-Fock to DMC
|   ├── gms                        - Hatree-Fock calculation using GAMESS
|   |   ├── h2o.hf.inp               - GAMESS input
|   |   ├── h2o.hf.dat               - GAMESS punch file containing orbitals
|   |   └── h2o.hf.out               - GAMESS output with orbitals and other info
|   ├── convert                    - Convert GAMESS wavefunction to QMCPACK format
|   |   ├── h2o.hf.out               - GAMESS output
|   |   ├── h2o.ptcl.xml             - converted particle positions
|   |   └── h2o.wfs.xml              - converted wave function
|   ├── opt                        - VMC optimization
|   |   └── optm.xml                 - QMCPACK VMC optimization input
|   ├── dmc_timestep               - Check DMC timestep bias
|   |   └── dmc_ts.xml               - QMCPACK DMC input
```

```
|   └── dmc_walkers                - Check DMC population control bias
|       └── dmc_wk.xml               - QMCPACK DMC input template
|
├── ex2_slater-jastrow-wf-options - explore jastrow and orbital options
|   ├── jastrow                    - Jastrow options
|   |   ├── 12j                      - no 3-body Jastrow
|   |   ├── 1j                       - only 1-body Jastrow
|   |   └── 2j                       - only 2-body Jastrow
|   └── orbitals                   - Orbital options
|       ├── pbe                      - PBE orbitals
|       |   └── gms                    - DFT calculation using GAMESS
|       |       └── h2o.pbe.inp         - GAMESS DFT input
|       ├── pbe0                     - PBE0  orbitals
|       ├── blyp                     - BLYP  orbitals
|       └── b3lyp                    - B3LYP orbitals
|
├── ex3_multi-slater-jastrow
|   ├── cisd                       - CISD wave function
|   |   ├── gms                      - CISD calculation using GAMESS
|   |   |   ├── h2o.cisd.inp           - GAMESS input
|   |   |   ├── h2o.cisd.dat           - GAMESS punch file containing orbitals
|   |   |   └── h2o.cisd.out           - GAMESS output with orbitals and other info
|   |   └── convert                  - Convert GAMESS wavefunction to QMCPACK format
|   |       └── h2o.hf.out             - GAMESS output
|   ├── casci                      - CASCI wave function
|   |   └── gms                      - CASCI calculation using GAMESS
|   └── soci                       - SOCI wave function
|       ├── gms                      - SOCI calculation using GAMESS
|       ├── thres0.01                - VMC optimization with few determinants
|       └── thres0.0075              - VMC optimization with more determinants
|
└── pseudo
    ├── H.BFD.gamess             - BFD pseudopotential for H in GAMESS format
    ├── O.BFD.CCT.gamess         - BFD pseudopotential for O in GAMESS format
    ├── H.xml                    - BFD pseudopotential for H in QMCPACK format
    └── O.xml                    - BFD pseudopotential for H in QMCPACK format
```

19.3 Exercise #1: Basics

The purpose of this exercise is to show how to generate wave-functions for QMCPACK using GAMESS and to optimize the resulting wave-functions using VMC. This will be followed by a study of the time-step and walker population dependence of DMC energies. The exercise will be performed on a water molecule at the equilibrium geometry.

19.3.1 Generation of a Hatree-Fock wave-function with GAMESS

From the top directory, go to "ex1_first-run-hartree-fock/gms". This directory contains an input file for a HF calculation of a water molecule using BFD ECPs and the corresponding cc-pVTZ basis set. The input file should be named: "h2o.hf.inp". Study the input file. If the student wishes, he can refer to section A for a more detailed description of the GAMESS input syntax. There will be a better time to do this soon, so we recommend that the student continues with the exercise at

this point. After you are done, execute GAMESS with this input and store the standard output in a file named "h2o.hf.output". Finally, in the "convert" folder, use convert4qmc to generate the QMCPACK particleset and wavefunction files. It is always useful to rename the files generated by convert4qmc to something meaningful, since by default they are called sample.Gaussian-G2.xml and sample.Gaussian-G2.ptcl.xml. In a standard computer (without cross-compilation), these tasks could be accomplished by the following commands.

```
cd ${TRAINING TOP}/ex1_first-run-hartree-fock/gms
jobrun_vesta rungms h2o.hf
cd ../convert
cp ../gms/h2o.hf.output
jobrun_vesta convert4qmc -gamessAscii h2o.hf.output -add3BodyJ
mv sample.Gaussian-G2.xml h2o.wfs.xml
mv sample.Gaussian-G2.ptcl.xml h2o.ptcl.xml
```

The HF energy of the system is -16.9600590022 Ha. To search for the energy in the output file quickly, you can use

```
grep "TOTAL ENERGY =" h2o.hf.output
```

As the job runs on VESTA, it is a good time to review section B, which contains a description on the use of the converter.

19.3.2 Optimize the wave-function

When the execution of the previous steps is completed, there should be 2 new files called h2o.wfs.xml and h2o.ptcl.xml. Now we will use VMC to optimize the Jastrow parameters in the wave-function. From the top directory, go to "ex1_first-run-hartree-fock/opt". Copy the xml files generated in the previous step to the current directory. This directory should already contain a basic QMCPACK input file for an optimization calculation (optm.xml) Open optm.xml with your favorite text editor and modify the name of the files that contain the wavefunction and particleset XML blocks. These files are included with the commands:

```
<include href=ptcl.xml/>
<include href=wfs.xml/>
```

(the particle set must be defined before the wave-function). The name of the particle set and wave-function files should now be h2o.ptcl.xml and h2o.wfs.xml, respectively. Study both files and submit when you are ready. Notice that the location of the ECPs has been set for you, in your own calculations you have to make sure you obtain the ECPs from the appropriate libraries and convert them to QMCPACK format using ppconvert. This is a good time to study section C, which contains a review of the main parameters in the optimization XML block, while this calculation finishes. The previous steps can be accomplished by the following commands:

```
cd ${TRAINING TOP}/ex1_first-run-hartree-fock/opt
cp ../convert/h2o.wfs.xml ./
cp ../convert/h2o.ptcl.xml ./
# edit optm.xml to include the correct ptcl.xml and wfs.xml
jobrun_vesta qmcpack optm.xml
```

Use the analysis tool qmca to analyze the results of the calculation. Obtain the VMC energy and variance for each step in the optimization and plot it using your favorite program. Remember that qmca has built-in functions to plot the analyzed data.

```
qmca -q e *scalar.dat -p
```

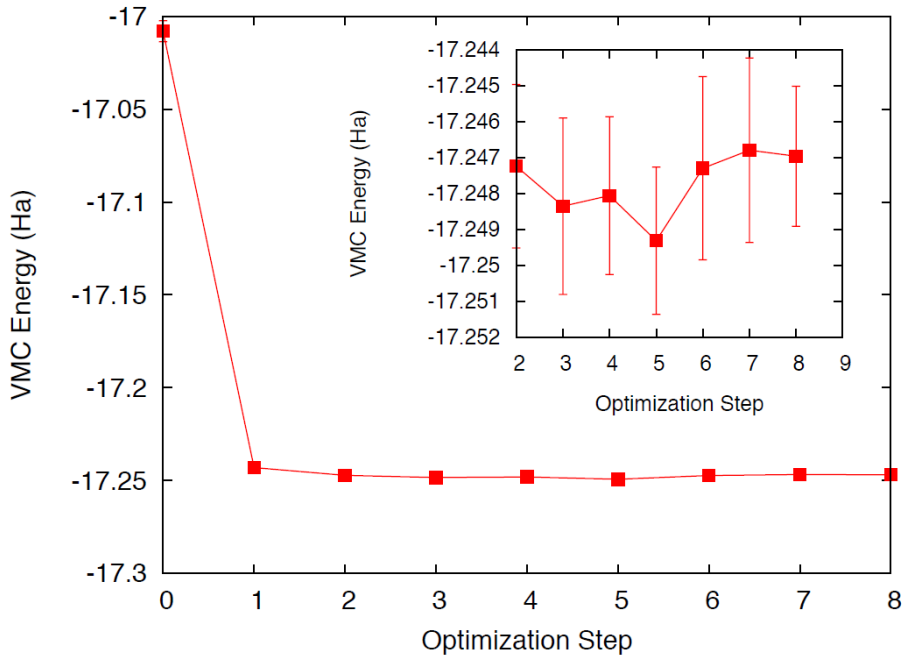


Figure 19.1: VMC energy as a function of optimization step.

The resulting energy as a function of optimization step should look qualitatively similar to figure 19.1. The energy should decrease quickly as a function of the number of optimization steps. After 6-8 steps, the energy should be converged to ∼2-3mHa. To improve convergence, we would need to increase the number of samples used during the optimization. You can check this for yourself on your free time. With optimized wave-functions we are in a position to perform VMC and DMC calculations. The modified wave-function files after each step are written in a file named ID.sNNN.opt.xml, where ID is the identifier of the calculation defined in the input file (this is defined in the project XML block with parameter "id") and NNN is a series number which increases with every executable xml block in the input file.

19.3.3 Time-step Study

Now we will study the dependence of the DMC energy with time-step. From the top directory, go to "ex1_first-run-hartree-fock/dmc_timestep". This folder contains a basic xml input file (dmc_ts.xml) that performs a short VMC calculation and three DMC calculations with varying time-steps (0.1, 0.05, 0.01). Link the particle set and the last optimization file from the previous folder (the file called jopt-h2o.sNNN.opt.xml with the largest value of NNN). Rename the optimized wave-function to any suitable name if you wish, for example h2o.opt.xml, and change the name of the particle set and wave-function files in the input file. An optimized wave-function can be found in the reference files (same location) in case it is needed.

The main steps needed to perform this exercise are:

```
cd \$\{TRAINING TOP\}/ex1_first-run-hartree-fock/dmc_timestep
cp ../opt/h2o.ptcl.xml ./
cp ../opt/jopt-h2o.s007.opt.xml h2o.opt.wfs.xml
# edit dmc_ts.xml to include the correct ptcl.xml and wfs.xml
```

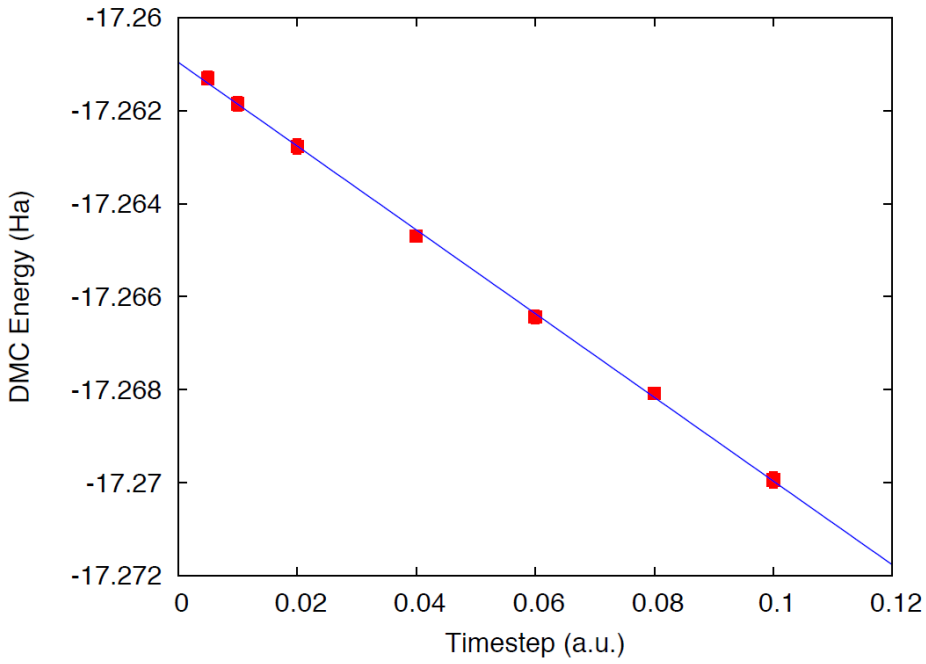


Figure 19.2: DMC energy as a function of timestep.

```
jobrun_vesta qmcpack dmc_ts.xml
```

While these runs complete, go to section D and review the basic VMC and DMC input blocks. Notice that in the current DMC blocks, as the time-step is decreased the number of blocks is also increased. Why is this?

When the simulations are finished, use qmca to analyze the output files and to plot the DMC energy as a function of time-step. Results should be qualitatively similar to those presented in figure 19.2, in this case we present more time-steps with well converged results to better illustrate the time-step dependence. In realistic calculations, the time-step must be chosen small enough so that the resulting error is below the desire accuracy. Alternatively, various calculations can be performed and the results extrapolated to the zero time-step limit.

19.3.4 Walker Population Study

Now we will study the dependence of the DMC energy with the number of walkers in the simulation. Remember that, in principle, the DMC distribution is reached in the limit of an infinite number of walkers. In practice, the energy and most properties converge to high accuracy with ∼100-1000 walkers. The actual number of walkers needed in a calculation will depend on the accuracy of the VMC wave-function and on the complexity and size of the system. Also notice that using too many walkers is not a problem, at worse it will be inefficient since it will cost more computer time than necessary. In fact, this is the strategy used when running QMC calculations on large parallel computers since we can reduce the statistical error bars efficiently by running with large walker populations distributed across all processors.

From the top directory, go to “ex1_first-run-hartree-fock/dmc_walkers”. Copy the optimized wave-function and particle set files used in the previous calculations to the current folder, these are the ones generated on step 2 of this exercise. An optimized wave-function can be found in

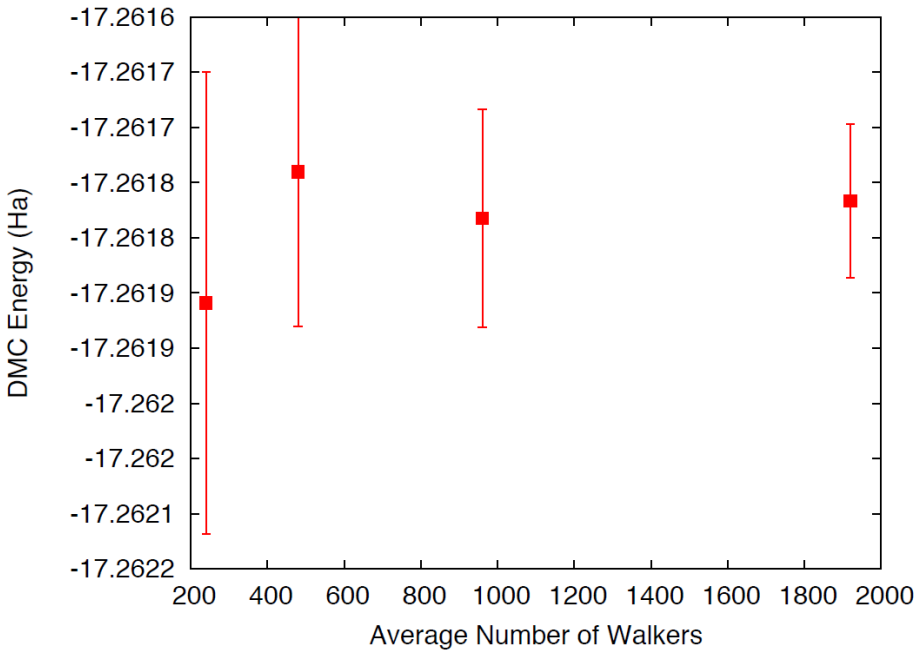


Figure 19.3: DMC energy as a function of the average number of walkers.

the reference files (same location) in case it is needed. The directory contains a sample DMC input file and submission script. Make 3 directories named NWx, with x values 120,240,480 and copy the input file to each one. Go to "NW120", and, in the input file, change the name of the wave-function and particle set files (in this case they will be located one directory above, so use "../dmc_timestep/h2.opt.xml" for example), change the pseudopotential directory to point to one directory above, change "targetWalkers" to 120, change the number of steps to 100, the time-step to 0.01 and the number of blocks to 400. Notice that "targetWalkers" is one way to set the desired (average) number of walkers in a DMC calculation. One can alternatively set "samples" in the <qmc method=``vmc'' block to carry over de-correlated VMC configurations as DMC walkers. For your own simulations we generally recommend setting ~2*(#threads) walkers per node (slightly smaller than this value).

The main steps needed to perform this exercise are:

```
cd ${TRAINING TOP}/ex1_first-run-hartree-fock/dmc_walkers
cp ../opt/h2o.ptcl.xml ./
cp ../opt/jopt-h2o.s007.opt.xml h2o.opt.wfs.xml
# edit dmc_wk.xml to include the correct ptcl.xml and wfs.xml and
#  use the correct pseudopotential directory
mkdir NW120
cp dmc_wk.xml NW120
# edit dmc_wk.xml to use the desired number of walkers,
#  and collect the desired amount of statistics
jobrun_vesta qmcpack dmc_wk.xml
# repeat for NW240, NW480
```

Repeat the same procedure in the other folders by setting (targetWalkers=240, steps=100, timestep=0.01, blocks=200) in NW240 and (targetWalkers=480, steps=100, timestep=0.01, blocks=100) in NW480. When the simulations complete, use qmca to analyze and plot the

energy as a function of the number of walkers in the calculation. As always, figure 19.3 shows representative results of the dependence of the energy on the number of walkers for a single water molecule. As shown, less than 240 walkers are needed to obtain an accuracy of 0.1 mHa.

19.4 Exercise #2 Slater-Jastrow Wave-Function Options

From this point on in the tutorial we assume familiarity with the basic parameters in the optimization, VMC and DMC XML input blocks of QMCPACK. In addition, we assume familiarity with the submission system. As a result, the folder structure will not contain any prepared input or submission files, the student will generate them using input files from exercise 1. In the case of QMCPACK sample files, you will find optm.xml, vmc dmc.xml and submit.csh files. Some of the options in these files can be left unaltered, but many of them will need to be tailored to the particular calculation.

In this exercise we will study the dependence of the DMC energy on the choices made in the wave-function ansatz. In particular, we will study the influence/dependence of the VMC energy with the various terms in the Jastrow. We will also study the influence of the VMC and DMC energies on the single particle orbitals used to form the Slater determinant in single determinant wave-functions. For this we will use wave-functions generated with various exchange-correlation functionals in DFT. Finally, we will optimize a simple multi-determinant wave-function and study the dependence of the energy o the number of configurations used in the expansion. All of these exercises will be performed on the water molecule at equilibrium.

19.4.1 Influence of Jastrow on VMC energy with HF wave-function

In this section we will study the dependence of the VMC energy on the various Jastrow terms, e.g. one-body, two-body and three-body. From the top directory, go to "ex2_slater-jastrow-wf-options/jastrow". We will compare the single determinant VMC energy using a two-body Jastrow term, both one- and two-body terms and finally one-, two- and three-body terms. Since we are interested in the influence of the Jastrow, we will use the HF orbitals calculated in exercise #1. Make three folders named 2j,12j,123j. For both 2j and 12j , copy the input file optm.xml from "ex1_first-run-hartree-fock/opt" . This input file performs both wave-function optimization and a VMC calculation. Remember to correct relative paths to the pseudopotential directory. Copy the un-optimized HF wave-function and particle set files from "ex1_first-run-hartree-fock/convert", if you followed the instructions in exercise #1 these should be named h2o.wfs.xml and h2o.ptcl.xml. Otherwise, you can obtained them from the REFERENCE files. Modify the file h2o.wfs.xml to remove the appropriate jastrow blocks. For example, for a two-body Jastrow (only), you need to eliminate the jastrow blocks named `<jastrow name="J1"` and `<jastrow name="J3"`. In the case of 12j, remove only `<jastrow name="J3"`. Recommended settings for the optimization run are: nodes=32, threads=16, blocks=250, samples=128000, time-step=0.5, 8 optimization loops, and in the VMC section we recommend walkers=16, blocks=1000, steps=1, substeps=100. Notice that samples should always be set to blocks*threads per node*nodes = 32*16*250=128000. Repeat the process in both 2j and 12j cases. For the 123j case, the wave-function has already been optimized in the previous exercise. Copy the optimized HF wave-function and the particle set from "ex1_first-run-hartree-fock/opt". Copy the input file from any of the previous runs and remove the optimization block from the input, just leave the VMC step. In all three cases, modify the submission script and submit the run.

These simulations will take several minutes to complete. This is an excellent opportunity to go to section E and review the wavefunction XML block used by QMCPACK. When the simulation

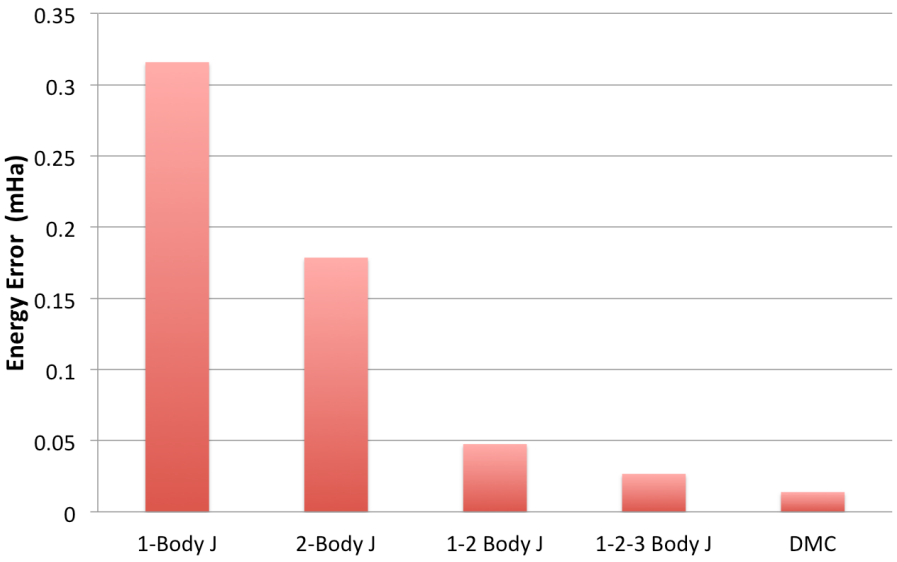


Figure 19.4: VMC energy as a function of Jastrow type.

are completed, use qmca to analyze the output files. Using your favorite plotting program (e.g. gnu plot), plot the energy and variance as a function of the Jastrow form. Figure 19.4 shows a typical result for this calculation. As can be seen, the VMC energy and variance depends strongly on the form of the Jastrow. Since the DMC error bar is directly related to the variance of the VMC energy, improving the Jastrow will always lead to a reduction in the DMC effort. In addition, systematic approximations (time-step, number of walkers, etc) are also reduced with improved wave-functions.

19.4.2 Generation of wave-functions from DFT using GAMESS

In this section we will use GAMESS to generate wave-functions for QMCPACK from DFT calculations. From the top folder, go to "ex2_slater-jastrow-wf-options/orbitals". In order to demonstrate the variation in DMC energies with the choice of DFT orbitals, we will choose the following set of exchange-correlation functionals (PBE, PBE0, BLYP, B3LYP). For each functional, make a directory using your preferred naming convention (e.g. the name of the functional). Go into each folder and copy a GAMESS input file from "ex1_first-run-hartree-fock/gms" .Rename the file with your preferred naming convention, we suggest using h2o.[dft].inp, where [dft] is the name of the functional used in the calculation. At this point, this input file should be identical to the one used to generate the HF wave-function in exercise #1. In order to perform a DFT calculation we only need to add "DFTTYP" to the mathescapemathescapemathescapemathescape$CONTRL ... $END section and set it to the desired functional type, for example "DFTTYP=PBE" for a PBE functional. This variable must be set to (PBE, PBE0, BLYP, B3LYP) to obtain the appropriate functional in GAMESS. For a complete list of implemented functionals, see the GAMESS input manual.

19.4.3 Optimization and DMC calculations with DFT wave-functions

In this section we will optimize the wave-function generated in the previous step and perform DMC calculations. From the top directory, go to "ex2_slater-jastrow-wf-options/orbitals". The

steps required to achieve this are identical to those used to optimize the wave-function with HF orbitals. Make individual folders for each calculation and obtain the necessary files to perform optimization, VMC and DMC calculations from "ex1_first-run-hartree-fock/opt" and "ex1_first-run-hartree-fock/dmc_ts", for example. For each functional, make the appropriate modifications to the input files and copy the particle set and wave-function files from the appropriate directory in "ex2_slater-jastrow-wf-options/orbitals/[dft]". We recommend the following settings: nodes=32, threads=16, (in optimization) blocks=250, samples=128000, timestep=0.5, 8 optimization loops, (in VMC) walkers=16, blocks=100, steps=1, substeps=100, (in DMC) blocks 400, targetWalkers=960, timestep=0.01. Submit the runs and analyze the results using qmca .

How do the energies compare against each other? How do they compare against DMC energies with HF orbitals?

19.5 Exercise #3: Multi-Determinant Wave-Functions

In this exercise we will study the dependence of the DMC energy on the set of orbitals and the type of configurations included in a multi-determinant wave-function.

19.5.1 Generation of a CISD wave-functions using GAMESS

In this section we will use GAMESS to generate a multi-determinant wave-function with Configuration Interaction with Single and Double excitations (CISD). In CISD, the Schrodinger equation is solved exactly in a basis of determinants including the HF determinant and all its single and double excitations.

Go to "ex3_multi-slater-jastrow/cisd/gms" and you'll see input and output files named h2o.cisd.inp and h2o.cisd.out. Due to technical problems with GAMESS in the BGQ architecture of VESTA, we are unable to use CISD properly in GAMESS. For this reason, the output of the calculation is already provided in the directory.

There will be time in the next step to study the GAMESS input files and the description in section A. Since the output is already provided, the only thing needed is to use the converter to generate the appropriate QMCPACK files.

```
jobrun_vesta convert4qmc h2o.cisd.out -ci h2o.cisd.out \
-readInitialGuess 57 -threshold 0.0075
```

We used the PRTMO=.T. flag in the GUESS section to include orbitals in the output file. You should read these orbitals from the output (-readInitialGuess 40). The highest occupied orbital in any determinant should be 34, so reading 40 orbitals is a safe choice. In this case, it is important to rename the xml files with meaningful names, for example h2o.cisd.wfs.xml. A threshold of 0.0075 is sufficient for the calculations in the training.

19.5.2 Optimization of Multi-Determinant wave-function

In this section we will optimize the wave-function generated in the previous step. There is no difference in the optimization steps if a single determinant and a multi-determinant wave-function. QMCPACK will recognize the presence of a multi-determinant wavefunction and will automatically optimize the linear coefficients by default. Go to "ex3_multi-slater-jastrow/cisd" and make a folder called thres0.01. Copy the particle set and wavefunction files created in the previous step to the current directory. With your favorite text editor, open the wave-function file h2o.wfs.xml. Look for the multideterminant XML block and change the "cutoff" parameter in detlist to 0.01. Then

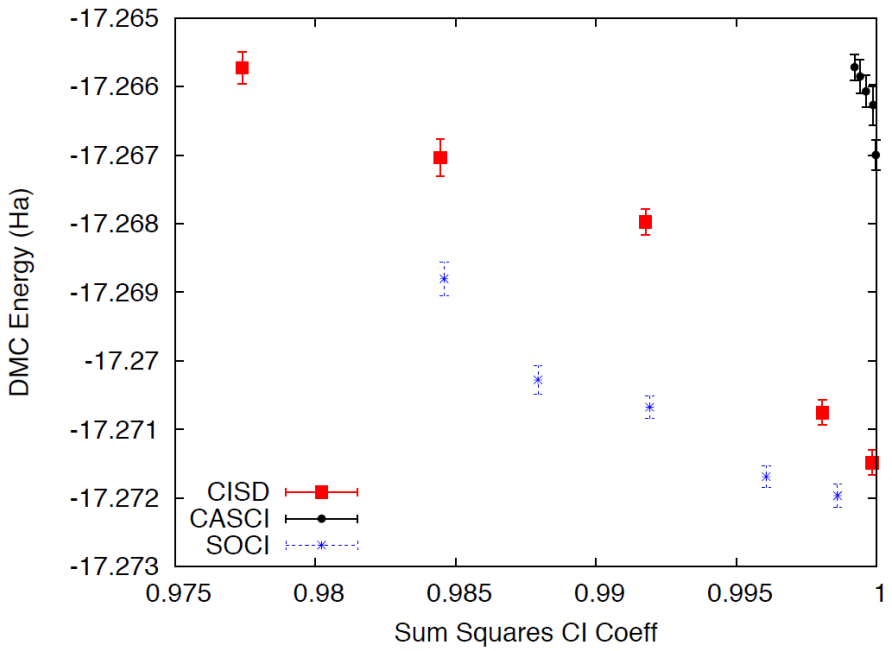


Figure 19.5: DMC energy as a function of the sum of the square of CI coefficients from CISD.

follow the same steps used in the subsection "Optimization and DMC calculations with DFT wave-functions" to optimize the wave-function. Similar to this case, design a QMCPACK input file that performs wave-function optimization followed by VMC and DMC calculations. Submit the calculation.

This is a good time to review the GAMESS input file description in Appendix section A. When the run is completed, go to the previous directory and make a new folder named thres0.0075. Repeat the steps performed above to optimize the wave-function with a cutoff of 0.01, but use a cutoff of 0.0075 this time. This will increase the number of determinants used in the calculation. Notice the "cutoff" parameter in the XML should be less than the "-threshold 0.0075" flag passed to the converted, which is further bounded by the PRTTOL flag in the GAMESS input.

After the wave-function is generated, we are ready to optimize. Instead of starting from an un-optimized wave-function, we can start from optimized wave-function from thres0.01 to speed up convergence. You will need to modify the file and change the cutoff in detlist to 0.0075 with a text editor. Repeat the optimization steps and submit the calculation.

When you are done, use qmca to analyze the results. Compare the energies at these two coefficient cutoffs with the energies obtained with DFT orbitals. Due to the time limitations of this tutorial it is not practical to optimize the wave-functions with a smaller cutoff, since this would require more samples and longer runs due to the larger number of optimizable parameters. Figure 19.5 shows the results of such exercise, the DMC energy as a function of the cutoff in the wave-function. As can be seen, a large improvement in the energy is obtained as the number of configurations is increased.

19.5.3 CISD, CASCI and SOCI

Go to "ex3_multi-slater-jastrow" and inspect folders for the remaining wave-function types: CASCI and SOCI. Follow steps in the previous exercise and obtain the optimized wave-functions for these determinant choices. Notice the SOCI GAMESS output is not included because it is large. Already converted XML inputs can be found in "ex3_multi-slater-jastrow/soci/thres*".

A CASCI wave-function is produced from a CI calculation that includes all the determinants in a complete active space (CAS) calculation, in this case using the orbitals from a previous CASSCF calculation. In this case we used a CAS(8,8) active space, that includes all determinants generated by distributing 8 electrons in the lowest 8 orbitals. A second-order CI (SOCI) calculation is similar to the CAS-CI calculation, but in addition to the determinants in the CAS it also includes all single and double excitations from all of them, leading to a much larger determinant set. Since we now have considerable experience optimizing wave-functions and calculating DMC energies, we will leave it to the student to complete the remaining tasks on its own. If you need help, refer to previous exercises in the tutorial. Perform optimizations for both wave-functions using cutoffs in the CI expansion of 0.01 an 0.0075. If there is enough time left, try to optimize the wave-functions with a cutoff of 0.005. Analyze the results and plot the energy as a function of cutoff for all three cases, CISD, CAS-CI and SOCI.

Figure 19.5 shows the result of similar calculations using more samples and smaller cutoffs. The results should be similar to those produced in the tutorial. For reference, the exact energy of the water molecule with ECPs is approximately -17.276 Ha. From the results of the tutorial, how does the selection of determinants is related to the expected DMC energy? What about the choice in the set of orbitals?

19.6 Appendix A: GAMESS input

In this section we provide a brief description of the GAMESS input needed to produce trial wave-function for QMC calculations with QMCPACK. We assume basic familiarity with GAMESS input structure, in particular regarding the input of atomic coordinates and the definition of gaussian basis sets. This section will focus on the generation of the output files needed by the converter tool, convert4qmc. For a description of the converter, see B.

Only a subset of the methods available in GAMESS can be used to generate wave-functions for QMCPACK and we restrict our description here to these. For a complete description of all the options and methods available in GAMESS, please refer to the official documentation which could be found in "http://www.msg.ameslab.gov/gamess/documentation.html".

Currently, convert4qmc can process output for the following methods in GAMESS (in SCFTYP) : RHF, ROHF, and MCSCF. Both HF as well as DFT calculations (any DFT type) could be used in combination with RHF and ROHF calculations. For MCSCF and CI calculations, ALDET, ORMAS and GUGA drivers can be used (see below for details).

19.6.1 HF input

The following input will perform a restricted HF calculation on a closed-shell singlet (multiplicity=1). This will generate RHF orbitals for any molecular system defined in `$DATA ... $END`.

mathescapemathescapemathescapemathescape

```
$CONTRL SCFTYP=RHF RUNTYP=ENERGY MULT=1
ISPHER=1 EXETYP=RUN COORD=UNIQUE MAXIT=200 $END
$SYSTEM MEMORY=150000000 $END
$GUESS GUESS=HUCKEL $END
$SCF DIRSCF=.TRUE. $END
$DATA
...
Atomic Coordinates and basis set
...
$END
```

Main options:

1. SCFTYP: Type of SCF method, options: RHF, ROHF, MCSCF, UHF and NONE.
2. RUNTYP: Type of run. For QMCPACK wave-function generation this should always be ENERGY.
3. MULT: Multiplicity of the molecule.
4. ISPHER: Use spherical harmonics (1) or cartesian basis functions (-1).
5. COORD: Input structure for the atomic coordinates in $DATA.

19.6.2 DFT calculations

The main difference between the input for a RHF/ROHF calculation and a DFT calculation is the definition of the DFTTYP parameter. If this is set in the $CONTROL section, a DFT calculation will be performed with the appropriate functional. Notice that while the default values are usually adequate, DFT calculations have many options involving the integration grids and accuracy settings. Make sure you study the input manual to be aware of these. Refer to the input manual for a list of the implemented exchange-correlation functionals.

19.6.3 Multi-Configuration Self-Consistent Field (MCSCF)

MCSCF calculations are performed by setting SCFTYP=MCSCF in the $CONTROL section. If this option is set, a M
section must be added to the input file with the options for the calculation. An example section for the water molecule used in the tutorial is shown below.

mathescapemathescapemathescapemathescape

```
$MCSCF CISTEP=GUGA MAXIT=1000 FULLNR=.TRUE. ACURCY=1.0D-5 $END
```

The most important parameter is CISTEP, which defines the CI package used. The only options compatible with QMCPACK are: ALDET, GUGA, and ORMAS. Depending on the package used, additional input sections are needed.

19.6.4 Configuration Interaction (CI)

Configuration interaction (full CI, truncated CI, CAS-CI, etc) calculations are performed by setting mathescapemathescapemathescapemathescapeSCFTYP=NONE and mathescapemathescapemathescapemathescapemathescapeCITYP=GUGA,ALDET,ORMAS. Each one of this packages requires further input sections, which are typically slightly different to the input sections needed for MCSCF runs.

19.6.5 GUGA: Unitary Group CI package

The GUGA package is the only alternative if one wants CSFs with GAMESS. Below we provide a very brief description of input sections needed to perform MCSCF, CASCI, truncated CI and SOCI with this package. For a complete description of these methods and all the options available, please refer to the GAMESS input manual.

GUGA-MCSCF

The following input section performs a CASCI calculation, with a CAS that includes 8 electrons in 8 orbitals (4 DOC and 4 VAL), e.g. CAS(8,8). NMCC is the number of frozen orbitals (doubly occupied orbitals in all determinants), NDOC is the number of double occupied orbitals in the reference determinant, NVAL is the number of singly occupied orbitals in the reference (for spin polarized cases), and NVAL is the number of orbitals in the active space. Since FORS is set to .TRUE., all configurations in the active space will be included. ISTSYM defines the symmetry of the desired state.

mathescapemathescapemathescapemathescape

```
$MCSCF CISTEP=GUGA MAXIT=1000 FULLNR=.TRUE. ACURCY=1.0D-5 $END
$DRT GROUP=C2v NMCC=0 NDOC=4 NALP=0 NVAL=4 ISTSYM=1 MXNINT= 500000 FORS=.TRUE. $END
```

GUGA-CASCI

The following input section performs a CASCI calculation, with a CAS that includes 8 electrons in 8 orbitals (4 DOC and 4 VAL), e.g. CAS(8,8). NFZC is the number of frozen orbitals (doubly occupied orbitals in all determinants). All other parameters are identical to those in the MCSCF input section.

mathescapemathescapemathescapemathescape

```
$CIDRT GROUP=C2v NFZC=0 NDOC=4 NALP=0 NVAL=4 NPRT=2 ISTSYM=1 FORS=.TRUE. MXNINT=
    500000 $END
$GUGDIA PRTTOL=0.001 CVGTOL=1.0E-5 ITERMX=1000 $END
```

GUGA-Truncated CI

The following input sections will lead to a truncated CI calculation, in this particular case it will perform a CISD calculation since IEXCIT is set to 2. Other values in IEXCIT will lead to different CI truncations, for example IEXCIT=4 will lead to CISDTQ. Notice that only the lowest 30 orbitals will be included in the generation of the excited determinants in this case. For a full CISD calculation, NVAL should be set to the total number of virtual orbitals.

mathescapemathescapemathescapemathescape

```
$CIDRT GROUP=C2v NFZC=0 NDOC=4 NALP=0 NVAL=30 NPRT=2 ISTSYM=1 IEXCIT=2 MXNINT= 500000
    $END
$GUGDIA PRTTOL=0.001 CVGTOL=1.0E-5 ITERMX=1000 $END
```

GUGA-SOCI

The following input section performs a SOCI calculation, with a CAS that includes 8 electrons in 8 orbitals (4 DOC and 4 VAL), e.g. CAS(8,8). Since SOCI is set to .TRUE., all single and double determinants from all determinants in the CAS(8,8) will be included.

mathescapemathescapemathescapemathescape

```
$CIDRT GROUP=C2v NFZC=0 NDOC=4 NALP=0 NVAL=4 NPRT=2 ISTSYM=1 SOCI=.TRUE. NEXT=30
    MXNINT= 500000 $END
$GUGDIA PRTTOL=0.001 CVGTOL=1.0E-5 ITERMX=1000 $END
```

19.6.6 ECP

To use Effective Core Potentials (ECP) in GAMESS, you must define a {`$ECP ... $END`} block. There must be a definition of a potential for every atom in the system, including symmetry equivalent ones. In addition, they must appear in the particular order expected by GAMESS. Below is an example of an ECP input block for a single water molecule using BFD ECPs. To turn on the use of ECPs, the option "ECP=READ" must be added to the CONTROL input block.

mathescapemathescapemathescapemathescape

```
$ECP
O-QMC GEN 2 1
3
6.00000000 1 9.29793903
55.78763416 3 8.86492204
-38.81978498 2 8.62925665
1
38.41914135 2 8.71924452
H-QMC GEN 0 0
3
1.000000000000 1 25.000000000000
25.000000000000 3 10.821821902641
-8.228005709676 2 9.368618758833
H-QMC
$END
```

19.7 Appendix B: convert4qmc

To generate the particleset and wavefunction XML blocks required by QMCPACK in calculations with molecular systems, the converter convert4qmc must be used. The converter will read the standard output from the appropriate Quantum Chemistry calculation and will generate all the necessary input for QMCPACK. Below we describe the main options of the converter for GAMESS output. In general, there are 3 ways to use the converter depending on the type of calculation performed. The minimum syntax for each option is found below. For a description of the xml files produced by the converter, see section E.

1. For all single determinant calculations (HF and DFT with any DFTTYP):

```
convert4qmc -gamessAscii single det.out
```

   - single det.out is the standard output generated by GAMESS.

2. *(This option is not recommended. Use option below to avoid mistakes.)* For multi-determinant calculations where the orbitals and configurations are read from different files (for example when using orbitals from a MCSCF run and configurations from a subsequent CI run):

```
convert4qmc -gamessAscii orbitals multidet.out -ci cicoeff multidet.out
```

   - orbitals_multidet.out is the standard output from the calculation that generates the orbitals. cicoeff multidet.out is the standard output from the calculation that calculates the CI expansion.

3. For multi-determinant calculations where the orbitals and configurations are read from the same file, using PRTMO=.T. in the GUESS input block:

```
convert4qmc -gamessAscii multi det.out -ci multi det.out -readInitialGuess Norb
```

   - multi_det.out is the standard output from the calculation that calculates the CI expansion.

Options:

- **-gamessAscii file.out**: Standard output of GAMESS calculation. With the exception of determinant configurations and coefficients in multi-determinant calculations, everything else is read from this file including: atom coordinates, basis sets, single particle orbitals, ECPs, number of electrons, multiplicity, etc.
- **-ci file.out**: In multi-determinant calculations, determinant configurations and coefficients are read from this file. Notice that single particle orbitals are NOT read from this file. Recognized CI packages are: ALDET, GUGA and ORMAS. Output produced with the GUGA package MUST have the option "NPRT=2" in the CIDRT or DRT input blocks.
- **-threshold cutoff**: Cutoff in multi-determinant expansion. Only configurations with coefficients above this value are printed.

- **-zeroCI**: Sets to zero the CI coefficients of all determinants, with the exception of the first one.
- **-readInitialGuess Norb**: Reads Norb initial orbitals ("INITIAL GUESS ORBITALS") from GAMESS output. These are orbitals generated by the GUESS input block and printed with the option "PRTMO=.T.". Notice that this is useful only in combination with the option "GUESS=MOREAD" and in cases where the orbitals are not modified in the GAMESS calculation, e.g. CI runs. This is the recommended option in all CI calculations.
- **-NaturalOrbitals Norb**: Read Norb "NATURAL ORBITALS" from GAMESS output. The natural orbitals must exists in the output, otherwise the code aborts.
- **-add3BodyJ**: Adds three-body Jastrow terms (e-e-I) between electron pairs (both same spin and opposite spin terms) and all ion species in the system. The radial function is initialized to zero and the default cutoff is 10.0 bohr. The converter will add a one- and two-body Jastrow to the wavefunction block by default.

19.7.1 Useful notes

- The type of single particle orbitals read by the converter depends on the type of calculation and on the options used. By default, when neither -readInitialGuess or -NaturalOrbitals are used, the following orbitals are read in each case (notice that -readInitialGuess or -NaturalOrbitals are mutually exclusive):
  - RHF and ROHF: "EIGENVECTORS"
  - MCSCF: "MCSCF OPTIMIZED ORBITALS"
  - GUGA, ALDET, ORMAS: Cannot read orbitals without -readInitialGuess or -NaturalOrbitals options.
- The single particle orbitals and printed CI coefficients in MCSCF calculations are not consistent in GAMESS. The printed CI coefficients correspond to the next-to-last iteration, they are not recalculated with the final orbitals. So in order to get appropriate CI coefficients from MCSCF calculations, a subsequent CI (no SCF) calculation is needed to produce consistent orbitals. In principle, it is possible to read the orbitals from the MCSCF output and the CI coefficients and configurations from the output of the following CI calculations. This could lead to problems in principle, since GAMESS will rotate initial orbitals by default in order to obtain an initial guess consistent with the symmetry of the molecule. This last step is done by default and can change the orbitals reported in the MCSCF calculation before the CI is performed. In order to avoid this problem, it is highly recommended to use option #3 above to read all the information from the output of the CI calculation, this requires the use of "PRTMO=.T." in the GUESS input block. Since the orbitals are printed after any symmetry rotation, the resulting output will always be consistent.

19.8 Appendix C: Wave-function Optimization XML block

Listing 19.1: "Sample XML optimization block."

```
<loop max="10">
  <qmc method="linear" move="pbyp" checkpoint="-1" gpu="no">
  <parameter name="blocks">     10  </parameter>
    <parameter name="warmupsteps"> 25 </parameter>
    <parameter name="steps"> 1 </parameter>
    <parameter name="substeps"> 20 </parameter>
    <parameter name="timestep"> 0.5 </parameter>
    <parameter name="samples"> 10240  </parameter>
    <cost name="energy">                    0.95 </cost>
    <cost name="unreweightedvariance">      0.0 </cost>
    <cost name="reweightedvariance">        0.05 </cost>
    <parameter name="useDrift">  yes </parameter>
    <parameter name="bigchange">10.0</parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="usebuffer"> yes </parameter>
    <parameter name="nonlocalpp"> yes </parameter>
    <parameter name="MinMethod">quartic</parameter>
    <parameter name="exp0">-6</parameter>
    <parameter name="alloweddifference"> 1.0e-5 </parameter>
    <parameter name="stepsize">  0.15 </parameter>
    <parameter name="nstabilizers"> 1 </parameter>
  </qmc>
</loop>
```

Options:

- bigchange: (default 50.0) largest parameter change allowed
- usebuffer: (default no) Save useful information during VMC
- nonlocalpp: (default no) Include non-local energy on 1-D min
- MinMethod: (default quartic) Method to calculate magnitude of parameter change quartic: fit quartic polynomial to 4 values of the cost function obtained using reweighting along chosen direction linemin: direct line minimization using reweighting rescale: no 1-D minimization. Uses Umrigars suggestions.
- stepsize: (default 0.25) step size in either quartic or linemin methods.
- alloweddifference: (default 1e-4) Allowed increased in energy
- exp0: (default -16.0) Initial value for stabilizer (shift to diagonal of H) Actual value of stabilizer is 10 exp0
- nstabilizers: (default 3) Number of stabilizers to try
- stabilizaterScale: (default 2.0) Increase in value of exp0 between iterations.
- max its: (default 1) number of inner loops with same sample
- minwalkers: (default 0.3) minimum value allowed for the ratio of effective samples to actual number of walkers in a reweighting step. The optimization will stop if the effective number of walkers in any reweighting calculation drops below this value. Last set of acceptable parameters are kept.

- maxWeight: (defaul 1e6) Maximum weight allowed in reweighting. Any weight above this value will be reset to this value.

Recommendations:

- Set samples to equal to (#threads)*blocks.
- Set steps to 1. Use substeps to control correlation between samples.
- For cases where equilibration is slow, increase both substeps and warmupsteps.
- For hard cases (e.g. simultaneous optimization of long MSD and 3-Body J), set exp0 to 0 and do a single inner iteration (max its=1) per sample of configurations.

19.9 Appendix D: VMC and DMC XML block

Listing 19.2: "Sample XML blocks for VMC and DMC calculations."

```
<qmc method="vmc" move="pbyp" checkpoint="-1">
  <parameter name="useDrift">yes</parameter>
  <parameter name="warmupsteps">100</parameter>
  <parameter name="blocks">100</parameter>
  <parameter name="steps">1</parameter>
  <parameter name="substeps">20</parameter>
  <parameter name="walkers">30</parameter>
  <parameter name="timestep">0.3</parameter>
  <estimator name="LocalEnergy" hdf5="no"/>
</qmc>
<qmc method="dmc" move="pbyp" checkpoint="-1">
  <parameter name="nonlocalmoves">yes</parameter>
  <parameter name="targetWalkers">1920</parameter>
  <parameter name="blocks">100</parameter>
  <parameter name="steps">100</parameter>
  <parameter name="timestep">0.1</parameter>
  <estimator name="LocalEnergy" hdf5="no"/>
</qmc>
```

General Options:

- **move**: (default "walker") Type of electron move. Options: "pbyp" and "walker".
- **checkpoint**: (default "-1") (If > 0) Generate checkpoint files with given frequency. The calculations can be restarted/continued with the produced checkpoint files.
- **useDrift**: (default "yes") Defines the sampling mode. useDrift = "yes" will use Langevin acceleration to sample the VMC and DMC distributions, while useDrift="no" will use random displacements in a box.
- **warmupSteps**: (default 0) Number of steps warmup steps at the beginning of the calculation. No output is produced for these steps.
- **blocks**: (default 1) Number of blocks (outer loop).
- **steps**: (default 1) Number of steps per blocks (middle loop).
- **sub steps**: (default 1) Number of substeps per step (inner loop). During sub steps, the local energy is not evaluated in VMC calculations, which leads to faster execution. In VMC calculations, set sub steps to the average autocorrelation time of the desired quantity.
- **time step**: (default 0.1) Electronic time step in bohr.
- **samples**: (default 0) Number of walker configurations saved during the current calculation.
- **walkers**: (default #threads) In VMC, sets the number of walkers per node. The total number of walkers in the calculation will be equal to walkers*(# nodes).

Options unique to DMC:

- **targetWalkers**: (default #walkers from previous calculation, e.g. VMC.) Sets the target number of walkers. The actual population of walkers will fluctuate around this value. The walkers will be distributed across all the nodes in the calculation. On a given node, the walkers are split across all the threads in the system.
- **nonlocalmoves**: (default "no") Set to "yes" to turns on the use of Casula's T-moves.

19.10 Appendix E: Wave-function XML block

Listing 19.3: "Basic framework for a single determinant determinantset XML block."

```
<wavefunction name="psi0" target="e">
  <determinantset type="MolecularOrbital" name="LCAOBSet"
   source="ion0" transform="yes">
    <basisset name="LCAOBSet">
      <atomicBasisSet name="Gaussian-G2" angular="cartesian" type="Gaussian"
   elementType="O" normalized="no">
      ...
      </atomicBasisSet>
    </basisset>
    <slaterdeterminant>
      <determinant id="updet" size="4">
        <occupation mode="ground"/>
        <coefficient size="57" id="updetC">
        ...
        </coefficient>
      </determinant>
      <determinant id="downdet" size="4">
        <occupation mode="ground"/>
        <coefficient size="57" id="downdetC">
        ...
        </coefficient>
      </determinant>
    </slaterdeterminant>

  </determinantset>

  <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
  ...
  </jastrow>

</wavefunction>
```

In this section we describe the basic format of a QMCPACK wavefunction XML block. Everything listed in this section is generated by the appropriate converter tools. Little to no modification is needed when performing standard QMC calculations. As a result, this section is meant mainly for illustration purposes. Only experts should attempt to modify these files (with very few exceptions like the cutoff of CI coefficients and the cutoff in Jastrow functions) since changes can lead to unexpected results.

A QMCPACK wavefunction XML block is a combination of a determinantset, which contains the anti-symmetric part of the wave-function, and one or more jastrow blocks. The syntax of the anti-symmetric block depends on whether the wave-function is a single determinant or a multi-determinant expansion. Listing 19.3 shows the general structure of the single determinant case. The determinantset block is composed of a basisset block, which defines the atomic orbital basis set, and a slaterdeterminant block, which defines the single particle orbitals and occupation numbers of the Slater determinant. Listing 19.4 shows a (piece of a) sample of a slaterdeterminant block. The slaterdeterminant block consists of 2 determinant blocks, one for each electron spin. The parameter "size" in the determinant block refers to the number of single particle orbitals present while the "size" parameter in the coefficient block refers to the number of atomic basis functions per single particle orbital.

Listing 19.4: "Sample XML block for the single slater determinant case."

```
    <slaterdeterminant>
      <determinant id="updet" size="5">
        <occupation mode="ground"/>
        <coefficient size="134" id="updetC">
 9.55471000000000e-01 -3.87000000000000e-04  6.51140000000000e-02
   2.17700000000000e-03
 1.43900000000000e-03  4.00000000000000e-06 -4.58000000000000e-04
  -5.20000000000000e-05
-2.40000000000000e-05  6.00000000000000e-06 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -5.26000000000000e-04  2.63000000000000e-04
   2.63000000000000e-04
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
  -1.27000000000000e-04
 6.30000000000000e-05  6.30000000000000e-05 -0.00000000000000e+00
  -0.00000000000000e+00
-0.00000000000000e+00 -3.20000000000000e-05  1.60000000000000e-05
   1.60000000000000e-05
-0.00000000000000e+00 -0.00000000000000e+00 -0.00000000000000e+00
   7.00000000000000e-06
```

Listing 19.5 shows the general structure of the multi-determinant case. Similar to the single determinant case, the determinantset must contain a basisset block. This definition is identical to the one described above. In this case, the definition of the single particle orbitals must be done independently from the definition of the determinant configurations, the latter is done in the sposet block while the former is done on the multideterminant block. Notice that 2 sposet sets must be defined, one for each electron spin. The name of reach sposet set is required in the definition of the multideterminant block. The determinants are defined in terms of occupation numbers based on these orbitals.

Listing 19.5: "Basic framework for a multi-determinant determinantset XML block."

```
<wavefunction id="psi0" target="e">
  <determinantset name="LCAOBSet" type="MolecularOrbital" transform="yes"
  source="ion0">
    <basisset name="LCAOBSet">
      <atomicBasisSet name="Gaussian-G2" angular="cartesian" type="Gaussian"
  elementType="O" normalized="no">
      ...
      </atomicBasisSet>
      ...
    </basisset>
    <sposet basisset="LCAOBSet" name="spo-up" size="8">
```

```
        <occupation mode="ground"/>
        <coefficient size="40" id="updetC">
        ...
</coefficient>
      </sposet>
      <sposet basisset="LCAOBSet" name="spo-dn" size="8">
        <occupation mode="ground"/>
        <coefficient size="40" id="downdetC">
        ...
      </coefficient>
      </sposet>
      <multideterminant optimize="yes" spo_up="spo-up" spo_dn="spo-dn">
        <detlist size="97" type="CSF" nca="0" ncb="0" nea="4" neb="4" nstates="8"
    cutoff="0.001">
          <csf id="CSFcoeff_0" exctLvl="0" coeff="0.984378" qchem_coeff="0.984378"
    occ="22220000">
            <det id="csf_0-0" coeff="1" alpha="11110000" beta="11110000"/>
          </csf>
          ...
        </detlist>
      </multideterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
    ...
    </jastrow>
  </wavefunction>
```

There are various options in the multideterminant block that users should be aware of.

- cutoff: (IMPORTANT!) Only configurations with (absolute value) "qchem coeff" larger than this value will be read by QMCPACK.
- optimize: Turn on/off the optimization of linear CI coefficients.
- coeff: (in csf) Current coefficient of given configuration. Gets updated during wavefunction optimization.
- qchem coeff: (in csf) Original coefficient of given configuration from GAMESS calculation. This is used when applying a cutoff to the configurations read from the file. The cutoff is applied on this parameter and not on the optimized coefficient.
- nca and nab: number of core orbitals for up/down electrons. A core orbital is an orbital that is doubly occupied in all determinant configurations, not to be confused with core electrons. These are not explicitly listed on the definition of configurations.
- nea and neb: number of up/down active electrons (those being explicitly correlated).
- nstates: number of correlated orbitals
- size (in detlist): contains the number of configurations in the list.

The remaining part of the determinantset block is the definition of jastrow factor. Any number of these can be defined. Figure 19.6 shows a sample jastrow block including one-, two- and three-body terms. This is the standard block produced by convert4qmc with the option -add3BodyJ (this particular example is for a water molecule). Optimization of individual radial functions can

be turned on/off using the “optimize” parameter. It can be added to any coefficients block, even though it is currently not present in the J1 and J2 blocks.

Listing 19.6: ”Sample Jastrow XML block.”

```
<jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation rcut="10" size="10" speciesA="u" speciesB="u">
        <coefficients id="uu" type="Array">0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0</coefficients>
      </correlation>
      <correlation rcut="10" size="10" speciesA="u" speciesB="d">
        <coefficients id="ud" type="Array">0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0</coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" source="ion0" print="yes">
      <correlation rcut="10" size="10" cusp="0" elementType="O">
        <coefficients id="eO" type="Array">0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0</coefficients>
      </correlation>
      <correlation rcut="10" size="10" cusp="0" elementType="H">
        <coefficients id="eH" type="Array">0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   0.0</coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J3" type="eeI" function="polynomial" source="ion0" print="yes">
      <correlation ispecies="O" especies="u" isize="3" esize="3" rcut="10">
        <coefficients id="uuO" type="Array" optimize="yes">
        </coefficients>
      </correlation>
      <correlation ispecies="O" especies1="u" especies2="d" isize="3" esize="3"
   rcut="10">
        <coefficients id="udO" type="Array" optimize="yes">
        </coefficients>
      </correlation>
      <correlation ispecies="H" especies="u" isize="3" esize="3" rcut="10">
        <coefficients id="uuH" type="Array" optimize="yes">
        </coefficients>
      </correlation>
      <correlation ispecies="H" especies1="u" especies2="d" isize="3" esize="3"
   rcut="10">
        <coefficients id="udH" type="Array" optimize="yes">
        </coefficients>
      </correlation>
      </jastrow>
```

This training assumes basic familiarity with the UNIX operating system. In particular, we use simple scripts written in “csh”. In addition, we assume that the student has obtained all the necessary files and executables, and that the location of the training files are located at ${TRAINING TOP}.

The goal of the training not only to familiarize the student with the execution and options in QMCPACK, but also to introduce him/her to important concepts in quantum Monte Carlo calculations and many-body electronic structure calculations.

Chapter 20

Lab 4: Condensed Matter Calculations

20.1 Topics covered in this Lab

- tiling DFT primitive cells into QMC supercells
- reducing finite size errors via extrapolation
- reducing finite size erors via averaging over twisted boundary conditions
- using the B-spline mesh factor to reduce memory requirements
- using a coarsely resolved vacuum buffer region to reduce memory requirements
- calculating the DMC total energies of representative 2D and 3D extended systems

20.2 Lab directories and files

```
labs/lab4_condensed_matter/
├── Be-2at-setup.py           - DFT only for prim to conv cell
├── Be-2at-qmc.py             - QMC only for prim to conv cell
├── Be-16at-qmc.py            - DFT and QMC for prim to 16 atom cell
├── graphene-setup.py         - DFT and OPT for graphene
├── graphene-loop-mesh.py     - VMC scan over orbital bspline mesh factors
├── graphene-final.py         - DMC for final meshfactor
└── pseudopotentials          - pseudopotential directory
    ├── Be.ncpp                 - Be PP for Quantum ESPRESSO
    ├── Be.xml                  - Be PP for QMCPACK
    ├── C.BFD.upf               - C  PP for Quantum ESPRESSO
    └── C.BFD.xml               - C  PP for QMCPACK
```

The goal of this lab will be to introduce you to the somewhat specialized problems involved in performing diffusion Monte Carlo calculations on condensed matter as opposed to the atoms and molecules that were the focus of earlier labs. Calculations will be performed on two different systems. Firstly, we will perform a series of calculations on BCC beryllium focusing on the necessary methodology to limit finite size effects. Secondly, we will perform calculations on graphene as an example of a system where qmcpack's ability to handle cases with mixed periodic and open

boundary conditions is useful. This example will also focus on strategies to limit memory usage for such systems. All of the calculations performed in this lab will utilize the Nexus workflow management system that vastly simplifies the process by automating the steps of generating trial wavefunctions and performing DMC calculations.

20.3 Preliminaries

For any DMC calculation, we must start with a trial wavefunction. As is typical for our calculations of condensed matter, we will produce this wavefunction using density functional theory. Specifically, we will use quantum espresso to generate a slater determinant of single particle orbitals. This is done as a three step process. First, we calculate the converged charge density by performing a DFT calculation with a fine grid of k-points to fully sample the Brilloiun zone. Next, a non-self consistent calculation is performed at the specific k-points needed for the supercell and twists needed in the DMC calculation (more on this later). Finally, a wavefunction is converted from the binary representation used by quantum espresso to the portable hdf5 representation used by qmcpack.

The choice of k-points necessary to generate the wavefunctions dependes on both the supercell chosen for the DMC calculation and by the supercell twist vectors needed. Recall that the wavefunction in a plane wave DFT calculation is written using Bloch's theorem as:

$$\Psi(\vec{r}) = e^{i\vec{k}\cdot\vec{r}}u(\vec{r}) \tag{20.1}$$

Where $\vec{k}$ is confined ot the first Brillouin zone of the cell chosen and $u(\vec{r})$ is periodic in this simulation cell. A plane wave DFT calculation stores the periodic part of the wavefunction as a linear combination of plane waves for each single particle orbital at all k-points selected. The symmetry of the system allows us to generate an arbitrary supercell of the primitive cell as follows: Consider the set of primitive lattice vectors, $\{\mathbf{a}_1^p, \mathbf{a}_2^p, \mathbf{a}_3^p\}$. We may write these vectors in a matrix, $\mathbf{L}_p$, whose rows are the primitive lattice vectors. Consider a non-singular matrix of integers, $\mathbf{S}$. A corresponding set of supercell lattice vectors, $\{\mathbf{a}_1^s, \mathbf{a}_2^s, \mathbf{a}_3^s\}$, can be constructed by the matrix product

$$\mathbf{a}_i^s = S_{ij}\mathbf{a}_j^p \tag{20.2}$$

If the primitive cell contains N_p atoms, the supercell will then contain $N_s = |\det(\mathbf{S})|N_p$ atoms.

Now, the wavefunciton at any point in this new supercell can be related to the wavefunction in the primitive cell by finding the linear combination of primitive lattice vectors that maps this point back to the primitive cell:

$$\vec{r}' = \vec{r} + x\mathbf{a}_1^p + y\mathbf{a}_2^p + z\mathbf{a}_3^p = \vec{r} + \vec{T} \tag{20.3}$$

where x, y, z are integers. Now the wavefunction in the supercell at point $\vec{r}'$ can be written in terms of the wavefunction in the primitive cell at $\vec{r}'$ as:

$$\Psi(\vec{r}) = \Psi(\vec{r}')e^{i\vec{T}\cdot\vec{k}} \tag{20.4}$$

where $\vec{k}$ is confined to the first Brillouin zone of the primitive cell. We have also chosen the supercell twist vector which places a constraint on the form of the wavefunction in the supercell. The combination of these two constraints allows us to identify family of N k-points in the primitive cell that satisfy the constraints. Thus for a given supercell tiling matrix and twist angle, we can write the wavefunction everywhere in the supercell by knowing the wavefunction a N k-points in the primitive cell. This means that the memory necessary to store the wavefunction in a supercell is only linear in the size of the supercell rather than the quadratic cost if symmetry were neglected.

20.4 Total energy of BCC beryllium

As was discussed in this morning's lectures when performing calculations of periodic solids with QMC, it is essential to work with a reasonable size supercell rather than the primitive cells that are common in mean field calculations. Specifically, all of the finite size correction schemes discussed in the morning require that the exchange-correlation hole be considerably smaller than the periodic simulation cell. Additionally, finite size effects are lessened as the distance between the electrons in the cell and their periodic images increases, so it is advantageous to generate supercells that are as spherical as possible so as to maximize this distance. However, there is a competing consideration in that for calculating total energies we often want to be able to extrapolate the energy per particle to the thermodynamic limit by means of the following formula in 3 dimensions:

$$E_{\text{inf}} = C + E_N/N \tag{20.5}$$

This formula derived assuming the shape of the supercells is consistent (more specifically that the periodic distances scale uniformly with system size), meaning we will need to do a uniform tiling, ie, 2x2x2, 3x3x3 etc. As a 3x3x3 tiling is 27 times larger than the supercell and the practical limit of DMC is on the order of 200 atoms (depending on Z), sometimes it is advantageous to choose a less spherical supercell with fewer atoms rather than a more spherical one that is too expensive to tile.

In the case of a BCC crystal, it is possible to tile the one atom primitive cell to a cubic supercell by only doubling the number of electrons. This is the best possible combination of a small number of atoms that can be tiled and a regular box that maximizes the distance between periodic images. We will need to determine the tiling matrix S that generates this cubic supercell by solving the following equation for the coefficients of the S matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \tag{20.6}$$

We will now use Nexus to generate the trial wavefunction for this BCC beryllium.

Fortunately, the Nexus will handle determination of the proper k-vectors given the tiling matrix. All that is needed is to place the tiling matrix in the Be-2at-setup.py file. Now the definition of the physical system is:

```
bcc_Be = generate_physical_system(
    lattice    = 'cubic',
    cell       = 'primitive',
    centering  = 'I',
    atoms      = 'Be',
    constants  = 3.490,
    units      = 'A',
    net_charge = 0,
    net_spin   = 0,
    Be         = 2,
    tiling     = [[a,b,c],[d,e,f],[g,h,i]],
    kgrid      = kgrid,
    kshift     = (.5,.5,.5)
    )
```

Where the tiling line should be replaced with the row major tiling matrix from above. This script file will now perform a converged DFT calculation to generate the charge density in a directory

called bcc-beryllium/scf and perform a non self consistend DFT calculation to generate single particle orbitals in the directory bcc-beryllium/nscf. Fortunately, Nexus will calculate the required k-points needed to tile the wavefunction to the supercell, so all that is necessary is the granularity of the supercell twists and whether this grid is shifted from the origin. Once this is finished, it performs the conversion from pwscf's binary format to the hdf5 format used by qmcpack. Finally, it will optimize the coefficients of one-body and two-body jastrow factors in the supercell defined by the tiling matrix.

Run these calculations by executing the script Be-2at-setup.py. You will notice that such small calcuations as are required to generate the wavefunction of Be in a one atom cell are rather inefficient to run on a high performance computer such as vesta in terms of the time spent doing calculations versus time waiting on the scheduler and booting compute nodes. One of the benefits of the portable hdf format that is used by qmcpack is that you can generate data like wavefunctions on a local workstation or other convenient resource and only use high performance clusters for the more expensive QMC calculations.

In this case, the wavefunction is generated in the directory bcc-beryllium/nscf-2at_222/pwscf_output in a file called pwscf.pwscf.h5. It can be useful for debugging purposes to be able to verify the contents of this file are what you expect. For instance, you can use the tool h5ls to check the geometry of the cell where the dft calculations were performed, or number of k-points or electrons in the calculation. This is done with the command: h5ls -d pwscf.pwscf.h5/supercell or h5ls -d pwscf.pwscf.h5/electrons.

In the course of running Be-2at-setup.py, you will get an error when attempting to perform the vmc and wavefunction optimization calculations. This is due to the fact that the wavefunction has been generated supercell twists of the form (+/- 1/4, +/- 1/4, +/- 1/4). In the case that the supercell twist contains only 0 or 1/2, it is possible to operate entirely with real arithmetic. The executable that has been indicated in Be-2at-setup.py has been compiled for this case. Note that where this is possible, the memory usage is a factor of two less than the general case and the calculations are somewhat faster. However, it is often necessary to perform calculations away from these special twist angles in order to reduce finite size effects. To fix this, delete the directory bcc-beryllium/opt-2at, change the line in near the top of Be-2at-setup.py from

```
qmcpack    = '/soft/applications/qmcpack/Binaries/qmcpack'
```

to

```
qmcpack    = '/soft/applications/qmcpack/Binaries/qmcpack_comp'
```

and rerun the script.

When the optimiztion calculation has finished, check that everything as proceeded correctly by looking at the output in the opt-2at directory. Firstly, you can grep the output file for Delta to see if the cost function has indeed been decreasing during the optimization. You should find something like:

```
OldCost: 4.8789147e-02 NewCost: 4.0695360e-02 Delta Cost:-8.0937871e-03
OldCost: 3.8507795e-02 NewCost: 3.8338486e-02 Delta Cost:-1.6930674e-04
OldCost: 4.1079105e-02 NewCost: 4.0898345e-02 Delta Cost:-1.8076319e-04
OldCost: 4.2681333e-02 NewCost: 4.2356598e-02 Delta Cost:-3.2473514e-04
OldCost: 3.9168577e-02 NewCost: 3.8552883e-02 Delta Cost:-6.1569350e-04
OldCost: 4.2176276e-02 NewCost: 4.2083371e-02 Delta Cost:-9.2903058e-05
OldCost: 4.3977361e-02 NewCost: 4.2865751e-02 Delta Cost:-1.11161830-03
OldCost: 4.1420944e-02 NewCost: 4.0779569e-02 Delta Cost:-6.4137501e-04
```

Which shows that the starting wavefunction was fairly good and that most of the optimizaiton occurred in the first step. Confirm this by using qmca to look at how the energy and variance

changed over the course of the calculation with the comand: qmca -q ev -e 10 *.scalar.dat executed in the opt-2at directory. You should get output like the following:

```
                 LocalEnergy              Variance              ratio
opt  series 0  -2.159139 +/- 0.001897   0.047343 +/- 0.000758   0.0219
opt  series 1  -2.163752 +/- 0.001305   0.039389 +/- 0.000666   0.0182
opt  series 2  -2.160913 +/- 0.001347   0.040879 +/- 0.000682   0.0189
opt  series 3  -2.162043 +/- 0.001223   0.041183 +/- 0.001250   0.0190
opt  series 4  -2.162441 +/- 0.000865   0.039597 +/- 0.000342   0.0183
opt  series 5  -2.161287 +/- 0.000732   0.039954 +/- 0.000498   0.0185
opt  series 6  -2.163458 +/- 0.000973   0.044431 +/- 0.003583   0.0205
opt  series 7  -2.163495 +/- 0.001027   0.040783 +/- 0.000413   0.0189
```

Now that the optimization has completed successfully, we can perform dmc calculations. The first goal of the calculations will be to try to eliminate the one body finite size effects by twist averaging. The script Be-2at-qmc.py has the necessary input. Note on line 42 two twist grids are specified, (2,2,2) and (3,3,3). Change the tiling matrix in this input file as in Be-2at-qmc.py and start the calculations. Note that this workflow takes advantage of qmcpack's ability to group jobs. If you look in the directory dmc-2at_222 at the job submission script, (dmc.qsub.in) you will note that rather than operating on an xml input file, qmcapp is targeting a text file called dmc.in. This file is a simple text file that contains the names of the 8 xml input files needed for this job, one for each twist. When operated in this mode, qmcpack will use mpi groups to run multiple copies of itself within the same mpi context. This is often useful both in terms of organizing calculations and also for taking advantage of the large job sizes that computer centers often encourage.

The dmc calculations in this case are designed to complete in a few minutes. When they have finished running, first look at the scalar.dat files corresponding to the dmc calculations at the various twists in dmc-2at_222. Using a command like 'qmca -q ev -e 32 *.s001.scalar.dat' (with a suitably chosen number of blocks for the equilibration), you will see that the dmc energy in each calcuation is nearly identical within the statistical uncertainty of the calculations. In the case of a large supercell, this is often indicative of a situation where the Brilloiun zone is so small that the one body finite size effects are nearly converged without any twist averaging. In this case, however, this is because of the symmetry of the system. For this cubic supercell, all of the twist angles chosen in this shifted 2x2x2 grid are equivalent by symmetry. In the case where substantial resources are required to equilibrate the dmc calculations, it can be beneficial to avoid repeating such twists and instead simply weight them properly. In this case however where the equilibration is inexpensive, there is no benefit to adding such complexity as the calculations can simply be averaged together and the result is equivalent to performing a single longer calcuation.

Using the command qmc -a -q ev -e 16 *.s001.scalar.dat, average the dmc energies in dmc-2at_222 and dmc-2at_333 to see whether the one body finite size effects are converged with a 3x3x3 grid of twists. As beryllium as a metal, the convergence is quite poor (0.025 Ha / Be or 0.7 eV / Be). If this were a production calculation it would be necessary to perform calculations on much larger grids of supercell twists to eliminate the one body finite size effects.

In this case there are several other calculations that would warrant a high priority. A script Be-16at-qmc.py has been provided where you can imput the appropriate tiling matrix for a 16 atom cell and perform calculations to estimate the two body finite size effects which will also be quite large in the 2 atom calculations. This script will take approximately 30 minutes to run to completion, so depending on interest, you can either run it, or also work to modify the scripts to address the other technical issues that would be necessary for a production calculation such as calculating the population bias or the timestep error in the dmc calculations.

Another useful exercise would be to attempt to validate this pseudopotential by calculating the

ionization potential and electron affinity of the isolated atom and comparing to the experimental values: IP = 9.3227 eV , EA = 2.4 eV.

20.5 Handling a 2D system: graphene

In this section we will examine a calculation of an isolated sheet of graphene. As graphene is a two dimensional system, we will take advantage of qmcpack's ability to mix periodic and open boundary conditions to eliminate and spurious interaction of the sheet with its images in the z direction. Run the script graphene-setup.py which will generate the wavefunction and optimize one and two body jastrow factors. In the script, notice line 160: bconds = 'ppn' in the generate_qmcpack function which specifies this mix of open and periodic boundary conditions. As a consequence of this, the atoms will need to be kept away from this open boundary in the z direction as the electronic wavefunction will not be defined outside of the simulation box in this direction. For this reason, all of the atom positions in at the beginning of the file have z coordinates 7.5. At this point, run the script graphene-setup.py.

Aside from the change in boundary conditions, the main thing that distinguished this kind of calculation from the beryllium example above is the large amount of vacuum in the cell. While this is a very small calculation designed to run quickly in the tutorial, in general a more converged calculation would quickly become memory limited on an architecture like BG/Q. When the initial wavefunciton optimizaiton has completed to your satisfaction, run the script graphene-loop-mesh.py. This examines within variational Monte Carlo an approache to reducing the memory required to store the wavefunction. In graphene-loop-mesh.py, the spacing between the b-spline points is varied uniformly. The mesh spacing is a prefactor to the linear spacing between the spline points, so the memory usage goes as the cube of the meshfactor. When you run the calculations, examine the .s000.scalar.dat files with qmca to determine the lowest possible mesh spacing that preserves both the vmc energy and the variance.

Finally, edit the file graphene-final.py which will perform two DMC calculations. In the first, (qmc1) replace the following lines:

```
meshfactor   = xxx,
precision    = '---',
```

using the values you have determined to perform the calculation with as small as possible of wavefunction. Note that we can also use single precision arithmetic to store the wavefunction by specifying precision='single'. When you run the script, compare the output of the two DMC calculations in terms of energy and variance. Also see if you can calculate the fraction of memory that you were able to save by using a meshfactor other than 1 and single precision arithmetic.

20.6 Conclusion

Upon completion of this lab, you should be able to use Nexus to perform DMC calculations on periodic solids when provided with a pseudopotential. You should also be able to reduce the size of the wavefunction in a solid state calculation in cases where memory is a limiting factor.

Chapter 21

Lab 5: Excited State Calculations

21.1 Topics covered in this Lab

- Tiling DFT primitive cells into optimal QMC supercells
- Fundamentals of between neutral and charged calculations
- Calculating quasiparticle excitation energies of condensed matter systems
- Calculating optical excitation energies of condensed matter systems

21.2 Lab directories and files

```
labs/lab5_excited_properties/
├── band.py           - Band structure calculation for Carbon Diamond
├── optical.py        - VMC optical gap calculation using the tiling matrix from band.py
├── quasiparticle.py  - VMC quasiparticle gap calculation using the tiling matrix from band.py
└── pseudopotentials      - pseudopotential directory
    ├── C.BFD.upf         - C PP for Quantum ESPRESSO
    └── C.BFD.xml         - C PP for QMCPACK
```

The goal of this lab is to perform neutral and charged excitation calculations in condensed matter systems using QMCPACK. Throughout this lab, a working knowledge of *Lab4 Condensed Matter Calculations* is assumed. First, we will introduce the concepts of neutral and charged excitations. We will briefly discuss these in relation to the specific experimental studies that must be used to benchmark DMC results. Secondly, we will perform charged (quasiparticle) and neutral (optical) excitations calculations on C-diamond.

21.3 Basics and excited state experiments

Although VMC and DMC methods are better suited for studying ground state properties of materials, they can still provide useful information regarding the excited states. Unlike the applications of band structure theory such as DFT and GW, it is more challenging to obtain the complete excitation spectra using DMC. However, it is relatively straightforward to calculate the band gap minimum of a condensed matter system using DMC.

We will briefly discuss the two main ways of obtaining the band gap minimum through experiments: photoemission and absorption studies. The energy required to remove an electron from a

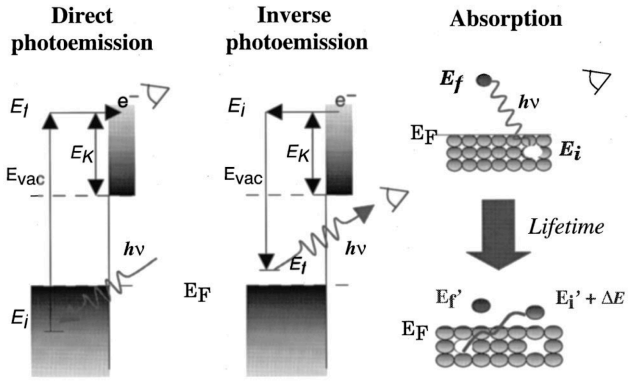


Figure 21.1: Direct and inverse photoemission experiments involve charged excitations, whereas optical absorption experiments involves excitation that are just enough to be excited to the conduction band. From ref. [42]

neutral system is called the ionization potential (IP), which is available from direct photoemission experiments. In contrast, the emission energy of a negatively charged system (or the energy required to convert a negatively charged system to a neutral system) known as electron affinity (EA) and it is available from inverse photoemission experiments. Outline of these experiments are shown in Fig. 21.1.

Following the explanation in the previous paragraph and Fig. 21.1, the *quasiparticle* band gap of a material can be defined as:

$$E_g = EA - IP = (E_{N+1}^{CBM} - E_N^{K'}) - (E_N^{K'} - E_{N-1}^{VBM}) = E_{N+1}^{CBM} + E_{N-1}^{VBM} - 2 * E_N^{K'} \tag{21.1}$$

where N is the number of electrons in the neutral system and E_N is the ground state energy of the neutral system. CBM and VBM stand for the conduction band minimum and valence band maximum, respectively. K' can formally be arbitrary at the infinite limit. However, in practical calculations, a supertwist which accommodates both CBM and VBM can be more efficient in terms of computational time and systematic finite size error cancellation. In the literature, the quasiparticle gap is also called the electronic gap. The term electronic comes from the fact that in both photoemission experiments, it is assumed that the perturbed electron is non-interacting with the sample.

Additionally, one can also perform absorption experiments where electrons are perturbed at relatively lower energies, just enough to be excited into the conduction band. In absorption experiments, electrons are perturbed at lower energies. Therefore, they are not completely free and the system is still considered neutral. Since a *quasihole* and *quasielectron* are formed simultaneously, it creates a bound state, unlike the free electron in the quasiparticle gap as described above. This process is also known as *optical* excitation, which is schematically shown in Fig. 21.1, under "Absorption". The optical gap can be formulated as follows:

$$E_g^{K_1 \to K_2} = E^{K_1 \to K_2} - E_0 \tag{21.2}$$

where $E^{K_1 \to K_2}$ is the energy of the system when a valence electron at wavevector K_1 is promoted to the conduction band at wavevector K_2. Therefore, the $E_g^{K_1 \to K_2}$ is called the optical gap for

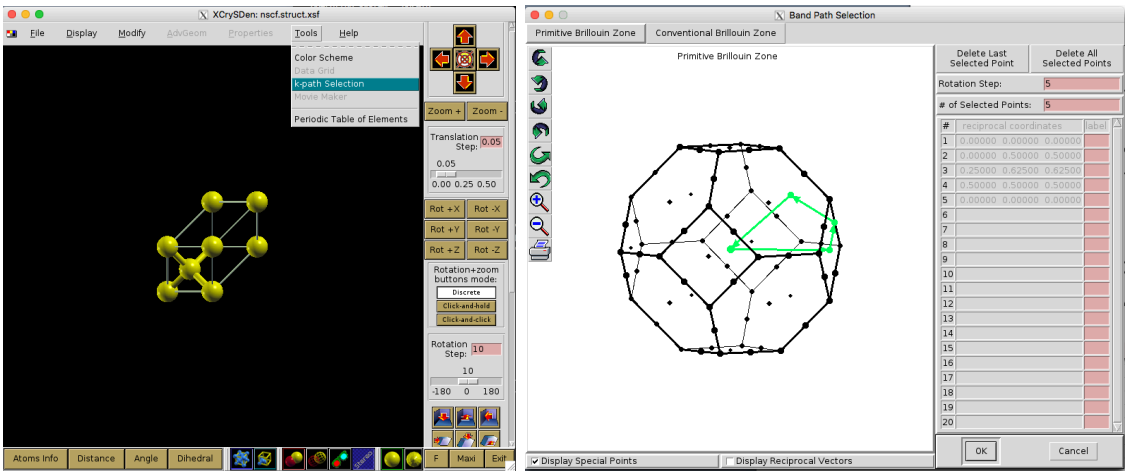


Figure 21.2: Visualizing the Brillouin Zone using XCRYSDEN.

promoting an electron at K_1 to K_2. If both CBM and VBM are on the same k-vector then the material is called direct band gap, since it can directly emit photons without any external perturbation (phonons). However, if CBM and VBM share different k-vectors, then the photon emitting electron has to transfer some of its momenta to the crystal lattice and then decay to the ground state. As this process involves an intermediate step, this property is called the indirect band gap. Difference between the optical and electronic band gaps are called the exciton binding energy. Exciton binding energy is very important for optoelectronic applications such as lasers. Since the recombination usually occurs between free holes and free electrons, a bound electron and hole state means that the spectrum of emission energies will be narrower. In the examples that follow, we will investigate the optical excitations of C-diamond.

21.4 Preparation for the excited state calculations

In this section, we will study the preparation steps to perform excited state calculations with quantum Monte Carlo. Here, the most basic steps are listed in the implementation order:

1. Identify the high symmetry k-points of the standardized primitive cell
2. Perform DFT band structure calculation along high symmetry paths
3. Find a supertwist which includes all the k-points of interest
4. Identify the indexing of k-points in the supertwist to be used in QMCPACK

21.4.1 Identifying high-symmetry k-points

Primitive cell is the most basic, non-unique repeat unit of a crystal in the real space. However, the translations of the repeat unit, the Bravais lattice is unique for each crystal, and can be represented using discrete translation operations, R_n:

$$\mathbf{R_n} = n_1\mathbf{a_1} + n_2\mathbf{a_2} + n_3\mathbf{a_3} \tag{21.3}$$

a_n are the real space lattice vectors in three dimensions. Thanks to the periodicity of the Bravais lattice, a crystal can also be represented using periodic functions in the reciprocal space:

$$f(\mathbf{R_n} + \mathbf{r}) = \sum_m f_m e^{iG_m(\mathbf{R_n}+\mathbf{r})} \tag{21.4}$$

where G_m are called as the reciprocal lattice vectors. Equation 21.4 also satisfies the equality $G_m \cdot R_n = 2\pi N$. High-symmetry structures can be represented using a subspace of the BZ, which is called as the irreducible Brillouin Zone (iBZ). If we choose series of paths of high-symmetry k-points which encapsulates the iBZ, we can determine the band gap and electronic structure of the material. For more discussion, please refer to any solid state physics textbook.

There are multiple practical ways to find the high-symmetry k-point path. For example, one can use pymatgen, [43] XCRYSDEN [44] or SeeK-path [45]. Figure 21.2 shows the procedure for visualizing the Brillouin Zone using XCRYSDEN after the structure file is loaded. However, the primitive cell is not unique, and the actual shape of the BZ can depend on the structure used. In our example, we use the python libraries of SeeK-path, using a wrapper written in Nexus.

SeeK-path includes routines to standardize primitive cells, which will be useful for our work.

SeeK-path can be installed easily using `pip`:

```
>pip install --user seekpath
```

In the `band.py` script, identification of high symmetry k-points and band structure calculations are done within the workflow. In the script, where the `dia` PhysicalSystem object is used as the input structure, `dia2_structure` is the standardized primitive cell and `dia2_kpath` is the respective k-path around the iBZ. `dia2_kpath` has a dictionary of the k-path in various coordinate systems, please make sure you are using the right one.

```
from structure import get_primitive_cell, get_kpath
dia2_structure   = get_primitive_cell(structure=dia.structure)['structure']
dia2_kpath       = get_kpath(structure=dia2_structure)
```

21.4.2 DFT band structure calculation along high symmetry paths

After the high-symmetry kpoints are identified, one can perform band structure calculations in DFT. For an insulating structure, DFT can provide VBM and CBM wavevectors which would be of interest to the DMC calculations. However, if available, CBM and VBM from DFT would need to be compared to the experiments. Basically, `band.py` will:

1. Perform an SCF calculation in QE using a high density reciprocal grid.
2. Identifies the high-symmetry k-points on the iBZ and provides a k-path.
3. Perform a 'band' calculation in QE explicitly writing all the k-points on the path. (Make sure to add extra unoccupied bands)
4. Plot the band structure curves and the location of VBM/CBM if available.

In Fig. 21.3, C-diamond is shown to have an indirect band gap between the red and green dots (CBM and VBM respectively). VBM is located at Γ. CBM is not located on a high symmetry k-point in this case. Therefore, we can use the symbol Δ to denote the CBM wavevector in the rest of this document. In `band.py` script, once the band structure calculation is finished, you can use the following lines to get the exact location of VBM and CBM using:

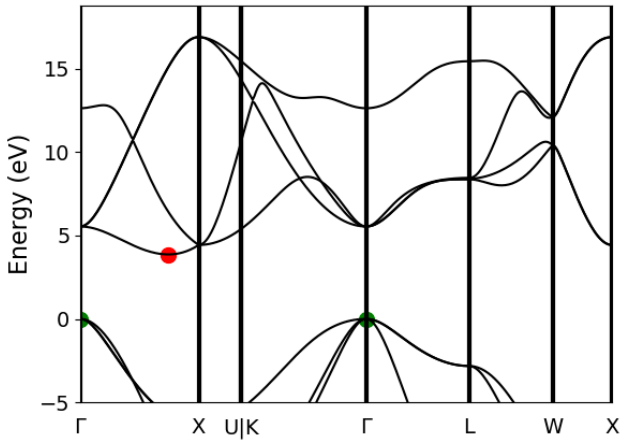


Figure 21.3: Band structure calculation of C-diamond performed at DFT-LDA level. Conduction band minimum (CBM) are shown with red points, and the valence band maximum(VBM) are shown with the green points both at Γ. DFT-LDA calculations suggest that the material has an indirect band gap from $\Gamma \rightarrow \Delta$. However, $\Gamma \rightarrow \Gamma$ transition can also be investigated for more complete check.

```
p = band.load_analyzer_image()
print "VBM:\n{0}".format(p.bands.vbm)
print "CBM:\n{0}".format(p.bands.cbm)
```

Output must be the following:

```
VBM:
  band_number     = 3
  energy          = 13.2874
  index           = 0
  kpoint_2pi_alat = [0. 0. 0.]
  kpoint_rel      = [0. 0. 0.]
  pol             = up

CBM:
  band_number     = 4
  energy          = 17.1545
  index           = 51
  kpoint_2pi_alat = [0.        0.1095605 0.       ]
  kpoint_rel      = [0.3695652 0.        0.3695652]
  pol             = up
```

21.4.3 Finding a supertwist which includes all the k-points of interest

Using the VBM and CBM wavevectors defined in the previous section, we now construct the supertwist which will hopefully contain both VBM and CBM. In Fig. 21.4, we provide a simple example using 2D rectangular lattice. Let us assume that we are interested in the indirect transition, $\Gamma \rightarrow X_1$. In Fig. 21.4a, the first BZ of the primitive cell is shown as the square centered on Γ, which is drawn using dashed lines. Due to the periodicity of the lattice, this primitive cell BZ repeats itself with spacings equal to the reciprocal lattice vectors: $(2\pi/a, 0)$ and $(0, 2\pi/a)$ (or (1,0) and (0,1) in crystal coordinates). We are interested in the first BZ, where X_1 is at (0,0.5). In Fig. 21.4b, the

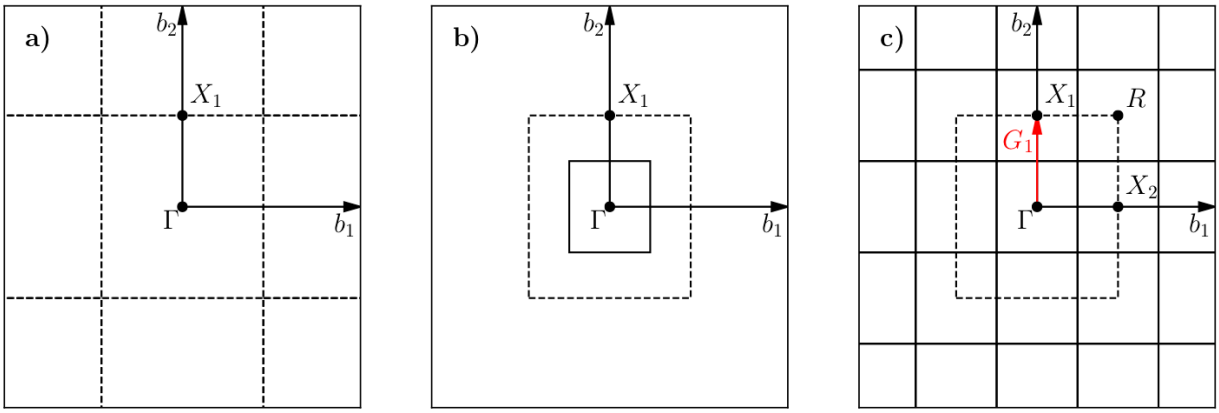


Figure 21.4: a) First Brillouin Zone (BZ) of the primitive cell centered on Γ. Dashed lines indicate zone boundaries. b) First BZ of the 2x2 supercell inside the first BZ of the primitive cell. First BZ boundaries of the supercell are shown using solid lines. c) Periodic translations of the first BZ of the supercell showing that Γ and X_1 are periodic images of each other given the supercell BZ.

first BZ of the 2x2 supercell is the smaller square, drawn using solid lines. In Fig. 21.4c, the BZ of the 2x2 supercell also repeats in the space, similar to Fig. 21.4a. Therefore, in the 2x2 supercell, X_1, X_2 and R are only the periodic images of Γ. 2x2 supercell calculation can be performed in reciprocal space using [2,2] tiling matrix. Therefore, individual kpoints (twists) of the primitive cell are combined in the supercell calculation, which are then called as supertwists. In more complex primitive cell (hence BZ), more general criteria would be constructing a set of supercell reciprocal lattice vectors which contains the $\Gamma \rightarrow X_1$ (e.g. G_1 in Fig. 21.4) vector within their convex hull. Under this constraint, Wigner-Seitz radius of the simulation cell can be maximized to in an effort to reduce finite size errors.

For the case of the indirect band gap in Diamond, one may need to deal with using several approximations to generate a supertwist which corresponds to a reasonable simulation cell. Δ in Diamond band gap is at `[0.3695653, 0., 0.3695653]`. In your calculations, the Δ wavevector and the eigenvalues you find can be slightly different in value. Closest simple fraction to this number with the smallest denominator is 1/3. If we use $\Delta' = [1/3, 0., 1/3]$, we could use 3x1x3 supercell as the simple choice and include both Δ' and Γ in the same supertwist exactly. Near Δ, the LDA band curvature is very low and using Δ' can indeed be a good approximation. We can compare the eigenvalues using their index numbers:

```
>>> print p.bands.up[51] ## CBM, Δ ##
  eigs            = [-3.2076  4.9221  7.5433  7.5433 17.1545 19.7598 28.3242 28.3242]
  index           = 51
  kpoint_2pi_alat = [0.        0.1095605 0.       ]
  kpoint_rel      = [0.3695652 0.        0.3695652]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
>>> print p.bands.up[46] ## Δ' ##
  eigs            = [-4.0953  6.1376  7.9247  7.9247 17.1972 20.6393 27.3653 27.3653]
  index           = 46
  kpoint_2pi_alat = [0.        0.0988193 0.       ]
  kpoint_rel      = [0.3333333 0.        0.3333333]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
```

This shows that the eigenvalues of the first unoccupied bands in Δ and Δ' are 17.1545 and 17.1972 eV respectively, meaning that according to LDA, a correction of nearly -40 meV is obtained. After electronic transitions between Γ and Δ' are studied using DMC, one can apply the LDA correction to extrapolate the results to Γ and Δ transitions.

21.4.4 Identifying the indexing of k-points of interest in the supertwist

At this stage, we must have performed *scf* calculation using a converged k-point grid and then an *nscf* calculation using the supertwist kpoints given above. We will be using the orbitals from neutral DFT calculations, therefore we need to explicitly define the band and twist indexes of the excitations in QMCPACK (e.g. in order to define electron promotion). In C-diamond, we can give an example by finding the band and twist indexes of Γ and Δ'. For this end, one can run a mock VMC calculation and read `einspline.tile_300010003 .spin_0.tw_0.g0.bandinfo.dat` file. Einspline file prints out the eigenstates information from DFT calculations. Therefore, we can obtain the band and the state index from this file, which can later be used to define the electron promotion. Below, you can see an explanation of how the band and twist indexes are defined using a portion of the `einspline.tile_300010003.spin_0.tw_0.g0.bandinfo.dat` file. Spin_0 in the file name suggests that we are reading the spin up eigenstates. Band, state, twistindex and bandindex numbers all start from zero. We know that we have 72 electrons in the simulation cell, where 36 of them are spin-up polarized. Since state number starts from 0, state number 35 must be occupied while state 36 should be unoccupied. States 35 and 36 have the same reciprocal crystal coordinates (K1,K2,K3) as Γ and Δ', respectively. Therefore, one should promote an electron from state number 35 to 36 to study the indirect band gap here.

```
#  Band State TwistIndex BandIndex Energy Kx Ky Kz K1 K2 K3 KmK
33 33 0  1     0.488302  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000      1
34 34 0  2     0.488302  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000      1
35 35 0  3     0.488302  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000      1
36 36 4  4     0.631985  0.0000 -0.6209  0.0000 -0.3333 -0.0000 -0.3333      1
37 37 8  4     0.631985  0.0000 -1.2418  0.0000 -0.6667 -0.0000 -0.6667      1
38 38 0  4     0.691907  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000      1
```

However, one should always check whether this is really what we want. It can be seen that band # 33, 34 and 35 are degenerate (energy eigenvalues are listed in the 5th column), but also they have the same reciprocal coordinates in (K1,K2,K3). This is actually expected as one can see from Fig. 21.3, in the band diagram the band structure is threefold degenerate at Γ. Here, we can choose the state with the largest band index: (0,3). Following the (twistindex, bandindex) notation, we can say that Γ to Δ' transition can be defined as from (0,3) to (4,4).

Alternatively, one can also read the band and twist indexes using PwscfAnalyzer and determine the band/twist indexes on the go:

```
p = nscf.load_analyzer_image()
print 'band information'
print p.bands.up
print 'twist 0 k-point:',p.bands.up[0].kpoint_rel
print 'twist 4 k-point:',p.bands.up[4].kpoint_rel
print 'twist 0 band 3 eigenvalue:',p.bands.up[0].eigs[3]
print 'twist 4 band 4 eigenvalue:',p.bands.up[4].eigs[4]
```

Giving output:

```
  0
    eigs            = [-8.0883 13.2874 13.2874 13.2874 18.8277 18.8277 18.8277
    25.9151]
```

```
  index           = 0
  kpoint_2pi_alat = [0. 0. 0.]
  kpoint_rel      = [0. 0. 0.]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
1
  eigs            = [-5.0893  3.8761 10.9518 10.9518 21.5031 21.5031 21.5361
  28.2574]
  index           = 1
  kpoint_2pi_alat = [-0.0494096  0.0494096  0.0494096]
  kpoint_rel      = [0.3333333 0.        0.       ]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
2
  eigs            = [-5.0893  3.8761 10.9518 10.9518 21.5031 21.5031 21.5361
  28.2574]
  index           = 2
  kpoint_2pi_alat = [-0.0988193  0.0988193  0.0988193]
  kpoint_rel      = [0.6666667 0.        0.       ]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
3
  eigs            = [-5.0893  3.8761 10.9518 10.9518 21.5031 21.5031 21.5361
  28.2574]
  index           = 3
  kpoint_2pi_alat = [ 0.0494096  0.0494096 -0.0494096]
  kpoint_rel      = [0.        0.        0.3333333]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
4
  eigs            = [-4.0954  6.1375  7.9247  7.9247 17.1972 20.6393 27.3652
  27.3652]
  index           = 4
  kpoint_2pi_alat = [0.        0.0988193 0.       ]
  kpoint_rel      = [0.3333333 0.        0.3333333]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
5
  eigs            = [-0.6681  2.3791  3.7836  8.5596 19.3423 26.2181 26.6666
  28.0506]
  index           = 5
  kpoint_2pi_alat = [-0.0494096  0.1482289  0.0494096]
  kpoint_rel      = [0.6666667 0.        0.3333333]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
6
  eigs            = [-5.0893  3.8761 10.9518 10.9518 21.5031 21.5031 21.5361
  28.2574]
  index           = 6
  kpoint_2pi_alat = [ 0.0988193  0.0988193 -0.0988193]
  kpoint_rel      = [0.        0.        0.6666667]
  occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
  pol             = up
7
  eigs            = [-0.6681  2.3791  3.7836  8.5596 19.3423 26.2181 26.6666
  28.0506]
  index           = 7
  kpoint_2pi_alat = [ 0.0494096  0.1482289 -0.0494096]
  kpoint_rel      = [0.3333333 0.        0.6666667]
```

```
    occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
    pol             = up
  8
    eigs            = [-4.0954  6.1375  7.9247  7.9247 17.1972 20.6393 27.3652
    27.3652]
    index           = 8
    kpoint_2pi_alat = [0.        0.1976385 0.       ]
    kpoint_rel      = [0.6666667 0.        0.6666667]
    occs            = [1. 1. 1. 1. 0. 0. 0. 0.]
    pol             = up

twist 0 k-point: [0. 0. 0.]
twist 4 k-point: [0.3333333 0.        0.3333333]
twist 0 band 3 eigenvalue: 13.2874
twist 4 band 4 eigenvalue: 17.1972
```

21.5 Quasiparticle (electronic) gap calculations

In quasiparticle calculations, it is essential to work with reasonably large sized supercells in order to avoid spurious "1/N effects". Since quasiparticle calculations involve charged cells, large simulation cells ensure that the extra charge is diluted over the simulation cell. Coulombic interactions are conditionally convergent for neutral periodic systems, but they are divergent for the charged systems. A typical workflow for a quasiparticle calculation includes:

1. SCF calculation in a neutral charged cell with QE using a high-density reciprocal grid.
2. Choose a tiling matrix which will at least approximately include VBM and CBM k-points.
3. 'nscf'/'p2q' calculations using the tiling matrix
4. VMC/DMC calculations for the neutral, positively and negatively charged cells in QMCPACK
5. Check the convergence of the quasiparticle gap with respect to the simulation cell size

```
<particleset name="e" random="yes">
  <group name="u" size="36" mass="1.0"> ##Change size to 35
    <parameter name="charge"              >    -1                   </parameter>
    <parameter name="mass"                >    1.0                  </parameter>
  </group>
...
...
<determinantset>
  <slaterdeterminant>
    <determinant id="updet" group="u" sposet="spo_u" size="36"> ##Change size to 35
      <occupation mode="ground" spindataset="0"/>
    </determinant>
    <determinant id="downdet" group="d" sposet="spo_d" size="36">
      <occupation mode="ground" spindataset="1"/>
    </determinant>
  </slaterdeterminant>
</determinantset>
```

Going back to equation 21.1, one can see that it is essential to include VBM and CBM wavevectors in the same twist for quasiparticle calculations as well. Therefore, the added electron will sit at CBM while the subtracted electron will be removed from VBM. However, for the charged cell

calculations, one may need to make changes in the input files for the fourth step. Alternatively, in `quasiparticle.py` file the changes in the qmc input are shown for negatively charged system:

```
qmc.input.simulation.qmcsystem.particlesets.e.groups.u.size +=1
(qmc.input.simulation.qmcsystem.wavefunction.determinantset
 .slaterdeterminant.determinants.updet.size += 1)
```

Here, the number of up electrons are increased by one (negatively charged system), and QMCPACK is instructed to read more one orbital in the up channel from the .h5 file.

QE uses symmetry in order to reduce the number of k-points required for the calculation. Therefore, all symmetry tags in QE (`nosym`, `noinv` and `nosym_evc`) must be set to false. An easy way to check whether this is the case is to see that all KmK values `einspline` files are equal to 1. Above, the input for the neutral cell is given, while the changes are denoted as comments for the positively charged cell. Notice that, we have used `det_format = "old"` in the `vmc_+/-e.py` files.

21.6 Optical gap calculations

Routines for the optical gap calculations are very similar to the quasiparticle gap calculations. The first three items in the quasiparticle band gap calculations can be reused for the optical gap calculations. However, at the VMC/DMC level, one should explicitly state the electronic transitions that are performed. Therefore, compared to the quasiparticle calculations, only the item number 4 is different for optical gap calculations. Here, the modified input file is given for the $\Gamma \rightarrow \Delta'$ transition, which can be compared to the ground state input file in the previous section.

```
<determinantset>
  <slaterdeterminant>
    <determinant id="updet" group="u" sposet="spo_u" size="36">
      <occupation mode="excited" spindataset="0" format="band" pairs="1" >
        0 3 4 4
      </occupation>
    </determinant>
    <determinant id="downdet" group="d" sposet="spo_d" size="36">
      <occupation mode="ground" spindataset="1"/>
    </determinant>
  </slaterdeterminant>
</determinantset>
```

We have used the (twistindex, bandindex) notation in the annihilaion/creation order for the up spin electrons. After resubmitting the batch job, in the output, you should be able to see the following lines in the `vmc.out` file:

```
Sorting the bands now:
  Occupying bands based on (ti,bi) data.
removing orbital 35
adding orbital 36
We will read 36 distinct orbitals.
There are 0 core states and 36 valence states.
```

And the `einspline.tile_300010003.spin_0.tw_0.g0.bandinfo.dat` file must be changed in the following way:

```
#  Band State TwistIndex BandIndex Energy Kx Ky Kz K1 K2 K3 KmK
33 33 0 1 0.499956      0.0000  0.0000 0.0000  0.0000 0.0000  0.0000 1
34 34 0 2 0.500126      0.0000  0.0000 0.0000  0.0000 0.0000  0.0000 1
35 35 4 4 0.637231      0.0000 -0.6209 0.0000 -0.3333 0.0000 -0.3333 1
36 36 0 3 0.502916      0.0000  0.0000 0.0000  0.0000 0.0000  0.0000 1
```

```
37 37 8 4 0.637231      0.0000 -1.2418 0.0000 -0.6667 0.0000 -0.6667 1
38 38 0 4 0.699993      0.0000  0.0000 0.0000  0.0000 0.0000  0.0000 1
```

Alternatively, one can define the excitations within Nexus as shown in `optical.py` file:

```
qmc = generate_qmcpack(
    ...
    excitation = ['up', '0 3 4 4'], # (ti, bi) notation
    #excitation = ['up', '-35 + 36'], # Orbital (state) index notation
    ...
    )
```

Chapter 22

Additional Tools

QMCPACK provides a set of lightweight executables that address certain common problems in QMC workflow and analysis. These range from conversion utilities between different file formats and QMCPACK (e.g. ppconvert and convert4qmc), (qmc-extract-eshdf-kvectors), to post-processing utilities (trace-density and qmcfinitesize), and to many others. In this chapter, we cover the use cases, syntax, and features of all additional tools provided with QMCPACK.

22.1 Initialization Tools

22.1.1 qmc-get-supercell

22.2 Post-Processing

22.2.1 qmca

`qmca` is a versatile tool to analyze and plot the raw data from QMCPACK *.scalar.dat files. It is a python executable and part of the Nexus suite of tools. It can be found in `qmcpack/nexus/executables`. For more detail, see section 12.1.

22.2.2 qmc-fit

`qmc-fit` is a curve fitting tool to obtain statistical error bars on fitted parameters. It is useful for DMC timestep extrapolation. For more details, see section 12.2.

22.2.3 qmcfinitesize

`qmcfinitesize` is a utility to compute many-body finite-size corrections to the energy. It is a C++ executable that is built alongside the qmcpack executable. It can be found in `build/bin`

22.3 Converters

22.3.1 convert4qmc

Convert4qmc allows conversion of orbitals and wavefunctions from quantum chemistry output files to QMCPACK XML and HDF5 input files. It is a small C++ executable that is built alongside the QMCPACK executable, and can be found in `build/bin`.

To date, convert4qmc supports the following codes: GAMESS[35], PySCF[36], Quantum Package[28] and GAMESS-FMO[37, 35]

General use

General usage of convert4qmc can be prompted by running with no options:

```
>convert4qmc

Defaults : -gridtype log -first 1e-6 -last 100 -size 1001 -ci required -threshold
    0.01 -TargetState 0 -prefix sample

 convert [-gaussian|-casino|-gamesxml|-gamess|-gamessFMO|-QP|-pyscf|-orbitals]
 filename
[-nojastrow -hdf5 -prefix title -addCusp -production -NbImages NimageX NimageY
    NimageZ]
[-psi_tag psi0 -ion_tag ion0 -gridtype log|log0|linear -first ri -last rf]
[-size npts -ci file.out -threshold cimin -TargetState state_number
-NaturalOrbitals NumToRead -optDetCoeffs]
Defaults : -gridtype log -first 1e-6 -last 100 -size 1001 -ci required
-threshold 0.01 -TargetState 0 -prefix sample
When the input format is missing, the  extension of filename is used to determine
the format
 *.Fchk -> gaussian; *.out -> gamess; *.data -> casino; *.xml -> gamesxml
```

As an example, to convert a GAMESS calculation using a single determinant, the following usage is sufficient:

```
convert4qmc -gamess MyGamessOutput.out
```

By default, the converter will generate multiple files:

`convert4qmc` output

output	**file type**	**default**	**description**
`*.qmc.in-wfs.xml`	XML	default	Main input file for QMCPACK
`*.qmc.in-wfnoj.xml`	XML	default	Main input file for QMCPACK
`*.structure.xml`	XML	default	file containing the structure of the system
`*.wfj.xml`	XML	default	Wavefunction file with 1, 2 and 3 body Jastrows
`*.wfnoj.xml`	XML	default	Wavefunction file with no Jastrows.
`*.orbs.h5`	HDF5	with -hdf5	HDF5 file containing all wavefunction data

If no `-prefix` option is specified, the prefix is taken from the input file name. For instance, if the GAMESS output file is **Mysim**.out the files generated by convert4qmc will use the prefix **Mysim** and output files will be `Mysim.qmc.`**`in`**`-wfs.xml`, `Mysim.structure.xml` and so on.

- Files **.in-wfs.xml** and **.in-wfnoj.xml**
  These are the input files for QMCPACK . The geometry and the wavefunction are stored in external files `*.structure.xml` and `*.wfj.xml` (referenced from `*.`**`in`**`-wfs.xml`) or `*.qmc.wfnoj.xml` (referenced from `*.qmc.`**`in`**`-wfnoj.xml`). The Hamiltonian section is

included and the presence or not of an ECP is detected during the conversion. If use of an ECP is detected, a default ECP name is added (e.g. `H.qmcpp.xml`) and it is the responsibility of the user to modify the ECP name to match the one used to generate the wavefunction.

```
  <?xml version="1.0"?>
<simulation>
  <!--

Example QMCPACK input file produced by convert4qmc

It is recommend to start with only the initial VMC block and adjust
parameters based on the measured energies, variance, and statistics.

-->
  <!--Name and Series number of the project.-->
  <project id="gms" series="0"/>
  <!--Link to the location of the Atomic Coordinates and the location of
      the Wavefunction.-->
  <include href="gms.structure.xml"/>
  <include href="gms.wfnoj.xml"/>
  <!--Hamiltonian of the system. Default ECP filenames are assumed.-->
  <hamiltonian name="h0" type="generic" target="e">
    <pairpot name="ElecElec" type="coulomb" source="e" target="e"
                                                  physical="true"/>
    <pairpot name="IonIon" type="coulomb" source="ion0" target="ion0"/>
    <pairpot name="PseudoPot" type="pseudo" source="ion0" wavefunction="psi0"
                                                      format="xml">
      <pseudo elementType="H" href="H.qmcpp.xml"/>
      <pseudo elementType="Li" href="Li.qmcpp.xml"/>
    </pairpot>
  </hamiltonian>
```

The `qmc.in-wfnoj.xml` file will have one VMC block with a minimum number of blocks to reproduce the HF/DFT energy used to generate the trial wavefunction.

```
  <qmc method="vmc" move="pbyp" checkpoint="-1">
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="warmupSteps">100</parameter>
    <parameter name="blocks">20</parameter>
    <parameter name="steps">50</parameter>
    <parameter name="substeps">8</parameter>
    <parameter name="timestep">0.5</parameter>
    <parameter name="usedrift">no</parameter>
  </qmc>
</simulation>
```

If the `qmc.in-wfj.xml` file is used, Jastrow optimization blocks followed by a VMC and DMC block are included. These blocks contain default values to allow the user to test the accuracy of a system, however, they need to be updated and optimized for each system. The initial values might only be suitable for a small molecule.

```
  <loop max="4">
    <qmc method="linear" move="pbyp" checkpoint="-1">
      <estimator name="LocalEnergy" hdf5="no"/>
      <parameter name="warmupSteps">100</parameter>
      <parameter name="blocks">20</parameter>
      <parameter name="timestep">0.5</parameter>
      <parameter name="walkers">1</parameter>
      <parameter name="samples">16000</parameter>
      <parameter name="substeps">4</parameter>
      <parameter name="usedrift">no</parameter>
      <parameter name="MinMethod">OneShiftOnly</parameter>
      <parameter name="minwalkers">0.0001</parameter>
    </qmc>
  </loop>
  <!--

Example follow-up VMC optimization using more samples for greater accuracy

-->
  <loop max="10">
    <qmc method="linear" move="pbyp" checkpoint="-1">
      <estimator name="LocalEnergy" hdf5="no"/>
      <parameter name="warmupSteps">100</parameter>
      <parameter name="blocks">20</parameter>
      <parameter name="timestep">0.5</parameter>
      <parameter name="walkers">1</parameter>
      <parameter name="samples">64000</parameter>
      <parameter name="substeps">4</parameter>
      <parameter name="usedrift">no</parameter>
      <parameter name="MinMethod">OneShiftOnly</parameter>
      <parameter name="minwalkers">0.3</parameter>
    </qmc>
  </loop>
  <!--

Production VMC and DMC

Examine the results of the optimization before running these blocks.
e.g. Choose the best optimized jastrow from all obtained, put in
wavefunction file, do not reoptimize.

-->
  <qmc method="vmc" move="pbyp" checkpoint="-1">
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="warmupSteps">100</parameter>
    <parameter name="blocks">200</parameter>
    <parameter name="steps">50</parameter>
    <parameter name="substeps">8</parameter>
    <parameter name="timestep">0.5</parameter>
    <parameter name="usedrift">no</parameter>
    <!--Sample count should match targetwalker count for
      DMC. Will be obtained from all nodes.-->
    <parameter name="samples">16000</parameter>
  </qmc>
  <qmc method="dmc" move="pbyp" checkpoint="20">
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="targetwalkers">16000</parameter>
    <parameter name="reconfiguration">no</parameter>
```

```
    <parameter name="warmupSteps">100</parameter>
    <parameter name="timestep">0.005</parameter>
    <parameter name="steps">100</parameter>
    <parameter name="blocks">100</parameter>
    <parameter name="nonlocalmoves">yes</parameter>
  </qmc>
</simulation>
```

- File **.structure.xml**
  This file will be referenced from the main qmcpack input. It contains the geometry of the system, position of the atoms, number of atoms, atomic types and charges, and number of electrons.

- Files **.wfj.xml** and **.wfnoj.xml**
  These files contain the basis set detail, orbital coefficients, and the 1, 2, 3 body Jastrow (in the case of **.wfj.xml**). If the wavefunction is multideterminant, the expansion will be at the end of the file. It is recommended to use the option **-hdf5** when large molecules are studied to store the data more compactly in an HDF5 file.

- File **.orbs.h5**
  This file is only generated if the option **-hdf5** is added as follows:

  ```
  convert4qmc -gamess MyGamessOutput.out -hdf5
  ```

  In this case, the **.wfj.xml** or **.wfnoj.xml** files will point to this HDF file. Information about the basis set, orbital coefficients and the multideterminant expansion is put in this file and removed from the wavefunction files, making them smaller.

`convert4qmc` input type

Option name	description
`-orbitals`	Generic HDF5 input file. Mainly automatically generated from Quantum Package and PyS
`-pyscf`	PySCF code
`-QP`	Quantum Package code
`-gamess`	Gamess code
`-gamesFMO`	Gamess FMO
`-gaussian`	Gaussian code
`-casino`	Casino code
`-gamesxml`	Gamess xml format code

Command line options

- **-multidet**
  This option is to be used when a multideterminant expansion (Mainly a CI expansion) is present in an HDF5 file. The trial wavefunction file will not display the full list of multideterminants and will add a path to the hdf5 file as follow (full example for the C2 molecule in qmcpack/tests/molecules/C2_pp).

`convert4qmc` command line options

Option name	Value	default	description
`-nojastrow`	-	-	Force no jastrow. qmc.in.wfj will not be generated
`-hdf5`	-	-	Force the wf to be in HDF5 format
`-prefix`	string	-	All created files will have the name of the string
`-multidet`	string	-	HDF5 file containing a multideterminant expansion
`-addCusp`	-	-	Force to add orbital cusp correction (ONLY for all-electron)
`-production`	-	-	Generates specific blocks in the input
`-psi_tag`	string	psi0	Name of the electrons particles inside QMCPACK
`-ion_tag`	string	ion0	Name of the ion particles inside QMCPACK

```
<?xml version="1.0"?>
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="MolecularOrbital" name="LCAOBSet" source="ion0"
    transform="yes" href="C2.h5">
      <sposet basisset="LCAOBSet" name="spo-up" size="58">
        <occupation mode="ground"/>
        <coefficient size="58" spindataset="0"/>
      </sposet>
      <sposet basisset="LCAOBSet" name="spo-dn" size="58">
        <occupation mode="ground"/>
        <coefficient size="58" spindataset="0"/>
      </sposet>
      <multideterminant optimize="no" spo_up="spo-up" spo_dn="spo-dn">
        <detlist size="202" type="DETS" nca="0" ncb="0" nea="4" neb="4"
    nstates="58" cutoff="1e-20" href="C2.h5"/>
      </multideterminant>
    </determinantset>
  </wavefunction>
</qmcsystem>
```

To generate such trial wavefunction, the converter has to be invoked as follow:

```
> convert4qmc -orbitals C2.h5 -multidet C2.h5
```

- **-nojastrow**
  This option generates only an input file `*.qmc.in.wfnoj.xml` containing no Jastrow optimization blocks and references a wavefunction file `*.wfnoj.xml` containing no jastrow section.

- **-hdf5**
  This option generates the `*.orbs.h5` HDF5 file containing the basis set and the orbital coefficients. If the wavefunction contains a multideterminant expansion from Quantum Package, it will also be stored in this file. This option minimizes the size of the `*.wfj.xml`, which points to the HDF file, as in the following example:

```
 <?xml version="1.0"?>
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="MolecularOrbital" name="LCAOBSet" source="ion0"
```

```
      transform="yes" href="test.orbs.h5">
    <slaterdeterminant>
      <determinant id="updet" size="39">
        <occupation mode="ground"/>
        <coefficient size="411" spindataset="0"/>
      </determinant>
      <determinant id="downdet" size="35">
        <occupation mode="ground"/>
        <coefficient size="411" spindataset="0"/>
      </determinant>
    </slaterdeterminant>
  </determinantset>
 </wavefunction>
</qmcsystem>
```

Jastrow functions will be included if the option "-nojastrow" was not specified. Note that when initially optimization a wavefunction, temporarily removing/disabling the three body Jastrow is recommended.

- **-prefix**
  Sets the prefix for all output generated by convert4qmc.
  If not specified, convert4qmc will use the defaults for:

  - **Gamess**
    If the Gamess output file is named "**Name**.out" or "**Name**.output", all files generated by convert4qmc will carry **Name** as a prefix (i.e **Name**.qmc.in.xml).

  - **PySCF**
    If the PySCF output file is named "**Name**.H5", all files generated by convert4qmc will carry **Name** as a prefix (i.e **Name**.qmc.in.xml).

  - **Quantum Package**
    If the Quantum Package output file is named "**Name**.dump", all files generated by convert4qmc will carry **Name** as a prefix (i.e **Name**.qmc.in.xml).

  - **Generic HDF5 input**
    If a generic HDF5 file (either from PySCF or Quantum Package in the HDF5 format) is named "**Name**.H5", all files generated by convert4qmc will carry **Name** as a prefix (i.e **Name**.qmc.in.xml).

- **-addCusp**
  This option is very important for all-electron (AE) calculations. In this case, orbitals have to be corrected for the electron-nuclear cusp. The cusp correction scheme follows the algorithm described by Ma *et. al.* [38] When this option is present, the wavefunction file has a new set of tags:

```
qmcsystem>
 <wavefunction name="psi0" target="e">
```

```
<determinantset type="MolecularOrbital" name="LCAOBSet" source="ion0"
  transform="yes" cuspCorrection="yes">
  <basisset name="LCAOBSet">
```

The tag "cuspCorrection" in the `wfj.xml` (or `wfnoj.xml`) wavefunction file will force correction of the orbitals at the beginning of the QMCPACK run.
In the "orbitals" section of the wavefunction file, a new tag "cuspInfo" will be added for orbitals spin-up and orbitals spin-down:

```
<slaterdeterminant>
     <determinant id="updet" size="2"
         cuspInfo="../CuspCorrection/updet.cuspInfo.xml">
       <occupation mode="ground"/>
       <coefficient size="135" id="updetC">

<determinant id="downdet" size="2"
        cuspInfo="../CuspCorrection/downdet.cuspInfo.xml">
       <occupation mode="ground"/>
       <coefficient size="135" id="downdetC">
```

These tags will point to the files `updet.cuspInfo.xml` and `downdet.cuspInfo.xml`. By default, the conveter assumes that the files are located in the relative path **../CuspCorrection/**. If the directory `../CuspCorrection` does not exist, or if the files are not present in that directory, QMCPACK will run the cusp correction algorithm to generate both files. If the files exist, then /qmcpack will apply the corrections to the orbitals.

Important notes:
- The AoS and SoA codes for cusp correction are unfortunately serial. Since the correction needs to be applied for every ion and then for every orbital on that ion, this operation can be extremely costly (slow) for large systems. It is recommended to run the correction on a single fast processor and then transfer `updet.cuspInfo.xml` and `downdet.cuspInfo.xml` to the HPC cluster where longer runs will be performed.

- **-psi_tag**
  QMCPACK builds the wavefunction as a named object. In the vast majority of cases, one wavefunction is simulated at a time, but there may be situations where we want to distinguish different parts of a wavefunction, or even use multiple wavefunctions. This option can change the name for these cases.

```
<wavefunction name="psi0" target="e">
```

- **-ion_tag**
  Similar to the **-psi_tag** but for the type of ions.

```
<particleset name="ion0" size="2">
```

- **-production**

  Without this option, input files with standard optimization, VMC, and DMC blocks are generated. When the "-production" option is specified, an input file containing more complex options that may be more suitable for large runs at HPC centers is generated. This option is for users who are already familiar with QMC and QMCPACK . Feedback on the standard and production sample inputs is encouraged.

The following options are specific to using MCSCF multideterminants from Gamess.

`convert4qmc` MCSCF arguments

option	Value	default	description
`-ci`	String	none	Name of the file containing the CI expansion
`-threshold`	double	1e-20	Cutoff of the weight of the determinants
`-TargetState`	int	none	?
`-NaturalOrbitals`	int	none	?
`-optDetCoeffs`	-	no	Enable the optimization of CI coefficients.

- keyword **-ci**
  Path/name of the file containing the CI expansion in a gamess Format.

- keyword **-threshold**
  The CI expansion contains coefficients (weights) for each determinant. This option sets the maximum coefficient to include in the QMC run. By default it is set to 1e-20 (meaning all determinants in a expansion are taken into account). At the same time, if the threshold is set to a different value, for example $1e-5$, any determinant with a weight, $abs(weight) < 1e-5$ will be discarded and the determinant will not be considered.

- keyword **-TargetState**
  ?

- keyword **-NaturalOrbitals**
  ?

- keyword **-optDetCoeffs**
  This flag enables the optimization of the CI expansion coefficients. By default, the optimization of the coefficients is disabled during wavefunction optimization runs.

Examples and more thorough descriptions of these options can be found in the Lab section of this manual: Chapter-19

Grid options

These parameters control how the basis set is projected on a grid. The default parameters are chosen to be very efficient. Unless you have a very good reason, it is not recommended to modify them.

convert4qmc Grid Keywords			
Tags			
keyword	**Value**	**default**	**description**
-gridtype	log|log0|linear	log	grid type
-first	double	1e-6	first point of the grid
-last	double	100	last point of the grid
-size	int	1001	number of point in the grid

- **-gridtype**
  Grid type can be logarithmic, logarithmic base 10 or linear

- **-first**
  First value of the grid

- **-last**
  last value of the grid

- **-size**
  Number of points in the grid between "first" and "last".

Supported codes

- **PySCF**

  PySCF[36] is an all-purpose quantum chemistry code that can run calculations from simple Hartree-Fock to DFT, MCSCF, and CCSD, and for both isolated systems and periodic boundary conditions. PySCF can be downloaded from https://github.com/sunqm/pyscf. Many examples and tutorials can be found on the PySCF website and all types of single determinants calculations are compatible with QMCPACK , thanks to active support from the authors of PySCF. A few additional steps are necessary to generate an output readable by convert4qmc.

  This example shows how to run a Hartree-Fock calculation for the LiH dimer molecule from PySCF and convert the wavefunction for QMCPACK .

  – **Python path**

  PySCF is a Python based code. A python module named **PyscfToQmcpack** containing the function **savetoqmcpack** is provided by QMCPACK and is located at qmcpack/src/QMCTools/PyscfToQmcpack.py. To be accessible to the PySCF script, this path must be added to the PYTHONPATH environment variable. For the bash shell, this can be done as follows:

```
export PYTHONPATH=/PATH_TO_QMCPACK/qmcpack/src/QMCTools:\$PYTHONPATH
```

– **PySCF Input File**

Copy and paste the following code in a file named LiH.py

```
#!/usr/bin/env python
from pyscf import gto, scf, df
import numpy

cell = gto.M(
   atom ='''
Li 0.0 0.0 0.0
H  0.0 0.0 3.0139239778''',
   basis ='cc-pv5z',
   unit="bohr",
   spin=0,
   verbose = 5,
   cart=False,
)
mf = scf.ROHF(cell)
mf.kernel()

###SPECIFIC TO QMCPACK###
title='LiH'
from PyscfToQmcpack import savetoqmcpack

savetoqmcpack(cell,mf,title)
```

The arguments to the function **savetoqmcpack** are:

* **cell**
  This is the object returned from gto.M, containing the type of atoms, geometry, basisset, spin etc..

* **mf**
  This is an object representing the pyscf level of theory; in this example, ROHF. This object contains the orbitals coefficients of the calculations.

* **title**
  The name of the output file generated by pyscf. By default, the name of the generated file will be "default" if nothing is specified.

By adding the three lines below the "SPECIFIC TO QMCPACK" comment in the input file, the script will dump all the necessary data for QMCPACK in an HDF5 file using the value of "title" as an output name. PySCF is run as follows:

```
>python LiH.py
```

The generated HDF5 can be read by convert4qmc to generate the appropriate QMCPACK input files.

– **Generating input files**

As described in the previous section, generating input files for pyscf is as follows:

```
> convert4qmc -pyscf LiH.h5
```

The HDF5 file produced by "savetoqmcpack" contains the wavefunction in a form directly readable by QMCPACK . The wavefunction files from convert4qmc reference this HDF file, as if the "-hdf5" option were specified (converting from PySCF implies the "-hdf5" option is always present).

An implementation of periodic boundary conditions with Gaussian orbitals from PySCF is under development.

- **Quantum Package**
Quantum Package[28] (QP) is a quantum chemistry code developed by the LCPQ laboratory in Toulouse, France. It can be downloaded from https://github.com/LCPQ/quantum_package, and the tutorial within is quite extensive. The tutorial section of QP can guide you on how to install and run the code.
After a QP calculation, the data needed for convert4qmc can be generated through:

```
qp_run save_for_qmcpack Myrun.ezfio &> Myrun.dump
```

Convert4qmc can read this format and generate QMCPACK input files in xml and HDF5 format. For example:

```
convert4qmc -QP Myrun.dump
```

The main reason to use Quantum Package is to access the CIPSI algorithm to generate a multideterminant wavefunction. CIPSI is the preferred choice for generating a selected CI trial wavefunction for QMCPACK . An example on how to use QP for Hartree-Fock and selected CI can be found in Chapter-14.1.1 of this manual. The converter code is actively maintained and co-developed by both QMCPACK and QP developers.

It is recommended to use a trial wavefunction stored in HDF5 format to reduce the reading time when a multideterminant expansion is too large (past 1k determinants). In order to do so, one can use 2 paths;
using the *-hdf5* option in the converter as follow:

- **Using -hdf5 tag**

```
convert4qmc -QP Myrun.dump -hdf5
```

This will read the multideterminant expansion in the *Myrun.dump* file and store them in the *Myrun.dump.orbs.h5*. Note that this method will be deprecated as Quantum Package generates automatically a compatible HDF5 file usable by QMCPACK directly.

- **Using h5 file**

  Quantum Package v2.0 (released in 2019) generates directly an HDF5 that mimics completely the QMCPACK readable format. This file can be generated after a CIPSI, Hartree Fock or Range Separated DFT in Quantum Package as follow

```
qp_run save_for_qmcpack Myrun.ezfio > Myrun.dump
```

  In addition to the *Myrun.dump*, an HDF5 file always named *QMC.h5* is also created containing all relevant information to start a qmcrun. Input files can be generated as follow

```
convert4qmc -orbitals QMC.h5 -multidet QMC.h5
```

  Note that the *QMC.h5* combined with the tags *-orbitals* and *-multidet* allow the user to choose orbitals from a different code such as PYSCF and the multideterminant section from Quantum Package. These two codes are fully compatible aand this route is also the only possible route for multideterminants for solids.

- **GAMESS**
  QMCPACK can use the output of GAMESS[35] for any type of single determinant calculation (HF or DFT) or multideterminant (MCSCF) calculation. A description with an example can be found in the Advanced Molecular Calculations Lab (section-19).

22.3.2 pw2qmcpack.x

pw2qmcpack.x is an executable that converts PWSCF wavefunctions to QMCPACK readable HDF5 format. This utility is built alongside the Quantum Espresso post-processing utilities. This utility is written in Fortran90, and is distributed as a patch of the Quantum Espresso source code. The patch, as well as automated QE download and patch scripts, can be found in `qmcpack/external_codes/quantum_espresso`.

pw2qmcpack can be used in serial in small systems and should be used in parallel for a large systems for best performance. The K_POINT gamma optimization is not supported.

Listing 22.1: Sample `pw2qmcpack.x` input file `p2q.in`

```
&inputpp
  prefix     = 'bulk_silicon'
  outdir     = './'
  write_psir = .false.
/
```

This example will cause pw2qmcpack.x to convert wavefunctions saved from PWSCF with the prefix “bulk_silicon”. Perform the conversion via, e.g.:

```
mpirun -np 1 pw2qmcpack.x < p2q.in>& p2q.out
```

Due to the large plane wave energy cutoffs in the pw.x calculation required by accurate pseudopotentials and the large system sizes that are of interest, one limitation of QE can be easily reached: that `wf_collect=.true.` results in problems of writing and loading correct plane wave coefficients on disks by pw.x because of the 32bit integer limits. Thus, pw2qmcpack.x fails to convert the orbitals for QMCPACK. Since Quantum ESPRESSO v5.3.0, the converter has been fully parallelized to overcome this limitation completely.

By setting `wf_collect=.false.` (by default `.false.` in v6.1 and before and `.true.` since v6.2), pw.x doesn't collect the whole wave function into individual files for each k point but instead write one smaller file for each processor. By just running pw2qmcpack.x in the same parallel setup (MPI tasks and k-pools) as the last scf/nscf calculation with pw.x, the orbitals distributed among processors will first be aggregated by the converter into individual temporal HDF5 files for each k-pool and then merged into the final file. In large calculations, users should benefit from a significant reduction of time in writing the wave function by pw.x thanks to avoiding the wavefunction collection.

pw2qmcpack has been included in the test suite of QMCPACK, see instructions about how to activate the tests in Sec. 3.12. There are tests labeled "no-collect" running the pw.x with setting `wf_collect=.false.`. The input files are stored at examples/solids/dft-inputs-polarized-no-collect. The scf, nscf, pw2qmcpack runs are performed on 16, 12, 12 MPI tasks with 16, 2, 2 k-pools respectively.

22.3.3 ppconvert

ppconvert is a utility to convert pseudopotentials between different commonly used formats. It is a stand alone C++ executable that is not built by default but accessible via adding `-DBUILD_PPCONVERT=1` to cmake and then typing `make ppconvert`. Currently it converts CASINO, FHI, UPF (generated by OPIUM), BFD, GAMESS formats to several formats including XML (QMCPACK) and UPF (Quantum ESPRESSO). See all the formats via `ppconvert -h`. For the output formats requiring Kleinman-Bylander projectors, the atom will be solved with DFT if the projectors are not provided in the input formats. This requires providing reference states and sometimes needs extra tuning for heavy elements. To avoid ghost states, the local channel can be changed via the `--local_channel` option. Ghost state considerations are similar to those of DFT calculations, but may be worse if ghost states were not considered during the original pseudopotential construction. To make the self-consistent calculation converge, the density mixing parameter may need to be reduced via the `--density_mix` option.

Note that the reference state should include only the valence electrons. One reference state should be included for each channel in the pseudopotential. For example, for a sodium atom with a neon core, the reference state would be "1s(1)". `--s_ref` needs to include a 1s state, `--p_ref` needs to include a 2p state, `--d_ref` needs to include a 3d state, etc. If not specified, a corresponding state with zero occupation is added. If the reference state is chosen as the neon core, setting empty reference states "" is techinically correct. In practice, reasonable reference states should be picked with care. For pseudopotentials with semi-core electrons in the valence, the reference state can be long. For example, Ti pseudopotential has 12 valence electrons. When using the neutral atom state, `--s_ref`, `--p_ref`, `--d_ref` are all set as "1s(2)2p(6)2s(2)3d(2)". When using an ionized state, the three reference states are all set as "1s(2)2p(6)2s(2)" or "1s(2)2p(6)2s(2)3d(0)". Unfortunately, if the generated UPF file is used in Quantum ESPRESSO, the calculation may be incorrect due to the presence of "ghost" states. Potentially these can be removed by adjusting the local chanel, e.g. by setting `--local_channel 1` which chooses the p channel as the local channel instead of d. For this reason, validation of UPF pseudopotentials is always required from the third

row and strongly encouraged in general. For example, check that the expected ionization potential and electron affinities are obtained for the atom and check that dimer properties are consistent with those obtained by a Quantum Chemistry code or a plane-wave code that does not use the Kleinman-Bylander projectors.

22.4 Obtaining pseudopotentials

22.4.1 Pseudopotentiallibrary.org

An open website collecting community developed and tested pseudopotentials for QMC and other many-body calculations is being developed at https://pseudopotentiallibrary.org. This site includes potentials in QMCPACK format and an increasing range of electronic structure and quantum chemistry codes. We recommend using potentials from this site if available and suitable for your science application.

22.4.2 Opium

Opium is a pseudopotential generation code available from the website http://opium.sourceforge.net/. Opium can generate pseudopotentials with either Hartree-Fock or DFT methods. Once you have a useable pseudopotential param file (for example, `Li.param`), generate pseudopotentials for use in Quantum ESPRESSO with the `upf` format as follows:

Listing 22.2: Generate UPF-formatted pseudopotential with Opium

```
opium Li.param Li.log all upf
```

This generates a UPF-formatted pseudopotential (`Li.upf`, in this case) for use in Quantum ESPRESSO. The pseudopotential conversion tool `ppconvert` can then convert UPF to FSAtom xml format for use in QMCPACK:

Listing 22.3: Convert UPF-formatted pseudopotential to FSAtom xml format

```
ppconvert --upf_pot Li.upf --xml Li.xml
```

22.4.3 Burkatzki-Filippi-Dolg

Burkatzki *et al.* developed a set of energy-consistent pseudopotenitals for use in QMC [4, 6], available at http://www.burkatzki.com/pseudos/index.2.html. To convert for use in QMCPACK, select a pseudopotential (choice of basis set is irrelevant to conversion) in GAMESS format and copy the ending (pseudopotential) lines beginning with(element symbol)-QMC GEN:
mathescapemathescapemathescapemathescape

Listing 22.4: BFD Li pseudopotential in GAMESS format

```
Li-QMC GEN 2 1
3
1.00000000 1 5.41040609
5.41040609 3 2.70520138
-4.60151975 2 2.07005488
1
7.09172172 2 1.34319829
```

Save these lines to a file (here, named `Li.BFD.gamess`; the exact name may be anything as long as it is passed to `ppconvert` after –gamess_pot). Then, convert using `ppconvert` with the following:

Listing 22.5: Convert GAMESS-formatted pseudopotential to FSAtom xml format

```
ppconvert --gamess_pot Li.BFD.gamess --s_ref "2s(1)" --p_ref "2p(0)" --xml
  Li.BFD.xml
```

Listing 22.6: Convert GAMESS-formatted pseudopotential to Quantum ESPRESSO UPF format

```
ppconvert --gamess_pot Li.BFD.gamess --s_ref "2s(1)" --p_ref "2p(0)" --log_grid
  --upf Li.BFD.upf
```

22.4.4 CASINO

The QMC code CASINO also makes available its pseudopotentials available at the website https://vallico.net/casinoqmc/pplib/. To use one in QMCPACK, select a pseudopotential and download its summary file (`summary.txt`), its tabulated form (`pp.data`), and (for ppconvert to construct the projectors to convert to Quantum ESPRESSO's UPF format) a CASINO atomic wavefunction for each angular momentum channel (`awfn.data_*`). Then, to convert using ppconvert, issue the following command:

Listing 22.7: Convert CASINO-formatted pseudopotential to Quantum ESPRESSO UPF format

```
ppconvert --casino_pot pp.data --casino_us awfn.data_s1_2S --casino_up
    awfn.data_p1_2P --casino_ud awfn.data_d1_2D --upf Li.TN-DF.upf
```

QMCPACK can directly read in the CASINO-formated pseudopotential (`pp.data`), but four parameters found in the pseudopotential summary file must be specified in the pseudo element (`l-local`, `lmax`, `nrule`, `cutoff`)[see Section 9.2.2 for details]:

Listing 22.8: XML syntax to use CASINO-formatted pseudopotentials in QMCPACK

```
<pairpot type="pseudo" name="PseudoPot" source="ion0" wavefunction="psi0"
    format="xml">
   <pseudo elementType="Li" href="Li.pp.data" format="casino" l-local="s" lmax="2"
    nrule="2" cutoff="2.19"/>
   <pseudo elementType="H" href="H.pp.data" format="casino" l-local="s" lmax="2"
    nrule="2" cutoff="0.5"/>
</pairpot>
```

Chapter 23

External Tools

This chapter provides some information on using QMCPACK with external tools.

23.1 LLVM Sanitizer Libraries

Using cmake set one of these flags for using the clang sanitizer libraries with or without lldb.

```
-DLLVM_SANITIZE_ADDRESS    link with the Clang address sanitizer library
-DLLVM_SANITIZE_MEMORY     link with the Clang memory sanitizer library
```

These set the basic flags required to build with either of these sanitizer libraries. They require a build of clang with dynamic libraries somehow visible, i.e. through `LD_FLAGS=-L/your/path/to/llvm/lib`. You must link through clang, which is generally the default when building with it. Depending on your system and linker this may be incompatible with the "Release" build so set `-DCMAKE_BUILD_TYPE=Debug`. They have been tested with the default spack install of llvm@7.0.0 and manually built llvm 7.0.1. See the above links for additional information on use, runtime and build options of the sanitizers.

In general the address sanitizer libraries will catch most pointer based errors. ASAN can also catch memory links but requires additional options be set. MSAN will catch more subtle memory management errors but is difficult to use without a full set of MSAN instrumented libraries.

23.2 Intel VTune

Intel's VTune profiler has an API that allows program control over collection (pause/resume) and can add information to the profile data (e.g. delineating tasks).

23.2.1 VTune API

If the variable `USE_VTUNE_API` is set, QMCPACK will check that the include file (`ittnotify.h`) and the library (`libittnotify.a`) can be found. To provide CMake with the VTune paths, add the include path to `CMAKE_CXX_FLAGS` and the the library path to `CMAKE_LIBRARY_PATH`.

An example of options to be passed to CMake

```
-DCMAKE_CXX_FLAGS=-I/opt/intel/vtune_amplifier_xe/include \
-DCMAKE_LIBRARY_PATH=/opt/intel/vtune_amplifier_xe/lib64
```

23.3 NVIDIA Tools Extensions (NVTX)

NVIDIA's Tools Extensions (NVTX) API enables programmers to annotate their source code when used with the NVIDIA profilers.

23.3.1 NVTX API

If the variable `USE_NVTX_API` is set, QMCPACK will add the library (`libnvToolsExt.so`) to the qmcpack target. To add NVTX annotations to a function, it is necessary to include the `nvToolsExt.h` header file and then make the appropriate calls into the NVTX API. For more information about the NVTX API, see https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvtx. Any additional calls to the NVTX API should be guarded by the `USE_NVTX_API` compiler define.

23.3.2 Timers as Tasks

To aid in connecting the timers in the code to the profile data, the start/stop of timers will be recorded as a task if `USE_VTUNE_TASKS` is set.

In addition to compiling with `USE_VTUNE_TASKS`, an option needs to be set at runtime to collect the task API data. In the GUI, select the checkbox labeled "Analyze user tasks" when setting up the analysis type. For the command line, set the **`enable`**`-user-tasks` knob to **`true`**. For example,

```
amplxe-cl -collect hotspots -knob enable-user-tasks=true ...
```

Collection with the timers set at "fine" can generate too much task data in the profile. Collection with the timers at "medium" collects a more reasonable amount of task data.

23.4 Scitools Understand

Scitools Understand (https://scitools.com/) is a tool for static code analysis. The easiest configuration route is to use the JSON output from CMake which the Understand project importer can read directly:

1. Configure QMCPACK by running cmake with `CMAKE_EXPORT_COMPILE_COMMANDS=ON`, e.g.

   ```
   cmake -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++
   -DQMC_MPI=0 -DCMAKE_EXPORT_COMPILE_COMMANDS=ON ../qmcpack/
   ```

2. Run Understand and create a new C++ project. At the import files and settings dialog, import the `compile_commands.json` created by cmake in the build directory.

Chapter 24

Contributing to the Manual

This section briefly describes how to contribute to the manual. This is primarily "by developers, for developers". This section should iterate until a consistent view on style/contents is reached.

Desirable:

- Use the table templates below when describing XML input.
- Instead of **\texttt** or **\verb** use
  - `\ishell` for shell text
  - `\ixml` for xml text
  - `\icode` for C++ text

  Except within tabularx or math environments
- Instead of **\begin**`{verbatim}` environments use the appropriate **\begin**`{lstlisting}[style=<see qmcpack_listings.sty>]`
- **\begin**`{shade}` can be used in place of **\begin**`{lstlisting}[style=SHELL]`
- Unicode rules
  - Do not use characters for which well established latex idiom exists, especially dashes, quotes, and apostrophes.
  - Use math mode markup instead of unicode characters for equations.
  - Be cautious of WYSIWYG word processors, cutting and pasting can pickup characters promoted to unicode by the program.
  - Take a look at your text multibyte expanded i.e. open in (emacs and 'esc-x toggle-enable-multibyte-characters'), see any unicode you didn't intend?
- Place unformatted text targeted at developers working on the latex in comments. Include generously.
- Encapsulate formatted text aimed at developers (like this entire chapter), in `\dev{}`. Text encapsulated in this way will be removed from the user version of the manual by editing the definition of `\dev{}` in `qmcpack_manual.tex`. Existing but deprecated or partially functioning features fall in this category.

- Newly added entries to a Bib file should be complete as possible. Use a tool such as JabRef or Zotero that can automate creation of these entries from just a DOI.

Forbidden:

- Including images instead of using lstlisting sections for text.
- Packages the LaTeX community considers deprecated.
- Do not use packages, features, or fonts not included in texlive 2017 unless you insure they degrade reasonably for 2017.
- Don't add packages unless they are bringing great value and are supported by tex4ht (unless you are willing to add the support).
- Tex files and Bib files are UTF8 encoded, do not save them in other encodings. Some may report being ASCII encoded since they contain no unicode characters.

Missing sections (these are opinions, not decided priorities):

- Description of XML input in general. Discuss XML format, use of attributes and `<parameter/>`'s in general, case sensitivity (input is generally case sensitive), and behavior of QMCPACK when unrecognized XML elements are encountered (they are generally ignored without notification).
- Overview of the input file in general, broad structure, and at least one full example that works in isolation.

Information currently missing for a complete reference specification:

- Noting how many instances of each child element are allowed. Examples: `simulation`–1 only, `method`–1 or more, `jastrow`–0 or more.

Below are template tables for describing XML elements in reference fashion. A number of examples can be found in *e.g.* Chapter 9. Preliminary style is (please weigh in with opinions): typewriter text (**`\texttt{}`**) for XML element, attribute, and parameter names, normal text for literal information in datatype/values/default columns, bold (**`\textbf{}`**) text if an attribute/parameter must take on a particular value (values column), italics (**`\textit{}`**) for descriptive (non-literal) information in the values column (e.g. *anything*, *non-zero*, etc.), required/optional attributes/parameters noted by `some_attr`^{r}/`some_attr`^{o} superscripts. Valid datatypes are text, integer, real, boolean, and arrays of each. Fixed lengh arrays can be noted, *e.g.* by "real array(3)".

Template for a generic XML element:

`generic` element				
parent elements:	`parent1 parent2`			
child elements:	`child1 child2 child3 ...`			
attributes				
name	**datatype**	**values**	**default**	**description**
`attr1`^{r}	text			
`attr2`^{r}	integer			
`attr3`^{o}	real			
`attr4`^{o}	boolean			
`attr5`^{o}	text array			
`attr6`^{o}	integer array			
`attr7`^{o}	real array			
`attr8`^{o}	boolean array			
parameters				
name	**datatype**	**values**	**default**	**description**
`param1`^{r}	text			
`param2`^{r}	integer			
`param3`^{o}	real			
`param4`^{o}	boolean			
`param5`^{o}	text array			
`param6`^{o}	integer array			
`param7`^{o}	real array			
`param8`^{o}	boolean array			
body text				
	Long form description of body text format			

"Factory" elements are XML elements that share a tag, but whose contents change based on the value an attribute (or sometimes multiple attributes take). The attribute(s) that determine the allowed contents is referred to below as the "type selector" (*e.g.* for **<estimator/>** elements, the type selector is usually the `type` attribute). These types of elements are frequently encountered as they correspond (sometimes loosely, sometimes literally) to polymorphic classes in QMCPACK that are built in "factories". This name is true to the underlying code, but may be obscure to the general user (is there a better name to retain the general meaning?).

The template below should be provided each time a new "factory" type is encountered (like **<estimator/>**). The table lists all types of possible elements (see "type options" below) and any attributes that are common to all possible related elements. Specific "derived" elements are then described one at a time with the template above, noting the type selector in addition to the XML tag (*e.g.* "`estimator type=density` element").

Template for shared information about "factory" elements.

`generic` factory element

parent elements:	`parent1 parent2`
child elements:	`child1 child2 child3 ...`
type selector:	`some` attribute
type options :	Selection1
	Selection2
	Selection3
	...
shared attributes:	

name	**datatype**	**values**	**default**	**description**
`attr1`	text			
`attr2`	integer			
...				

Chapter 25

Unit Testing

Unit testing is a standard software engineering practice to aid in ensuring a quality product. A good suite of unit tests provides confidence in refactoring and changing code, provides some documentation on how classes and functions are used, and can drive a more decoupled design.

If unit tests do not already exist for a section of code, you are encouraged to add them when modifying that section of code. New code additions should also include unit tests. When possible, fixes for specific bugs should also include a unit test that would have caught the bug.

25.1 Unit testing framework

The Catch framework is used for unit testing. See the project site for a tutorial and documentation: https://github.com/philsquared/Catch

Catch consists solely of header files. It is distributed as a single include file about 400KB in size. In QMCPACK, it is stored in `external_codes/catch`.

25.2 Unit test organization

The source for the unit tests is located in the `tests` directory under each directory in `src` (e.g. `src/QMCWavefunctions/tests`). All of the tests in each `tests` directory get compiled into an executable. After building the project, the individual unit test executables can be found in `build/tests/bin`. For example, the tests in `src/QMCWavefunctions/tests` are compiled into `build/tests/bin/test_wavefunction`.

All the unit test executables are collected under CTest with the `unit` label. When checking the whole code, it's useful to run through CMake (`cmake -L unit`). When working on an individual directory, it's useful to run the individual executable.

Some of the tests reference input files. The unit test CMake setup places those input files in particular locations under the `tests` directory (e.g. `tests/xml_test`). The individual test needs to be run from that directory to find the expected input files.

Command line options are available on the unit test executables. Some of the more useful ones are

List command line options.

List all the tests in the executable.

A test name can be given on the command line to execute just that test. This is useful when iterating on a particular test, or when running in the debugger. Test names often contain spaces, so most command line environments require enclosing the test name in single or double quotes.

25.3 Example

The first example is one test from `src/Numerics/tests/test_grid_functor.cpp`

Listing 25.1: Unit test example using Catch

```
TEST_CASE("double_1d_grid_functor", "[numerics]")
{
  LinearGrid<double> grid;
  OneDimGridFunctor<double> f(&grid);

  grid.set(0.0, 1.0, 3);

  REQUIRE(grid.size() == 3);
  REQUIRE(grid.rmin() == 0.0);
  REQUIRE(grid.rmax() == 1.0);
  REQUIRE(grid.dh() == Approx(0.5));
  REQUIRE(grid.dr(1) == Approx(0.5));
}
```

The test function declaration is `TEST_CASE("double_1d_grid_functor","[numerics]")`. The first argument is the test name, and it must be unique in the test suite. The second argument is an optional list of tags. Each tag is a name surrounded by brackets (`"[tag1][tag2]"`). It can also be the empty string.

The `REQUIRE` macro accepts expressions with C++ comparison operators and records an error if the value of the expression is false.

Floating point numbers may have small differences due to roundoff, etc. The `Approx` class adds some tolerance to the comparison. Place it on either side of the comparison (e.g. `Approx(a) == 0.3` or `a = Approx(0.3)`). To adjust the tolerance, use the `epsilon` and `scale` methods to `Approx` (`REQUIRE(Approx(a).epsilon(0.001) = 0.3);`.

25.3.1 Expected output

When running the test executables individually, the output of a run with no failures should look like

```
===============================================================================
All tests passed (26 assertions in 4 test cases)
```

A test with failures will look like

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
test_numerics is a Catch v1.4.0 host application.
Run with -? for options

-------------------------------------------------------------------------------
double_1d_grid_functor
-------------------------------------------------------------------------------
/home/user/qmcpack/src/Numerics/tests/test_grid_functor.cpp:29
...............................................................................

/home/user/qmcpack/src/Numerics/tests/test_grid_functor.cpp:39: FAILED:
  REQUIRE( grid.dh() == Approx(0.6) )
with expansion:
  0.5 == Approx( 0.6 )

===============================================================================
test cases:  4 |  3 passed | 1 failed
assertions: 25 | 24 passed | 1 failed
```

25.4 Adding tests

There are three scenarios covered here: adding a new test in an existing file, adding a new test file, or adding a new **test** directory.

25.4.1 Adding a test to existing file

Copy an existing test, or from the example shown here. Be sure to change the test name.

25.4.2 Adding a test file

When adding a new test file, create a file in the test directory, or copy from an existing file. Add the file name to the `ADD_EXECUTABLE` in the `CMakeLists.txt` file in that directory.

One (and only one) file must define the `main` function for the test executable by defining `CATCH_CONFIG_MAIN` before including the Catch header. If more than one file defines this value, there will be linking errors about multiply defined values.

Some of the tests need to shut down MPI properly to avoid extraneous error messages. Those tests include `Message/catch_mpi_main.hpp` instead of defining `CATCH_CONFIG_MAIN`.

25.4.3 Adding a test directory

Copy the CMakeLists.txt file from an existing `tests` directory. Change the `SRC_DIR` name and the files in the `ADD_EXECUTABLES` line. The libraries to link in `TARGET_LINK_LIBRARIES` may need to be updated.

Add the new test directory to `src/CMakeLists.txt` in the `BUILD_UNIT_TESTS` section near the end.

25.5 Testing with random numbers

Many algorithms and parts of the code depend on random numbers, which makes validating the results difficult. One solution is to verify that certain properties hold for any random number. This approach is valuable at some levels of testing, but is unsatisfying at the unit test level.

The `Utilities` directory contains a 'fake' random number generator that can be used for deterministic tests of these parts of the code. Currently it outputs a single, fixed value every time it is called, but it could be expanded to produce more varied, but still deterministic, sequences. See `src/QMCDrivers/test_vmc.cpp` for an example of using the fake random number generator.

Chapter 26

QMCPACK Design and Feature Documentation

This section contains information on the overall design of QMCPACK. Also included in this section are detailed explanations/derivations of major features and algorithms present in the code.

26.1 QMCPACK Design

TBD.

26.2 Feature: Optimized Long-Ranged Breakup (Ewald)

Consider a group of particles interacting with long-ranged central potentials, $v^{\alpha\beta}(|r_i^\alpha - r_j^\beta|)$, where the Greek superscripts represent the particle species (eg. α = electron, β = proton), and Roman subscripts refer to particle number within a species. We can then write the total interaction energy for the system as,

$$V = \sum_\alpha \left\{ \sum_{i<j} v^{\alpha\alpha}(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\alpha|) + \sum_{\beta<\alpha} \sum_{i,j} v^{\alpha\beta}(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta|) \right\} \tag{26.1}$$

26.2.1 The Long-Range Problem

Consider such a system in periodic boundary conditions in a cell defined by primitive lattice vectors $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$. Let $\mathbf{L} \equiv n_1\mathbf{a}_1 + n_2\mathbf{a}_2 + n_3\mathbf{a}_3$ be a direct lattice vector. Then the interaction energy per cell for the periodic system is given by

$$\begin{aligned} V = &\sum_{\mathbf{L}} \sum_\alpha \left\{ \overbrace{\sum_{i<j} v^{\alpha\alpha}(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\alpha + \mathbf{L}|)}^{\text{homologous}} + \overbrace{\sum_{\beta<\alpha} \sum_{i,j} v^{\alpha\beta}(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta + \mathbf{L}|)}^{\text{heterologous}} \right\}, \\ &+ \underbrace{\sum_{\mathbf{L}\neq\mathbf{0}} \sum_\alpha N^\alpha v^{\alpha\alpha}(|\mathbf{L}|)}_{\text{Madelung}} \end{aligned} \tag{26.2}$$

where N^α is the number particles of species α. If the potentials $v^{\alpha\beta}(r)$ are indeed long-range, the summation over direct lattice vectors will not converge in this naive form. A solution to the problem was posited by Ewald. We break the central potentials into two pieces – a short range and a long range part define by

$$v^{\alpha\beta}(r) = v_s^{\alpha\beta}(r) + v_l^{\alpha\beta}(r). \tag{26.3}$$

We will perform the summation over images for the short-range part in real space, while performing the sum for the long-range part in reciprocal space. For simplicity, we choose $v_s^{\alpha\beta}(r)$ so that it is identically zero at the half the box length. This eliminates the need to sum over images in real space.

26.2.2 Reciprocal-Space Sums

Heterologous terms

We begin with (26.2), starting with the heterologous terms, i.e. the terms involving particles of different species. The short-range terms are trivial, so we neglect them here.

$$\text{heterologous} = \frac{1}{2} \sum_{\alpha\neq\beta} \sum_{i,j} \sum_{\mathbf{L}} v_l^{\alpha\beta}(\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta + \mathbf{L}) \tag{26.4}$$

We insert the resolution of unity in real space twice,

$$\begin{aligned}
\text{heterologous} &= \frac{1}{2}\sum_{\alpha\neq\beta}\int_{\text{cell}} d\mathbf{r}\, d\mathbf{r}' \sum_{i,j}\delta(\mathbf{r}_i^\alpha-\mathbf{r})\delta(\mathbf{r}_j^\beta-\mathbf{r}')\sum_{\mathbf{L}} v_l^{\alpha\beta}(|\mathbf{r}-\mathbf{r}'+\mathbf{L}|) \qquad (26.5)\\
&= \frac{1}{2\Omega^2}\sum_{\alpha\neq\beta}\int_{\text{cell}} d\mathbf{r}\, d\mathbf{r}' \sum_{\mathbf{k},\mathbf{k}',i,j} e^{i\mathbf{k}\cdot(\mathbf{r}_i^\alpha-\mathbf{r})}e^{i\mathbf{k}'\cdot(\mathbf{r}_j^\beta-\mathbf{r}')}\sum_{\mathbf{L}} v_l^{\alpha\beta}(|\mathbf{r}-\mathbf{r}'+\mathbf{L}|)\\
&= \frac{1}{2\Omega^2}\sum_{\alpha\neq\beta}\int_{\text{cell}} d\mathbf{r}\, d\mathbf{r}' \sum_{\mathbf{k},\mathbf{k}',\mathbf{k}'',i,j} e^{i\mathbf{k}\cdot(\mathbf{r}_i^\alpha-\mathbf{r})}e^{i\mathbf{k}'\cdot(\mathbf{r}_j^\beta-\mathbf{r}')}e^{i\mathbf{k}''\cdot(\mathbf{r}-\mathbf{r}')}v_{\mathbf{k}''}^{\alpha\beta},.
\end{aligned}$$

Here, the $\mathbf{k}$ summations are over reciprocal lattice vectors given by $\mathbf{k} = m_1\mathbf{b}_1 + m_2\mathbf{b}_2 + m_3\mathbf{b}_3$, where

$$\begin{aligned}
\mathbf{b}_1 &= 2\pi\frac{\mathbf{a}_2\times\mathbf{a}_3}{\mathbf{a}_1\cdot(\mathbf{a}_2\times\mathbf{a}_3)}\\
\mathbf{b}_2 &= 2\pi\frac{\mathbf{a}_3\times\mathbf{a}_1}{\mathbf{a}_1\cdot(\mathbf{a}_2\times\mathbf{a}_3)} \qquad (26.6)\\
\mathbf{b}_3 &= 2\pi\frac{\mathbf{a}_1\times\mathbf{a}_2}{\mathbf{a}_1\cdot(\mathbf{a}_2\times\mathbf{a}_3)}.
\end{aligned}$$

We note that $\mathbf{k}\cdot\mathbf{L} = 2\pi(n_1m_1 + n_2m_2 + n_3m_3)$.

$$\begin{aligned}
v_{k''}^{\alpha\beta} &= \frac{1}{\Omega}\int_{\text{cell}} d\mathbf{r}''\sum_{\mathbf{L}} e^{-i\mathbf{k}''\cdot(|\mathbf{r}''+\mathbf{L}|)}v^{\alpha\beta}(|\mathbf{r}''+\mathbf{L}|), \qquad (26.7)\\
&= \frac{1}{\Omega}\int_{\text{all space}} d\tilde{\mathbf{r}}\, e^{-i\mathbf{k}''\cdot\tilde{\mathbf{r}}}v^{\alpha\beta}(\tilde{r}), \qquad (26.8)
\end{aligned}$$

where Ω is the volume of the cell. Here we have used the fact that summing over all cells of the integral over the cell is equivalent to integrating over all space.

$$\text{hetero} = \frac{1}{2\Omega^2}\sum_{\alpha\neq\beta}\int_{\text{cell}} d\mathbf{r}\, d\mathbf{r}' \sum_{\mathbf{k},\mathbf{k}',\mathbf{k}'',i,j} e^{i(\mathbf{k}\cdot\mathbf{r}_i^\alpha+\mathbf{k}'\cdot\mathbf{r}_j^\beta)}e^{i(\mathbf{k}''-\mathbf{k})\cdot\mathbf{r}}e^{-i(\mathbf{k}''+\mathbf{k}')\cdot\mathbf{r}'}v_{\mathbf{k}''}^{\alpha\beta}. \qquad (26.9)$$

We have

$$\frac{1}{\Omega}\int d\mathbf{r}\; e^{i(\mathbf{k}-\mathbf{k}')\cdot\mathbf{r}} = \delta_{\mathbf{k},\mathbf{k}'}, \qquad (26.10)$$

Then, performing the integrations we have

$$\text{hetero} = \frac{1}{2}\sum_{\alpha\neq\beta}\sum_{\mathbf{k},\mathbf{k}',\mathbf{k}'',i,j} e^{i(\mathbf{k}\cdot\mathbf{r}_i^\alpha+\mathbf{k}'\cdot\mathbf{r}_j^\beta)}\delta_{\mathbf{k},\mathbf{k}''}\delta_{-\mathbf{k}',\mathbf{k}''}v_{\mathbf{k}''}^{\alpha\beta}. \qquad (26.11)$$

We now separate the summations, yielding

$$\text{hetero} = \frac{1}{2}\sum_{\alpha\neq\beta}\sum_{\mathbf{k},\mathbf{k}'}\underbrace{\left[\sum_i e^{i\mathbf{k}\cdot\mathbf{r}_i^\alpha}\right]}_{\rho_{\mathbf{k}}^\alpha}\underbrace{\left[\sum_j e^{i\mathbf{k}'\cdot\mathbf{r}_j^\beta}\right]}_{\rho_{\mathbf{k}'}^\beta}\delta_{\mathbf{k},\mathbf{k}''}\delta_{-\mathbf{k}',\mathbf{k}''}v_{\mathbf{k}''}^{\alpha\beta}. \qquad (26.12)$$

Summing over $\mathbf{k}$ and $\mathbf{k}'$, we have

$$\text{hetero} = \frac{1}{2}\sum_{\alpha\neq\beta}\sum_{\mathbf{k}''}\rho^{\alpha}_{\mathbf{k}''}\,\rho^{\beta}_{-\mathbf{k}''}v^{\alpha\beta}_{k''}. \tag{26.13}$$

We can simplify the calculation a bit further by rearranging the sums over species,

$$\begin{aligned}\text{hetero} &= \frac{1}{2}\sum_{\alpha>\beta}\sum_{\mathbf{k}}\left(\rho^{\alpha}_{\mathbf{k}}\rho^{\beta}_{-\mathbf{k}}+\rho^{\alpha}_{-\mathbf{k}}\rho^{\beta}_{\mathbf{k}}\right)v^{\alpha\beta}_{k} &(26.14)\\ &= \sum_{\alpha>\beta}\sum_{\mathbf{k}}\mathcal{R}e\left(\rho^{\alpha}_{\mathbf{k}}\rho^{\beta}_{-\mathbf{k}}\right)v^{\alpha\beta}_{k}. &(26.15)\end{aligned}$$

Homologous Terms

We now consider the terms involving particles of the same species interacting with each other. The algebra is very similar to that above, with the slight difficulty of avoiding the self-interaction term.

$$\begin{aligned}\text{homologous} &= \sum_{\alpha}\sum_{L}\sum_{i<j}v^{\alpha\alpha}_{l}(|\mathbf{r}^{\alpha}_{i}-\mathbf{r}^{\alpha}_{j}+\mathbf{L}|) &(26.16)\\ &= \frac{1}{2}\sum_{\alpha}\sum_{L}\sum_{i\neq j}v^{\alpha\alpha}_{l}(|\mathbf{r}^{\alpha}_{i}-\mathbf{r}^{\alpha}_{j}+\mathbf{L}|) &(26.17)\end{aligned}$$

$$\begin{aligned}\text{homologous} &= \frac{1}{2}\sum_{\alpha}\sum_{L}\left[-N^{\alpha}v^{\alpha\alpha}_{l}(|\mathbf{L}|)+\sum_{i,j}v^{\alpha\alpha}_{l}(|\mathbf{r}^{\alpha}_{i}-\mathbf{r}^{\alpha}_{j}+\mathbf{L}|)\right] &(26.18)\\ &= \frac{1}{2}\sum_{\alpha}\sum_{\mathbf{k}}\left(|\rho^{\alpha}_{k}|^{2}-N\right)v^{\alpha\alpha}_{k} &(26.19)\end{aligned}$$

Madelung Terms

Let us now consider the Madelung term for a single particle of species α. This term corresponds to the interaction of a particle with all of its periodic images.

$$\begin{aligned}v^{\alpha}_{M} &= \frac{1}{2}\sum_{\mathbf{L}\neq\mathbf{0}}v^{\alpha\alpha}(|\mathbf{L}|) &(26.20)\\ &= \frac{1}{2}\left[-v^{\alpha\alpha}_{l}(0)+\sum_{\mathbf{L}}v^{\alpha\alpha}(|\mathbf{L}|)\right] &(26.21)\\ &= \frac{1}{2}\left[-v^{\alpha\alpha}_{l}(0)+\sum_{\mathbf{k}}v^{\alpha\alpha}_{\mathbf{k}}\right] &(26.22)\end{aligned}$$

$\mathbf{k}=\mathbf{0}$ terms

Thus far, we have neglected what happens at the special point $\mathbf{k}=\mathbf{0}$. For many long-range potentials, such as the coulomb potential, $v^{\alpha\alpha}_{k}$ diverges for $k=0$. However, we recognize that for a charge-neutral system, the divergent part of the terms cancel each other. If all the potential in the system were precisely coulomb, the $\mathbf{k}=\mathbf{0}$ terms would cancel precisely, yielding zero. For systems

involving pseudopotentials, however, it may be the case the resulting term is finite, but nonzero. Consider the terms from $\mathbf{k} = \mathbf{0}$,

$$V_{k=0} = \sum_{\alpha>\beta} N^\alpha N^\beta v_{k=0}^{\alpha\beta} + \frac{1}{2}\sum_\alpha (N^\alpha)^2 v_{k=0}^{\alpha\alpha} \tag{26.23}$$

$$= \frac{1}{2}\sum_{\alpha,\beta} N^\alpha N^\beta v_{k=0}^{\alpha\beta}. \tag{26.24}$$

Next, we must compute $v_{k=0}^{\alpha\beta}$.

$$v_{k=0}^{\alpha\beta} = \frac{4\pi}{\Omega}\int_0^\infty dr\ r^2 v_l^{\alpha\beta}(r) \tag{26.25}$$

We recognize that this integral will not converge because of the large-r behavior. However, we recognize that when we do the sum in (26.24), the large-r parts of the integrals will cancel precisely. Therefore, we define

$$\tilde{v}_{k=0}^{\alpha\beta} = \frac{4\pi}{\Omega}\int_0^{r_{\text{end}}} dr\ r^2 v_l^{\alpha\beta}(r), \tag{26.26}$$

where r_{end} is some cutoff value after which the potential tails precisely cancel.

Neutralizing Background Terms

For systems with a net charge, such as the one-component plasma (jellium), we add a uniform background charge which makes the system neutral. When we do this, we must add a term which comes from the interaction of the particle with the neutral background. It is a constant term, independent of the particle positions. In general, we have a compensating background for each species, which largely cancels out for neutral systems.

$$V_{\text{background}} = -\frac{1}{2}\sum_\alpha (N^\alpha)^2 v_{s\mathbf{0}}^{\alpha\alpha} - \sum_{\alpha>\beta} N_\alpha N_\beta v_{s\mathbf{0}}^{\alpha\beta}, \tag{26.27}$$

where $v_{s\mathbf{0}}^{\alpha\beta}$ is given by

$$v_{s\mathbf{0}}^{\alpha\beta} = \frac{1}{\Omega}\int_0^{r_c} d^3r\ v_s^{\alpha\beta}(r) \tag{26.28}$$

$$= \frac{4\pi}{\Omega}\int_0^{r_c} r^2 v_s(r)\ dr$$

26.2.3 Combining Terms

Here, we sum all of the terms we computed in the sections above,

$$\begin{aligned} V &= \sum_{\alpha>\beta}\left[\sum_{i,j} v_s(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta|) + \sum_{\mathbf{k}} \mathcal{R}e\left(\rho_{\mathbf{k}}^\alpha \rho_{-\mathbf{k}}^\beta\right) v_k^{\alpha\beta} - N^\alpha N^\beta v_{s\mathbf{0}}^{\alpha\beta}\right] \\ &+ \sum_{\alpha}\left[N^\alpha v_M^\alpha + \sum_{i>j} v_s(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\alpha|) + \frac{1}{2}\sum_{\mathbf{k}}\left(|\rho_{\mathbf{k}}^\alpha|^2 - N\right) v_{\mathbf{k}}^{\alpha\alpha} - \frac{1}{2}\left(N_\alpha\right)^2 v_{s\mathbf{0}}^{\alpha\alpha}\right] \\ &= \sum_{\alpha>\beta}\left[\sum_{i,j} v_s(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta|) + \sum_{\mathbf{k}} \mathcal{R}e\left(\rho_{\mathbf{k}}^\alpha \rho_{-\mathbf{k}}^\beta\right) v_k^{\alpha\beta} - N^\alpha N^\beta v_{s\mathbf{0}}^{\alpha\beta} + \tilde{V}_{k=0}\right] \\ &+ \sum_{\alpha}\left[-\frac{N^\alpha v_l^{\alpha\alpha}(0)}{2} + \sum_{i>j} v_s(|\mathbf{r}_i^\alpha - \mathbf{r}_j^\alpha|) + \frac{1}{2}\sum_{\mathbf{k}}|\rho_{\mathbf{k}}^\alpha|^2 v_{\mathbf{k}}^{\alpha\alpha} - \frac{1}{2}\left(N_\alpha\right)^2 v_{s\mathbf{0}}^{\alpha\alpha} + \tilde{V}_{k=0}\right] \end{aligned} \tag{26.29}$$

26.2.4 Computing the Reciprocal Potential

Now we return to (26.8). Without loss of generality, we define for convenience $\mathbf{k} = k\hat{\mathbf{z}}$.

$$v_k^{\alpha\beta} = \frac{2\pi}{\Omega}\int_0^\infty dr \int_{-1}^1 d\cos(\theta)\; r^2 e^{-ikr\cos(\theta)} v_l^{\alpha\beta}(r) \tag{26.30}$$

We do the angular integral first. By inversion symmetry, the imaginary part of the integral vanishes, yielding

$$v_k^{\alpha\beta} = \frac{4\pi}{\Omega k}\int_0^\infty dr\; r\sin(kr) v_l^{\alpha\beta}(r). \tag{26.31}$$

26.2.5 The Coulomb Potential

For the case of the Coulomb potential, the above integral is not formally convergent if we do the integral naively. We may remedy the situation by including a convergence factor, $e^{-k_0 r}$. For a potential of the form $v^{\text{coul}}(r) = q_1 q_2/r$, this yields

$$\begin{aligned} v_k^{\text{screened coul}} &= \frac{4\pi q_1 q_2}{\Omega k}\int_0^\infty dr\;\; \sin(kr) e^{-k_0 r} && (26.32) \\ &= \frac{4\pi q_1 q_2}{\Omega(k^2 + k_0^2)} && (26.33) \end{aligned}$$

Allowing the convergence factor to tend to zero, we have

$$v_k^{\text{coul}} = \frac{4\pi q_1 q_2}{\Omega k^2} \tag{26.34}$$

For more generalized potentials with a coulomb tail, we cannot evaluate (26.31) numerically but must handle the coulomb part analytically. In this case, we have

$$v_k^{\alpha\beta} = \frac{4\pi}{\Omega}\left\{\frac{q_1 q_2}{k^2} + \int_0^\infty dr\; r\sin(kr)\left[v_l^{\alpha\beta}(r) - \frac{q_1 q_2}{r}\right]\right\} \tag{26.35}$$

26.2.6 Efficient calculation methods

Fast computation of $\rho_{\mathbf{k}}$

We wish to quickly calculate the quantity

$$\rho_{\mathbf{k}}^{\alpha} \equiv \sum_i e^{i\mathbf{k}\cdot r_i^{\alpha}} \tag{26.36}$$

First, we write

$$\begin{aligned} \mathbf{k} &= m_1\mathbf{b}_1 + m_2\mathbf{b}_2 + m_3\mathbf{b}_3 && (26.37)\\ \mathbf{k}\cdot\mathbf{r}_i^{\alpha} &= m_1\mathbf{b}_1\cdot\mathbf{r}_i^{\alpha} + m_2\mathbf{b}_2\cdot\mathbf{r}_i^{\alpha} + m_3\mathbf{b}_3\cdot\mathbf{r}_i^{\alpha} && (26.38)\\ e^{i\mathbf{k}\cdot r_i^{\alpha}} &= \underbrace{\left[e^{i\mathbf{b}_1\cdot\mathbf{r}_i^{\alpha}}\right]}_{C_1^{i\alpha}}^{m_1} \underbrace{\left[e^{i\mathbf{b}_2\cdot\mathbf{r}_i^{\alpha}}\right]}_{C_2^{i\alpha}}^{m_2} \underbrace{\left[e^{i\mathbf{b}_3\cdot\mathbf{r}_i^{\alpha}}\right]}_{C_3^{i\alpha}}^{m_3} && (26.39) \end{aligned}$$

Now, we note that

$$[C_1^{i\alpha}]^{m_1} = C_1^{i\alpha}[C^{i\alpha}]^{(m_1-1)}. \tag{26.40}$$

This allows us to recursively build up an array of the $C^{i\alpha}$s, and then compute $\rho_{\mathbf{k}}$ for all $\mathbf{k}$-vectors by looping over all k-vectors, requiring only two complex multiplies per particle per $\mathbf{k}$.

Algorithm 1 Algorithm to quickly calculate $\rho_{\mathbf{k}}^{\alpha}$.

```
--list-tests Create list of k-vectors and corresponding (m1, m2, m3) indices.
  for all α ∈ species do
    Zero out ρ_k^α
    for all i ∈ particles do
      for j ∈ [1 ··· 3] do
        Compute C_j^{iα} ≡ e^{i b_j · r_i^α}
        for m ∈ [−m_max ... m_max] do
          Compute [C_j^{iα}]^m and store in array
        end for
      end for
      for all (m1, m2, m3) ∈ index list do
        Compute e^{i k · r_i^α} = [C_1^{iα}]^{m1} [C_2^{iα}]^{m2} [C_3^{iα}]^{m3} from array
      end for
    end for
  end for
```

26.2.7 Gaussian Charge Screening Breakup

This original approach to the short and long-ranged breakup adds an opposite screening charge of gaussian shape around each point charge. It then removes the charge in the long-ranged part of the potential. In this potential,

$$v_{\text{long}}(r) = \frac{q_1 q_2}{r}\text{erf}(\alpha r), \tag{26.41}$$

where α is an adjustable parameter used to control how short-ranged the potential should be. If the box size is L, a typical value for α might be $7/(Lq_1q_2)$. We should note that this form

for the long-ranged potential should also work for any general potential with a coulomb tail, e.g. pseudo-Hamiltonian potentials. For this form of the long-ranged potential, we have in k-space

$$v_k = \frac{4\pi q_1 q_2 \exp\left[\frac{-k^2}{4\alpha^2}\right]}{\Omega k^2}. \tag{26.42}$$

26.2.8 Optimized Breakup Method

In this section, we undertake the task of choosing a long-range/short-range partitioning of the potential which is optimal in that it minimizes the error for given real and k-space cutoffs r_c and k_c. Here, we modify slightly the method introduced Natoli and Ceperley[10]. We choose $r_c = \frac{1}{2}\min\{L_i\}$, so that we require the nearest image in real space summation. k_c is then chosen so as to satisfy our accuracy requirements.

Here we modify our notation slightly to accommodate details not required above. We restrict our discussion to the interaction of two particles species (which may be the same), and drop our species indices. Thus we are looking for short and long-range potentials defined by,

$$v(r) = v^s(r) + v^\ell(r) \tag{26.43}$$

Define v_k^s and v_k^ℓ to be the respective Fourier transforms of the above. The goal is to choose $v_s(r)$ such that its value and first two derivatives vanish at r_c, while making $v^\ell(r)$ as smooth as possible so that k-space components, v_k^ℓ, are very small for $k > k_c$. Here, we describe how to do this is an optimal way.

Define the periodic potential, V_p, as

$$V_p(\mathbf{r}) = \sum_l v(|\mathbf{r} + \mathbf{l}|), \tag{26.44}$$

where $\mathbf{r}$ is the displacement between the two particles and $\mathbf{l}$ is a lattice vector. Let us then define our approximation to this potential, V_a, as

$$V_a(\mathbf{r}) = v^s(r) + \sum_{|\mathbf{k}|<k_c} v_k^\ell e^{i\mathbf{k}\cdot\mathbf{r}} \tag{26.45}$$

Now, we seek to minimize the RMS error over the cell,

$$\chi^2 = \frac{1}{\Omega}\int_\Omega d^3\mathbf{r}\ \ |V_p(\mathbf{r}) - V_a(\mathbf{r})|^2 \tag{26.46}$$

We may write

$$V_p(\mathbf{r}) = \sum_{\mathbf{k}} v_k e^{i\mathbf{k}\cdot\mathbf{r}}, \tag{26.47}$$

where

$$v_k = \frac{1}{\Omega}\int d^3\mathbf{r}\ e^{-i\mathbf{k}\cdot\mathbf{r}} v(r). \tag{26.48}$$

We now need a basis in which to represent the broken up potential. We may choose to represent either $v^s(r)$ or $v^\ell(r)$ in a real-space basis. Natoli and Ceperley chose the prior in their paper. We choose the latter for a number of reasons. First, singular potentials are difficult to represent in a linear basis unless the singularity is explicitly included. This requires a separate basis for each type of singularity. The short-range potential may have an arbitrary number of features for $r < r_c$ and

still be a valid potential. By construction, however, we desire that $v^\ell(r)$ be smooth in real-space so that its Fourier transform falls off quickly with increasing k. We therefore expect that, in general, $v^\ell(r)$ should be well-represented by fewer basis functions than $v^s(r)$. Therefore, we define,

$$v^\ell(r) \equiv \begin{cases} \sum_{n=0}^{J-1} t_n h_n(r) & \text{for } r \leq r_c \\ v(r) & \text{for } r > r_c. \end{cases} \tag{26.49}$$

where the $h_n(r)$ are a set of J basis functions. We require that the two cases agree on the value and first two derivatives at r_c. We may then define

$$c_{nk} \equiv \frac{1}{\Omega} \int_0^{r_c} d^3\mathbf{r}\ e^{-i\mathbf{k}\cdot\mathbf{r}} h_n(r). \tag{26.50}$$

Similarly, we define

$$x_k \equiv -\frac{1}{\Omega} \int_{r_c}^{\infty} d^3\mathbf{r}\ e^{-i\mathbf{k}\cdot\mathbf{r}} v(r) \tag{26.51}$$

Therefore,

$$v_k^\ell = -x_k + \sum_{n=0}^{J-1} t_n c_{nk} \tag{26.52}$$

Because $v^s(r)$ goes identically to zero at the box edge, inside the cell we may write

$$v^s(\mathbf{r}) = \sum_{\mathbf{k}} v_k^s e^{i\mathbf{k}\cdot\mathbf{r}} \tag{26.53}$$

We then write

$$\chi^2 = \frac{1}{\Omega} \int_\Omega d^3\mathbf{r} \left| \sum_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{r}} (v_k - v_k^s) - \sum_{|\mathbf{k}|\leq k_c} v_k^\ell \right|^2 \tag{26.54}$$

We see that if we define

$$v^s(r) \equiv v(r) - v^\ell(r) \tag{26.55}$$

Then

$$v_k^\ell + v_k^s = v_k, \tag{26.56}$$

which then cancels out all terms for $|\mathbf{k}| < k_c$. Then we have

$$\chi^2 = \frac{1}{\Omega} \int_\Omega d^3\mathbf{r} \left| \sum_{|\mathbf{k}|>k_c} e^{i\mathbf{k}\cdot\mathbf{r}} (v_k - v_k^s) \right|^2 \tag{26.57}$$

$$= \frac{1}{\Omega} \int_\Omega d^3\mathbf{r} \left| \sum_{|\mathbf{k}|>k_c} e^{i\mathbf{k}\cdot\mathbf{r}} v_k^\ell \right|^2 \tag{26.58}$$

$$= \frac{1}{\Omega} \int_\Omega d^3\mathbf{r} \left| \sum_{|\mathbf{k}|>k_c} e^{i\mathbf{k}\cdot\mathbf{r}} \left(-x_k + \sum_{n=0}^{J-1} t_n c_{nk} \right) \right|^2 \tag{26.59}$$

We expand the summation,

$$\chi^2 = \frac{1}{\Omega} \int_\Omega d^3\mathbf{r} \sum_{\{|\mathbf{k}|,|\mathbf{k}'|\}>k_c} e^{i(\mathbf{k}-\mathbf{k}')\cdot\mathbf{r}} \left(x_k - \sum_{n=0}^{J-1} t_n c_{nk} \right) \left(x_k - \sum_{m=0}^{J-1} t_m c_{mk'} \right) \tag{26.60}$$

We take the derivative w.r.t. t_m,

$$\frac{\partial(\chi^2)}{\partial t_m} = \frac{2}{\Omega}\int_\Omega d^3\mathbf{r} \sum_{\{|\mathbf{k}|,|\mathbf{k}'|\}>k_c} e^{i(\mathbf{k}-\mathbf{k}')\cdot\mathbf{r}} \left(x_k - \sum_{n=0}^{J-1} t_n c_{nk} \right) c_{mk'} \tag{26.61}$$

We integrate w.r.t. $\mathbf{r}$, yielding a Kronecker δ.

$$\frac{\partial(\chi^2)}{\partial t_m} = 2 \sum_{\{|\mathbf{k}|,|\mathbf{k}'|\}>k_c} \delta_{\mathbf{k},\mathbf{k}'} \left(x_k - \sum_{n=0}^{J-1} t_n c_{nk} \right) c_{mk'} \tag{26.62}$$

Summing over $\mathbf{k}'$ and equating the derivative to zero, we find the minimum of our error function is given by

$$\sum_{n=0}^{J-1} \sum_{|\mathbf{k}|>k_c} c_{mk} c_{nk} t_n = \sum_{|\mathbf{k}|>k_c} x_k c_{mk}, \tag{26.63}$$

which is equivalent in form to equation (19) in [10], where we have x_k, instead of V_k. Thus, we see that we may optimize the short-range or long-range potential in simply by choosing to use V_k or x_k in the above equation. We now define

$$A_{mn} \equiv \sum_{|\mathbf{k}|>k_c} c_{mk} c_{nk} \tag{26.64}$$

$$b_m \equiv \sum_{|\mathbf{k}|>k_c} x_k c_{mk} \tag{26.65}$$

Thus, it becomes clear that our minimization equations can be cast in the canonical linear form,

$$\mathbf{A}\mathbf{t} = \mathbf{b}. \tag{26.66}$$

Solution by SVD

In practice, we note that the matrix $\mathbf{A}$ frequently becomes singular in practice. For this reason, we use the singular value decomposition to solve for t_n. This factorization decomposes A as

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{26.67}$$

where $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = 1$ and $\mathbf{S}$ is diagonal. In this form, we have

$$\mathbf{t} = \sum_{i=0}^{J-1} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{\mathbf{S}_{ii}} \right) \mathbf{V}_{(i)}, \tag{26.68}$$

where the parenthesized subscripts refer to columns. The advantage of this form is that if $\mathbf{S}_{ii}$ is zero or very near zero, the contribution of the i^{th} of $\mathbf{V}$, may be neglected, since it represents a numerical instability and has little physical meaning. It represents the fact that the system cannot distinguish between two linear combinations of the basis functions. Using the SVD in this manner is guaranteed to be stable. This decomposition is available in LAPACK in the DGESVD subroutine.

Constraining Values

Often, we wish to constrain the value of t_n to have a fixed value to enforce a boundary condition, for example. To do this, we define

$$\mathbf{b}' \equiv \mathbf{b} - t_n \mathbf{A}_{(n)}. \tag{26.69}$$

We then define $\mathbf{A}^*$ as $\mathbf{A}$ with the n^{th} row and column removed, and $\mathbf{b}^*$ as $\mathbf{b}'$ with the n^{th} element removed. Then we solve the reduced equation $\mathbf{A}^*\mathbf{t}^* = \mathbf{b}^*$, and finally insert t_n back into the appropriate place in $\mathbf{t}^*$ to recover the complete, constrained vector $\mathbf{t}$. This may be trivially generalized to an arbitrary number of constraints.

The LPQHI basis

The above discussion was general and independent of the basis used to represent $v^\ell(r)$. In this section, we introduce a convenient basis of localized interpolant functions, similar to those used for splines, which have a number of properties which are convenient for our purposes.

First, we divide the region from 0 to r_c into $M-1$ subregions, bounded above and below by points we term *knots*, defined by $r_j \equiv j\Delta$, where $\Delta \equiv r_c/(M-1)$. We then define compact basis elements, $h_{j\alpha}$ which span the region $[r_{j-1}, r_{j+1}]$, except for $j=0$ and $j=M$. For $j=0$, only the region $[r_0, r_1]$, while for $j=M$, only $[r_{M-1}, r_M]$. Thus the index j identify the knot the element is centered on, while α is an integer from 0 to 2 indicating one of three function shapes. The dual index can be mapped to the single index above by the relation, $n = 3j + \alpha$. The basis functions are then defined as

$$h_{j\alpha}(r) = \begin{cases} \Delta^\alpha \sum_{n=0}^{5} S_{\alpha n} \left(\frac{r-r_j}{\Delta}\right)^n, & r_j < r \le r_{j+1} \\ (-\Delta)^\alpha \sum_{n=0}^{5} S_{\alpha n} \left(\frac{r_j-r}{\Delta}\right)^n, & r_{j-1} < r \le r_j \\ 0, & \text{otherwise,} \end{cases} \tag{26.70}$$

where the matrix $S_{\alpha n}$ is given by

$$S = \begin{bmatrix} 1 & 0 & 0 & -10 & 15 & -6 \\ 0 & 1 & 0 & -6 & 8 & -3 \\ 0 & 0 & \frac{1}{2} & -\frac{3}{2} & \frac{3}{2} & -\frac{1}{2} \end{bmatrix}. \tag{26.71}$$

Figure 26.1 shows plots of these function shapes.

The basis functions have the property that at the left and right extremes, i.e. r_{j-1} and r_{j+1}, their values and first two derivatives are zero. At the center, r_j, we have the properties,

$$h_{j0}(r_j) = 1, \quad h'_{j0}(r_j) = 0, \quad h''_{j0}(r_j) = 0 \tag{26.72}$$
$$h_{j1}(r_j) = 0, \quad h'_{j1}(r_j) = 1, \quad h''_{j1}(r_j) = 0 \tag{26.73}$$
$$h_{j2}(r_j) = 0, \quad h'_{j2}(r_j) = 0, \quad h''_{j2}(r_j) = 1 \tag{26.74}$$

These properties allow the control of the value and first two derivatives of the represented function at any knot value simply by setting the coefficients of the basis functions centered around that knot. Used in combination with the method described in section 26.2.8 above, boundary conditions can easily be enforced. In our case, we wish require that

$$h_{M0} = v(r_c), \quad h_{M1} = v'(r_c), \quad \text{and} \quad h_{M2} = v''(r_c). \tag{26.75}$$

This ensures that v^s and its first two derivatives vanish at r_c.

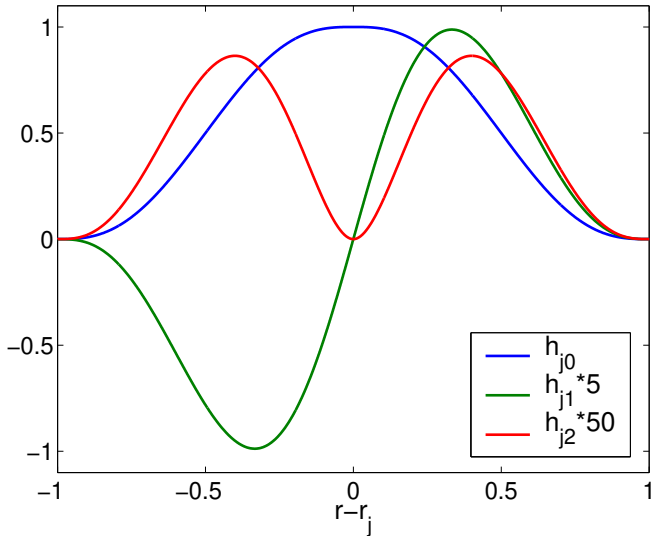


Figure 26.1: Basis functions h_{j0}, h_{j1}, and h_{j2} are shown. We note at the left and right extremes, the values and first two derivatives of the functions are zero, while at the center, h_{j0} has a value of 1, h_{j1} has a first derivative of 1, and h_{j2} has a second derivative of 1.

Fourier coefficients

We wish now to calculate the Fourier transforms of the basis functions, defined as

$$c_{j\alpha k} \equiv \frac{1}{\Omega}\int_0^{r_c} d^3\mathbf{r} e^{-i\mathbf{k}\cdot\mathbf{r}} h_{j\alpha}(r) \tag{26.76}$$

We then may write,

$$c_{j\alpha k} = \begin{cases} \Delta^\alpha \sum_{n=0}^5 S_{\alpha n} D^+_{0kn}, & j=0 \\ \Delta^\alpha \sum_{n=0}^5 S_{\alpha n}(-1)^{\alpha+n} D^-_{Mkn}, & j=M \\ \Delta^\alpha \sum_{n=0}^5 S_{\alpha n}\left[D^+_{jkn} + (-1)^{\alpha+n} D^-_{jkn}\right] & \text{otherwise,} \end{cases} \tag{26.77}$$

where

$$D^\pm_{jkn} \equiv \frac{1}{\Omega}\int_{r_j}^{r_{j\pm1}} d^3\mathbf{r}\ e^{-i\mathbf{k}\cdot\mathbf{r}} \left(\frac{r-r_j}{\Delta}\right)^n. \tag{26.78}$$

We then further make the definition that

$$D^\pm_{jkn} = \pm\frac{4\pi}{k\Omega}\left[\Delta \mathrm{Im}\left(E^\pm_{jk(n+1)}\right) + r_j \mathrm{Im}\left(E^\pm_{jkn}\right)\right] \tag{26.79}$$

It can then be shown that

$$E^\pm_{jkn} = \begin{cases} -\frac{i}{k} e^{ikr_j}\left(e^{\pm ik\Delta} - 1\right) & \text{if } n=0, \\ -\frac{i}{k}\left[(\pm1)^n e^{ik(r_j\pm\Delta)} - \frac{n}{\Delta} E^\pm_{jk(n-1)}\right] & \text{otherwise.} \end{cases} \tag{26.80}$$

Note that these equations correct typographical errors present in [10].

Enumerating k-points

We note that the summations over k which have been ubiquitous in this paper requires enumeration of the k-vectors. In particular, we should sum over all $|\mathbf{k}| > k_c$. In practice, we must limit our summation to some finite cutoff value $k_c < |\mathbf{k}| < k_{\max}$, where $k_{\max}$ should be of order $3000/L$, where L is the minimum box dimension. Enumerating these vectors in a naive fashion even for this finite cutoff would prove quite prohibitive, as it requires $\sim 10^9$ vectors.

Our first optimization come in realizing that all quantities in this calculation require only $|\mathbf{k}|$, and not $\mathbf{k}$ itself. Thus, we may take advantage of the great degeneracy of $|\mathbf{k}|$. We create a list of (k, N) pairs, where N is the number of vectors with magnitude k. We make nested loops over n_1, n_2, and n_3, yielding $\mathbf{k} = n_1\mathbf{b}_1 + n_2\mathbf{b}_2 + n_3\mathbf{b}_3$. If $|\mathbf{k}|$ is in the required range, we check to see if there is already an entry with that magnitude on our list, incrementing the corresponding N if there is, or creating a new entry if not. Doing so typically saves a factor of ~ 200 in storage and computation.

This reduction is still not sufficient for large k_max, since it requires that we still look over 10^9 entries. To further reduce cost, we may pick an intermediate cutoff, k_{cont}, above which we will approximate the degeneracy assuming a continuum of k-points. We stop our exact enumeration at k_{cont}, and then add ~ 1000 points, k_i, uniformly spaced between k_{cont} and $k_{\max}$. We then approximate the degeneracy by

$$N_i = \frac{4\pi}{3}\frac{\left(k_b^3 - k_a^3\right)}{(2\pi)^3/\Omega}, \tag{26.81}$$

where $k_b = (k_i + k_{i+1})/2$ and $k_a = (k_i + k_{i-1})$. In doing so, we typically reduce our total number of k-points to sum over ~ 2500 from the 10^9 we had to start.

Calculating x_k's

The coulomb potential

For $v(r) = \frac{1}{r}$, x_k is given by

$$x_k^{\text{coulomb}} = -\frac{4\pi}{\Omega k^2}\cos(kr_c) \tag{26.82}$$

The $1/r^2$ potential

For $v(r) = \frac{1}{r^2}$, x_k is given by

$$x_k^{1/r^2} = \frac{4\pi}{\omega k}\left[\text{Si}(kr_c) - \frac{\pi}{2}\right], \tag{26.83}$$

where the *sin integral*, $\text{Si}(z)$, is given by

$$\text{Si}(z) \equiv \int_0^z \frac{\sin\ t}{t} dt. \tag{26.84}$$

The $1/r^3$ potential

For $v(r) = \frac{1}{r^3}$, x_k is given by

$$x_k^{1/r^3} = \frac{4\pi}{\Omega k}\left[k\text{Ci}(kr_c) - \frac{\sin(kr_c)}{r_c}\right], \tag{26.85}$$

where the *cosine integral*, $\text{Ci}(z)$, is given by

$$\text{Ci}(z) \equiv -\int_z^\infty \frac{\cos t}{t} dt. \tag{26.86}$$

The $1/r^4$ potential

For $v(r) = \frac{1}{r^4}$, x_k is given by

$$x_k^{1/r^4} = -\frac{4\pi}{\Omega k}\left\{\frac{k\cos(kr_c)}{2r_c} + \frac{\sin(kr_c)}{2r_c^2} + \frac{k^2}{2}\left[\mathrm{Si}(kr_c) - \frac{\pi}{2}\right]\right\} \tag{26.87}$$

26.3 Feature: Optimized Long-Ranged Breakup (Ewald) 2

Given a lattice of vectors $\mathbf{L}$, its associated reciprocal lattice of vectors $\mathbf{k}$ and a function $\psi(\mathbf{r})$ periodic on the lattice we define its Fourier transform $\widetilde{\psi}(\mathbf{k})$ as

$$\widetilde{\psi}(\mathbf{k}) = \frac{1}{\Omega}\int_{\Omega} d\mathbf{r}\psi(\mathbf{r})e^{-i\mathbf{kr}} \tag{26.88}$$

where we indicated both the cell domain and the cell volume by Ω. $\psi(\mathbf{r})$ can then be expressed as

$$\psi(\mathbf{r}) = \sum_{\mathbf{k}} \widetilde{\psi}(\mathbf{k})e^{i\mathbf{kr}} \tag{26.89}$$

The potential generated by charges sitting on the lattice positions at a particular point $\mathbf{r}$ inside the cell is given by

$$V(\mathbf{r}) = \sum_{\mathbf{L}} v(|\mathbf{r}+\mathbf{L}|) \tag{26.90}$$

and its Fourier transform can be explicitly written as a function of V or v

$$\widetilde{V}(\mathbf{k}) = \frac{1}{\Omega}\int_{\Omega} d\mathbf{r}V(\mathbf{r})e^{-i\mathbf{kr}} = \frac{1}{\Omega}\int_{\mathbb{R}^3} d\mathbf{r}v(\mathbf{r})e^{-i\mathbf{kr}} \tag{26.91}$$

where $\mathbb{R}^3$ denotes the whole 3-dimensional space. We now want to find the best ("best" to be defined later) approximate potential of the form

$$V_a(\mathbf{r}) = \sum_{k\leq k_c} \widetilde{Y}(k)e^{i\mathbf{kr}} + W(r) \tag{26.92}$$

where $W(r)$ has been chosen to go to 0 smoothly when $r = r_c$, being r_c lower or equal to the Wigner-Seitz radius of the cell. Note also the cutoff k_c on the momentum summation.

The best form of $\widetilde{Y}(k)$ and $W(r)$ is given by minimizing

$$\chi^2 = \frac{1}{\Omega}\int d\mathbf{r}\left(V(\mathbf{r}) - W(\mathbf{r}) - \sum_{k\leq k_c}\widetilde{Y}(k)e^{i\mathbf{kr}}\right)^2 \tag{26.93}$$

or the reciprocal space equivalent

$$\chi^2 = \sum_{k\leq k_c}(\widetilde{V}(k) - \widetilde{W}(k) - \widetilde{Y}(k))^2 + \sum_{k>k_c}(\widetilde{V}(k) - \widetilde{W}(k))^2 \tag{26.94}$$

Eq.26.94 follows from Eq.26.93 and the unitarity (norm conservation) of the Fourier transform.

This last condition is minimized by

$$\widetilde{Y}(k) = \widetilde{V}(k) - \widetilde{W}(k) \qquad \min_{\widetilde{W}(k)} \sum_{k>k_c}(\widetilde{V}(k) - \widetilde{W}(k))^2 \tag{26.95}$$

We now use a set of basis function $c_i(r)$ vanishing smoothly at r_c to expand $W(r)$ i.e.

$$W(r) = \sum_i t_i c_i(r) \qquad \text{or} \qquad \widetilde{W}(k) = \sum_i t_i \widetilde{c}_i(k) \tag{26.96}$$

Inserting the reciprocal space expansion of $\widetilde{W}$ in the second condition of Eq.26.95 and minimizing with respect to t_i leads immediately to the linear system $\mathbf{At} = \mathbf{b}$ where

$$A_{ij} = \sum_{k>k_c}\widetilde{c}_i(k)\widetilde{c}_j(k) \qquad b_j = \sum_{k>k_c} V(k)\widetilde{c}_j(k) \tag{26.97}$$

26.3.1 Basis functions

The basis functions are splines. We define a uniform grid with $N_{\rm knot}$ uniformly spaced knots at position $r_i = i\frac{r_c}{N_{\rm knot}}$ where $i \in [0, N_{\rm knot} - 1]$. On each knot we center $m + 1$ piecewise polynomials $c_{i\alpha}(r)$ with $\alpha \in [0, m]$, defined as

$$c_{i\alpha}(r) = \begin{cases} \Delta^{\alpha} \sum_{n=0}^{\mathcal{N}} S_{\alpha n} (\frac{r-r_i}{\Delta})^n & r_i < r \leq r_{i+1} \\ \Delta^{-\alpha} \sum_{n=0}^{\mathcal{N}} S_{\alpha n} (\frac{r_i-r}{\Delta})^n & r_{i-1} < r \leq r_i \\ 0 & |r - r_i| > \Delta \end{cases} \tag{26.98}$$

These functions and their derivatives are, by construction, continuous and odd (even) (with respect to $r - r_i \to r_i - r$) when α is odd (even). We further ask them to satisfy

$$\frac{d^{\beta}}{dr^{\beta}} c_{i\alpha}(r)\Big|_{r=r_i} = \delta_{\alpha\beta} \quad \beta \in [0, m] \tag{26.99}$$

$$\frac{d^{\beta}}{dr^{\beta}} c_{i\alpha}(r)\Big|_{r=r_{i+1}} = 0 \quad \beta \in [0, m] \tag{26.100}$$

(The parity of the functions guarantees that the second constraint is satisfied at r_{i-1} as well). These constraints have a simple interpretation: the basis functions and their first m derivatives are 0 on the boundary of the subinterval where they are defined; the only function to have a non zero β-th derivative in r_i is $c_{i\beta}$. These $2(m + 1)$ constraints therefore impose $\mathcal{N} = 2m + 1$. Inserting the definitions of Eq.(26.98) in the constraints of Eq.(26.100) leads to the set of $2(m + 1)$ linear equation that fixes the value of $S_{\alpha n}$:

$$\Delta^{\alpha-\beta} S_{\alpha\beta} \beta! = \delta_{\alpha\beta} \tag{26.101}$$

$$\Delta^{\alpha-\beta} \sum_{n=\beta}^{2m+1} S_{\alpha n} \frac{n!}{(n-\beta)!} = 0 \tag{26.102}$$

One can further simplify inserting the first of these equations into the second and write the linear system as

$$\sum_{n=m+1}^{2m+1} S_{\alpha n} \frac{n!}{(n-\beta)!} = \begin{cases} -\frac{1}{(\alpha-\beta)!} & \alpha \geq \beta \\ 0 & \alpha < \beta \end{cases} \tag{26.103}$$

26.3.2 Fourier components of the basis functions in 3D

$k \neq 0$, non coulomb case.

We now need to evaluate the Fourier transform $\widetilde{c}_{i\alpha}(k)$. Let us start by writing the definition

$$\widetilde{c}_{i\alpha}(k) = \frac{1}{\omega} \int_{\Omega} d\mathbf{r} e^{-i\mathbf{kr}} c_{i\alpha}(r) \tag{26.104}$$

Because $c_{i\alpha}$ is different from zero only inside the spherical crown defined by $r_{i-1} < r < r_i$ one can conveniently compute the integral in spherical coordinates as

$$\widetilde{c}_{i\alpha}(k) = \Delta^{\alpha} \sum_{n=0}^{\mathcal{N}} S_{\alpha n} \left[D_{in}^{+}(k) + w_{\text{knot}}(-1)^{\alpha+n} D_{in}^{-}(k) \right] \tag{26.105}$$

where we used the definition $w_{\text{knot}} = 1 - \delta_{i0}$ and

$$D_{in}^{\pm}(k) = \pm \frac{4\pi}{k\Omega} \text{Im} \left[\int_{r_i}^{r_i \pm \Delta} dr \left(\frac{r - r_i}{\Delta} \right)^n r e^{ikr} \right] \tag{26.106}$$

obtained by integrating the angular part of the Fourier transform. Using the identity

$$\left(\frac{r - r_i}{\Delta} \right)^n r = \Delta \left(\frac{r - r_i}{\Delta} \right)^{n+1} + \left(\frac{r - r_i}{\Delta} \right)^n r_i \tag{26.107}$$

and the definition

$$E_{in}^{\pm}(k) = \int_{r_i}^{r_i \pm \Delta} dr \left(\frac{r - r_i}{\Delta} \right)^n e^{ikr} \tag{26.108}$$

we rewrite Eq.26.106 as

$$D_{in}^{\pm}(k) = \pm \frac{4\pi}{k\Omega} \text{Im} \left[\Delta E_{i(n+1)}^{\pm}(k) + r_i E_{in}^{\pm}(k) \right] \tag{26.109}$$

Finally, using integration by part, one can define $E_{in}^{\pm}$ recursively

$$E_{in}^{\pm}(k) = \frac{1}{ik} \left[(\pm)^n e^{ik(r_i \pm \Delta)} - \frac{n}{\Delta} E_{i(n-1)}^{\pm}(k) \right] \tag{26.110}$$

starting from the $n = 0$ term

$$E_{i0}^{\pm}(k) = \frac{1}{ik} e^{ikr_i} \left(e^{\pm ik\Delta} - 1 \right) \tag{26.111}$$

$k \neq 0$, coulomb case.

To efficiently treat the coulomb divergence at the origin it is convenient to use a basis set $c_{i\alpha}^{\text{coul}}$ of the form

$$c_{i\alpha}^{\text{coul}} = \frac{c_{i\alpha}}{r} \tag{26.112}$$

An equation identical to Eq.26.106 holds but with the modified definition

$$D_{in}^{\pm}(k) = \pm \frac{4\pi}{k\Omega} \text{Im} \left[\int_{r_i}^{r_i \pm \Delta} dr \left(\frac{r - r_i}{\Delta} \right)^n e^{ikr} \right] \tag{26.113}$$

which can be simply expressed using $E_{in}^{\pm}(k)$ as

$$D_{in}^{\pm}(k) = \pm \frac{4\pi}{k\Omega} \text{Im} \left[E_{in}^{\pm}(k) \right] \tag{26.114}$$

$k = 0$ **coulomb and non coulomb case.**

The definitions of $D_{in}(k)$ given so far are clearly incompatible with the choice $k = 0$ (they involve division by k). For the non-coulomb case the starting definition is

$$D_{in}^{\pm}(0) = \pm\frac{4\pi}{\Omega}\int_{r_i}^{r_i\pm\Delta} r^2\left(\frac{r-r_i}{\Delta}\right)^n dr \tag{26.115}$$

Using the definition $I_n^{\pm} = (\pm)^{n+1}\Delta/(n+1)$ we can express this as

$$D_{in}^{\pm}(0) = \pm\frac{4\pi}{\Omega}\left[\Delta^2 I_{n+2}^{\pm} + 2r_i\Delta I_{n+1}^{\pm} + 2r_i^2 I_n^{\pm}\right] \tag{26.116}$$

For the coulomb case one get

$$D_{in}^{\pm}(0) = \pm\frac{4\pi}{\Omega}\left(\Delta I_{n+1}^{\pm} + r_i I_n^{\pm}\right) \tag{26.117}$$

26.3.3 Fourier components of the basis functions in 2D

Eq.26.105 still holds provided we define

$$D_{in}^{\pm}(k) = \pm\frac{2\pi}{\Omega\Delta^n}\sum_{j=0}^{n}\binom{n}{j}(-r_i)^{n-j}\int_{r_i}^{r_i\pm\Delta} dr r^{j+1-C}J_0(kr) \tag{26.118}$$

where $C = 1(= 0)$ for the coulomb(non coulomb) case. Eq.26.118 is obtained using the integral definition of the zero order Bessel function of the first kind

$$J_0(z) = \frac{1}{\pi}\int_0^{\pi} e^{iz\cos\theta}d\theta \tag{26.119}$$

and the binomial expansion for $(r - r_i)^n$. The integrals can be computed recursively using the following identities

$$\int dz J_0(z) = \frac{z}{2}\left[\pi J_1(z)H_0(z) + J_0(z)(2 - \pi H_1(z))\right] \tag{26.120}$$

$$\int dz z J_0(z) = zJ_1(z) \tag{26.121}$$

$$\int dz z^n J_0(z) = z^n J_1(z) + (n-1)x^{n-1}J_0(z) - (n-1)^2\int dz z^{n-2}J_0(z) \tag{26.122}$$

Eq.26.122 is obtained using Eq.26.121, integration by part and the identity $\int J_1(z)dz = -J_0(z)$. In Eq.26.120 H_0 and H_1 are Struve functions.

26.3.4 Construction of the matrix elements

Using the above equations one can construct the matrix elements in Eq.26.97 and proceed solving for the t_i. It is sometimes desirable to put some constraints on the value of t_i. For example, when the coulomb potential is concerned one may want to set $t_0 = 1$. If the first g variable are constrained by $t_m = \gamma_m$ with $m = [1, g]$ one can simply redefine Eq.26.97 as

$$\begin{aligned} A_{ij} &= \sum_{k>k_c} \widetilde{c}_i(k)\widetilde{c}_j(k) \quad i,j \notin [1,g] \\ b_j &= \sum_{k>k_c} \left(\widetilde{V}(k) - \sum_{m=1}^{g} \gamma_m \widetilde{c}_m(k) \right) \widetilde{c}_j(k) \quad j \notin [1,g] \end{aligned} \tag{26.123}$$

26.4 Feature: Cubic Spline Interpolation

We present the basic equations and algorithms necessary to construct and evaluate cubic interpolating splines in one, two, and three dimensions. Equations are provided for both natural and periodic boundary conditions.

26.4.1 One Dimension

Let us consider the problem in which we have a function $y(x)$ specified at a discrete set of points x_i, such that $y(x_i) = y_i$. We wish to construct a piece-wise cubic polynomial interpolating function, $f(x)$, which satisfies the following conditions:

- $f(x_i) = y_i$
- $f'(x_i^-) = f'(x_i^+)$
- $f''(x_i^-) = f''(x_i+)$

Hermite Interpolants

In our piecewise representation, we wish to store only the values, y_i, and first derivatives, y'_i, of our function at each point x_i, which we call *knots*. Given this data, we wish to construct the piecewise cubic function to use between x_i and x_{i+1} which satisfies the above conditions. In particular, we wish to find the unique cubic polynomial, $P(x)$ satisfying

$$\begin{aligned} P(x_i) &= y_i && (26.124)\\ P(x_{i+1}) &= y_{i+1} && (26.125)\\ P'(x_i) &= y'_i && (26.126)\\ P'(x_{i+1}) &= y'_{i+1} && (26.127) \end{aligned}$$

$$\begin{aligned} h_i &\equiv x_{i+1} - x_i && (26.128)\\ t &\equiv \frac{x - x_i}{h_i}. && (26.129) \end{aligned}$$

We then define the basis functions,

$$\begin{aligned} p_1(t) &= (1+2t)(t-1)^2 && (26.130)\\ q_1(t) &= t(t-1)^2 && (26.131)\\ p_2(t) &= t^2(3-2t) && (26.132)\\ q_2(t) &= t^2(t-1) && (26.133) \end{aligned}$$

On the interval, $(x_i, x_{i+1}]$, we define the interpolating function,

$$P(x) = y_i p_1(t) + y_{i+1} p_2(t) + h\left[y'_i q_1(t) + y'_{i+1} q_2(t)\right] \tag{26.134}$$

It can be easily verified that $P(x)$ satisfies conditions (26.124) through (26.127). It is now left to determine the proper values for the y'_is such that the continuity conditions given above are satisfied.

By construction, the value of the function and derivative will match at the knots, i.e.

$$P(x_i^-) = P(x_i^+), \qquad P'(x_i^-) = P'(x_i^+). \tag{26.135}$$

Then we must now enforce only the second derivative continuity:

$$\begin{aligned} P''(x_i^-) &= P''(x_i^+) \qquad (26.136)\\ \frac{1}{h_{i-1}^2}\left[6y_{i-1} - 6y_i + h_{i-1}\left(2y'_{i-1} + 4y'_i\right)\right] &= \frac{1}{h_i^2}\left[-6y_i + 6y_{i+1} + h_i\left(-4y'_i - 2y'_{i+1}\right)\right] \end{aligned}$$

Let us define

$$\lambda_i \equiv \frac{h_i}{2(h_i + h_{i-1})} \qquad (26.137)$$

$$\mu_i \equiv \frac{h_{i-1}}{2(h_i + h_{i-1})} = \frac{1}{2} - \lambda_i. \qquad (26.138)$$

Then we may rearrange,

$$\lambda_i y'_{i-1} + y'_i + \mu_i y'_{i+1} = \underbrace{3\left[\lambda_i \frac{y_i - y_{i-1}}{h_{i-1}} + \mu_i \frac{y_{i+1} - y_i}{h_i}\right]}_{d_i} \qquad (26.139)$$

This equation holds for all $0 < i < (N-1)$, so we have a tridiagonal set of equations. The equations for $i = 0$ and $i = N-1$ depend on the boundary conditions we are using.

Periodic boundary conditions

For periodic boundary conditions, we have

$$\begin{array}{ccccccccccc} y'_0 & + & \mu_0 y'_1 & & & & \cdots & & +\lambda_0 y'_{N-1} & = & d_0 \\ \lambda_1 y'_0 & + & y'_1 & + & \mu_1 y'_2 & & \cdots & & & = & d_1 \\ & & \lambda_2 y'_1 & + & y'_2 + & \mu_2 y'_3 & \cdots & & & = & d_2 \\ & & & & \vdots & & & & & & \\ \mu_{N-1} y'_0 & & & & & & +\lambda_{N-1} y'_{N-1} & & +y'_{N-2} & = & d_3 \end{array} \qquad (26.140)$$

Or, in matrix form, we have,

$$\begin{pmatrix} 1 & \mu_0 & 0 & 0 & \cdots & 0 & \lambda_0 \\ \lambda_1 & 1 & \mu_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & 1 & \mu_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \lambda_{N-3} & 1 & \mu_{N-3} & 0 \\ 0 & 0 & 0 & 0 & \lambda_{N-2} & 1 & \mu_{N-2} \\ \mu_{N-1} & 0 & 0 & 0 & 0 & \lambda_{N-1} & 1 \end{pmatrix} \begin{pmatrix} y'_0 \\ y'_1 \\ y'_2 \\ \vdots \\ y'_{N-3} \\ y'_{N-2} \\ y'_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-3} \\ d_{N-2} \\ d_{N-1} \end{pmatrix}. \qquad (26.141)$$

The system is tridiagonal except for the two elements in the upper right and lower left corners. These terms complicate the solution a bit, although it can still be done in $\mathcal{O}(N)$ time. We first proceed down the rows, eliminating the the first non-zero term in each row by subtracting the appropriate multiple of the previous row. At the same time, we also eliminate the first element in the last row, shifting the position of the first non-zero element to the right with each iteration. When we get to the final row, we will have the value for y'_{N-1}. We can then proceed back upward, backsubstituting values from the rows below to calculate all the derivatives.

Complete boundary conditions

If we specify the first derivatives of our function at the end points, we have what is known as *complete* boundary conditions. The equations in that case are trivial to solve:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ \lambda_1 & 1 & \mu_1 & 0 & \dots & 0 & 0 \\ 0 & \lambda_2 & 1 & \mu_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \lambda_{N-3} & 1 & \mu_{N-3} & 0 \\ 0 & 0 & 0 & 0 & \lambda_{N-2} & 1 & \mu_{N-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y'_0 \\ y'_1 \\ y'_2 \\ \vdots \\ y'_{N-3} \\ y'_{N-2} \\ y'_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-3} \\ d_{N-2} \\ d_{N-1} \end{pmatrix}. \tag{26.142}$$

This system is completely tridiagonal and we may solve trivially by performing row eliminations downward, then proceeding upward as before.

Natural boundary conditions

If we do not have information about the derivatives at the boundary conditions, we may construct a *natural spline*, which assumes the the second derivatives are zero at the end points of our spline. In this case our system of equations is the following:

$$\begin{pmatrix} 1 & \frac{1}{2} & 0 & 0 & \dots & 0 & 0 \\ \lambda_1 & 1 & \mu_1 & 0 & \dots & 0 & 0 \\ 0 & \lambda_2 & 1 & \mu_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \lambda_{N-3} & 1 & \mu_{N-3} & 0 \\ 0 & 0 & 0 & 0 & \lambda_{N-2} & 1 & \mu_{N-2} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} y'_0 \\ y'_1 \\ y'_2 \\ \vdots \\ y'_{N-3} \\ y'_{N-2} \\ y'_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-3} \\ d_{N-2} \\ d_{N-1} \end{pmatrix}, \tag{26.143}$$

with

$$d_0 = \frac{3}{2}\frac{y_1 - y_1}{h_0}, \qquad d_{N-1} = \frac{3}{2}\frac{y_{N-1} - y_{N-2}}{h_{N-1}}. \tag{26.144}$$

26.4.2 Bicubic Splines

It is possible to extend the cubic spline interpolation method to functions of two variables, i.e. $F(x, y)$. In this case, we have a rectangular mesh of points given by $F_{ij} \equiv F(x_i, y_j)$. In the case of 1D splines, we needed to store the value of the first derivative of the function at each point, in addition to the value. In the case of *bicubic splines*, we need to store four quantities for each mesh point:

$$F_{ij} \equiv F(x_i, y_i) \tag{26.145}$$
$$F^x_{ij} \equiv \partial_x F(x_i, y_i) \tag{26.146}$$
$$F^y_{ij} \equiv \partial_y F(x_i, y_i) \tag{26.147}$$
$$F^{xy} \equiv \partial_x \partial_y F(x_i, y_i) \tag{26.148}$$

Consider the point (x, y) at which we wish to interpolate F. We locate the rectangle which contains this point, such that $x_i <= x < x_{i+1}$ and $y_i <= x < y_{i+1}$. Let

$$h \equiv x_{i+1} - x_i \tag{26.149}$$
$$l \equiv y_{i+1} - y_i \tag{26.150}$$
$$u \equiv \frac{x - x_i}{h} \tag{26.151}$$
$$v \equiv \frac{y - y_i}{l} \tag{26.152}$$

Then, we calculate the interpolated value as

$$F(x,y) = \begin{pmatrix} p_1(u) \\ p_2(u) \\ hq_1(u) \\ hq_2(u) \end{pmatrix}^T \begin{pmatrix} F_{i,j} & F_{i+1,j} & F^y_{i,j} & F^y_{i,j+1} \\ F_{i+1,j} & F_{i+1,j+1} & F^y_{i+1,j} & F^y_{i+1,j+1} \\ F^x_{i,j} & F^x_{i,j+1} & F^{xy}_{i,j} & F^{xy}_{i,j+1} \\ F^x_{i+1,j} & F^x_{i+1,j+1} & F^{xy}_{i+1,j} & F^{xy}_{i+1,j+1} \end{pmatrix} \begin{pmatrix} p_1(v) \\ p_2(v) \\ kq_1(v) \\ kq_2(v) \end{pmatrix} \tag{26.153}$$

Construction bicubic splines

We now address the issue of how to compute the derivatives that are needed for the interpolation. The algorithm is quite simple. For every x_i, we perform the tridiagonal solution as we did in the 1D splines to compute F^y_{ij}. Similarly, we perform a tridiagonal solve for every value of F^x_{ij}. Finally, in order to compute the cross-derivative we may *either* to the tridiagonal solve in the y direction of F^x_{ij}, *or* solve in the x direction for F^y_{ij} to obtain the cross-derivatives, F^{xy}_{ij}. Hence, only minor modifications to the $1D$ interpolations are necessary.

26.4.3 Tricubic Splines

Bicubic interpolation required two four-component vectors and a 4x4 matrix. By extension, tricubic interpolation requires three 4-component vectors and a 4x4x4 tensor. We summarize the forms of these vectors below.

$$h \equiv x_{i+1} - x_i \tag{26.154}$$
$$l \equiv y_{i+1} - y_i \tag{26.155}$$
$$m \equiv z_{i+1} - z_i \tag{26.156}$$
$$u \equiv \frac{x - x_i}{h} \tag{26.157}$$
$$v \equiv \frac{y - y_i}{l} \tag{26.158}$$
$$w \equiv \frac{z - z_i}{m} \tag{26.159}$$

$$\vec{a} = \begin{pmatrix} p_1(u) & p_2(u) & hq_1(u) & hq_2(u) \end{pmatrix}^T \tag{26.160}$$
$$\vec{b} = \begin{pmatrix} p_1(v) & p_2(v) & kq_1(v) & kq_2(v) \end{pmatrix}^T \tag{26.161}$$
$$\vec{c} = \begin{pmatrix} p_1(w) & p_2(w) & lq_1(w) & lq_2(w) \end{pmatrix}^T \tag{26.162}$$

$$\left(\begin{array}{llll}
A_{000}=F_{i,j,k} & A_{001}=F_{i,j,k+1} & A_{002}=F^{z}_{i,j,k} & A_{003}=F^{z}_{i,j,k+1}\\
A_{010}=F_{i,j+1,k} & A_{011}=F_{i,j+1,k+1} & A_{012}=F^{z}_{i,j+1,k} & A_{013}=F^{z}_{i,j+1,k+1}\\
A_{020}=F^{y}_{i,j,k} & A_{021}=F^{y}_{i,j,k+1} & A_{022}=F^{yz}_{i,j,k} & A_{023}=F^{yz}_{i,j,k+1}\\
A_{030}=F^{y}_{i,j+1,k} & A_{031}=F^{y}_{i,j+1,k+1} & A_{032}=F^{yz}_{i,j+1,k} & A_{033}=F^{yz}_{i,j+1,k+1}\\
\\
A_{100}=F_{i+1,j,k} & A_{101}=F_{i+1,j,k+1} & A_{102}=F^{z}_{i+1,j,k} & A_{103}=F^{z}_{i+1,j,k+1}\\
A_{110}=F_{i+1,j+1,k} & A_{111}=F_{i+1,j+1,k+1} & A_{112}=F^{z}_{i+1,j+1,k} & A_{113}=F^{z}_{i+1,j+1,k+1}\\
A_{120}=F^{y}_{i+1,j,k} & A_{121}=F^{y}_{i+1,j,k+1} & A_{122}=F^{yz}_{i+1,j,k} & A_{123}=F^{yz}_{i+1,j,k+1}\\
A_{130}=F^{y}_{i+1,j+1,k} & A_{131}=F^{y}_{i+1,j+1,k+1} & A_{132}=F^{yz}_{i+1,j+1,k} & A_{133}=F^{yz}_{i+1,j+1,k+1}\\
\\
A_{200}=F^{x}_{i,j,k} & A_{201}=F^{x}_{i,j,k+1} & A_{202}=F^{xz}_{i,j,k} & A_{203}=F^{xz}_{i,j,k+1}\\
A_{210}=F^{x}_{i,j+1,k} & A_{211}=F^{x}_{i,j+1,k+1} & A_{212}=F^{xz}_{i,j+1,k} & A_{213}=F^{xz}_{i,j+1,k+1}\\
A_{220}=F^{xy}_{i,j,k} & A_{221}=F^{xy}_{i,j,k+1} & A_{222}=F^{xyz}_{i,j,k} & A_{223}=F^{xyz}_{i,j,k+1}\\
A_{230}=F^{xy}_{i,j+1,k} & A_{231}=F^{xy}_{i,j+1,k+1} & A_{232}=F^{xyz}_{i,j+1,k} & A_{233}=F^{xyz}_{i,j+1,k+1}\\
\\
A_{300}=F^{x}_{i+1,j,k} & A_{301}=F^{x}_{i+1,j,k+1} & A_{302}=F^{xz}_{i+1,j,k} & A_{303}=F^{xz}_{i+1,j,k+1}\\
A_{310}=F^{x}_{i+1,j+1,k} & A_{311}=F^{x}_{i+1,j+1,k+1} & A_{312}=F^{xz}_{i+1,j+1,k} & A_{313}=F^{xz}_{i+1,j+1,k+1}\\
A_{320}=F^{xy}_{i+1,j,k} & A_{321}=F^{xy}_{i+1,j,k+1} & A_{322}=F^{xyz}_{i+1,j,k} & A_{323}=F^{xyz}_{i+1,j,k+1}\\
A_{330}=F^{xy}_{i+1,j+1,k} & A_{331}=F^{xy}_{i+1,j+1,k+1} & A_{332}=F^{xyz}_{i+1,j+1,k} & A_{333}=F^{xyz}_{i+1,j+1,k+1}
\end{array}\right) \tag{26.163}$$

Now, we can write

$$F(x,y,z)=\sum_{i=0}^{3}a_i\sum_{j=0}^{3}b_j\sum_{k=0}^{3}c_k\ A_{i,j,k} \tag{26.164}$$

The appropriate derivatives of F may be computed by a generalization of the method used for bicubic splines above.

26.5 Feature: B-spline Orbital Tiling (Band Unfolding)

In continuum quantum Monte Carlo simulations, it is necessary to evaluate the electronic orbitals of a system at real-space positions hundreds of millions of times. It has been found that if these orbitals are represented in a localized, B-spline basis, each evaluation takes a small, constant time which is independent of system size.

Unfortunately, the memory required for storing the B-spline grows with the second power of the system size. If we are studying perfect crystals, however, this can be reduced to linear scaling if we *tile* the primitive cell. In this approach, a supercell is constructed by tiling the primitive cell $N_1 \times N_2 \times N_3$ in the three lattice directions. The orbitals are then represented in real space only in the primitive cell, and an $N_1 \times N_2 \times N_3$ k-point mesh. To evaluate an orbital at any point in the supercell, it is only necessary to wrap that point back into the primitive cell, evaluate the spline, and then multiply the phase factor, $e^{-i\mathbf{k}\cdot\mathbf{r}}$.

Here, we show that this approach can be generalized to a tiling constructed with a 3×3 nonsingular matrix of integers, of which the above approach is a special case. This generalization brings with it a number of advantages. The primary reason for performing supercell calculations in QMC is to reduce finite-size errors. These errors result from three sources: 1) the quantization of the crystal momentum; 2) the unphysical periodicity of the exchange-correlation hole of the electron; and 3) the kinetic-energy contribution from the periodicity of the long-range jastrow correlation functions. The first source of error can be largely eliminated by twist averaging. If the simulation cell is large enough that XC hole does not "leak" out of the simulation cell, the second source can be eliminated either through use of the MPC interaction or the *a postiori* correction of Chiesa et. al.

The satisfaction of the leakage requirement is controlled by whether the minimum distance, L_{min} from one supercell image to the next is greater than the width of the XC hole. Therefore, given a choice, it is best to use a cell which is as nearly cubic as possible, since this choice maximizes L_{min} for a given number of atoms. Most often, however, the primitive cell is not cubic. In these cases, if we wish to choose the optimal supercell to reduce finite size effects, we cannot utilize the simple primitive tiling scheme. In the generalized scheme we present, it is possible to choose far better supercells (from the standpoint of finite-size errors), while retaining the storage efficiency of the original tiling scheme.

26.5.1 The mathematics

Consider the set of primitive lattice vectors, $\{\mathbf{a}_1^{\text{p}}, \mathbf{a}_2^{\text{p}}, \mathbf{a}_3^{\text{p}}\}$. We may write these vectors in a matrix, $\mathbf{L}_p$, whose rows are the primitive lattice vectors. Consider a non-singular matrix of integers, $\mathbf{S}$. A corresponding set of supercell lattice vectors, $\{\mathbf{a}_1^{\text{s}}, \mathbf{a}_2^{\text{s}}, \mathbf{a}_3^{\text{s}}\}$, can be constructed by the matrix product

$$\mathbf{a}_i^{\text{s}} = S_{ij}\mathbf{a}_j^{\text{p}} \tag{26.165}$$

If the primitive cell contains N_p atoms, the supercell will then contain $N_s = |\det(\mathbf{S})|N_p$ atoms.

26.5.2 Example: FeO

As an example, consider the primitive cell for antiferromagnetic FeO (wustite) in the rocksalt structure. The primitive vectors, given in units of the lattice constant, are given by

$$\mathbf{a}_1^s = \frac{1}{2}\hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{y}} + \hat{\mathbf{z}} \tag{26.166}$$

$$\mathbf{a}_2^s = \frac{1}{2}\hat{\mathbf{x}} + \hat{\mathbf{y}} + \frac{1}{2}\hat{\mathbf{z}} \tag{26.167}$$

$$\mathbf{a}_3^s = \hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{y}} + \frac{1}{2}\hat{\mathbf{z}} \tag{26.168}$$

This primitive cell contains two iron atoms and two oxygen atoms. It is a very elongated cell with acute angles, and thus has a short minimum distance between adjacent images.

The smallest cubic cell consistent with the AFM ordering can be constructed with the matrix

$$\mathbf{S} = \begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix} \tag{26.169}$$

This cell has $2\det(\mathbf{S}) = 32$ iron atoms and 32 oxygen atoms. In this example, we may perform the simulation in the 32-iron supercell, while storing the orbitals only in the 2-iron primitive cell, for a savings of a factor of 16.

The k-point mesh

In order to be able to use the generalized tiling scheme, we need to have the appropriate number of bands to occupy in the supercell. This may be achieved by appropriately choosing the k-point mesh. In this section, we explain how these points are chosen.

For simplicity, let us assume that the supercell calculation will be performed at the Γ-point. We may lift this restriction very easily later. The fact that supercell calculation is performed at Γ implies that the k-points used in the primitive-cell calculation must be **G**-vectors of the superlattice. This still leaves us with an infinite set of vectors. We may reduce this set to a finite number by considering that the orbitals must form an linearly independent set. Orbitals with k-vectors $\mathbf{k}_1^p$ and $\mathbf{k}_2^p$ will differ by at most a constant factor if $\mathbf{k}_1^p - \mathbf{k}_2^p = \mathbf{G}^p$, where $\mathbf{G}^p$ is a reciprocal lattice vector of the primitive cell.

Combining these two considerations gives us a prescription for generating our k-point mesh. The mesh may be taken to be the set of k-point which are G-vectors of the superlattice, reside within the first Brillouin zone (FBZ) of the primitive lattice, whose members do not differ a G-vector of the primitive lattice. Upon constructing such a set, we find that the number of included k-points is equal to $|\det(\mathbf{S})|$, precisely the number we need. This can by considering the fact that the supercell has a volume $|\det(\mathbf{S})|$ times that of the primitive cell. This implies that the volume of the supercell's FBZ is $|\det(\mathbf{S})|^{-1}$ times that of the primitive cell. Hence, $|\det(\mathbf{S})|$ G-vectors of the supercell will fit in the FBZ of the primitive cell. Removing duplicate k-vectors, which differ from another by a reciprocal lattice vector, avoids double-counting vectors which lie on zone faces.

Formulae

Let $\mathbf{A}$ be the matrix whose rows are the direct lattice vectors, $\{\mathbf{a}_i\}$. The, let the matrix $\mathbf{B}$ be defined as $2\pi(\mathbf{A}^{-1})^\dagger$. Its rows are the primitive reciprocal lattice vectors. Let $\mathbf{A}_p$ and $\mathbf{A}_s$ represent

the primitive and superlattice matrices, respectively, and similarly for their reciprocals. Then we have

$$\begin{aligned}
\mathbf{A}_s &= \mathbf{S}\mathbf{A}_p && (26.170)\\
\mathbf{B}_s &= 2\pi\left[(\mathbf{S}\mathbf{A}_p)^{-1}\right]^\dagger && (26.171)\\
&= 2\pi\left[\mathbf{A}_p^{-1}\mathbf{S}{-}1\right]^\dagger && (26.172)\\
&= 2\pi(\mathbf{S}^{-1})^\dagger(\mathbf{A}_p^{-1})^\dagger && (26.173)\\
&= (\mathbf{S}^{-1})^\dagger\mathbf{B}_p && (26.174)
\end{aligned}$$

Consider a k-vector, $\mathbf{k}$. It may be alternatively be written in basis of reciprocal lattice vectors as $\mathbf{t}$.

$$\begin{aligned}
\mathbf{k} &= (\mathbf{t}^\dagger\mathbf{B})^\dagger && (26.175)\\
&= \mathbf{B}^\dagger\mathbf{t} && (26.176)\\
\mathbf{t} &= (\mathbf{B}^\dagger)^{-1}\mathbf{k} && (26.177)\\
&= (\mathbf{B}^{-1})^\dagger\mathbf{k} && (26.178)\\
&= \frac{\mathbf{A}\mathbf{k}}{2\pi} && (26.179)
\end{aligned}$$

We may then express a twist vector of the primitive lattice, $\mathbf{t}_p$ in terms of the superlattice.

$$\begin{aligned}
\mathbf{t}_s &= \frac{\mathbf{A}_s\mathbf{k}}{2\pi} && (26.180)\\
&= \frac{\mathbf{A}_s\mathbf{B}_p^\dagger\mathbf{t}_p}{2\pi} && (26.181)\\
&= \frac{\mathbf{S}\mathbf{A}_p\mathbf{B}_p^\dagger\mathbf{t}_p}{2\pi} && (26.182)\\
&= \frac{2\pi\mathbf{S}\mathbf{A}_p\mathbf{A}_p^{-1}\mathbf{t}_p}{2\pi} && (26.183)\\
&= \mathbf{S}\mathbf{t}_p && (26.184)
\end{aligned}$$

This gives the simple result that twist-vectors transform in precisely the same way as direct lattice vectors.

26.6 Feature: Hybrid orbital representation

$$\phi(\mathbf{r}) = \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} Y_\ell^m(\hat{\Omega}) u_{\ell m}(r), \tag{26.185}$$

where $u_{lm}(r)$ are complex radial functions represented in some radial basis (e.g. splines).

26.6.1 Real Spherical Harmonics

If $\phi(\mathbf{r})$ can be written as purely real, we can change the representation so that

$$\phi(\mathbf{r}) = \sum_{l=0}^{l_{\max}} \sum_{m=-\ell}^{\ell} Y_{\ell m}(\hat{\Omega}) \bar{u}_{lm}(r), \tag{26.186}$$

where $\bar{Y}_\ell^m$ are the *real* spherical harmonics defined by

$$Y_{\ell m} = \begin{cases} Y_\ell^0 & \text{if } m = 0 \\ \frac{1}{2}\left(Y_\ell^m + (-1)^m Y_\ell^{-m}\right) = \mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right] & \text{if } m > 0 \\ \frac{1}{i2}\left(Y_\ell^{-m} - (-1)^m Y_\ell^m\right) = \mathrm{Im}\left[\mathrm{Y}_\ell^{-\mathrm{m}}\right] & \text{if } m < 0. \end{cases} \tag{26.187}$$

We need then to relate $\bar{u}_{\ell m}$ to $u_{\ell m}$. We wish to express,

$$\mathrm{Re}\left[\phi(\mathbf{r})\right] = \sum_{\ell=0}^{\ell_{\max}} \sum_{\mathrm{m}=-\ell}^{\ell} \mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}(\mathbf{r})\mathrm{u}_{\ell\mathrm{m}}(\mathrm{r})\right] \tag{26.188}$$

in terms of $\bar{u}_{\ell m}(r)$ and $Y_{\ell m}$.

$$\mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}\mathrm{u}_{\ell\mathrm{m}}\right] = \mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right]\mathrm{Re}\left[\mathrm{u}_{\ell\mathrm{m}}\right] - \mathrm{Im}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right]\mathrm{Im}\left[\mathrm{u}_{\ell\mathrm{m}}\right] \tag{26.189}$$

For $m > 0$,

$$\mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right] = \mathrm{Y}_{\ell\mathrm{m}} \qquad \text{and} \qquad \mathrm{Im}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right] = \mathrm{Y}_{\ell\,-\mathrm{m}}. \tag{26.190}$$

For $m < 0$,

$$\mathrm{Re}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right] = (-1)^{\mathrm{m}}\mathrm{Y}_{\ell\,-\mathrm{m}} \qquad \text{and} \qquad \mathrm{Im}\left[\mathrm{Y}_\ell^{\mathrm{m}}\right] = -(-1)^{\mathrm{m}}\mathrm{Y}_{\ell\mathrm{m}}. \tag{26.191}$$

Then for $m > 0$,

$$\bar{u}_{\ell m} = \mathrm{Re}\left[\mathrm{u}_{\ell\mathrm{m}}\right] + (-1)^{\mathrm{m}}\mathrm{Re}\left[\mathrm{u}_{\ell\,-\mathrm{m}}\right] \tag{26.192}$$

$$\bar{u}_{\ell\,-m} = -\mathrm{Im}\left[\mathrm{u}_{\ell\mathrm{m}}\right] + (-1)^{\mathrm{m}}\mathrm{Im}\left[\mathrm{u}_{\ell\,-\mathrm{m}}\right]. \tag{26.193}$$

26.6.2 Projecting to atomic orbitals

Inside a muffin tin, orbitals are represented as product of spherical harmonics and 1D radial functions, primarily represented by splines. For a muffin tin centered at $\mathbf{I}$,

$$\phi_n(\mathbf{r}) = \sum_{\ell,m} Y_\ell^m(\mathbf{r} \hat{-} \mathbf{I}) u_{lm}\left(|\mathbf{r} - \mathbf{I}|\right) \tag{26.194}$$

Let use consider the case that our original representation for $\phi(\mathbf{r})$ is of the form

$$\phi_{n,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{G}} c^n_{\mathbf{G}+\mathbf{k}} e^{i(\mathbf{G}+\mathbf{k})\cdot\mathbf{r}} \tag{26.195}$$

Recall that

$$e^{i\mathbf{k}\cdot\mathbf{r}} = 4\pi \sum_{\ell,m} i^\ell j_\ell(|\mathbf{r}||\mathbf{k}|) Y_\ell^m(\hat{\mathbf{k}}) \left[Y_\ell^m(\hat{\mathbf{r}})\right]^* . \tag{26.196}$$

Conjugating,

$$e^{-i\mathbf{k}\cdot\mathbf{r}} = 4\pi \sum_{\ell,m} (-i)^\ell j_\ell(|\mathbf{r}||\mathbf{k}|) \left[Y_\ell^m(\hat{\mathbf{k}})\right]^* Y_\ell^m(\hat{\mathbf{r}}). \tag{26.197}$$

Setting $\mathbf{k} \to -k$,

$$e^{i\mathbf{k}\cdot\mathbf{r}} = 4\pi \sum_{\ell,m} i^\ell j_\ell(|\mathbf{r}||\mathbf{k}|) \left[Y_\ell^m(\hat{\mathbf{k}})\right]^* Y_\ell^m(\hat{\mathbf{r}}). \tag{26.198}$$

Then,

$$e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{I})} = 4\pi \sum_{\ell,m} i^\ell j_\ell(|\mathbf{r}-\mathbf{I}||\mathbf{k}|) \left[Y_\ell^m(\hat{\mathbf{k}})\right]^* Y_\ell^m(\mathbf{r} \hat{-} \mathbf{I}). \tag{26.199}$$

$$e^{i\mathbf{k}\cdot\mathbf{r}} = 4\pi e^{i\mathbf{k}\cdot\mathbf{I}} \sum_{\ell,m} i^\ell j_\ell(|\mathbf{r}-\mathbf{I}||\mathbf{k}|) \left[Y_\ell^m(\hat{\mathbf{k}})\right]^* Y_\ell^m(\mathbf{r} \hat{-} \mathbf{I}). \tag{26.200}$$

Then

$$\phi_{n,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{G}} 4\pi c^n_{\mathbf{G}+\mathbf{k}} e^{i(\mathbf{G}+\mathbf{k})\cdot\mathbf{I}} \sum_{\ell,m} i^\ell j_\ell(|\mathbf{G}+\mathbf{k}||\mathbf{r}-\mathbf{I}|) \left[Y_\ell^m(\mathbf{G} \hat{+} \mathbf{k})\right]^* Y_\ell^m(\mathbf{r} \hat{-} \mathbf{I}) \tag{26.201}$$

Comparing to (26.194),

$$u^n_{\ell m}(r) = 4\pi i^\ell \sum_G c^n_{\mathbf{G}+\mathbf{k}} e^{i(\mathbf{G}+\mathbf{k})\cdot\mathbf{I}} j_\ell\left(|\mathbf{G}+\mathbf{k}|r|\right) \left[Y_\ell^m(\mathbf{G} \hat{+} \mathbf{k})\right]^* . \tag{26.202}$$

If we had adopted the opposite sign convention for Fourier transforms (as is unfortunately the case in wfconvert), we would have

$$u^n_{\ell m}(r) = 4\pi (-i)^\ell \sum_G c^n_{\mathbf{G}+\mathbf{k}} e^{-i(\mathbf{G}+\mathbf{k})\cdot\mathbf{I}} j_\ell\left(|\mathbf{G}+\mathbf{k}|r|\right) \left[Y_\ell^m(\mathbf{G} \hat{+} \mathbf{k})\right]^* . \tag{26.203}$$

26.7 Feature: Electron-electron-ion Jastrow factor

The general form of the 3-body Jastrow we describe here depends on the three inter-particle distances, (r_{ij}, r_{iI}, r_{jI}).

$$J_3 = \sum_{I \in \text{ions}} \sum_{i,j \in \text{elecs}; i \neq j} U(r_{ij}, r_{iI}, r_{jI}) \tag{26.204}$$

Note that we constrain the form of U such that $U(r_{ij}, r_{iI}, r_{jI}) = U(r_{ij}, r_{jI}, r_{iI})$, so as to preserve the particle symmetry of the wave function. We then compute the gradient as

$$\nabla_i J_3 = \sum_{I \in \text{ions}} \sum_{j \neq i} \left[\frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{ij}} \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} + \frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{iI}} \frac{\mathbf{r}_i - \mathbf{I}}{|\mathbf{r}_i - \mathbf{I}|} \right] \tag{26.205}$$

To compute the laplacian, we take

$$\begin{aligned} \nabla_i^2 J_3 &= \nabla_i \cdot (\nabla_i J_3) \\ &= \sum_{I \in \text{ions}} \sum_{j \neq i} \left[\frac{\partial^2 U}{\partial r_{ij}^2} + \frac{2}{r_{ij}} \frac{\partial U}{\partial r_{ij}} + 2 \frac{\partial^2 U}{\partial r_{ij} \partial r_{iI}} \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{iI}}{r_{ij} r_{iI}} + \frac{\partial^2 U}{\partial r_{iI}^2} + \frac{2}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \right] \end{aligned} \tag{26.206}$$

We now wish to compute the gradient of these terms w.r.t. the ion position, I.

$$\nabla_I J_3 = -\sum_{j \neq i} \left[\frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{iI}} \frac{\mathbf{r}_i - \mathbf{I}}{|\mathbf{r}_i - \mathbf{I}|} + \frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{jI}} \frac{\mathbf{r}_j - \mathbf{I}}{|\mathbf{r}_j - \mathbf{I}|} \right] \tag{26.207}$$

For the gradient w.r.t. i of the gradient w.r.t. I, the result is a tensor,

$$\begin{aligned} \nabla_I \nabla_i J_3 &= \nabla_I \sum_{j \neq i} \left[\frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{ij}} \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} + \frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{iI}} \frac{\mathbf{r}_i - \mathbf{I}}{|\mathbf{r}_i - \mathbf{I}|} \right] \\ &= -\sum_{j \neq i} \left[\frac{\partial^2 U}{\partial r_{ij} r_{iI}} \hat{\mathbf{r}}_{ij} \otimes \hat{\mathbf{r}}_{iI} + \left(\frac{\partial^2 U}{\partial r_{iI}^2} - \frac{1}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \right) \hat{\mathbf{r}}_{iI} \otimes \hat{\mathbf{r}}_{iI} + \right. \\ &\qquad \left. \frac{\partial^U}{\partial r_{ij} r_{jI}} \hat{\mathbf{r}}_{ij} \otimes \hat{\mathbf{r}}_{jI} + \frac{\partial^2 U}{\partial r_{iI} \partial r_{jI}} \hat{\mathbf{r}}_{iI} \otimes \hat{\mathbf{r}}_{jI} + \frac{1}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \overleftrightarrow{\mathbf{1}} \right] \end{aligned} \tag{26.208}$$

$$\begin{aligned} \nabla_I \nabla_i J_3 &= \nabla_I \sum_{j \neq i} \left[\frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{ij}} \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} + \frac{\partial U(r_{ij}, r_{iI}, r_{jI})}{\partial r_{iI}} \frac{\mathbf{r}_i - \mathbf{I}}{|\mathbf{r}_i - \mathbf{I}|} \right] \\ &= \sum_{j \neq i} \left[-\frac{\partial^2 U}{\partial r_{ij} \partial r_{iI}} \hat{\mathbf{r}}_{ij} \otimes \hat{\mathbf{r}}_{iI} + \left(-\frac{\partial^2 U}{\partial r_{iI}^2} + \frac{1}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \right) \hat{\mathbf{r}}_{iI} \otimes \hat{\mathbf{r}}_{iI} - \frac{1}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \overleftrightarrow{\mathbf{1}} \right] \end{aligned} \tag{26.209}$$

For the laplacian,

$$
\begin{aligned}
\nabla_I \nabla_i^2 J_3 &= \nabla_I \left[\nabla_i \cdot (\nabla_i J_3) \right] \\
&= \nabla_I \sum_{j \neq i} \left[\frac{\partial^2 U}{\partial r_{ij}^2} + \frac{2}{r_{ij}} \frac{\partial U}{\partial r_{ij}} + 2 \frac{\partial^2 U}{\partial r_{ij} \partial r_{iI}} \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{iI}}{r_{ij} r_{iI}} + \frac{\partial^2 U}{\partial r_{iI}^2} + \frac{2}{r_{iI}} \frac{\partial U}{\partial r_{iI}} \right] \\
&= \sum_{j \neq i} \left[\frac{\partial^3 U}{\partial r_{iI} \partial^2 r_{ij}} + \frac{2}{r_{ij}} \frac{\partial^2 U}{\partial r_{iI} \partial r_{ij}} + 2 \left(\frac{\partial^3 U}{\partial r_{ij} \partial^2 r_{iI}} - \frac{1}{r_{iI}} \frac{\partial^2 U}{\partial r_{ij} \partial r_{iI}} \right) \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{iI}}{r_{ij} r_{iI}} + \frac{\partial^3 U}{\partial^3 r_{iI}} - \frac{2}{r_{iI}^2} \frac{\partial U}{\partial r_{iI}} + \frac{2}{r_{iI}} \frac{\partial^2 U}{\partial^2 r_{iI}} \right] \\
&\quad \sum_{j \neq i} \left[\frac{\partial^3 U}{\partial r_{ij}^2 \partial r_{jI}} + \frac{2}{r_{ij}} \frac{\partial^2 U}{\partial r_{jI} \partial r_{ij}} + 2 \frac{\partial^3 U}{\partial r_{ij} \partial r_{iI} \partial r_{jI}} \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{iI}}{r_{ij} r_{iI}} + \frac{\partial^3 U}{\partial r_{iI}^2 \partial r_{jI}} + \frac{2}{r_{iI}} \frac{\partial^2 U}{\partial r_{iI} \partial r_{jI}} \right] \frac{\mathbf{I} - \mathbf{r}_j}{|\mathbf{r}_j - \mathbf{I}|} + \\
&\quad \sum_{j \neq i} \left[-\frac{2}{r_{iI}} \frac{\partial^2 U}{\partial r_{ij} \partial r_{iI}} \right] \frac{\mathbf{r}_{ij}}{r_{ij}}
\end{aligned}
$$

26.8 Feature: Reciprocal-Space Jastrow Factors

26.8.1 Two-body Jastrow

$$J_2 = \sum_{\mathbf{G}\neq\mathbf{0}} \sum_{i\neq j} a_{\mathbf{G}} e^{i\mathbf{G}\cdot(\mathbf{r}_i-\mathbf{r}_j)} \tag{26.211}$$

This may be rewritten as

$$J_2 = \sum_{\mathbf{G}\neq\mathbf{0}} \sum_{i\neq j} a_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}_i} e^{-i\mathbf{G}\cdot\mathbf{r}_j} \tag{26.212}$$

$$= \sum_{\mathbf{G}\neq\mathbf{0}} a_{\mathbf{G}} \left\{ \underbrace{\left[\sum_i e^{i\mathbf{G}\cdot\mathbf{r}_i}\right]}_{\rho_{\mathbf{G}}} \underbrace{\left[\sum_j e^{-i\mathbf{G}\cdot\mathbf{r}_j}\right]}_{\rho_{-\mathbf{G}}} - 1 \right\} \tag{26.213}$$

The -1 is just a constant term and may be subsumed into the $a_{\mathbf{G}}$ coefficient by a simple redefinition. This leaves a simple, but general, form:

$$J_2 = \sum_{\mathbf{G}\neq\mathbf{0}} a_{\mathbf{G}} \rho_{\mathbf{G}} \rho_{-\mathbf{G}} \tag{26.214}$$

We may now further constrain this on physical grounds. First, we recognize that J_2 should be real. Since $\rho_{-\mathbf{G}} = \rho^*_{\mathbf{G}}$, it follows that $\rho_{\mathbf{G}}\rho_{-\mathbf{G}} = |\rho_{\mathbf{G}}|^2$ is real, so that $a_{\mathbf{G}}$ must be real. Furthermore, we group the $\mathbf{G}$'s into $(+\mathbf{G}, -\mathbf{G})$ pairs, and sum over only the positive vectors to save time.

26.8.2 One-body Jastrow

The one-body Jastrow has a similar form, but depends on the displacement from the electrons to the ions in the system.

$$J_1 = \sum_{\mathbf{G}\neq\mathbf{0}} \sum_{\alpha} \sum_{i\in\mathbf{I}^\alpha} \sum_{j\in\text{elec.}} b^\alpha_{\mathbf{G}} e^{i\mathbf{G}\cdot(\mathbf{I}^\alpha_i-\mathbf{r}_j)}, \tag{26.215}$$

where α denotes the different ionic species. We may rewrite this in terms of $\rho^\alpha_{\mathbf{G}}$,

$$J_1 = \sum_{\mathbf{G}\neq\mathbf{0}} \left[\sum_\alpha b^\alpha_{\mathbf{G}} \rho^\alpha_{\mathbf{G}}\right] \rho_{-\mathbf{G}}, \tag{26.216}$$

where

$$\rho^\alpha_{\mathbf{G}} = \sum_{i\in\mathbf{I}^\alpha} e^{i\mathbf{G}\cdot\mathbf{I}^\alpha_i}. \tag{26.217}$$

We note that in the above equation, for a single configuration of the ions, the sum in brackets can be rewritten as a single constant. This implies that the per-species one-body coefficients, $b^\alpha_{\mathbf{G}}$, are underdetermined for single configuration of the ions. In general, if we have N species, we need N linearly independent ion configurations to uniquely determine $b^\alpha_{\mathbf{G}}$. For this reason, we will drop the α superscript of $b_{\mathbf{G}}$ for now.

If we do desire to find a reciprocal space one-body Jastrow that is transferable to systems with different ion positions and N ionic species, we must perform compute $b_{\mathbf{G}}$ for N different ion configurations. We may then construct N equations at each value of $\mathbf{G}$ to solve for the N unknown values, $b^\alpha_{\mathbf{G}}$.

In the two-body case, $a_{\mathbf{G}}$ was constrained to be real by the fact that $\rho_{\mathbf{G}}\rho_{-\mathbf{G}}$ was real. However, in the one-body case, there is no such guarantee about $\rho^{\alpha}_{\mathbf{G}}\rho_{\mathbf{G}}$. Therefore, in general, $b_{\mathbf{G}}$ may be complex.

26.8.3 Symmetry considerations

For a crystal, many of the **G**-vectors will be equivalent by symmetry. It is useful then, to divide the **G**-vectors into symmetry-related groups and then to require that they share a common coefficient. Two vectors, **G** and **G**$'$, may be considered symmetry related if, for all α and β,

$$\rho^{\alpha}_{\mathbf{G}}\rho^{\beta}_{-\mathbf{G}} = \rho^{\alpha}_{\mathbf{G}'}\rho^{\beta}_{-\mathbf{G}'}. \tag{26.218}$$

For the one-body term, we may also omit from our list of **G**-vectors those for which all species structure factors are zero. This is equivalent to saying that, if we are tiling a primitive cell, we should include only the **G**-vectors of the primitive cell, and not the supercell. Note that this is not the case for the two-body term, since the exchange-correlation hole should not have the periodicity of the primitive cell.

26.8.4 Gradients and Laplacians

$$\begin{aligned}
\nabla_{\mathbf{r}_i} J_2 &= \sum_{\mathbf{G}\neq 0} a_{\mathbf{G}} \left[\left(\nabla_{\mathbf{r}_i}\rho_{\mathbf{G}}\right)\rho_{-\mathbf{G}} + \text{c.c.}\right] && (26.219)\\
&= \sum_{\mathbf{G}\neq \mathbf{0}} 2\mathbf{G}a_{\mathbf{G}}\mathbf{Re}\left(ie^{i\mathbf{G}\cdot\mathbf{r}_i}\rho_{-\mathbf{G}}\right) && (26.220)\\
&= \sum_{\mathbf{G}\neq \mathbf{0}} -2\mathbf{G}a_{\mathbf{G}}\mathbf{Im}\left(e^{i\mathbf{G}\cdot\mathbf{r}_i}\rho_{-\mathbf{G}}\right) && (26.221)
\end{aligned}$$

The Laplacian is then given by

$$\begin{aligned}
\nabla^2 J_2 &= \sum_{\mathbf{G}\neq \mathbf{0}} a_{\mathbf{G}} \left[\left(\nabla^2\rho_{\mathbf{G}}\right)\rho_{-\mathbf{G}} + \text{c.c.} + 2\left(\nabla\rho_{\mathbf{G}}\right)\cdot\left(\nabla\rho_{-\mathbf{G}}\right)\right] && (26.222)\\
&= \sum_{\mathbf{G}\neq \mathbf{0}} a_{\mathbf{G}} \left[-2G^2\mathbf{Re}(e^{i\mathbf{G}\cdot\mathbf{r}_i}\rho_{-\mathbf{G}}) + 2\left(i\mathbf{G}e^{i\mathbf{G}\cdot\mathbf{r}_i}\right)\cdot\left(-i\mathbf{G}e^{-i\mathbf{G}\cdot\mathbf{r}_i}\right)\right] && (26.223)\\
&= 2\sum_{\mathbf{G}\neq \mathbf{0}} G^2 a_{\mathbf{G}} \left[-\mathbf{Re}\left(e^{i\mathbf{G}\cdot\mathbf{r}_i}\rho_{-\mathbf{G}}\right) + 1\right] && (26.224)
\end{aligned}$$

Chapter 27

Development Guide

The section gives guidance on how to extend the functionality of QMCPACK. Future examples will likely include topics such as the addition of a jastrow function or add a new QMC method.

27.1 QMCPACK Coding Standards

The following document presents what we collectively have agreed are best practices for the code. This includes both a formatting style, naming conventions, documentation conventions, and certain prescriptions for C++ languague use. At the moment only the formatting can be enforced in an objective fashion.

New development should follow these guidelines and contributors are expected to adhere to them as they represent an integral part of our effort to continue QMCPACK as a worldclass and sustainable quantum Monte Carlo code. While some of the source code has a ways to go to live up to these ideas, new code even in old files should follow the new conventions not the local conventions of the file whenever possible. Work on the code with continuous improvement in mind and not a commitment to stasis.

At anytime the current workflow conventions for the project are described in the wiki on the github repository. It will save you and all the maintainers considerable time if you read these and ask questions up front.

A PR should follow these standards before inclusion in the mainline. You can be sure of properly following the formatting conventions if you use clang-format. The mechanics of clang-format setup and use are can be found at `https://github.com/QMCPACK/qmcpack/wiki/Source-formatting`.

The clang-format file found at `qmcpack/src/.clang-format` that should be run over all code touched in a PR before a pull request is prepared. We also encourage developers to run clang-tidy with the `qmcpack/src/.clang-tidy` configuration over all new code.

As much as possible try to break up refactoring, reformatting, feature, and bugs into separate small PR's. Aim for something that would take a reviewer no more than an hour. In this way we can maintain a good collective development velocity.

27.2 Files

Each file should start with the header

```
//////////////////////////////////////////////////////////////////////////////////////
// This file is distributed under the University of Illinois/NCSA Open Source License.
```

```
// See LICENSE file in top directory for details.
//
// Copyright (c) 2018 QMCPACK developers
//
// File developed by: Name, email, affiliation
//
// File created by: Name, email, affiliation
//////////////////////////////////////////////////////////////////////////////////////
```

If you make significant changes to an existing file, add yourself to the list of developed by authors.

27.2.1 File organization

Header files should be placed in the same directory as their implementations. Unit tests should be written for all new functionality. These tests should be placed in a `tests` subdirectory below the implementations.

27.2.2 File Names

Each class should be defined in a separate file whose name is the same as the class name. Use separate `.cpp` implementation files whenever possible to aid in incremental compilation.

The filenames of tests are composed by the filename of the object tested and the prefix `test_`. The filenames of *fake* and *mock* objects used in tests are composed by the prefixes `fake_` and `mock_`, respectively, and the filename of the object that is imitated.

27.2.3 Header files

All header files should be self-contained i.e. it is not dependent on following any other header when it is included. Nor should they include files that are not necessary for their use, i.e. headers only needed by the implementation. Implementation files should not include files only for the benefit of files they include.

There are many header files that violate this currently. Each header must use `#define` guards to prevent multiple inclusion. The symbol name of the `#define` guards should be `NAMESPACE(s)_CLASSNAME_H`.

27.2.4 Includes

Header files should be included with the full path based on the `src` directory. For example, the file `qmcpack/src/QMCWaveFunctions/SPOSet.h` should be included as

```
#include "QMCWaveFunctions/SPOSet.h"
```

Even if the included file is located in the same directory as the including file this rule should be obeyed. Header files from external projects and standard libraries should be includes using `<iostream>` convention, while headers that are part of the QMCPACK project should be included using the `"our_header.h"` convention.

For readability it is suggested to use the standard order of includes:

1. related header
2. std C library headers
3. std C++ library headers

4. Other libraries' headers
5. QMCPACK headers

In each section the included files should be sorted in alphabetical order.

27.3 Naming

The balance between description and ease of implementation should be balanced such that the code remains self documenting within a single terminal window. If an extremely short variable name is used its scope must be shorter than ~ 40 lines. An exception is made for template parameters which must be in all CAPS.

27.3.1 Namespace Names

Namespace names should be one word, lowercase.

27.3.2 Type and Class Names

Type and class names should start with a capital letter and have a capital letter for each new word. Underscores _ are not allowed.

27.3.3 Variable Names

Variable names should not begin with a capital letter this is reserved for type and class names. Underscores (_) should be used to separate words.

27.3.4 Class Data Members

Class private/protected data members names should follow the convention of variable names with a trailing underscore _.

27.3.5 (Member) Function Names

Function names should start with a lowercase character and have a capital letter for each new word.

27.3.6 Lambda Expressions

Named lambda expressions follow the naming convention for functions:

```
auto myWhatever = [](int i) { return i + 4; };
```

27.3.7 Macro Names

Macro names should be all uppercase and can include underscores (_). The underscore is not allowed as first or last character.

27.3.8 Test Case and Test Names

Test code files should be named

```
class DiracMatrix;
//leads to
test_dirac_matrix.cpp
//which contains test cases named
TEST_CASE("DiracMatrix_update_row","[wavefunction][fermion]")
```

Where the test case covers the `updateRow` and `[wavefunction][fermion]` indicates this test belongs to the fermion wavefunction functionality.

27.4 Comments

27.4.1 Comment Style

Use the `//Comment` syntax for actual comments. Use:

```
/** base class for Single-particle orbital sets
 *
 * SPOSet stands for S(ingle)P(article)O(rbital)Set which contains
 * a number of single-particle orbitals with capabilities of
 * evaluating \f$ \psi_j({\bf r}_i)\f$
 */
```

or

```
///index in the builder list of sposets
int builder_index;
```

27.4.2 Documentation

Doxygen will be used for source documentation. Doxygen commands should be used when appropriate guidance on this TBD.

File Docs

Do not put the file name after the `\file` doxygen command. Doxygen will fill it in for the file the tag appears in.

```
/** \file
 *  File level documentation
 */
```

Class Docs

Every class should have a short description (in the header of the file) of what it is and what is does. Comments for public class member functions follow the same rules as general function comments. Comments for private members are allowed, but not mandatory.

Function Docs

For function parameters whose type is non-const reference or pointer to non-const memory, it should be specified if they are input (In:), output (Out:) or input-output parameters (InOut:).

Example:

```
/** Updates foo and computes bar using in_1 .. in_5.
 * \param[in] in_3
 * \param[in] in_5
 * \param[in,out] foo
 * \param[out] bar
 */

//This is probably not what our clang-format would do
void computeFooBar(Type in_1, const Type& in_2, Type& in_3,
                   const Type* in_4, Type* in_5, Type& foo,
                   Type& bar);
```

Variable Documentation

Name should be self-descriptive. If you need documentation consider renaming first.

Golden Rule of Comments

If you modify a piece of code, also adapt the comments that belong to it if necessary.

27.5 Formatting and "Style"

Use the provided clang-format style in `src/.clang-format` to format `.h`, `.hpp`, `.cu` and `.cpp` files. Many of the rules below will be applied to the code by clang-format, which should allow you to ignore most of them if you always run it on your modified code.

You should use clang-format support and the .clangformat file with your editor, use a git pre-commit hook to run clang-format or run clang-format manually on every file you modify. However if you see numerous formatting updates outside of the code you have modified first commit the formatting changes in a separate PR.

27.5.1 Indentation

Indentation consists of 2 spaces. Do not use tabs in the code.

27.5.2 Line Length

The length of each line of your code should be at most *120* characters.

27.5.3 Horizontal Spacing

No trailing whitespaces should be added to any line. Use no space before a comma (`,`) and a semicolon (`;`) and add a space after them if they are not at the end of a line.

27.5.4 Preprocessor Directives

The preprocessor directives are not indented. The hash is the first character of the line.

27.5.5 Binary Operators

The assignment operators should always have spaces around them.

27.5.6 Unary Operators

Do not put any space between an unary operator and their argument.

27.5.7 Types

The `using` syntax is preferred to `typedef` for type aliases. If the actual type is not excessively long or complex simply just use it, renaming simple types makes code less understandable.

27.5.8 Pointers and References

Pointer or Reference operators should go with the type. But understand the compiler reads them right to left.

```
Type* var;
Type& var;

//Understand this is incompatible with multiple declarations
Type* var1, var2; // var1 is a pointer to Type but var2 is a Type.
```

27.5.9 Templates

The angle brackets of templates should not have any external and internal padding.

```
template<class C>
class Class1;

Class1<Class2<type1>> object;
```

27.5.10 Vertical Spacing

Use empty lines when it helps to improve the readability of the code, but do not use too many. Do not use empty lines after a brace which opens a scope, or before a brace which closes a scope. Each file should contain an empty line at the end of the file. Some editors add an empty line automatically, some do not.

27.5.11 Variable Declarations and Definitions

- Avoid declaring multiple variables in the same declaration, especially if they are not fundamental types:

```
int x, y;                         // Not recommended
Matrix a("my-matrix"), b(size);   // Not allowed

// Preferred
int x;
int y;
```

```
Matrix a("my-matrix");
Matrix b(10);
```

- Use the following order for keywords and modifiers in variable declarations:

```
// General type
[static] [const/constexpr] Type variable_name;

// Pointer
[static] [const] Type* [const] variable_name;

// Integer
// the int is not optional not all platforms support long, etc.
[static] [const/constexpr] [signedness] [size] int variable_name;

// Examples:
static const Matrix a(10);
const double* const d(3.14);
constexpr unsigned long l(42);
```

27.5.12 Function Declarations and Definitions

The return type should be on the same line as the function name. Parameters should be on the same line, too, unless they do not fit on it then one parameter per line aligned with the first parameter should be used.

Include the parameter names also in the declaration of a function, i.e.

```
// calculates a*b+c
double function(double a, double b, double c);

// avoid
double function(double, double, double);

//dont do this
double function(BigTemplatedSomething<double> a, BigTemplatedSomething<double> b,
                BigTemplatedSomething<double> c);

//do this
double function(BigTemplatedSomething<double> a,
                BigTemplatedSomething<double> b,
                BigTemplatedSomething<double> c);
```

27.5.13 Conditionals

Examples:

```
if (condition)
  statement;
else
  statement;

if (condition)
{
  statement;
```

```
}
else if (condition2)
{
  statement;
}
else
{
  statement;
}
```

27.5.14 Switch statement

Switch statements should always have a default case.

Example:

```
switch (var)
{
  case 0:
    statement1;
    statement2;
    break;

  case 1:
    statement1;
    statement2;
    break;

  default:
    statement1;
    statement2;
}
```

27.5.15 Loops

Examples:

```
for (statement; condition; statement)
  statement;

for (statement; condition; statement)
{
  statement1;
  statement2;
}

while (condition)
  statement;

while (condition)
{
  statement1;
  statement2;
}

do
{
  statement;
```

```
}
while (condition);
```

27.5.16 Class Format

`public`, `protected` and `private` keywords are not indented.

Example:

```
class Foo : public Bar
{
public:
  Foo();
  explicit Foo(int var);

  void function();
  void emptyFunction() {}

  void setVar(const int var)
  {
    var_ = var;
  }
  int getVar() const
  {
    return var_;
  }

private:
  bool privateFunction();

  int var_;
  int var2_;
};
```

Constructor Initializer Lists

Examples:

```
// When everything fits on one line:
Foo::Foo(int var) : var_(var)
{
  statement;
}

// If the signature and the initializer list do not
// fit on one line, the colon is indented by 4 spaces:
Foo::Foo(int var)
    : var_(var), var2_(var + 1)
{
  statement;
}

// If the initializer list occupies more lines,
// they are aligned in the following way:
Foo::Foo(int var)
    : some_var_(var),
      some_other_var_(var + 1)
{
```

```
  statement;
}

// No statements:
Foo::Foo(int var)
    : some_var_(var) {}
```

27.5.17 Namespace Formatting

The content of namespaces is not indented. A comment should indicate when a namespace is closed. (clang-format will add these if absent) If nested namespaces are used, a comment with the full namespace is required after opening a set of namespaces or an inner namespace.

Examples:

```
namespace ns
{
void foo();
}  // ns
```

```
namespace ns1
{
namespace ns2
{
// ns1::ns2::
void foo();

namespace ns3
{
// ns1::ns2::ns3::
void bar();
}  // ns3
}  // ns2

namespace ns4
{
namespace ns5
{
// ns1::ns4::ns5::
void foo();
}  // ns5
}  // ns4
}  // ns1
```

27.6 QMCPack C++ Guidance

The guidance here like any advice on how to program should not be treated as a set of rules but rather the hardwon wisdom of many hours of develop suffering. In the past many were ignored and the absolute worst results of that will affect whatever code you need to work with. Your PR should go much smoother if you don't ignore them.

27.6.1 Encapsulation

A class is not just a naming scheme for a set of variables and functions. It should provide a logical set of methods, could contain the state of a logical object and might allow access to object data through a well defined interface related variables. while preserving maximally ability to change internal implementation of the class.

Do not use `struct` as a way to avoid controlling access to the class. Only in rare cases where a class is a fully public data structure `struct` is this appropriate. Ignore (or fix one) the many examples of this in QMCPACK.

Do not use inheritance primarily as a means to break encapsulation. If your class could aggregate or compose another class do that and access it solely through its public interface. You will reduce dependencies.

27.6.2 Casting

In C++ source avoid C style casts, they are difficult to search for and imprecise in function. An exception is made for controlling implicit conversion of simple numerical types.

Explicit C++ style casts make it clear what the safety of the cast is and what sort of conversion is expected to be possible.

```
int c = 2;
int d = 3;
double a;
a = (double)c / d;  // Ok

const class1 c1;
class2* c2;
c2 = (class2*)&c1; // NO
SPOSetAdvanced* spo_advanced = new SPOSetAdvanced();

SPOSet* spo = (SPOSet*)spo_advanced; // NO
SPOSet* spo = static_cast<SPOSet*>(spo_advanced); // OK if upcast, dangerous if
    downcast
```

27.6.3 Pre-increment and pre-decrement

Use the pre-increment (pre-decrement) operator when a variable is incremented (decremented) and the value of the expression is not used. In particular, use the pre-increment (pre-decrement) operator for loop counters where i is not used:

```
for (int i = 0; i < N; ++i)
{
  doSomething();
}

for (int i = 0; i < N; i++)
{
  doSomething(i);
}
```

The post-increment and post-decrement operators create an unnecessary copy, that the compiler cannot optimize away in the case of iterators or other classes with overloaded increment and decrement operators.

27.6.4 Alternative Operator Representations

Alternative representations of operators and other tokens such as `and`, `or`, and `not` instead of `&&`, `||`, and `!` are not allowed. For the reason of consistency, the far more common primary tokens should always be used.

27.6.5 Use of const

- Add the `const` qualifier to all function parameters that are not modified in the function body.
- For parameters passed by value, only add the keyword in the function definition.
- Member functions should be specified const whenever possible.

```
// Declaration
int computeFoo(int bar, const Matrix& m)

// Definition
int computeFoo(const int bar, const Matrix& m)
{
  int foo = 42;

  // Compute foo without changing bar or m.
  // ...

  return foo;
}

class MyClass
{
  int count_
  ...
  int getCount() const { return count_;}
}
```

27.7 Scalar Estimator Implementation

27.7.1 Introduction: Life of a Specialized QMCHamiltonianBase

Almost all observables in QMCPACK are implemented as specialized derived classes of the QMCHamiltonianBase base class. Each observable is instantiated in HamiltonianFactory and added to QMCHamiltonian for tracking. QMCHamiltonian tracks two types of observables: main and auxiliary. Main observables contribute to the local energy. These observables are elements of the simulated Hamiltonian such as kinetic or potential energy. auxiliary observables are expectation values of matrix elements that do not contribute to the local energy. These Hamiltonians do not affect the dynamics of the simulation. In the code, the main observables are labeled by "physical" flag, the auxiliary observables have "physical" set to false.

Initialization

When an `<estimator type="est_type" name="est_name" other_stuff="value"/>` tag is present in the `<hamiltonian/>` section, it is first read by HamiltonianFactory. In general, the `type` of the estimator will determine which specialization of QMCHamiltonianBase to be instantiated, and a derived class with `myName="est_name"` will be constructed. Then, the put() method of this specific class will be called to read any other parameters in the `<estimator/>` xml node. Sometimes these parameters will be read by HamiltonianFactory instead, because it can access more objects than QMCHamiltonianBase.

Cloning

When `OpenMP` threads are spawned, the estimator will be cloned by the `CloneManager`, which is a parent class of many QMC drivers.

```
// In CloneManager.cpp
#pragma omp parallel for shared(w,psi,ham)
for(int ip=1; ip<NumThreads; ++ip)
{
  wClones[ip]=new MCWalkerConfiguration(w);
  psiClones[ip]=psi.makeClone(*wClones[ip]);
  hClones[ip]=ham.makeClone(*wClones[ip],*psiClones[ip]);
}
```

In the above snippet, `ham` is the reference to the estimator on the master thread. If the implemented estimator does not allocate memory for any array, then the default constructor should suffice for the `makeClone` method.

```
// In SpeciesKineticEnergy.cpp
QMCHamiltonianBase* SpeciesKineticEnergy::makeClone(ParticleSet& qp,
    TrialWaveFunction& psi)
{
  return new SpeciesKineticEnergy(*this);
}
```

If memory is allocated during estimator construction (usually when parsing the xml node in the `put` method), then the `makeClone` method should perform the same initialization on each thread.

```
QMCHamiltonianBase* LatticeDeviationEstimator::makeClone(ParticleSet& qp,
    TrialWaveFunction& psi)
{
  LatticeDeviationEstimator* myclone = new
     LatticeDeviationEstimator(qp,spset,tgroup,sgroup);
  myclone->put(input_xml);
  return myclone;
}
```

Evaluate

After the observable class (derived class of QMCHamiltonianBase) is constructed and prepared (by the put() method), it is ready to be used in a QMCDriver. A QMCDriver will call `H.auxHevaluate(W,thisWalker)` after every accepted move, where H is the QMCHamiltonian that holds all main and auxiliary Hamiltonian elements, W is a MCWalkerConfiguration and thisWalker is a pointer to the current walker being worked on. As shown below, this function goes through each auxiliary Hamiltonian element and evaluate it using the current walker configuration. Under

the hood, observables are calculated and dumped to the main particle set's property list for later collection.

```
// In QMCHamiltonian.cpp
// This is more efficient.
// Only calculate auxH elements if moves are accepted.
void QMCHamiltonian::auxHevaluate(ParticleSet& P, Walker_t& ThisWalker)
{
#if !defined(REMOVE_TRACEMANAGER)
  collect_walker_traces(ThisWalker,P.current_step);
#endif
  for(int i=0; i<auxH.size(); ++i)
  {
    auxH[i]->setHistories(ThisWalker);
    RealType sink = auxH[i]->evaluate(P);
    auxH[i]->setObservables(Observables);
#if !defined(REMOVE_TRACEMANAGER)
    auxH[i]->collect_scalar_traces();
#endif
    auxH[i]->setParticlePropertyList(P.PropertyList,myIndex);
  }
}
```

For estimators that contribute to the local energy (main observables), the return value of evaluate() is used in accumulating the local energy. For auxiliary estimators, the return value is not used (`sink` local variable above), only the value of Value is recorded property lists by the setObservables() method as shown in the above code snippet. By default, the setObservables() method will transfer `auxH[i]->Value` to `P.PropertyList[auxH[i]->myIndex]`. The same property list is also kept by the particle set being moved by QMCDriver. This list is updated by `auxH[i]->setParticlePropertyList(P.PropertyList,myIndex)`, where myIndex is the starting index of space allocated to this specific auxiliary Hamiltonian in the property list kept by the target particle set P.

Collection

The actual statistics are collected within the QMCDriver, which owns an EstimatorManager object. This object (or a clone in the case of multithreading) will be registered with each mover it owns. For each mover (such as VMCUpdatePbyP derived from QMCUpdateBase), an accumulate() call is made, which by default, makes an accumulate(walkerset) call to the EstimatorManager it owns. Since each walker has a property set, EstimatorManager uses that local copy to calculate statistics. The EstimatorManager performs block averaging and file I/O.

27.7.2 Single Scalar Estimator Implementation Guide

Almost all of the defaults can be utilized for a single scalar observable. With any luck, only the put() and evaluate() methods need to be implemented. As an example, this section will present a step-by-step guide to implement a `SpeciesKineticEnergy` estimator that calculates the kinetic energy of a specific species instead of the entire particle set. For example, a possible input to this estimator can be:

```
<estimator type="specieskinetic" name="ukinetic" group="u"/>
<estimator type="specieskinetic" name="dkinetic" group="d"/>.
```

This should create two extra columns in the scalar.dat file that contains the kinetic energy of the up and down electrons in two separate columns. If the estimator is properly implemented,then the sum of these two columns should be equal to the default `Kinetic` column.

Barebone

The first step is to create a barebone class structure for this simple scalar estimator. The goal is to be able to instantiate this scalar estimator with an xml node and have it print out a column of zeros in the scalar.dat file.

To achieve this, first create a header file "SpeciesKineticEnergy.h" in QMCHamiltonians folder, with only the required functions declared as follows:

```
// In SpeciesKineticEnergy.h
#ifndef QMCPLUSPLUS_SPECIESKINETICENERGY_H
#define QMCPLUSPLUS_SPECIESKINETICENERGY_H

#include <Particle/WalkerSetRef.h>
#include <QMCHamiltonians/QMCHamiltonianBase.h>

namespace qmcplusplus
{

class SpeciesKineticEnergy: public QMCHamiltonianBase
{
public:

  SpeciesKineticEnergy(ParticleSet& P):tpset(P){ };

  bool put(xmlNodePtr cur);         // read input xml node, required
  bool get(std::ostream& os) const; // class description, required

  Return_t evaluate(ParticleSet& P);
  inline Return_t evaluate(ParticleSet& P, std::vector<NonLocalData>& Txy)
  { // delegate responsity inline for speed
    return evaluate(P);
  }

  // pure virtual functions require overrider
  void resetTargetParticleSet(ParticleSet& P) { }                          // required
  QMCHamiltonianBase* makeClone(ParticleSet& qp, TrialWaveFunction& psi); // required

private:
  ParticleSet& tpset;

}; // SpeciesKineticEnergy

} // namespace qmcplusplus
#endif
```

Notice a local reference `tpset` to the target particle set `P` is saved in the constructor. The target particle set carries much information useful for calculating observables. Next, make "SpeciesKineticEnergy.cpp" and make vacuous definitions

```
// In SpeciesKineticEnergy.cpp
#include <QMCHamiltonians/SpeciesKineticEnergy.h>
namespace qmcplusplus
```

```
{

bool SpeciesKineticEnergy::put(xmlNodePtr cur)
{
  return true;
}

bool SpeciesKineticEnergy::get(std::ostream& os) const
{
  return true;
}

SpeciesKineticEnergy::Return_t SpeciesKineticEnergy::evaluate(ParticleSet& P)
{
  Value = 0.0;
  return Value;
}

QMCHamiltonianBase* SpeciesKineticEnergy::makeClone(ParticleSet& qp,
    TrialWaveFunction& psi)
{
  // no local array allocated, default constructor should be enough
  return new SpeciesKineticEnergy(*this);
}

} // namespace qmcplusplus
```

Now, head over to Hamiltonian Factory and instantiate this observable if an xml node is found requesting it. Look for "gofr" in HamiltonianFactory.cpp, for example, and follow the if block

```
// In HamiltonianFactory.cpp
#include <QMCHamiltonians/SpeciesKineticEnergy.h>
else if(potType =="specieskinetic")
{
  SpeciesKineticEnergy* apot = new SpeciesKineticEnergy(*target_particle_set);
  apot->put(cur);
  targetH->addOperator(apot,potName,false);
}
```

The last argument of addOperator(), i.e. the `false` flag, is **crucial**. This tells QMCPACK that the observable we implemented is not a physical Hamiltonian, thus it will not contribute to the local energy. Changes to the local energy will alter the dynamics of the simulation. Finally, add "SpeciesKineticEnergy.cpp" to HAMSRCS in "CMakeLists.txt" located in the QMCHamiltonians folder. Now, recompile QMCPACK and run it on an input that requests `<estimator type="specieskinetic" name="ukinetic"/>` in the `hamiltonian` block. A columns of zeros should appear in the scalar.dat file under the name "ukinetic".

Evaluate

The evaluate() method is where we perform the calculation of the desired observable. In a first iteration, we will simply hard-code the name and mass of the particles

```
// In SpeciesKineticEnergy.cpp
#include <QMCHamiltonians/BareKineticEnergy.h> // laplaician() defined here
SpeciesKineticEnergy::Return_t SpeciesKineticEnergy::evaluate(ParticleSet& P)
{
  std::string group="u";
```

```
  RealType minus_over_2m = -0.5;

  SpeciesSet& tspecies(P.getSpeciesSet());

  Value = 0.0;
  for (int iat=0; iat<P.getTotalNum(); iat++)
  {
    if (tspecies.speciesName[ P.GroupID(iat) ] == group)
    {
      Value += minus_over_2m*laplacian(P.G[iat],P.L[iat]);
    }
  }
  return Value;

  // Kinetic column has:
  // Value = -0.5*( Dot(P.G,P.G) + Sum(P.L) );
}
```

Voila, you should now be able to compile QMCPACK, rerun, and see that the values in the "ukinetic" column are no longer zero. Now, the only task left to make this basic observable complete is to read in the extra parameters instead of hard-coding them.

Parse Extra Input

The preferred method to parse extra input parameters in the xml node is to implement the put() function of our specific observable. Suppose we wish to read in a single string that tells us whether to record the kinetic energy of the up electron (group="u") or the down electron (group="d"). This is easily achievable using the OhmmsAttributeSet class

```
// In SpeciesKineticEnergy.cpp
#include <OhmmsData/AttributeSet.h>
bool SpeciesKineticEnergy::put(xmlNodePtr cur)
{
  // read in extra parameter "group"
  OhmmsAttributeSet attrib;
  attrib.add(group,"group");
  attrib.put(cur);

  // save mass of specified group of particles
  SpeciesSet& tspecies(tpset.getSpeciesSet());
  int group_id  = tspecies.findSpecies(group);
  int massind   = tspecies.getAttribute("mass");
  minus_over_2m = -1./(2.*tspecies(massind,group_id));

  return true;
}
```

where we may keep "group" and "minus_over_2m" as local variables to our specific class.

```
// In SpeciesKineticEnergy.h
private:
  ParticleSet& tpset;
  std::string  group;
  RealType minus_over_2m;
```

Notice, the above operations are made possible by the saved reference to the target particle set. Last but not least, compile and run a full example (i.e. a short DMC calculation) with the following xml nodes in your input

```
<estimator type="specieskinetic" name="ukinetic" group="u"/>
<estimator type="specieskinetic" name="dkinetic" group="d"/>.
```

Make sure the sum of the ”ukinetic” and ”dkinetic” columns is **exactly** the same as the Kinetic columns at **every block**.

For easy reference, the complete list of changes is summarized bellow

```
// In HamiltonianFactory.cpp
#include "QMCHamiltonians/SpeciesKineticEnergy.h"
else if(potType =="specieskinetic")
{
        SpeciesKineticEnergy* apot = new SpeciesKineticEnergy(*targetPtcl);
        apot->put(cur);
        targetH->addOperator(apot,potName,false);
}
```

```
// In SpeciesKineticEnergy.h
#include <Particle/WalkerSetRef.h>
#include <QMCHamiltonians/QMCHamiltonianBase.h>

namespace qmcplusplus
{

class SpeciesKineticEnergy: public QMCHamiltonianBase
{
public:

  SpeciesKineticEnergy(ParticleSet& P):tpset(P){ };

  // xml node is read by HamiltonianFactory, eg. the sum of following should be
      equivalent to Kinetic
  // <estimator name="ukinetic" type="specieskinetic" target="e" group="u"/>
  // <estimator name="dkinetic" type="specieskinetic" target="e" group="d"/>
  bool put(xmlNodePtr cur);         // read input xml node, required
  bool get(std::ostream& os) const; // class description, required

  Return_t evaluate(ParticleSet& P);
  inline Return_t evaluate(ParticleSet& P, std::vector<NonLocalData>& Txy)
  { // delegate responsity inline for speed
    return evaluate(P);
  }

  // pure virtual functions require overrider
  void resetTargetParticleSet(ParticleSet& P) { }                          // required
  QMCHamiltonianBase* makeClone(ParticleSet& qp, TrialWaveFunction& psi); // required

private:
  ParticleSet&       tpset; // reference to target particle set
  std::string        group; // name of species to track
  RealType   minus_over_2m; // mass of the species !! assume same mass
  // for multiple species, simply initialize multiple estimators

}; // SpeciesKineticEnergy

} // namespace qmcplusplus
#endif
```

```
// In SpeciesKineticEnergy.cpp
```

```
#include <QMCHamiltonians/SpeciesKineticEnergy.h>
#include <QMCHamiltonians/BareKineticEnergy.h> // laplaician() defined here
#include <OhmmsData/AttributeSet.h>

namespace qmcplusplus
{

bool SpeciesKineticEnergy::put(xmlNodePtr cur)
{
  // read in extra parameter "group"
  OhmmsAttributeSet attrib;
  attrib.add(group,"group");
  attrib.put(cur);

  // save mass of specified group of particles
  int group_id  = tspecies.findSpecies(group);
  int massind   = tspecies.getAttribute("mass");
  minus_over_2m = -1./(2.*tspecies(massind,group_id));

  return true;
}

bool SpeciesKineticEnergy::get(std::ostream& os) const
{ // class description
  os << "SpeciesKineticEnergy:␣" << myName << "␣for␣species␣" << group;
  return true;
}

SpeciesKineticEnergy::Return_t SpeciesKineticEnergy::evaluate(ParticleSet& P)
{
  Value = 0.0;
  for (int iat=0; iat<P.getTotalNum(); iat++)
  {
    if (tspecies.speciesName[ P.GroupID(iat) ] == group)
    {
      Value += minus_over_2m*laplacian(P.G[iat],P.L[iat]);
    }
  }
  return Value;
}

QMCHamiltonianBase* SpeciesKineticEnergy::makeClone(ParticleSet& qp,
    TrialWaveFunction& psi)
{ //default constructor
  return new SpeciesKineticEnergy(*this);
}

} // namespace qmcplusplus
```

27.7.3 Multiple Scalars

It is fairly straight-forward to created more than one column in the scalar.dat file with a single observable class. For example, if we want a single SpeciesKineticEnergy estimator to record the kinetic energies of all species in the target particle set simultaneously, we only have to write two new methods: addObservables() and setObservables(), then tweak the behavior of evaluate(). Firstly, we will have to override the default behavior of addObservables() to make room for more than one

column in the scalar.dat file as follows

```
// In SpeciesKineticEnergy.cpp
void SpeciesKineticEnergy::addObservables(PropertySetType& plist, BufferType&
    collectables)
{
  myIndex = plist.size();
  for (int ispec=0; ispec<num_species; ispec++)
  { // make columns named "$myName_u", "$myName_d" etc.
    plist.add(myName + "_" + species_names[ispec]);
  }
}
```

where "num_species" and "species_name" can be local variables initialized in the constructor. We should also initialize some local arrays to hold temporary data

```
// In SpeciesKineticEnergy.h
private:
  int num_species;
  std::vector<std::string> species_names;
  std::vector<RealType> species_kinetic,vec_minus_over_2m;

// In SpeciesKineticEnergy.cpp
SpeciesKineticEnergy::SpeciesKineticEnergy(ParticleSet& P):tpset(P)
{
  SpeciesSet& tspecies(P.getSpeciesSet());
  int massind = tspecies.getAttribute("mass");

  num_species = tspecies.size();
  species_kinetic.resize(num_species);
  vec_minus_over_2m.resize(num_species);
  species_names.resize(num_species);
  for (int ispec=0; ispec<num_species; ispec++)
  {
    species_names[ispec] = tspecies.speciesName[ispec];
    vec_minus_over_2m[ispec] = -1./(2.*tspecies(massind,ispec));
  }
}
```

Next, we need to override the default behavior of `setObservables()` to transfer multiple values to the property list kept by the main particle set, which eventually go into the scalar.dat file

```
// In SpeciesKineticEnergy.cpp
void SpeciesKineticEnergy::setObservables(PropertySetType& plist)
{ // slots in plist must be allocated by addObservables() first
  copy(species_kinetic.begin(),species_kinetic.end(),plist.begin()+myIndex);
}
```

Finally, we need to change the behavior of evaluate() to fill the local vector "species_kinetic" with appropriate observable values

```
SpeciesKineticEnergy::Return_t SpeciesKineticEnergy::evaluate(ParticleSet& P)
{
  std::fill(species_kinetic.begin(),species_kinetic.end(),0.0);

  for (int iat=0; iat<P.getTotalNum(); iat++)
  {
    int ispec = P.GroupID(iat);
    species_kinetic[ispec] += vec_minus_over_2m[ispec]*laplacian(P.G[iat],P.L[iat]);
  }
```

```
  Value = 0.0; // Value is no longer used
  return Value;
}
```

That's it! SpeciesKineticEnergy estimator no longer needs the "group" input and will automatically output the kinetic energy of every species in the target particle set in multiple columns. You should now be able to run with `<estimator type="specieskinetic" name="skinetic"/>` and check that the sum of all columns that starts with "skinetic" is equal to the default "Kinetic" column.

27.7.4 HDF5 Output

If we desire an observable that will output hundreds of scalars per simulation step (eg. SkEstimator), then it is preferred to output to the stat.h5 file instead of the scalar.dat file for better organization. A large chunk of data to be registered in the stat.h5 file is called a "Collectable" in QMCPACK. In particular, if a QMCHamiltonianBase object is initialized with `UpdateMode.set(COLLECTABLE,1)` then the "Collectables" object carried by the main particle set will be processed and written to the stat.h5 file, where "UpdateMode" is a bit set (i.e. a collection of flags) with the following enumeration

```
// In QMCHamiltonianBase.h
///enum for UpdateMode
enum {PRIMARY=0,
  OPTIMIZABLE=1,
  RATIOUPDATE=2,
  PHYSICAL=3,
  COLLECTABLE=4,
  NONLOCAL=5,
  VIRTUALMOVES=6
};
```

As a simple example, to put the two columns we produced in the previous section into the stat.h5 file, we will first need to declare that our observable uses "Collectables"

```
// In constructor add:
hdf5_out = true;
UpdateMode.set(COLLECTABLE,1);
```

Then make some room in the "Collectables" object carried by the target particle set.

```
// In addObservables(PropertySetType& plist, BufferType& collectables) add:
if (hdf5_out)
{
  h5_index = collectables.size();
  std::vector<RealType> tmp(num_species);
  collectables.add(tmp.begin(),tmp.end());
}
```

Next make some room in the stat.h5 file by overriding the registerCollectables() method

```
// In SpeciesKineticEnergy.cpp
void SpeciesKineticEnergy::registerCollectables(std::vector<observable_helper*>&
    h5desc, hid_t gid) const
{
  if (hdf5_out)
  {
    std::vector<int> ndim(1,num_species);
```

```
    observable_helper* h5o=new observable_helper(myName);
    h5o->set_dimensions(ndim,h5_index);
    h5o->open(gid);
    h5desc.push_back(h5o);
  }
}
```

Finally, edit evaluate() to use the space in the "Collectables" object.

```
// In SpeciesKineticEnergy.cpp
SpeciesKineticEnergy::Return_t SpeciesKineticEnergy::evaluate(ParticleSet& P)
{
  RealType wgt = tWalker->Weight; // MUST explicitly use DMC weights in Collectables!
  std::fill(species_kinetic.begin(),species_kinetic.end(),0.0);

  for (int iat=0; iat<P.getTotalNum(); iat++)
  {
    int ispec = P.GroupID(iat);
    species_kinetic[ispec] += vec_minus_over_2m[ispec]*laplacian(P.G[iat],P.L[iat]);
    P.Collectables[h5_index + ispec] +=
        vec_minus_over_2m[ispec]*laplacian(P.G[iat],P.L[iat])*wgt;
  }

  Value = 0.0; // Value is no longer used
  return Value;
}
```

That should be it. There should now be a new entry in the stat.h5 file containing the same columns of data as the scalar.dat file. After this check, we should clean up the code by

1. make "hdf5_out" and input flag by editing the put() method
2. disable output to scalar.dat when "hdf5_out" flag is on

27.8 Estimator Output

27.8.1 Estimator Definition

For simplicity, consider a local property $O(\boldsymbol{R})$, where $\boldsymbol{R}$ is the collection of all particle coordinates. An *estimator* for $O(\boldsymbol{R})$ is a weighted average over walkers

$$E[O] = \left(\sum_{i=1}^{N_{walker}^{tot}} w_i O(\boldsymbol{R}_i) \right) / \left(\sum_{i=1}^{N_{walker}^{tot}} w_i \right). \tag{27.1}$$

N_{walker}^{tot} is the total number of walkers collected in the entire simulation. Notice, N_{walker}^{tot} is typically far larger than the number of walkers held in memory at any given simulation step. w_i is the weight of walker i.

In a VMC simulation, the weight of every walkers is 1.0. Further, the number of walkers is constant at each step. Therefore, eq. (27.1) simplifies to

$$E_{VMC}[O] = \frac{1}{N_{step} N_{walker}^{ensemble}} \sum_{s,e} O(\boldsymbol{R}_{s,e}). \tag{27.2}$$

Each walker $\boldsymbol{R}_{s,e}$ is labeled by *step index* s, and *ensemble index* e.

In a DMC simulation, the weight of each walker is different and may change from step to step. Further, the ensemble size varies from step to step. Therefore, eq. (27.1) simplifies to

$$E_{DMC}[O] = \frac{1}{N_{step}} \sum_s \left\{ \left(\sum_e w_{s,e} O(\boldsymbol{R}_{s,e}) \right) / \left(\sum_e w_{s,e} \right) \right\}. \tag{27.3}$$

I will refer to the average in the {} as *ensemble average* and the remaining averages *block average*. The process of calculating $O(\boldsymbol{R})$ is *evaluate*.

27.8.2 Class Relations

A large number of classes are involved in the estimator collection process. They often have misleading class name or method name. Document gotchas in the following list:

1. `EstimatorManager` is an unused copy of `EstimatorManagerBase`. `EstimatorManagerBase` is the class used in the QMC drivers. (PR #371 explains this)

2. `EstimatorManagerBase::Estimators` is completely different from `QMCDriver::Estimators`, which is subtly different from `QMCHamiltonianBase::Estimators`. The first is a list of pointers to `ScalarEstimatorBase`. The second is the master estimator (one per MPI group). The third is the slave estimator that exists one per OpenMP thread.

3. `QMCHamiltonian` is NOT a parent class of `QMCHamiltonianBase`. Instead, `QMCHamiltonian` owns two lists of `QMCHamiltonianBase` named `H` and `auxH`.

4. `QMCDriver::H` is NOT the same as `QMCHamiltonian::H`. The first is a pointer to a `QMCHamiltonian`. `QMCHamiltonian::H` is a list.

5. `EstimatorManager::stopBlock(std::vector)` is completely different from `EstimatorManager::stopBlock(RealType)`, which is the same as `stopBlock(RealType, true)`, but is subtly different from `stopBlock(RealType, false)`. The first three methods are intended to be called by the master estimator which exists one per MPI group. The last method is intended to be called by the slave estimator which exists one per OpenMP thread.

27.8.3 Estimator Output Stages

Estimators take four conceptual stages to propagate to the output files: evaluate, load ensemble, unload ensemble, and collect. They are easier to understand in reverse order.

Collect Stage

File output is performed by the master `EstimatorManager` owned by `QMCDriver`. The first 8+ entries in `EstimatorManagerBase::AverageCache` will be written to scalar.dat. The remaining entries in `AverageCache` will be written to stat.h5. File writing is triggered by `EstimatorManagerBase::collectBlockAverages` inside `EstimatorManagerBase::stopBlock`.

```
// In EstimatorManagerBase.cpp::collectBlockAverages
  if(Archive)
  {
    *Archive << std::setw(10) << RecordCount;
    int maxobjs=std::min(BlockAverages.size(),max4ascii);
```

```
    for(int j=0; j<maxobjs; j++)
      *Archive << std::setw(FieldWidth) << AverageCache[j];
    for(int j=0; j<PropertyCache.size(); j++)
      *Archive << std::setw(FieldWidth) << PropertyCache[j];
    *Archive << std::endl;
    for(int o=0; o<h5desc.size(); ++o)
      h5desc[o]->write(AverageCache.data(),SquaredAverageCache.data());
    H5Fflush(h_file,H5F_SCOPE_LOCAL);
  }
```

`EstimatorManagerBase::collectBlockAverages` is triggered from master-thread estimator via either `stopBlock(std::vector)` or `stopBlock(RealType, true)`. Notice, file writing is NOT triggered by the slave-thread estimator method `stopBlock(RealType, false)`.

```
// In EstimatorManagerBase.cpp
void EstimatorManagerBase::stopBlock(RealType accept, bool collectall)
{
  //take block averages and update properties per block
  PropertyCache[weightInd]=BlockWeight;
  PropertyCache[cpuInd] = MyTimer.elapsed();
  PropertyCache[acceptInd] = accept;
  for(int i=0; i<Estimators.size(); i++)
    Estimators[i]->takeBlockAverage(AverageCache.begin(),SquaredAverageCache.begin());
  if(Collectables)
  {
    Collectables->takeBlockAverage(AverageCache.begin(),SquaredAverageCache.begin());
  }
  if(collectall)
    collectBlockAverages(1);
}
```

```
// In ScalarEstimatorBase.h
template<typename IT>
inline void takeBlockAverage(IT first, IT first_sq)
{
  first += FirstIndex;
  first_sq += FirstIndex;
  for(int i=0; i<scalars.size(); i++)
  {
    *first++ = scalars[i].mean();
    *first_sq++ = scalars[i].mean2();
    scalars_saved[i]=scalars[i]; //save current block
    scalars[i].clear();
  }
}
```

At the collect stage, `ScalarEstimatorBase::scalars` must be populated with ensemble-averaged data. Two derived classes of `ScalarEstimatorBase` are crucial: `LocalEnergyEstimator` will carry `Properties`, where as `CollectablesEstimator` will carry `Collectables`.

Unload Ensemble Stage

`LocalEnergyEstimator::scalars` are populated by `ScalarEstimatorBase::accumulate`, whereas `CollectablesEstimator::scalars` are populated by `CollectablesEstimator:: accumulate_all`.

Both accumulate methods are triggered by `EstimatorManagerBase::accumulate`. One confusing aspect about the unload stage is that `EstimatorManagerBase::accumulate` has a master and a slave call signature. A slave estimator such as `QMCUpdateBase::Estimators` should unload a subset of walkers. Thus, the slave estimator should call `accumulate(W,it,it_end)`. However, the master estimator, such as `SimpleFixedNodeBranch::myEstimator` should unload data from the entire walker ensemble. This is achieved by calling `accumulate(W)`.

```
void EstimatorManagerBase::accumulate(MCWalkerConfiguration& W)
{ // intended to be called by master estimator only
  BlockWeight += W.getActiveWalkers();
  RealType norm=1.0/W.getGlobalNumWalkers();
  for(int i=0; i< Estimators.size(); i++)
    Estimators[i]->accumulate(W,W.begin(),W.end(),norm);
  if(Collectables)//collectables are normalized by QMC drivers
    Collectables->accumulate_all(W.Collectables,1.0);
}
```

```
void EstimatorManagerBase::accumulate(MCWalkerConfiguration& W
 , MCWalkerConfiguration::iterator it
 , MCWalkerConfiguration::iterator it_end)
{ // intended to be called slaveEstimator only
  BlockWeight += it_end-it;
  RealType norm=1.0/W.getGlobalNumWalkers();
  for(int i=0; i< Estimators.size(); i++)
    Estimators[i]->accumulate(W,it,it_end,norm);
  if(Collectables)
    Collectables->accumulate_all(W.Collectables,1.0);
}
```

```
// In LocalEnergyEstimator.h
inline void accumulate(const Walker_t& awalker, RealType wgt)
{ // ensemble average W.Properties
  // expect ePtr to be W.Properties; expect wgt = 1/GlobalNumberOfWalkers
  const RealType* restrict ePtr = awalker.getPropertyBase();
  RealType wwght= wgt* awalker.Weight;
  scalars[0](ePtr[LOCALENERGY],wwght);
  scalars[1](ePtr[LOCALENERGY]*ePtr[LOCALENERGY],wwght);
  scalars[2](ePtr[LOCALPOTENTIAL],wwght);
  for(int target=3, source=FirstHamiltonian; target<scalars.size(); ++target,
      ++source)
    scalars[target](ePtr[source],wwght);
}
```

```
// In CollectablesEstimator.h
inline void accumulate_all(const MCWalkerConfiguration::Buffer_t& data, RealType wgt)
{ // ensemble average W.Collectables
  // expect data to be W.Collectables; expect wgt = 1.0
  for(int i=0; i<data.size(); ++i)
    scalars[i](data[i], wgt);
}
```

At the unload ensemble stage, the data structures `Properties` and `Collectables` must be populated by appropriately normalized values so that the ensemble average can be correctly taken. `QMCDriver` is responsible for the correct loading of data onto the walker ensemble.

Load Ensemble Stage

`Properties` in the Monte Carlo ensemble of walkers `QMCDriver::W` is populated by `QMCHamiltonian ::saveProperties`. The master `QMCHamiltonian::LocalEnergy`, `::KineticEnergy`, and `::Observables` must be properly populated at the end of the evaluate stage.

```
// In QMCHamiltonian.h
  template<class IT>
  inline
  void saveProperty(IT first)
  { // expect first to be W.Properties
    first[LOCALPOTENTIAL]= LocalEnergy-KineticEnergy;
    copy(Observables.begin(),Observables.end(),first+myIndex);
  }
```

`Collectables`'s load stage is combined with its evaluate stage.

Evaluate Stage

The master `QMCHamiltonian::Observables` is populated by slave `QMCHamiltonianBase ::setObservables`. However, the call signature must be `QMCHamiltonianBase::setObservables( QMCHamiltonian::Observables)`. This call signature is enforced by `QMCHamiltonian::evaluate` and `QMCHamiltonian::auxHevaluate`.

```
// In QMCHamiltonian.cpp
QMCHamiltonian::Return_t
QMCHamiltonian::evaluate(ParticleSet& P)
{
  LocalEnergy = 0.0;
  for(int i=0; i<H.size(); ++i)
  {
    myTimers[i]->start();
    LocalEnergy += H[i]->evaluate(P);
    H[i]->setObservables(Observables);
#if !defined(REMOVE_TRACEMANAGER)
    H[i]->collect_scalar_traces();
#endif
    myTimers[i]->stop();
    H[i]->setParticlePropertyList(P.PropertyList,myIndex);
  }
  KineticEnergy=H[0]->Value;
  P.PropertyList[LOCALENERGY]=LocalEnergy;
  P.PropertyList[LOCALPOTENTIAL]=LocalEnergy-KineticEnergy;
  // auxHevaluate(P);
  return LocalEnergy;
}
```

```
// In QMCHamiltonian.cpp
void QMCHamiltonian::auxHevaluate(ParticleSet& P, Walker_t& ThisWalker)
{
#if !defined(REMOVE_TRACEMANAGER)
```

```
  collect_walker_traces(ThisWalker,P.current_step);
#endif
  for(int i=0; i<auxH.size(); ++i)
  {
    auxH[i]->setHistories(ThisWalker);
    RealType sink = auxH[i]->evaluate(P);
    auxH[i]->setObservables(Observables);
#if !defined(REMOVE_TRACEMANAGER)
    auxH[i]->collect_scalar_traces();
#endif
    auxH[i]->setParticlePropertyList(P.PropertyList,myIndex);
  }
}
```

27.8.4 Estimator Use Cases

VMCSingleOMP pseudo code

```
bool VMCSingleOMP::run()
{
  masterEstimator->start(nBlocks);
  for (int ip=0; ip<NumThreads; ++ip)
    Movers[ip]->startRun(nBlocks,false);  // slaveEstimator->start(blocks, record)

  do // block
  {
    #pragma omp parallel
    {
      Movers[ip]->startBlock(nSteps);  // slaveEstimator->startBlock(steps)
      RealType cnorm = 1.0/static_cast<RealType>(wPerNode[ip+1]-wPerNode[ip]);
      do // step
      {
        wClones[ip]->resetCollectables();
        Movers[ip]->advanceWalkers(wit, wit_end, recompute);
        wClones[ip]->Collectables *= cnorm;
        Movers[ip]->accumulate(wit, wit_end);
      } // end step
      Movers[ip]->stopBlock(false);  // slaveEstimator->stopBlock(acc, false)
    } // end omp
    masterEstimator->stopBlock(estimatorClones);  // write files
  } // end block
  masterEstimator->stop(estimatorClones);
}
```

DMCOMP pseudo code

```
bool DMCOMP::run()
{
  masterEstimator->setCollectionMode(true);

  masterEstimator->start(nBlocks);
  for(int ip=0; ip<NumThreads; ip++)
    Movers[ip]->startRun(nBlocks,false);  // slaveEstimator->start(blocks, record)

  do // block
```

```
  {
    masterEstimator->startBlock(nSteps);
    for(int ip=0; ip<NumThreads; ip++)
      Movers[ip]->startBlock(nSteps);  // slaveEstimator->startBlock(steps)

    do // step
    {
      #pragma omp parallel
      {
      wClones[ip]->resetCollectables();
      // advanceWalkers
      } // end omp

      //branchEngine->branch
      { // In WalkerControlMPI.cpp::branch
      wgt_inv=WalkerController->NumContexts/WalkerController->EnsembleProperty.Weight;
      walkers.Collectables *= wgt_inv;
      slaveEstimator->accumulate(walkers);
      }
      masterEstimator->stopBlock(acc)  // write files
    }  // end for step
  }  // end for block

  masterEstimator->stop();
}
```

27.8.5 Summary

Two ensemble-level data structures `ParticleSet::Properties` and `::Collectables` serve as intermediaries between evaluate classes and output classes to scalar.dat and stat.h5. `Properties` appears in both scalar.dat and stat.h5, whereas `Collectables` appears only in stat.h5. `Properties` is overwritten by `QMCHamiltonian::Observables` at the end of each step. `QMCHamiltonian::Observables` is filled upon call to `QMCHamiltonian::evaluate` and `::auxHevaluate`. `Collectables` is zeroed at the beginning of each step and accumulated upon call to `::auxHevaluate`.

Data are outputted to scalar.dat in 4 stages: evaluate, load, unload, and collect. In the evaluate stage, `QMCHamiltonian::Observables` is populated by a list of `QMCHamiltonianBase`. In the load stage, `QMCHamiltonian::Observables` is transfered to `Properties` by `QMCDriver`. In the unload stage, `Properties` is copied to `LocalEnergyEstimator::scalars`. In the collect stage, `LocalEnergyEstimator::scalars` is block-averaged to `EstimatorManagerBase ::AverageCache` and dumped to file. For `Collectables`, the evaluate and load stages are combined in a call to `QMCHamiltonian::auxHevaluate`. In the unload stage, `Collectables` is copied to `CollectablesEstimator::scalars`. In the collect stage, `CollectablesEstimator ::scalars` is block-averaged to `EstimatorManagerBase::AverageCache` and dumped to file.

27.8.6 Appendix: dmc.dat

There is an additional data structure `ParticleSet::EnsembleProperty`, which is managed by `WalkerControlBase::EnsembleProperty` and directly dumped to dmc.dat via its own averaging procedure. dmc.dat is written by `WalkerControlBase::measureProperties`, which is called by `WalkerControlBase::branch`, which is called by `SimpleFixedNodeBranch ::branch` for example.

27.9 Slater-Backflow Wavefunction Implementation Details

For simplicity, consider N identical fermions of the same spin (e.g. up electrons) at spatial locations $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N\}$. Then the Slater determinant can be written as

$$S = \det M, \tag{27.4}$$

where each entry in the determinant is a single-particle orbital evaluated at a particle position

$$M_{ij} = \phi_i(\boldsymbol{r}_j). \tag{27.5}$$

When backflow transformation is applied to the Determinant, the particle coordinates $\boldsymbol{r}_i$ that go into the single-particle orbitals are replaced by quasi-particle coordinates $\boldsymbol{x}_i$

$$M_{ij} = \phi_i(\boldsymbol{x}_j), \tag{27.6}$$

where

$$\boldsymbol{x}_i = \boldsymbol{r}_i + \sum_{j=1, j\neq i}^{N} \eta(r_{ij})(\boldsymbol{r}_i - \boldsymbol{r}_j). \tag{27.7}$$

$r_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j|$. The integers i,j label the particle/quasi-particle. There is a one-to-one correspondence between the particles and the quasi-particles, which is simplest when $\eta = 0$.

27.9.1 Value

The evaluation of the Slater-Backflow wavefunction is almost identical to that of a Slater wavefunction. The only difference is that the quasi-particles coordinates are used to evaluate the single-particle orbitals. The actual value of the determinant is stored during the inversion of the matrix M (`cgetrf`→`cgetri`). Suppose $M = LU$, then $S = \prod_{i=1}^{N} L_{ii}U_{ii}$.

```
// In DiracDeterminantWithBackflow::evaluateLog(P,G,L)
Phi->evaluate(BFTrans->QP, FirstIndex, LastIndex, psiM,dpsiM,grad_grad_psiM);
psiMinv = psiM;
LogValue=InvertWithLog(psiMinv.data(),NumPtcls,NumOrbitals
  ,WorkSpace.data(),Pivot.data(),PhaseValue);
```

QMCPACK represents the complex value of the wavefunction in polar coordinates $S = e^U e^{i\theta}$. Specifically, `LogValue` U and `PhaseValue` θ are handled separately. In the following, I will consider derivatives of the log value only.

27.9.2 Gradient

To evaluate particle gradient of the log value of the Slater-Backflow wavefunction, we can use the $\log\det$ identity in eq. (27.8). This identity maps the derivative of $\log\det M$ with respect to a real variable p to a trace over $M^{-1}dM$

$$\frac{\partial}{\partial p}\log\det M = \mathrm{tr}\left(M^{-1}\frac{\partial M}{\partial p}\right). \tag{27.8}$$

Following Kwon, Ceperley, and Martin [18], the particle gradient

$$G_i^\alpha \equiv \frac{\partial}{\partial r_i^\alpha} \log\det M = \sum_{j=1}^{N}\sum_{\beta=1}^{3} F_{jj}^\beta A_{jj}^{\alpha\beta}, \tag{27.9}$$

where the quasi-particle gradient matrix

$$A_{ij}^{\alpha\beta} \equiv \frac{\partial x_j^\beta}{\partial r_i^\alpha}, \tag{27.10}$$

and the intermediate matrix

$$F_{ij}^\alpha \equiv \sum_k M_{ik}^{-1} dM_{kj}^\alpha, \tag{27.11}$$

with the single-particle orbital derivatives (w.r. to quasi-particle coordinates)

$$dM_{ij}^\alpha \equiv \frac{\partial M_{ij}}{\partial x_j^\alpha}. \tag{27.12}$$

Notice, I have made the name change of $\phi \to M$ from the notations of ref. [18]. This name change is intended to help the reader associate M with the QMCPACK variable `psiM`.

```
// In DiracDeterminantWithBackflow::evaluateLog(P,G,L)
for(int i=0; i<num; i++) // k in above formula
{
  for(int j=0; j<NumPtcls; j++)
  {
    for(int k=0; k<OHMMS_DIM; k++) // alpha in above formula
    {
      myG(i) += dot(BFTrans->Amat(i,FirstIndex+j),Fmat(j,j));
    }
  }
}
```

Eq. (27.9) is still relatively simple to understand. The A matrix maps changes in particle coordinates $d\boldsymbol{r}$ to changes in quasi-particle coordinates $d\boldsymbol{x}$. Dotting A into F propagates $d\boldsymbol{x}$ to dM. Thus $F \cdot A$ is the term inside the trace operator of eq. (27.8). Finally, performing the trace completes the evaluation of the derivative.

27.9.3 Laplacian

The particle laplacian is given in ref. [18] as

$$L_i \equiv \sum_\beta \frac{\partial^2}{\partial (r_i^\beta)^2} \log\det M = \sum_{j\alpha} B_{ij}^\alpha F_{jj}^\alpha - \sum_{jk}\sum_{\alpha\beta\gamma} A_{ij}^{\alpha\beta} A_{ik}^{\alpha\gamma} \times \left(F_{kj}^\alpha F_{jk}^\gamma - \delta_{jk} \sum_m M_{jm}^{-1} d2M_{mj}^{\beta\gamma} \right), \tag{27.13}$$

where the quasi-particle laplacian matrix

$$B_{ij}^\alpha \equiv \sum_\beta \frac{\partial^2 x_j^\alpha}{\partial (r_i^\beta)^2}, \tag{27.14}$$

with the second derivatives of the single-particles orbitals being

$$d2M_{ij}^{\alpha\beta} \equiv \frac{\partial^2 M_{ij}}{\partial x_j^\alpha \partial x_j^\beta}. \tag{27.15}$$

Schematically, L_i has contributions from 3 terms of the form $BF, AAFF, \mathrm{tr}(AA, Md2M)$, respectively. $A, B, M, d2M$, and F can be calculated and stored before the calculations of L_i. The first BF term can be directly calculated in a loop over quasi-particle coordinates $j\alpha$.

```
// In DiracDeterminantWithBackflow::evaluateLog(P,G,L)
for(int j=0; j<NumPtcls; j++)
  for(int a=0; a<OHMMS_DIM; k++)
    myL(i) += BFTrans->Bmat_full(i,FirstIndex+j)[a]*Fmat(j,j)[a];
```

Notice B_{ij}^α is stored in `Bmat_full`, NOT `Bmat`.

The remaining two terms both involve AA. Thus, it is best to define a temporary tensor AA

$$_iAA_{jk}^{\beta\gamma} \equiv \sum_\alpha A_{ij}^{\alpha\beta} A_{ij}^{\alpha\gamma}, \tag{27.16}$$

which we will overwrite for each particle i. Similarly, define FF

$$FF_{jk}^{\alpha\gamma} \equiv F_{kj}^\alpha F_{jk}^\gamma, \tag{27.17}$$

which is simply the outer product of $F \otimes F$. Then the $AAFF$ term can be calculated by fully contracting AA with FF.

```
// In DiracDeterminantWithBackflow::evaluateLog(P,G,L)
for(int j=0; j<NumPtcls; j++)
  for(int k=0; k<NumPtcls; k++)
    for(int i=0; i<num; i++)
    {
      Tensor<RealType,OHMMS_DIM> AA =
          dot(transpose(BFTrans->Amat(i,FirstIndex+j)),BFTrans->Amat(i,FirstIndex+k));
      HessType FF = outerProduct(Fmat(k,j),Fmat(j,k));
      myL(i) -= traceAtB(AA,FF);
    }
```

Finally, define the single-particle orbital derivative term

$$Md2M_j^{\beta\gamma} \equiv \sum_m M_{jm}^{-1} d2M_{mj}^\beta, \tag{27.18}$$

then the last term is given by the contraction of $Md2M$ (`q_j`) with the diagonal of AA.

```
for(int j=0; j<NumPtcls; j++)
{
  HessType q_j;
  q_j=0.0;
  for(int k=0; k<NumPtcls; k++)
    q_j += psiMinv(j,k)*grad_grad_psiM(j,k);
  for(int i=0; i<num; i++)
  {
    Tensor<RealType,OHMMS_DIM> AA = dot(
      transpose(BFTrans->Amat(i,FirstIndex+j)),
      BFTrans->Amat(i,FirstIndex+j)
    );
    myL(i) += traceAtB(AA,q_j);
  }
}
```

27.9.4 Wavefunction Parameter Derivative

In order to use the robust linear optimization method of ref. [20], the trial wavefunction needs to know its contributions to the overlap and hamiltonian matrices. In particular, we need derivatives of these matrices with respect to wavefunction parameters. As a consequence, the wavefunction ψ needs to be able to evaluate $\frac{\partial}{\partial p}\ln\psi$ and $\frac{\partial}{\partial p}\frac{\mathcal{H}\psi}{\psi}$, where p is a parameter.

When two-body backflow is considered, a wavefunction parameter p enters the η function only (eq. (27.7)). $\boldsymbol{r}$, ϕ, and M are do not explicitly dependent on p. Derivative of the log value is almost identical to particle gradient. Namely, eq. (27.9) applies upon the substitution $r_i^\alpha \to p$

$$\frac{\partial}{\partial p}\ln\det M = \sum_{j=1}^{N}\sum_{\beta=1}^{3} F_{jj}^{\beta}\left({}_pC_j^{\beta}\right), \tag{27.19}$$

where the quasi-particle derivatives are stored in `Cmat`

$${}_pC_i^{\alpha} \equiv \frac{\partial}{\partial p}x_i^{\alpha}. \tag{27.20}$$

The change in local kinetic energy is a lot more difficult to calculate

$$\frac{\partial T_{\text{local}}}{\partial p} = \frac{\partial}{\partial p}\left\{\left(\sum_{i=1}^{N}\frac{1}{2m_i}\nabla_i^2\right)\ln\det M\right\} = \sum_{i=1}^{N}\frac{1}{2m_i}\frac{\partial}{\partial p}L_i, \tag{27.21}$$

where L_i is the particle laplacian defined in eq. (27.13). In order to evaluate eq. (27.21), we need to calculate parameter derivatives of all three terms defined in the laplacian evalaution. Namely $(B)(F)$, $(AA)(FF)$, and $\mathrm{tr}(AA, Md2M)$, where I have put parentheses around previously identified data structures. After $\frac{\partial}{\partial p}$ hits, each of the 3 terms will split into two terms by the product rule. Each smaller term will contain a contraction of two data structures. Therefore, we will need to calculate the parameter derivatives of each data structure defined in the laplacian evaluation

$${}_pX_{ij}^{\alpha\beta} \equiv \frac{\partial}{\partial p}A_{ij}^{\alpha\beta} \tag{27.22}$$

$${}_pY_{ij}^{\alpha} \equiv \frac{\partial}{\partial p}B_{ij}^{\alpha} \tag{27.23}$$

$${}_pdF_{ij}^{\alpha} \equiv \frac{\partial}{\partial p}F_{ij}^{\alpha} \tag{27.24}$$

$${}_{pi}AA'^{\beta\gamma}_{jk} \equiv \frac{\partial}{\partial p}{}_iAA_{jk}^{\beta\gamma} \tag{27.25}$$

$${}_pFF'^{\alpha\gamma}_{jk} \equiv \frac{\partial}{\partial p}FF_{jk}^{\alpha\gamma} \tag{27.26}$$

$${}_pMd2M'^{\beta\gamma}_{j} \equiv \frac{\partial}{\partial p}Md2M_{j}^{\beta\gamma} \tag{27.27}$$

X and Y are stored as `Xmat`, and `Ymat_full` (NOT `Ymat`) in the code. dF is `dFa`. AA' is not fully store, intermediate values are stored in `Aij_sum` and `a_j_sum`. FF' is calculated on-the-fly as $dF \otimes F + F \otimes dF$. $Md2M'$ is not stored, intermediate values are stored in `q_j_prime`.

References

[1] D. Ceperley, "Ground state of the fermion one-component plasma: A monte carlo study in two and three dimensions," *Phys. Rev. B*, vol. 18, pp. 3126–3138, Oct. 1978.

[2] S. Fahy, X. W. Wang, and S. G. Louie, "Variational quantum Monte Carlo nonlocal pseudopotential approach to solids: Formulation and application to diamond, graphite, and silicon," *Physical Review B*, vol. 42, no. 6, pp. 3503–3522, 1990.

[3] N. D. Drummond, M. D. Towler, and R. J. Needs, "Jastrow correlation factor for atoms, molecules, and solids," *Physical Review B - Condensed Matter and Materials Physics*, vol. 70, no. 23, pp. 1–11, 2004.

[4] M. Burkatzki, C. Filippi, and M. Dolg, "Energy-consistent pseudopotentials for quantum monte carlo calculations," *The Journal of Chemical Physics*, vol. 126, no. 23, pp. –, 2007.

[5] T. Kato, "Fundamental properties of hamiltonian operators of the schrodinger type," *Transactions of the American Mathematical Society*, vol. 70, pp. 195–211, 1951.

[6] M. Burkatzki, C. Filippi, and M. Dolg, "Energy-consistent small-core pseudopotentials for 3d-transition metals adapted to quantum monte carlo calculations," *The Journal of Chemical Physics*, vol. 129, no. 16, pp. –, 2008.

[7] K. P. Esler, J. Kim, D. M. Ceperley, and L. Shulenburger, "Accelerating quantum monte carlo simulations of real materials on gpu clusters," *Computing in Science and Engineering*, vol. 14, no. 1, pp. 40–51, 2012.

[8] J. T. Krogel, M. Yu, J. Kim, and D. M. Ceperley, "Quantum energy density: Improved efficiency for quantum monte carlo calculations," *Phys. Rev. B*, vol. 88, p. 035137, Jul 2013.

[9] J. T. Krogel, J. Kim, and F. A. Reboredo, "Energy density matrix formalism for interacting quantum systems: Quantum monte carlo study," *Phys. Rev. B*, vol. 90, p. 035125, Jul 2014.

[10] V. Natoli and D. M. Ceperley, "An optimized method for treating long-range potentials," *Journal of Computational Physics*, vol. 117, no. 1, pp. 171 – 178, 1995.

[11] L. Mitas, E. L. Shirley, and D. M. Ceperley, "Nonlocal pseudopotentials and diffusion monte carlo," *The Journal of Chemical Physics*, vol. 95, no. 5, pp. 3467–3475, 1991.

[12] S. Chiesa, D. M. Ceperley, R. M. Martin, and M. Holzmann, "Finite-size error in many-body simulations with long-range interactions," *Phys. Rev. Lett.*, vol. 97, p. 076404, Aug 2006.

[13] D. Alfè and M. J. Gillan, "An efficient localized basis set for quantum Monte Carlo calculations on condensed matter," *Physical Review B*, vol. 70, no. 16, p. 161101, 2004.

[14] L. Zhao and E. Neuscamman, "An efficient variational principle for the direct optimization of excited states," *J. Chem. Theory. Comput.*, vol. 12, p. 3436, 2016.

[15] L. Zhao and E. Neuscamman, "A blocked linear method for optimizing large parameter sets in variational monte carlo," *J. Chem. Theory. Comput.*, 2017. DOI: 10.1021/acs.jctc.7b00119.

[16] J. T. Krogel, "Nexus: A modular workflow management system for quantum simulation codes," *Computer Physics Communications*, vol. 198, pp. 154 – 168, 2016.

[17] A. Mathuriya, Y. Luo, R. C. Clay, III, A. Benali, L. Shulenburger, and J. Kim, "Embracing a new era of highly efficient and productive quantum monte carlo simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, (New York, NY, USA), pp. 38:1–38:12, ACM, 2017.

[18] Y. Kwon, D. M. Ceperley, and R. M. Martin, "Effects of three-body and backflow correlations in the two-dimensional electron gas," *Phys. Rev. B*, vol. 48, pp. 12037–12046, Oct 1993.

[19] C. J. Umrigar, M. P. Nightingale, and K. J. Runge, "A diffusion Monte Carlo algorithm with very small timestep errors A diffusion Monte Carlo algorithm with very small time-step errors," *The Journal of Chemical Physics*, vol. 99, no. 4, p. 2865, 1993.

[20] J. Toulouse and C. J. Umrigar, "Optimization of quantum monte carlo wave functions by energy minimization," *The Journal of Chemical Physics*, vol. 126, no. 8, p. 084102, 2007.

[21] S. Zhang and H. Krakauer, "Quantum monte carlo method using phase-free random walks with slater determinants," *Phys. Rev. Lett.*, vol. 90, p. 136401, Apr 2003.

[22] W. Purwanto and S. Zhang, "Quantum monte carlo method for the ground state of many-boson systems," *Phys. Rev. E*, vol. 70, p. 056702, Nov 2004.

[23] S. Zhang, "Auxiliary-field quantum monte carlo for correlated electron systems," *Modeling and Simulation*, vol. 3, 2013.

[24] N. S. Blunt and E. Neuscamman, "Charge-transfer excited states: Seeking a balanced and efficient wave function ansatz in variational Monte Carlo," *The Journal of Chemical Physics*, vol. 147, p. 194101, Nov. 2017.

[25] M. Casula, "Beyond the locality approximation in the standard diffusion Monte Carlo method," *Physical Review B - Condensed Matter and Materials Physics*, vol. 74, pp. 1–4, 2006.

[26] M. Casula, S. Moroni, S. Sorella, and C. Filippi, "Size-consistent variational approaches to nonlocal pseudopotentials: Standard and lattice regularized diffusion Monte Carlo methods revisited," *Journal of Chemical Physics*, vol. 132, no. 15, 2010.

[27] M. C. E. Giner, A. Scemama, "Using perturbatively selected configuration interaction in quantum monte carlo calculations," *Canadian Journal of Chemistry*, vol. 91, p. 9, 2013.

[28] A. Scemamma, "Quantum package." https://github.com/LCPQ/quantum_package, 2013–2017.

[29] S. Diner, J. P. Malrieu, and P. Claverie, "The use of perturbation methods for the study of the effects of configuration interaction," *Theoretica chimica acta*, vol. 8, no. 5, pp. 390–403, 1967.

[30] R. K. Nesbet, "Configuration interaction in orbital theories," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 230, no. 1182, pp. 312–321, 1955.

[31] A. Scemama, Y. Garniron, M. Caffarel, and P.-F. Loos, "Deterministic construction of nodal surfaces within quantum monte carlo: the case of fes," *Journal of Chemical Theory and Computation*, vol. 0, no. ja, p. null, 0.

[32] Y. Garniron, E. Giner, J.-P. Malrieu, and A. Scemama, "Alternative definition of excitation amplitudes in multi-reference state-specific coupled cluster," *The Journal of Chemical Physics*, vol. 146, no. 15, p. 154107, 2017.

[33] Y. Garniron, A. Scemama, P.-F. Loos, and M. Caffarel, "Hybrid stochastic-deterministic calculation of the second-order perturbative contribution of multireference perturbation theory," *The Journal of Chemical Physics*, vol. 147, no. 3, p. 034101, 2017.

[34] A. Scemama, T. Applencourt, E. Giner, and M. Caffarel, "Quantum monte carlo with very large multideterminant wavefunctions," *Journal of Computational Chemistry*, vol. 37, no. 20, pp. 1866–1875, 2016.

[35] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, "General atomic and molecular electronic structure system," *Journal of Computational Chemistry*, vol. 14, no. 11, pp. 1347–1363, 1993.

[36] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K.-L. Chan, "Pyscf: the python-based simulations of chemistry framework," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 8, no. 1, pp. n/a–n/a, 2018.

[37] D. G. Fedorov, R. M. Olson, K. Kitaura, M. S. Gordon, and S. Koseki, "A new hierarchical parallelization scheme: Generalized distributed data interface (gddi), and an application to the fragment molecular orbital method (fmo)," *Journal of Computational Chemistry*, vol. 25, no. 6, pp. 872–880, 2004.

[38] A. Ma, M. D. Towler, N. D. Drummond, and R. J. Needs, "Scheme for adding electron–nucleus cusps to gaussian orbitals," *The Journal of Chemical Physics*, vol. 122, no. 22, p. 224322, 2005.

[39] T. Gaskell, "The collective treatment of a fermi gas: Ii," *Proceedings of the Physical Society*, vol. 77, no. 6, p. 1182, 1961.

[40] T. Gaskell, "The collective treatment of many-body systems: Iii," *Proceedings of the Physical Society*, vol. 80, no. 5, p. 1091, 1962.

[41] Y. Luo, K. P. Esler, P. R. C. Kent, and L. Shulenburger, "An efficient hybrid orbital representation for quantum monte carlo calculations," *The Journal of Chemical Physics*, vol. 149, no. 8, p. 084107, 2018.

[42] G. Onida, L. Reining, and A. Rubio, "Electronic excitations: density-functional versus many-body Green's-function approaches," *Reviews of Modern Physics*, vol. 74, no. 2, pp. 601–659, 2002.

[43] S. P. Ong, W. D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V. L. Chevrier, K. A. Persson, and G. Ceder, "Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis," *Computational Materials Science*, vol. 68, pp. 314 – 319, 2013.

[44] A. Kokalj, "XCrySDen—a new program for displaying crystalline structures and electron densities," *Journal of Molecular Graphics and Modelling*, vol. 17, no. 3, pp. 176 – 179, 1999.

[45] Y. Hinuma, G. Pizzi, Y. Kumagai, F. Oba, and I. Tanaka, "Band structure diagram paths based on crystallography," *Computational Materials Science*, vol. 128, pp. 140 – 184, 2017.

[46] T. McDaniel, E. F. D'Azevedo, Y. W. Li, K. Wong, and P. R. C. Kent, "Delayed slater determinant update algorithms for high efficiency quantum monte carlo," *The Journal of Chemical Physics*, vol. 147, p. 174107, nov 2017.

[47] Y. Luo and J. Kim, "An highly efficient delayed update algorithm for evaluating slater determinants in quantum monte carlo," *in preparation*, 2018.

Appendix A

Derivation of twist averaging efficiency

In this appendix we derive the relative statistical efficiency of twist averaging with an irreducible (weighted) set of k-points versus using uniform weights over an unreduced set of k-points (*e.g.* a full Monkhorst-Pack mesh).

Consider the weighted average of a set of statistical variables $\{x_m\}$ with weights $\{w_m\}$:

$$x_{TA} = \frac{\sum_m w_m x_m}{\sum_m w_m} \tag{A.1}$$

If produced by a finite quantum Monte Carlo run at a set of twist angles/k-points $\{k_m\}$, each variable mean $\langle x_m \rangle$ has a statistical error bar σ_m and we can also obtain the statistical error bar of the mean of the twist averaged quantity $\langle x_{TA} \rangle$:

$$\sigma_{TA} = \frac{\left(\sum_m w_m^2 \sigma_m^2\right)^{1/2}}{\sum_m w_m} \tag{A.2}$$

The error bar of each individual twist σ_m, is related to the autocorrelation time κ_m, intrinsic variance v_m, and number of post-equilibration Monte Carlo steps N_{step} in the following way:

$$\sigma_m^2 = \frac{\kappa_m v_m}{N_{step}} \tag{A.3}$$

In the setting of twist averaging, the autocorrelation time and variance for different twist angles are often very similar across twists and we have

$$\sigma_m^2 = \sigma^2 = \frac{\kappa v}{N_{step}} \tag{A.4}$$

If we define the total weight as W, *i.e.* $W \equiv \sum_{m=1}^{M} w_m$, for the weighted case with M irreducible twists the error bar is

$$\sigma_{TA}^{weighted} = \frac{\left(\sum_{m=1}^{M} w_m^2\right)^{1/2}}{W} \sigma \tag{A.5}$$

For uniform weighting with $w_m = 1$ the number of twists is W and we have

$$\sigma_{TA}^{uniform} = \frac{1}{\sqrt{W}} \sigma \tag{A.6}$$

We are interested in comparing the efficiency of choosing weights uniformly or based on the irreducible multiplicity of each twist angle for a given target error bar σ_{target}. The number of Monte Carlo steps required to reach this target for uniform weighting is

$$N_{step}^{uniform} = \frac{1}{W}\frac{\kappa v}{\sigma_{target}^2} \tag{A.7}$$

while for non-uniform weighting we have

$$\begin{aligned} N_{step}^{weighted} &= \frac{\sum_{m=1}^{M} w_m^2}{W^2}\frac{\kappa v}{\sigma_{target}^2} \\ &= \frac{\sum_{m=1}^{M} w_m^2}{W} N_{step}^{uniform} \end{aligned} \tag{A.8}$$

The Monte Carlo efficiency is defined as

$$\xi = \frac{1}{\sigma^2 t} \tag{A.9}$$

where σ is the error bar and t is the total cpu time required for the Monte Carlo run.

The main advantage made possible by irreducible twist weighting is to reduce the equilibration time overhead by having fewer twists, and hence fewer Monte Carlo runs to equilibrate. In the context of twist averaging, the total cpu time for a run can be considered to be

$$t = N_{twist}(N_{eq} + N_{step})t_{step} \tag{A.10}$$

where N_{twist} is the number of twists, N_{eq} is the number of Monte Carlo steps required to reach equilibrium, N_{step} is the number of Monte Carlo steps included in the statisical averaging as before, and t_{step} is the wall clock time required to complete a single Monte Carlo step. For uniform weighting $N_{twist} = W$ while for irreducible weighting $N_{twist} = M$.

We can now calculate the relative efficiency (η) of irreducible vs. uniform twist weighting with the aim of obtaining a target error bar σ_{target}:

$$\begin{aligned} \eta &= \frac{\xi_{TA}^{weighted}}{\xi_{TA}^{uniform}} \\ &= \frac{\sigma_{target}^2 t_{TA}^{uniform}}{\sigma_{target}^2 t_{TA}^{weighted}} \\ &= \frac{W(N_{eq} + N_{step}^{uniform})}{M(N_{eq} + N_{step}^{weighted})} \\ &= \frac{W(N_{eq} + N_{step}^{uniform})}{M(N_{eq} + \frac{\sum_{m=1}^{M} w_m^2}{W} N_{step}^{uniform})} \\ &= \frac{W}{M}\frac{1+f}{1+\frac{\sum_{m=1}^{M} w_m^2}{W} f} \end{aligned} \tag{A.11}$$

In this last expression, f is the ratio of the number of usable Monte Carlo steps to the number that must be discarded during equilibration ($f = N_{step}^{uniform}/N_{eq}$) and as before $W = \sum_m w_m$, which is the number of twist angles in the uniform weighting case. It is important to recall that $N_{step}^{uniform}$ in

f is defined relative to uniform weighting; it is the number of Monte Carlo steps required to reach a target accuracy in the case of uniform twist weights.

The formula for η above can be easily changed with the help of Eq. A.8 to reflect the number of Monte Carlo steps obtained in an irreducibly weighted run instead. A good exercise is to consider runs one has already completed with either uniform or irreducible weighting and calculate the expected efficiency change had the opposite type of weighting been used.

The break even point ($\eta = 1$) can be found at a usable step fraction of

$$f = \frac{W - M}{M\frac{\sum_{m=1}^{M} w_m^2}{W} - W} \tag{A.12}$$

The relative efficiency (η) above is useful to consider in view of certain scenarios. An important case is where the number of required sampling steps is no larger than the number of equilibration steps, *i.e.* $f \approx 1$. For a very simple case with 8 uniform twists with irreducible multiplicities of $w_m \in \{1, 3, 3, 1\}$ ($W = 8$, $M = 4$), the relative efficiency of irreducible vs. uniform weighting is $\eta = \frac{8}{4}\frac{2}{1+20/8} \approx 1.14$. In this case, irreducible weighting is about 14% more efficient than uniform weighting.

Another interesting case is where the number of sampling steps you can reach with uniform twists before wall clock time runs out is small relative to the number of equilibration steps ($f \to 0$). In this limit $\eta \approx W/M$. For our 8 uniform twist example, this would result in a relative efficiency of $\eta = 8/4 = 2$ making irreducible weighting twice as efficient.

A final case of interest is where the equilibration time is short relative to the available sampling time ($f \to \infty$), giving $\eta \approx W^2/(M\sum_{m=1}^{M} w_m^2)$. Again for our simple example we find $\eta = 8^2/(4 \times 20) \approx 0.8$, with uniform weighting being 25% more efficient than irreducible.

For this simple example, the crossover point for irreducible weighting being more efficient than uniform weighting is $f < 2$, *i.e.* when the available sampling period is less than twice the length of the equilibration period. The expected efficiency ratio and crossover point should be checked for the particular case under consideration to inform the choice between twist averaging methods.

Appendix B

QMCPACK papers

The following is a list of all papers, theses, and book chapters known to use QMCPACK. Please let the developers know if your paper is missing, if you know of other works, or an entry is incorrect. We list papers whether they cite QMCPACK directly or not. This list will be placed on the http://www.qmcpack.org website.

[1] D. Das, *Quantum Monte Carlo With a Stochastic Poisson Solver*. 2005.

[2] D. Das, R. M. Martin, and M. H. Kalos, "Quantum monte carlo method using a stochastic poisson solver," *Phys. Rev. E*, vol. 73, p. 046702, Apr 2006.

[3] J. E. Vincent, *Quantum Monte Carlo Calculations of the Electronic Excitations of Germanium Atoms, Molecules and Nanoclusters Using Core-Polarization Potentials*. 2006.

[4] J. E. Vincent, J. Kim, and R. M. Martin, "Quantum monte carlo calculations of the optical gaps of ge nanoclusters using core-polarization potentials," *Phys. Rev. B*, vol. 75, p. 045302, Jan 2007.

[5] T. D. Beaudet, M. Casula, J. Kim, S. Sorella, and R. M. Martin, "Molecular hydrogen adsorbed on benzene: Insights from a quantum monte carlo study," *The Journal of Chemical Physics*, vol. 129, no. 16, p. 164711, 2008.

[6] K. P. Esler, J. Kim, D. M. Ceperley, W. Purwanto, E. J. Walter, H. Krakauer, S. Zhang, P. R. C. Kent, R. G. Hennig, C. Umrigar, M. Bajdich, J. Kolorenč, L. Mitas, and A. Srinivasan, "Quantum monte carlo algorithms for electronic structure at the petascale; the endstation project," *Journal of Physics: Conference Series*, vol. 125, no. 1, p. 012057, 2008.

[7] B. Clark, *Strongly Correlated Systems Approach Through Quantum Monte Carlo*. PhD thesis, University of Illinois at Urbana-Champaign, 2009.

[8] J. R. Gergely, *Quantum Monte Carlo Methods for First Principles Simulation of Liquid Water*. PhD thesis, University of Illinois at Urbana-Champaign, 2009.

[9] K. P. Esler, R. E. Cohen, B. Militzer, J. Kim, R. J. Needs, and M. D. Towler, "Fundamental high-pressure calibration from all-electron quantum monte carlo calculations," *Phys. Rev. Lett.*, vol. 104, p. 185702, May 2010.

[10] J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. E. Stone, and J. C. Phillips, "Quantifying the impact of gpus on performance and energy efficiency in hpc clusters," in *International Conference on Green Computing*, pp. 317–324, Aug 2010.

[11] S. Huotari, J. A. Soininen, T. Pylkkänen, K. Hämäläinen, A. Issolah, A. Titov, J. McMinis, J. Kim, K. Esler, D. M. Ceperley, M. Holzmann, and V. Olevano, "Momentum distribution and renormalization factor in sodium and the electron gas," *Phys. Rev. Lett.*, vol. 105, p. 086403, Aug 2010.

[12] K. Hongo, M. A. Watson, R. S. Sánchez-Carrera, T. Iitaka, and A. Aspuru-Guzik, "Failure of conventional density functionals for the prediction of molecular crystal polymorphism: A quantum monte carlo study," *The Journal of Physical Chemistry Letters*, vol. 1, no. 12, pp. 1789–1794, 2010.

[13] M. Bajdich, P. R. C. Kent, J. Kim, and F. A. Reboredo, "Simple impurity embedded in a spherical jellium: Approximations of density functional theory compared to quantum monte carlo benchmarks," *Phys. Rev. B*, vol. 84, p. 075131, Aug 2011.

[14] M. Holzmann, B. Bernu, C. Pierleoni, J. McMinis, D. M. Ceperley, V. Olevano, and L. Delle Site, "Momentum distribution of the homogeneous electron gas," *Phys. Rev. Lett.*, vol. 107, p. 110402, Sep 2011.

[15] K. Ahuja, *Recycling Krylov subspaces and preconditioners*. PhD thesis, Virginia Polytechnic Institute and State University, 2011.

[16] K. Ahuja, B. K. Clark, E. de Sturler, D. M. Ceperley, and J. Kim, "Improved scaling for quantum monte carlo on insulators," *SIAM Journal on Scientific Computing*, vol. 33, no. 4, pp. 1837–1859, 2011.

[17] T. D. Beaudet, *Quantum Monte Carlo Study of Hydrogen Adsorption on Carbon and Transition Metal Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.

[18] B. K. Clark, M. A. Morales, J. McMinis, J. Kim, and G. E. Scuseria, "Computing the energy of a water molecule using multideterminants: A simple, efficient algorithm," *The Journal of Chemical Physics*, vol. 135, no. 24, p. 244105, 2011.

[19] K. Esler, J. Kim, D. Ceperley, and L. Shulenburger, "Accelerating quantum monte carlo simulations of real materials on GPU clusters," *Computing in Science & Engineering*, vol. 14, pp. 40–51, jan 2012.

[20] G. H. Bauer, T. Hoefler, W. T. Kramer, and R. A. Fiedler, "Analyses and modeling of applications used to demonstrate sustained petascale performance on blue waters," in *Proceedings of the Annual Meeting of the Cray Users Group*, 2012.

[21] J. Kim, K. P. Esler, J. McMinis, M. A. Morales, B. K. Clark, L. Shulenburger, and D. M. Ceperley, "Hybrid algorithms in quantum monte carlo," *Journal of Physics: Conference Series*, vol. 402, no. 1, p. 012008, 2012.

[22] M. A. Watson, K. Hongo, T. Iitaka, and A. Aspuru-Guzik, *A Benchmark Quantum Monte Carlo Study of Molecular Crystal Polymorphism: A Challenging Case for Density-Functional Theory*, ch. 9, pp. 101–117. 2012.

[23] S. Coghlan, K. Kumaran, R. M. Loy, P. Messina, V. Morozov, J. C. Osborn, S. Parker, K. M. Riley, N. A. Romero, and T. J. Williams, "Argonne applications for the ibm blue gene/q, mira," *IBM Journal of Research and Development*, vol. 57, pp. 12:1–12:11, Jan 2013.

[24] J. McMinis and N. M. Tubman, "Renyi entropy of the interacting fermi liquid," *Phys. Rev. B*, vol. 87, p. 081108, Feb 2013.

[25] C. Sudheer, S. Krishnan, A. Srinivasan, and P. Kent, "Dynamic load balancing for petascale quantum monte carlo applications: The alias method," *Computer Physics Communications*, vol. 184, pp. 284–292, feb 2013.

[26] B. Swingle, J. McMinis, and N. M. Tubman, "Oscillating terms in the renyi entropy of fermi gases and liquids," *Phys. Rev. B*, vol. 87, p. 235112, Jun 2013.

[27] J. T. Krogel, M. Yu, J. Kim, and D. M. Ceperley, "Quantum energy density: Improved efficiency for quantum monte carlo calculations," *Phys. Rev. B*, vol. 88, p. 035137, Jul 2013.

[28] S. Herbein, M. Matheny, M. Wezowicz, J. Krogel, J. Logan, J. Kim, S. Klasky, and M. Taufer, "Performance impact of i/o on qmcpack simulations at the petascale and beyond," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, pp. 92–99, Dec 2013.

[29] L. Shulenburger and T. R. Mattsson, "Quantum monte carlo applied to solids," *Phys. Rev. B*, vol. 88, p. 245117, Dec 2013.

[30] J. Krogel, *Energetics of neutral interstitials in germanium*. PhD thesis, University of Illinois at Urbana-Champaign, 2013.

[31] J. McMinis, *Benchmark studies using quantum Monte Carlo: pressure estimators, energy, and entanglement*. PhD thesis, University of Illinois at Urbana-Champaign, 2013.

[32] R. C. Clay, J. Mcminis, J. M. McMahon, C. Pierleoni, D. M. Ceperley, and M. A. Morales, "Benchmarking exchange-correlation functionals for hydrogen at high pressures using quantum monte carlo," *Phys. Rev. B*, vol. 89, p. 184106, May 2014.

[33] K. Foyevtsova, J. T. Krogel, J. Kim, P. Kent, E. Dagotto, and F. A. Reboredo, "Ab initioQuantum monte carlo calculations of spin superexchange in cuprates: The benchmarking case ofCa2cuo3," *Physical Review X*, vol. 4, jul 2014.

[34] J. T. Krogel, J. Kim, and F. A. Reboredo, "Energy density matrix formalism for interacting quantum systems: Quantum monte carlo study," *Phys. Rev. B*, vol. 90, p. 035125, Jul 2014.

[35] S. Herbein, S. Klasky, and M. Taufer, "Benchmarking the performance of scientific applications with irregular i/o at the extreme scale," in *2014 43rd International Conference on Parallel Processing Workshops*, pp. 292–301, Sept 2014.

[36] L. Shulenburger, M. P. Desjarlais, and T. R. Mattsson, "Theory of melting at high pressures: Amending density functional theory with quantum monte carlo," *Phys. Rev. B*, vol. 90, p. 140104, Oct 2014.

[37] N. M. Tubman, I. Kylänpää, S. Hammes-Schiffer, and D. M. Ceperley, "Beyond the born-oppenheimer approximation with quantum monte carlo methods," *Phys. Rev. A*, vol. 90, p. 042507, Oct 2014.

[38] Y. Lin, R. E. Cohen, S. Stackhouse, K. P. Driver, B. Militzer, L. Shulenburger, and J. Kim, "Equations of state and stability of $mgsio_3$ perovskite and post-perovskite phases from quantum monte carlo simulations," *Phys. Rev. B*, vol. 90, p. 184103, Nov 2014.

[39] T. R. Mattsson, S. Root, A. E. Mattsson, L. Shulenburger, R. J. Magyar, and D. G. Flicker, "Validating density-functional theory simulations at high energy-density conditions with liquid krypton shock experiments to 850 gpa on sandia's z machine," *Phys. Rev. B*, vol. 90, p. 184105, Nov 2014.

[40] M. Matheny, S. Herbein, N. Podhorszki, S. Klasky, and M. Taufer, "Using surrogate-based modeling to predict optimal i/o parameters of applications at the extreme scale," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 568–575, Dec 2014.

[41] A. Benali, L. Shulenburger, N. A. Romero, J. Kim, and O. A. von Lilienfeld, "Application of diffusion monte carlo to materials dominated by van der waals interactions," *Journal of Chemical Theory and Computation*, vol. 10, no. 8, pp. 3417–3422, 2014. PMID: 26588310.

[42] P. Ganesh, J. Kim, C. Park, M. Yoon, F. A. Reboredo, and P. R. C. Kent, "Binding and diffusion of lithium in graphite: Quantum monte carlo benchmarks and validation of van der waals density functional methods," *Journal of Chemical Theory and Computation*, vol. 10, no. 12, pp. 5318–5323, 2014. PMID: 26583215.

[43] L. Koziol and M. M. Morales, "A fixed-node diffusion monte carlo study of the 1,2,3-tridehydrobenzene triradical," *The Journal of Chemical Physics*, vol. 140, no. 22, p. 224316, 2014.

[44] C. L. Mendes, B. Bode, G. H. Bauer, J. Enos, C. Beldica, and W. T. Kramer, "Deploying a large petascale system: The blue waters experience," *Procedia Computer Science*, vol. 29, pp. 198 – 209, 2014.

[45] M. A. Morales, R. Clay, C. Pierleoni, and D. M. Ceperley, "First principles methods: A perspective from quantum monte carlo," *Entropy*, vol. 16, no. 1, pp. 287–321, 2014.

[46] M. A. Morales, J. R. Gergely, J. McMinis, J. M. McMahon, J. Kim, and D. M. Ceperley, "Quantum monte carlo benchmark of exchange-correlation functionals for bulk water," *Journal of Chemical Theory and Computation*, vol. 10, no. 6, pp. 2355–2362, 2014. PMID: 26580755.

[47] H. Shin, S. Kang, J. Koo, H. Lee, J. Kim, and Y. Kwon, "Cohesion energetics of carbon allotropes: Quantum monte carlo study," *The Journal of Chemical Physics*, vol. 140, no. 11, p. 114702, 2014.

[48] S. Sreepathi, M. L. Grodowitz, R. Lim, P. Taffet, P. C. Roth, J. Meredith, S. Lee, D. Li, and J. Vetter, "Application characterization using oxbow toolkit and pads infrastructure," in *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, Co-HPC '14, (Piscataway, NJ, USA), pp. 55–63, IEEE Press, 2014.

[49] J. McMinis, R. C. Clay, D. Lee, and M. A. Morales, "Molecular to atomic phase transition in hydrogen under high pressure," *Phys. Rev. Lett.*, vol. 114, p. 105305, Mar 2015.

[50] W. M. Hwu, L. W. Chang, H. S. Kim, A. Dakkak, and I. E. Hajj, "Transitioning hpc software to exascale heterogeneous computing," in *2015 Computational Electromagnetics International Workshop (CEM)*, pp. 1–2, July 2015.

[51] W. Zhou, J. Chen, Z. Wang, X. Xu, L. Xu, and Y. Tang, "Time-dimension communication characterization of representative scientific applications on tianhe-2," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 423–429, Aug 2015.

[52] M. Deible, "Theory and applications of quantum monte carlo," September 2015.

[53] S. Root, L. Shulenburger, R. W. Lemke, D. H. Dolan, T. R. Mattsson, and M. P. Desjarlais, "Shock response and phase transitions of mgo at planetary impact conditions," *Phys. Rev. Lett.*, vol. 115, p. 198501, Nov 2015.

[54] M. J. Deible, M. Kessler, K. E. Gasperich, and K. D. Jordan, "Quantum monte carlo calculation of the binding energy of the beryllium dimer," *The Journal of Chemical Physics*, vol. 143, no. 8, p. 084116, 2015.

[55] R. Gioiosa, R. W. Wisniewski, R. Murty, and T. Inglett, "Analyzing system calls in multi-os hierarchical environments," in *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '15, (New York, NY, USA), pp. 6:1–6:8, ACM, 2015.

[56] J. McMinis, M. A. Morales, D. M. Ceperley, and J. Kim, "The transition to the metallic state in low density hydrogen," *The Journal of Chemical Physics*, vol. 143, no. 19, p. 194703, 2015.

[57] C. L. Mendes, B. Bode, G. H. Bauer, J. Enos, C. Beldica, and W. T. Kramer, "Deployment and testing of the sustained petascale blue waters system," *Journal of Computational Science*, vol. 10, pp. 327 – 337, 2015.

[58] C. Mitra, J. T. Krogel, J. A. Santana, and F. A. Reboredo, "Many-body ab initio diffusion quantum monte carlo applied to the strongly correlated oxide nio," *The Journal of Chemical Physics*, vol. 143, no. 16, p. 164710, 2015.

[59] Q. Niu, *Characterization and Enhancement of Data Locality and Load Balancing for Irregular Applications*. PhD thesis, The Ohio State University, 2015.

[60] R. C. C. III and M. A. Morales, "Influence of single particle orbital sets and configuration selection on multideterminant wavefunctions in quantum monte carlo," *The Journal of Chemical Physics*, vol. 142, no. 23, p. 234103, 2015.

[61] J. A. Santana, J. T. Krogel, J. Kim, P. R. C. Kent, and F. A. Reboredo, "Structural stability and defect energetics of zno from diffusion quantum monte carlo," *The Journal of Chemical Physics*, vol. 142, no. 16, p. 164705, 2015.

[62] L. Shulenburger, A. Baczewski, Z. Zhu, J. Guan, and D. Tománek, "The nature of the interlayer interaction in bulk and few-layer phosphorus," *Nano Letters*, vol. 15, no. 12, pp. 8170–8175, 2015. PMID: 26523860.

[63] L. Shulenburger, T. R. Mattsson, and M. Desjarlais, "Beyond chemical accuracy: The pseudopotential approximation in diffusion monte carlo calculations of the hcp to bcc phase transition in beryllium," *arXiv preprint arXiv:1501.03850*, 2015.

[64] Y. Yang, I. Kylänpää, N. M. Tubman, J. T. Krogel, S. Hammes-Schiffer, and D. M. Ceperley, "How large are nonadiabatic effects in atomic and diatomic systems?," *The Journal of Chemical Physics*, vol. 143, no. 12, p. 124308, 2015.

[65] R. C. Clay, M. Holzmann, D. M. Ceperley, and M. A. Morales, "Benchmarking density functionals for hydrogen-helium mixtures with quantum monte carlo: Energetics, pressures, and forces," *Phys. Rev. B*, vol. 93, p. 035121, Jan 2016.

[66] J. T. Krogel, J. A. Santana, and F. A. Reboredo, "Pseudopotentials for quantum monte carlo studies of transition metal oxides," *Phys. Rev. B*, vol. 93, p. 075143, Feb 2016.

[67] R. Nazarov, L. Shulenburger, M. Morales, and R. Q. Hood, "Benchmarking the pseudopotential and fixed-node approximations in diffusion monte carlo calculations of molecules and solids," *Phys. Rev. B*, vol. 93, p. 094111, Mar 2016.

[68] M. Holzmann, R. C. Clay, M. A. Morales, N. M. Tubman, D. M. Ceperley, and C. Pierleoni, "Theory of finite size effects for electronic quantum monte carlo calculations of liquids and solids," *Phys. Rev. B*, vol. 94, p. 035126, Jul 2016.

[69] A. Benali, L. Shulenburger, J. T. Krogel, X. Zhong, P. R. C. Kent, and O. Heinonen, "Quantum monte carlo analysis of a charge ordered insulating antiferromagnet: the ti4o7 magneli phase," *Phys. Chem. Chem. Phys.*, vol. 18, pp. 18323–18335, 2016.

[70] J.-P. Davis, M. D. Knudson, L. Shulenburger, and S. D. Crockett, "Mechanical and optical response of [100] lithium fluoride to multi-megabar dynamic pressures," *Journal of Applied Physics*, vol. 120, no. 16, p. 165901, 2016.

[71] S. Herbein, S. McDaniel, N. Podhorszki, J. Logan, S. Klasky, and M. Taufer, "Performance characterization of irregular i/o at the extreme scale," *Parallel Computing*, vol. 51, pp. 17 – 36, 2016. Special Issue on Parallel Programming Models and SystemsSoftware for High-End Computing.

[72] J. T. Krogel, "Nexus: A modular workflow management system for quantum simulation codes," *Computer Physics Communications*, vol. 198, pp. 154 – 168, 2016.

[73] M. G. Lopez, C. Bergstrom, Y. W. Li, W. Elwasif, and O. Hernandez, *Using C++ AMP to Accelerate HPC Applications on Multiple Platforms*, pp. 563–576. Cham: Springer International Publishing, 2016.

[74] Y. Luo, A. Benali, L. Shulenburger, J. T. Krogel, O. Heinonen, and P. R. C. Kent, "Phase stability of tio 2 polymorphs from diffusion quantum monte carlo," *New Journal of Physics*, vol. 18, no. 11, p. 113049, 2016.

[75] T. McDaniel, E. D'Azevedo, Y. W. Li, P. Kent, M. Wong, and K. Wong, "Delayed update algorithms for quantum monte carlo simulation on gpu: Extended abstract," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, (New York, NY, USA), pp. 13:1–13:4, ACM, 2016.

[76] Q. Niu, J. Dinan, S. Tirukkovalur, A. Benali, J. Kim, L. Mitas, L. Wagner, and P. Sadayappan, "Global-view coefficients: a data management solution for parallel quantum monte carlo applications," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 13, pp. 3655–3671, 2016. cpe.3748.

[77] A. D. Powell and R. Dawes, "Calculating potential energy curves with fixed-node diffusion monte carlo: Co and n2," *The Journal of Chemical Physics*, vol. 145, no. 22, p. 224308, 2016.

[78] J. A. Santana, J. T. Krogel, P. R. C. Kent, and F. A. Reboredo, "Cohesive energy and structural parameters of binary oxides of groups iia and iiib from diffusion quantum monte carlo," *The Journal of Chemical Physics*, vol. 144, no. 17, p. 174707, 2016.

[79] N. M. Tubman, Y. Yang, S. Hammes-Schiffer, and D. M. Ceperley, *Interpolated Wave Functions for Nonadiabatic Simulations with the Fixed-Node Quantum Monte Carlo Method*, ch. 203, pp. 47–61. 2016.

[80] D. C.-M. Yang, *Pairing and entanglement: quantum Monte Carlo studies*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[81] L. Zhao and E. Neuscamman, "An efficient variational principle for the direct optimization of excited states," *Journal of Chemical Theory and Computation*, vol. 12, no. 8, pp. 3436–3440, 2016. PMID: 27379468.

[82] B. V. D. Goetz and E. Neuscamman, "Suppressing ionic terms with number-counting jastrow factors in real space," *Journal of Chemical Theory and Computation*, vol. 13, pp. 2035–2042, apr 2017.

[83] A. Mathuriya, Y. Luo, A. Benali, L. Shulenburger, and J. Kim, "Optimization and parallelization of b-spline based orbital evaluations in QMC on multi/many-core shared memory processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, may 2017.

[84] J. T. Krogel and P. R. C. Kent, "Magnitude of pseudopotential localization errors in fixed node diffusion quantum monte carlo," *The Journal of Chemical Physics*, vol. 146, p. 244101, jun 2017.

[85] L. Zhao and E. Neuscamman, "A blocked linear method for optimizing large parameter sets in variational monte carlo," *Journal of Chemical Theory and Computation*, vol. 13, pp. 2604–2611, jun 2017.

[86] A. L. Dzubak, J. T. Krogel, and F. A. Reboredo, "Quantitative estimation of localization errors of 3d transition metal pseudopotentials in diffusion monte carlo," *The Journal of Chemical Physics*, vol. 147, p. 024102, jul 2017.

[87] J. A. Santana, J. T. Krogel, P. R. C. Kent, and F. A. Reboredo, "Diffusion quantum monte carlo calculations of SrFeO3 and LaFeO3," *The Journal of Chemical Physics*, vol. 147, p. 034701, jul 2017.

[88] K. Gasperich, M. Deible, and K. D. Jordan, "H4: A model system for assessing the performance of diffusion monte carlo calculations using a single slater determinant trial function," *The Journal of Chemical Physics*, vol. 147, p. 074106, aug 2017.

[89] P. J. Robinson, S. D. P. Flores, and E. Neuscamman, "Excitation variance matching with limited configuration interaction expansions in variational monte carlo," *The Journal of Chemical Physics*, vol. 147, p. 164114, oct 2017.

[90] J. A. Santana, R. Mishra, J. T. Krogel, A. Y. Borisevich, P. R. C. Kent, S. T. Pantelides, and F. A. Reboredo, "Quantum many-body effects in defective transition-metal-oxide superlattices," *Journal of Chemical Theory and Computation*, vol. 13, pp. 5604–5609, oct 2017.

[91] H. Shin, J. Kim, H. Lee, O. Heinonen, A. Benali, and Y. Kwon, "Nature of interlayer binding and stacking of sp–sp2 hybridized carbon layers: A quantum monte carlo study," *Journal of Chemical Theory and Computation*, vol. 13, pp. 5639–5646, oct 2017.

[92] N. S. Blunt and E. Neuscamman, "Charge-transfer excited states: Seeking a balanced and efficient wave function ansatz in variational monte carlo," *The Journal of Chemical Physics*, vol. 147, p. 194101, nov 2017.

[93] A. L. Dzubak, C. Mitra, M. Chance, S. Kuhn, G. E. Jellison, A. S. Sefat, J. T. Krogel, and F. A. Reboredo, "MnNiO3 revisited with modern theoretical and experimental methods," *The Journal of Chemical Physics*, vol. 147, p. 174703, nov 2017.

[94] I. Kylänpää, J. Balachandran, P. Ganesh, O. Heinonen, P. R. C. Kent, and J. T. Krogel, "Accuracy of ab initio electron correlation and electron densities in vanadium dioxide," *Physical Review Materials*, vol. 1, nov 2017.

[95] T. McDaniel, E. F. D'Azevedo, Y. W. Li, K. Wong, and P. R. C. Kent, "Delayed slater determinant update algorithms for high efficiency quantum monte carlo," *The Journal of Chemical Physics*, vol. 147, p. 174107, nov 2017.

[96] J. A. R. Shea and E. Neuscamman, "Size consistent excited states via algorithmic transformations between variational principles," *Journal of Chemical Theory and Computation*, vol. 13, pp. 6078–6088, nov 2017.

[97] M. C. Bennett, C. A. Melton, A. Annaberdiyev, G. Wang, L. Shulenburger, and L. Mitas, "A new generation of effective core potentials for correlated calculations," *The Journal of Chemical Physics*, vol. 147, p. 224106, dec 2017.

[98] H. Shin, Y. Luo, P. Ganesh, J. Balachandran, J. T. Krogel, P. R. C. Kent, A. Benali, and O. Heinonen, "Electronic properties of doped and defective NiO: A quantum monte carlo study," *Physical Review Materials*, vol. 1, dec 2017.

[99] A. Benali, D. Ceperley, E. M. D'Azevedo, M. Dewing, P. R. C. Kent, J. Kim, J. T. Krogel, Y. W. Li, Y. Luo, T. McDaniel, M. A. Morales, A. Mathuriya, L. Shulenburger, and N. M. Tubman, *Development of QMCPACK for Exascale Scientific Computing*, ch. Exascale Scientific Applications: Scalability and Performance Portability, pp. 461–480. CRC Press, 2017.

[100] Q. Liu, N. Podhorszki, J. Choi, J. Logan, M. Wolf, S. Klasky, T. Kurc, and X. He, "StoreRush: An application-level approach to harvesting idle storage in a best effort environment," *Procedia Computer Science*, vol. 108, pp. 475–484, 2017.

[101] A. Mathuriya, Y. Luo, R. C. Clay, A. Benali, L. Shulenburger, and J. Kim, "Embracing a new era of highly efficient and productive quantum monte carlo simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17*, ACM Press, 2017.

[102] J. T. Krogel and F. A. Reboredo, "Kinetic energy classification and smoothing for compact b-spline basis sets in quantum monte carlo," *The Journal of Chemical Physics*, vol. 148, p. 044110, jan 2018.

[103] J. Kim, A. Baczewski, T. Beaudet, A. Benali, C. Bennett, M. Berrill, N. Blunt, E. J. L. Borda, M. Casula, D. Ceperley, S. Chiesa, bryan K clark, R. Clay, K. Delaney, M. Dewing, K. Esler, H. Hao, O. Heinonen, P. R. C. Kent, J. T. Krogel, I. Kylänpää, Y. W. Li, M. G. Lopez, Y. Luo, F. Malone, R. Martin, A. Mathuriya, J. McMinis, C. Melton, L. Mitas, M. A. Morales, E. Neuscamman, W. Parker, S. Flores, N. A. Romero, B. Rubenstein, J. Shea, H. Shin, L. Shulenburger, A. Tillack, J. Townsend, N. Tubman, B. van der Goetz, J. Vincent, D. C. Yang, Y. Yang, S. Zhang, and L. Zhao, "QMCPACK : An open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules, and solids," *Journal of Physics: Condensed Matter*, mar 2018.

[104] S. Song, M.-C. Kim, E. Sim, A. Benali, O. Heinonen, and K. Burke, "Benchmarks and reliable DFT results for spin gaps of small ligand fe(II) complexes," *Journal of Chemical Theory and Computation*, vol. 14, pp. 2304–2311, apr 2018.

[105] A. Benali, Y. Luo, H. Shin, D. Pahls, and O. Heinonen, "Quantum monte carlo calculations of catalytic energy barriers in a metallorganic framework with transition-metal-functionalized nodes," *The Journal of Physical Chemistry C*, vol. 122, pp. 16683–16691, jun 2018.

[106] Y. Sharma, J. Balachandran, C. Sohn, J. T. Krogel, P. Ganesh, L. Collins, A. V. Ievlev, Q. Li, X. Gao, N. Balke, O. S. Ovchinnikova, S. V. Kalinin, O. Heinonen, and H. N. Lee, "Nanoscale control of oxygen defects and metal–insulator transition in epitaxial vanadium dioxides," *ACS Nano*, vol. 12, pp. 7159–7166, jun 2018.

[107] P. R. C. Kent and G. Kotliar, "Toward a predictive theory of correlated materials," *Science*, vol. 361, pp. 348–354, jul 2018.

[108] A. Scemama, A. Benali, D. Jacquemin, M. Caffarel, and P.-F. Loos, "Excitation energies from diffusion monte carlo using selected configuration interaction nodes," *The Journal of Chemical Physics*, vol. 149, p. 034108, jul 2018.

[109] H. Shin, A. Benali, Y. Luo, E. Crabb, A. Lopez-Bezanilla, L. E. Ratcliff, A. M. Jokisaari, and O. Heinonen, "Zirconia and hafnia polymorphs: Ground-state structural properties from diffusion monte carlo," *Physical Review Materials*, vol. 2, jul 2018.

[110] J. Ahn, I. Hong, Y. Kwon, R. C. Clay, L. Shulenburger, H. Shin, and A. Benali, "Phase stability and interlayer interaction of blue phosphorene," *Physical Review B*, vol. 98, aug 2018.

[111] Y. Luo, K. P. Esler, P. R. C. Kent, and L. Shulenburger, "An efficient hybrid orbital representation for quantum monte carlo calculations," *The Journal of Chemical Physics*, vol. 149, p. 084107, aug 2018.

[112] K. Saritas, J. T. Krogel, P. R. C. Kent, and F. A. Reboredo, "Diffusion monte carlo: A pathway towards an accurate theoretical description of manganese oxides," *Physical Review Materials*, vol. 2, aug 2018.

[113] M. C. Bennett, G. Wang, A. Annaberdiyev, C. A. Melton, L. Shulenburger, and L. Mitas, "A new generation of effective core potentials from correlated calculations: 2nd row elements," *The Journal of Chemical Physics*, vol. 149, p. 104108, sep 2018.

[114] K. Alberi, M. B. Nardelli, A. Zakutayev, L. Mitas, S. Curtarolo, A. Jain, M. Fornari, N. Marzari, I. Takeuchi, M. L. Green, M. Kanatzidis, M. F. Toney, S. Butenko, B. Meredig, S. Lany, U. Kattner, A. Davydov, E. S. Toberer, V. Stevanovic, A. Walsh, N.-G. Park, A. Aspuru-Guzik, D. P. Tabor, J. Nelson, J. Murphy, A. Setlur, J. Gregoire, H. Li, R. Xiao, A. Ludwig, L. W. Martin, A. M. Rappe, S.-H. Wei, and J. Perkins, "The 2019 materials by design roadmap," *Journal of Physics D: Applied Physics*, vol. 52, p. 013001, oct 2018.

[115] A. Annaberdiyev, G. Wang, C. A. Melton, M. C. Bennett, L. Shulenburger, and L. Mitas, "A new generation of effective core potentials from correlated calculations: 3d transition metal series," *The Journal of Chemical Physics*, vol. 149, p. 134108, oct 2018.

[116] R. Archibald, J. T. Krogel, and P. R. C. Kent, "Gaussian process based optimization of molecular geometries using statistically sampled energy surfaces from quantum monte carlo," *The Journal of Chemical Physics*, vol. 149, p. 164116, oct 2018.

[117] K. Saritas, J. T. Krogel, and F. A. Reboredo, "Relative energies and electronic structures of CoO polymorphs through ab initio diffusion quantum monte carlo," *Physical Review B*, vol. 98, oct 2018.

[118] S. Zhang, F. D. Malone, and M. A. Morales, "Auxiliary-field quantum monte carlo calculations of the structural properties of nickel oxide," *The Journal of Chemical Physics*, vol. 149, p. 164102, oct 2018.

[119] K. Saritas, W. Ming, M.-H. Du, and F. A. Reboredo, "Excitation energies of localized correlated defects via quantum monte carlo: A case study of mn4+ doped phosphors," *The Journal of Physical Chemistry Letters*, vol. 10, pp. 67–74, nov 2018.

[120] M. E. Segovia and O. N. Ventura, "Diffusion and reptation quantum monte carlo study of the NaK molecule," *Molecular Physics*, pp. 1–10, nov 2018.

[121] N. S. Blunt and E. Neuscamman, "Excited-state diffusion monte carlo calculations: A simple and efficient two-determinant ansatz," *Journal of Chemical Theory and Computation*, vol. 15, pp. 178–189, dec 2018.

[122] F. D. Malone, S. Zhang, and M. A. Morales, "Overcoming the memory bottleneck in auxiliary field quantum monte carlo simulations with interpolative separable density fitting," *Journal of Chemical Theory and Computation*, vol. 15, pp. 256–264, dec 2018.

[123] Y. Luo and J. Kim, "An highly efficient delayed update algorithm for evaluating slater determinants in quantum monte carlo," *in preparation*, 2018.