QMCPACK

User's Guide and Developer's Manual Preview
May 30, 2016

Contents

Chapter 1

Introduction

QMCPACK is an open-source, high-performance electronic structure code that implements numerous Quantum Monte Carlo 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 designed with the modularity afforded by object-oriented programming. It makes extensive use of template metaprogramming to achieve high computational efficiency. Due to the modular architecture, the addition of new wavefunctions, algorithms, and observables is relatively straightforward. For parallelization QMCPACK utilizes 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 utilizes 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 a guide to installing and running it. Over time this manual will be expanded to including 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

If you are keen to get started this section describes how to quickly build and run QMCPACK on 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 may require hints on the locations of libraries and which versions to use, typical of any code, see Chapter 2. Section 2.6 includes complete examples 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
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 2 for instructions and specific build instructions for common systems.
6. Build QMCPACK: `make` or `make -j 16`, the latter for a faster parallel build on a system using, e.g., 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`
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 10 for information on analysing QMCPACK data.

```
export PATH=$PATH:location-of-qmcpack/nexus/executables
export PYTHONPATH=$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. 1.1. 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, timestep etc.

Congratulations, you have completed a DMC calculation with QMCPACK!

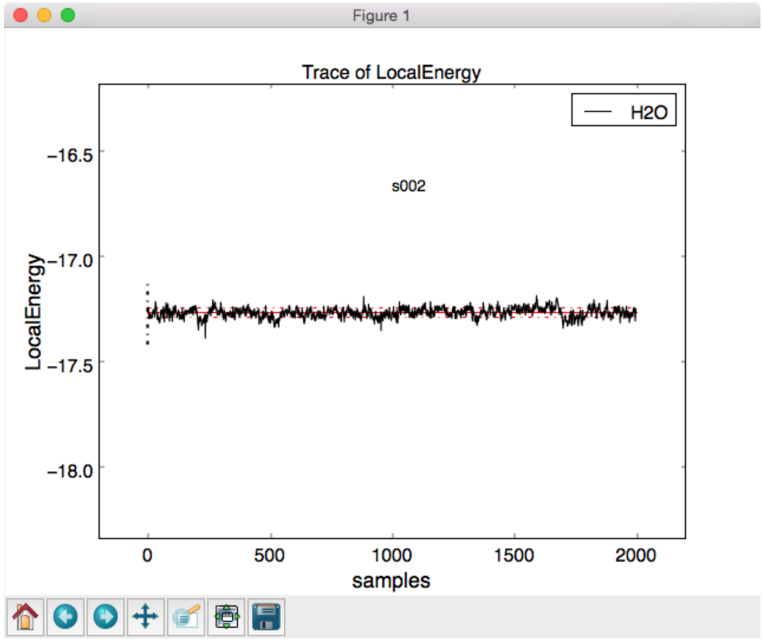


Figure 1.1: Trace of walker energies produced by qmca tool for simple water molecule example.

1.2 Authors and History

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

The primary and original author of the code is Jeongnim Kim. 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:

- "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 U. S. National Science Foundation.
- Materials Computational Center, supported by the U.S. 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 `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 algorithms, is highly parallel, and is also 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 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/NCSA Open Source License.

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. Please post on the QMCPACK Google group or contact the developers. If you are planning a development, early discussions are encouraged. We will be able to tell you if anyone else if working on a similar feature or if any related work has been done in the past. Credit for your contribution can be obtained, e.g., through citation of a paper, or becoming one of the authors on the next version of the standard QMCPACK reference citation.

Please note the following guidelines for a contributions:

- Additions should be fully synchronized with the latest release version and ideally the latest development SVN. Merging of code developed on older versions is error prone.
- Code should be cleanly formatted, commented, portable, and accessible to other programmers. i.e. 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 below. 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 to continue improving the accessibility and usability of QMCPACK, by 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 on these systems efficiently are expected to benefit *all* platforms due to improved vectorization, cache utilization and memory performance.

1.7.2 Documentation

This manual currently 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.

Due to its history as a research code, QMCPACK contains a variety of additional QMC methods, trial wavefunction forms, potentials (etc.) that, although not critical, may 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 an historical Jastrow form is needed for benchmarking.

Chapter 2

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.

2.1 Installation steps

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

1. Download the source code, Sections 2.2 or 2.3.
2. Verify that you have the required compilers, libraries and tools installed, Section 2.4.
3. Run the cmake configure step and build with make, Section 2.5 and 2.5.1. Some examples for common systems are given in Section 2.6.
4. Run the tests to verify QMCPACK, Section 2.7.
5. Build the ppconvert utility in QMCPACK, Section 2.8.
6. Download and patch Quantum Espresso. This patch adds the pw2qmcpack utility, Section 2.9.

Hints for high performance are in Section 2.10. Troubleshooting suggestions are in Section 2.11.

Note that there are two different QMCPACK executables that can be produced: the general one, which is the default, and the "complex" version which support periodic calculations at arbitrary twist angles and k-points. This second version is enabled via a cmake configuration parameter, see Section 2.5.3. The general version only supports wavefunctions that can be made real. If you run a calculation that needs the complex version, QMCPACK will stop and inform you.

2.2 Obtaining the latest release version

Major releases of QMCPACK are distributed from `http://www.qmcpack.org`. These releases undergo the most testing. Unless there are specific reasons we encourage all production calculations to use the latest release versions.

Releases are usually compressed tar files indicating the version number, date, and often the source code revision control number corresponding to the release.

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

2.3 Obtaining the latest development version

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

```
svn checkout https://svn.qmcpack.org/svn/trunk
```

Once checked-out, updates can be made via the standard `svn update`.

The subversion repository contains the day-to-day development source with the latest updates, bugfixes etc. This may 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 may not be fully consistent with the online documentation. We attempt to keep the development version fully working. However, please be sure to run the 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.

2.4 Prerequisites

The following 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 access 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 2.10 for performance suggestions.

- C/C++ compilers such as GCC, Intel, IBM XLC. CLANG-based compilers are not yet supported by the build system, but the source code is ready.
- MPI library such at OpenMPI `http://open-mpi.org`
- BLAS/LAPACK, numerical and linear algebra libraries. Use platform-optimized libraries where available, such as Intel MKL. ATLAS or other optimized open-source libraries may 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/`

- BOOST, peer-reviewed portable C++ source libraries, `http://www.boost.org`
- 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.

2.5 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 - at the time of writing this means CMake 3.4.3. The much older CMake 2.8 is known to work, but might not work optimally on your system.

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 environmental or CMake variables. When the libraries are installed in standard locations, e.g., /usr, /usr/local, there is no need to set environmental or cmake variables for the packages.

2.5.1 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. e.g. 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. i.e. Check the details below. We give examples for a number of common systems in Section 2.6.

2.5.2 Environment variables

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

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

2.5.3 Configuration options

In addition to reading the environmental 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 environmental 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)
```

- 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
```

- 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 (currently
                    unused, but likely to be used for future performance tests)
```

- libxml related

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

- FFTW related

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

2.5.4 Configure and build using cmake and make

To configure and build QMPACK, 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 where the filesystem 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.

2.5.5 Example configure and build

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

```
export CXX=icpc
export CC=icc
export MKL_HOME=/usr/local/intel/mkl/10.0.3.020
export LIBXML2_HOME=/usr/local
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
```

2.5.6 Build scripts

It is recommended to create a helper script that contains the configure line for CMake. This is particularly useful when avoiding environmental variables, packages are installed in custom locations, or if 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 that the script is called. Deleting all the files in the build directory is also acceptable. If you do so we recommend to add 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              \
  ..
```

2.6 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 may require small modifications to these recipes. See Section 2.10 for key points to check to obtain highest performance and Section 2.11 for troubleshooting hints.

2.6.1 Installing on Ubuntu Linux or other apt-get based distributions

The following is designed to obtain a working QMCPACK build on e.g. 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 subversion 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
```

2.6.2 Installing on CentOS Linux or other yum based distributions

The following is designed to obtain a working QMCPACK build on e.g. 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 only complicated by the need to install another repository to obtain HDF5 packages which 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++ subversion 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 OS. By default the 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
```

2.6.3 Installing on Mac OS X using Macports

These instructions assume a fresh installation of macports and for consistency with current Linux distributions, use the gcc 4.8.2 compiler. More recent versions are fine, but it is vital to ensure matching compilers and libraries are used for all packages and to force use of what is installed in /opt/local. As with the Linux examples above, this build is very good if not optimal, and is easily good enough to learn QMCPACK or experiment on a travel laptop.

Note that we utilize the Apple provided Accelerate framework for optimized BLAS.

Follow the Macports install instructions `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 gcc48
sudo port select gcc mp-gcc48  # Set default

sudo port install openmpi-devel-gcc48
sudo port select set mpi openmpi-devel-gcc48-fortran  # Set default

# Sanity check
mpiCXX -v
#should return  gcc version 4.8.2 (MacPorts gcc48 4.8.2_2) or similar.

sudo port install fftw-3 +gcc48
sudo port install cmake

sudo port install boost +gcc48
sudo port install libxml2
sudo port install hdf5-18 +gcc48

sudo port select set python python27
sudo port install py27-matplotlib  # For qmca
```

QMCPACK build:

```
export CXX=mpiCXX
export CC=/opt/local/bin/gcc
export LIBXML2_HOME=/opt/local/
export HDF5_HOME=/opt/local
export BOOST_HOME=/opt/local
export FFTW_HOME=/opt/local
cd build
cmake ..
make -j 6 # Adjust for available core count
ls -l bin/qmcpack
```

2.6.4 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 10.11 “El Capitain” in February 2016. 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 # Builds full gcc 5 from scratch, will take 30 minutes
   export HOMEBREW_CXX=g++-5
   export HOMEBREW_CC=gcc-5
   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 5.x.x
   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.

   ```
   ctest -R short
   ```

2.6.5 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 49152 compute nodes and each node has a 16-core PowerPC A2 processor with 16 GB DDR3 memory. Due to the fact that 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-xl and +cmake in $HOME/.soft. In order to build QMCPACK, a toolchain file is provided for setting up CMake and the cmake command should be executed twice.

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

In addition, adding a very useful cmake option -DCMAKE_VERBOSE_MAKEFILE=TRUE allows printing all the build commands during the make step. Alternatively you can use make VERBOSE=1.

2.6.6 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.

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

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

As noted in Section2.6.6 for the GPU, building on Crays requires only loading the appropriate library modules.

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

2.6.8 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.

```
module load cray-hdf5
module load cmake
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
mkdir build_eos
cd build_eos
cmake ..
make -j 8
ls -l bin/qmcpack
```

2.6.9 Installing on NERSC Edison Cray XC30

Edison is a Cray XC30 with dual 12-core Intel "Ivy Bridge" nodes installed at NERSC. The build settings are identical to eos.

```
module load cray-hdf5
module load cmake
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
mkdir build_edison
cd build_edison
cmake ..
make -j 8
ls -l bin/qmcpack
```

When the above was tested on 1 February 2016, the following module and software versions were present:

```
qmcpack@edison04:trunk> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.3                        16) alps/5.2.3-2.0502.9295.14.14.ari
  2) nsg/1.2.0                               17) rca/1.0.0-2.0502.57212.2.56.ari
  3) eswrap/1.1.0-1.020200.1130.0            18) atp/1.8.3
  4) switch/1.0-1.0502.57058.1.58.ari        19) PrgEnv-intel/5.2.56
  5) craype-network-aries                    20) craype-ivybridge
  6) craype/2.5.0                            21) cray-shmem/7.3.0
  7) intel/15.0.1.133                        22) cray-mpich/7.3.0
  8) cray-libsci/13.3.0                      23) slurm/edison
  9) udreg/2.3.2-1.0502.9889.2.20.ari        24) altd/2.0
 10) ugni/6.0-1.0502.10245.9.9.ari           25) darshan/2.3.0
 11) pmi/5.0.10-1.0000.11050.0.0.ari         26) subversion/1.7.9
 12) dmapp/7.0.1-1.0502.10246.8.47.ari       27) cray-hdf5/1.8.14
```

```
 13) gni-headers/4.0-1.0502.10317.9.2.ari    28) cmake/2.8.11.2
 14) xpmem/0.1-2.0502.57015.1.15.ari         29) fftw/3.3.4.6
 15) dvs/2.5_0.9.0-1.0502.1958.2.55.ari      30) boost/1.54
```

2.6.10 Installing on NERSC Cori (Phase 1) Cray XC40

Cori is a Cray XC40 with 16-core Intel "Haswell" nodes installed at NERSC. The build settings are identical to eos.

```
module load cray-hdf5
module load cmake
module load fftw
export FFTW_HOME=$FFTW_DIR/..
module load boost
mkdir build_cori
cd build_cori
cmake ..
make -j 8
ls -l bin/qmcpack
```

When the above was tested on 1 February 2016, the following module and software versions were present:

```
qmcpack@cori05:trunk> module list
Currently Loaded Modulefiles:
  1) nsg/1.2.0                              15) dvs/2.5_0.9.0-1.0502.2188.1.116.ari
  2) modules/3.2.10.3                       16) alps/5.2.4-2.0502.9774.31.11.ari
  3) eswrap/1.1.0-1.020200.1231.0           17) rca/1.0.0-2.0502.60530.1.62.ari
  4) switch/1.0-1.0502.60522.1.61.ari       18) atp/1.8.3
  5) intel/16.0.0.109                       19) PrgEnv-intel/5.2.82
  6) craype-network-aries                   20) craype-haswell
  7) craype/2.4.2                           21) cray-shmem/7.2.5
  8) cray-libsci/13.2.0                     22) cray-mpich/7.2.5
  9) udreg/2.3.2-1.0502.10518.2.17.ari      23) slurm/cori
 10) ugni/6.0-1.0502.10863.8.29.ari         24) cray-hdf5/1.8.14
 11) pmi/5.0.9-1.0000.10911.0.0.ari         25) gcc/5.1.0
 12) dmapp/7.0.1-1.0502.11080.8.76.ari      26) cmake/3.3.2
 13) gni-headers/4.0-1.0502.10859.7.8.ari   27) fftw/3.3.4.5
 14) xpmem/0.1-2.0502.64982.5.3.ari         28) boost/1.59
```

2.7 Testing and validation of QMCPACK

We **strongly encourage** running the included tests each time QMCPACK is built. These compare the results from the executable with known-good mean-field, quantum chemical, and other QMC results.

The tests included with QMCPACK currently test only the VMC code with single determinant wavefunction and simple spline Jastrow wavefunctions, and for gaussian and periodic spline basis sets. Although not yet comprehensive, it is extremely unlikely that, e.g., DMC will be correct if the

VMC tests do not pass. We check that the known mean field results are obtained with no Jastrow. When Jastrow functions are included we test against previous QMC data. The tests are statistical with a generous 3 σ tolerance, however the system sizes are small, typically < 10 electrons, so the error bars are typically small.

The "short" tests only take a few minutes on a 16 core machine. You can run these tests using the command below in the build directory:

```
ctest -R short   # Run the tests with "short" in their name
```

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 that are run varies between the standard, complex, and GPU compilations.

The full set of tests consist of significantly longer versions of the short tests. They require several hours each to run yielding a much more stringent test of the code. To run all the tests simply run ctest in the build directory:

```
ctest            # Run all the tests. This will take several hours.
```

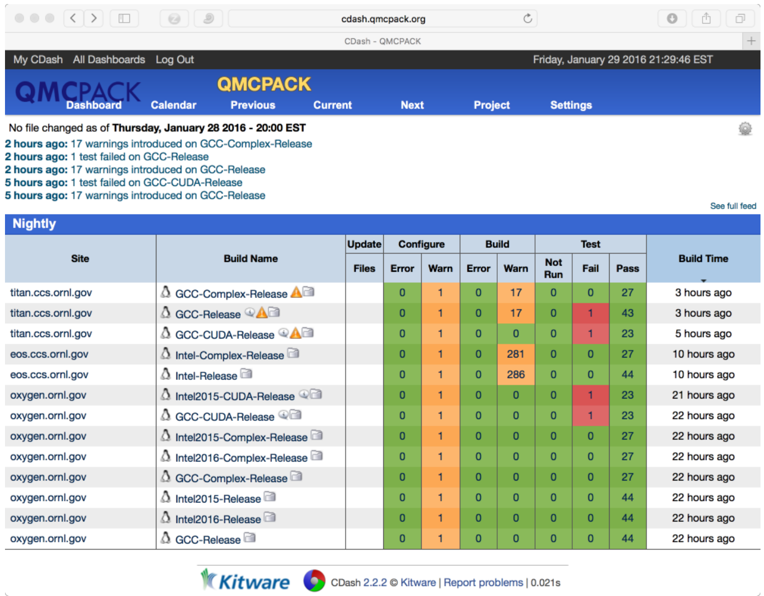
You can also run verbose tests which direct the QMCPACK output to the standard output:

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

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

The data files for the tests are located in the tests directory. The runs occur in build/src/QMCApp/test/test_name. The numerical comparisons and test definitions are in src/QMCApp/test/CMakeLists.txt. If *all* the QMC tests fail it is likely that the appropriate mpiexec (or aprun, srun) 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 outputs in build/src/QMCApp/test/test_name.

Note that because the tests are very small, consisting of only a few electrons, the performance is not representative of larger calculations. For example, while the calculations might fit in cache, there will be essentially no vectorization due to the small electron counts. **The tests should not be used for any benchmarking or performance analysis**. Dedicated larger runs are required.



cdash.qmcpack.org

CDash - QMCPACK

My CDash All Dashboards Log Out Friday, January 29 2016 21:29:46 EST

QMCPACK

Dashboard Calendar Previous Current Next Project Settings

No file changed as of **Thursday, January 28 2016 - 20:00 EST**

2 hours ago: 17 warnings introduced on GCC-Complex-Release
2 hours ago: 1 test failed on GCC-Release
2 hours ago: 17 warnings introduced on GCC-Release
5 hours ago: 1 test failed on GCC-CUDA-Release
5 hours ago: 17 warnings introduced on GCC-Release

See full feed

Nightly

Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
titan.ccs.ornl.gov	GCC-Complex-Release		0	1	0	17	0	0	27	3 hours ago
titan.ccs.ornl.gov	GCC-Release		0	1	0	17	0	1	43	3 hours ago
titan.ccs.ornl.gov	GCC-CUDA-Release		0	1	0	0	0	1	23	5 hours ago
eos.ccs.ornl.gov	Intel-Complex-Release		0	1	0	281	0	0	27	10 hours ago
eos.ccs.ornl.gov	Intel-Release		0	1	0	286	0	0	44	10 hours ago
oxygen.ornl.gov	Intel2015-CUDA-Release		0	1	0	0	0	1	23	21 hours ago
oxygen.ornl.gov	GCC-CUDA-Release		0	1	0	0	0	1	23	22 hours ago
oxygen.ornl.gov	Intel2015-Complex-Release		0	1	0	0	0	0	27	22 hours ago
oxygen.ornl.gov	Intel2016-Complex-Release		0	1	0	0	0	0	27	22 hours ago
oxygen.ornl.gov	GCC-Complex-Release		0	1	0	0	0	0	27	22 hours ago
oxygen.ornl.gov	Intel2015-Release		0	1	0	0	0	0	44	22 hours ago
oxygen.ornl.gov	Intel2016-Release		0	1	0	0	0	0	44	22 hours ago
oxygen.ornl.gov	GCC-Release		0	1	0	0	0	0	44	22 hours ago

Kitware CDash 2.2.2 © Kitware | Report problems | 0.021s

Figure 2.1: 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, 4 errors were found.

2.7.1 Automatic tests of QMCPACK

The QMCPACK developers run automatic tests of QMCPACK on several different computer systems, many on a continuous basis. We currently test the following combinations nights (workstation) or weekly (supercomputers):

- On a Red Hat Linux workstation:
  - GCC 4.8.2 with OpenMPI and CUDA 7.0 (GPU build, run on NVIDIA K40s)
  - GCC 4.8.2 with OpenMPI
  - Intel 2016 with Intel MPI and MKL
  - Intel 2015 with Intel MPI and MKL and CUDA 7.0 (GPU build, run on NVIDIA K40s)
  - Intel 2015 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

2.8 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 Quantum

Espresso, and the XML format used by QMCPACK itself. The utility also enables the atomic orbitals to recomputed via a numerical density functional calculation if they need to be reconstructed for use in an electronic structure calculation.

To build ppconvert follow the instructions in src/QMCTools/ppconvert/README. Currently ppconvert is not built automatically although we expect to automate it soon. The makefile must be updated to refer to suitable C++ compiler and link in BLAS. Due to the small size of the calculations, optimal settings are not essential.

2.9 Installing and patching Quantum Espresso

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

Quantum Espresso currently stores wavefunctions in a non-standard internal "save" format. To convert these to a conventional HDF5 format file we have developed a converter, pw2qmcpack. This is an add on to the Quantum Espresso distribution.

To simplify the process of patching Quantum Espresso we have developed a script that will automatically download and patch the source code. The patches are specific to each version. e.g. To download and patch QE v5.3.0:

```
cd external_codes/quantum_espresso
./download_and_patch_qe5.3.0.sh
```

After running the patch, you must configure Quantum Espresso with the HDF5 capability enabled, i.e.

```
cd espresso-5.3.0
./configure --with-hdf5 HDF5_DIR=/opt/local   # Specify HDF5 base directory
```

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

2.10 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 and also to apply the orbital coefficients in local basis calculations.
- 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, e.g. 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. i.e. Do not compute smaller error bars than necessary.

2.11 Troubleshooting the installation

Some tips to help troubleshoot installations of QMCPACK:

- First, build QMCPACK on a workstation that you control, or on any system that has 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 that 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, specify the appropriate directories and files directly (Section 2.5.3). e.g. 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 3

Running QMCPACK

To run QMCPACK, put one or more input file names on the command line:

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

3.1 Command line options

Command line options include

-debug Print more information about code internals. Helpful for understanding what functions are being called for a particular input file.

3.2 Input files

The input is an XML file, documented in section 5.

3.3 Output files

scalar.dat
dmc.dat
stat.h5
config.h5

3.4 Running in parallel

3.4.1 MPI

QMCPACK is fully parallelized with MPI. When performing an ensemble job, all the MPI ranks are first equally divided into groups which 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. On clusters with more than just one GPU per node (NVIDIA Tesla

K80), it requires to use the same number of MPI ranks as the number of GPUs per node in order to let each MPI rank take one GPU.

3.4.2 Use of 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 all that compute capability on a processor by using threads via 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 environmental variable OMP_NUM_THREADS. Even in the GPU accelerated version, using threads significantly reduces the time spent on the calculations performed by the CPU.

Performance consideration

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.

Memory consideration

When using threads, some memory objects shared by all the threads. Usually these memory 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 7.2.1.

The other memory objects which are distinct for each walker during random walk need to be associated with individual walkers and can not 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.

3.4.3 Running on GPU machines

The GPU version on the NVIDIA CUDA platform is fully incorporated into the main trunk. Currently some commonly used functionalities for solid-state and molecular systems using B-spline single-particle orbitals is supported. A detailed description of the GPU implementation can be found in Ref. [1].

Current GPU implementation assumes one MPI process per GPU. 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.

Supported GPU features

1. Quantum Monte Carlo methods:
   (a) Variational Monte Carlo (VMC).
   (b) Diffusion Monte Carlo (DMC).
   (c) Wavefunction optimization.

2. Boundary conditions:
   (a) Periodic and open boundary conditions are fully supported.
   (b) Twist-averaged boundary condition is supported for only real-valued wavefunctions.
   (c) Mixed boundary conditions and complex wavefunctions (e.g. fixed phase) are not yet supported.

3. Wavefunctions:
   (a) Single Slater determinants with 3D B-spline orbitals. Only real-valued wavefunctions is supported, but tiling complex orbitals to supercells is supported as long as each k-point is a multiple of half a G-vector of the supercell.
   (b) 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.
   (c) One-body and two-body Jastrows represented as 1D B-splines are supported.

4. Interaction types:
   (a) Semilocal (nonlocal and local) pseudopotentials.
   (b) Coulomb interaction (electron-electron, electron-ion).
   (c) Model periodic Coulomb (MPC) interaction.

Compiling the GPU code

To build the executable `qmcpack` with GPU support, follow these steps:

1. Make sure NVIDIA's CUDA compiler, nvcc, is in the search path. In most cases, CMake should be able to locate the nvcc compiler on the system automatically.

2. (a) Run CMake with the argument `QMC_CUDA` switched on:
   ```
   cd build
   cmake -D QMC_CUDA=1 ..
   make
   ```

   or

(b) If a CMake toolchain file is used, switch on `QMC_CUDA` by including this line in the toolchain file:

```
SET (QMC_CUDA 1)
```

Then compile the code as before:

```
cd build
cmake -D CMAKE_TOOLCHAIN_FILE=[toolchain name] ..
make
```

CMake variables for adjusting CUDA code build features

These values can be changed by passing them as CMake's command line options with the `-D` flag, or using a toolchain file to overwrite the default values.

1. `QMC_CUDA`

   `=0` (default): no GPU support, build QMCPACK as a CPU code

   `=1` : build QMCPACK with GPU support

2. `CUDA_PRECISION`

   `=float` (default): single precision arithmetics and data types will be used for most GPU kernels. Several precision critical kernels are always in double precision.

   `=double` : double precision arithmetics and data types will be used for all the GPU kernels.

Performance consideration

The relative speedup of the GPU implementation increases with both the number of electrons and the number of walkers running on a GPU. Typically, 128-256 walkers per GPU utilize sufficient number of threads to operate the GPU efficiently and to hide memory-access latency.

To achieve better performance, current implementation utilizes single precision operations on most GPU calculations, except for matrix inversions and Coulomb interaction where double precision is required to retain high accuracy. The single 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 1500 electrons.

Memory consideration

In the GPU implementation, each walker has an anonymous buffer on 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.

If the GPU memory is exhausted, reduce the number of walkers per GPU. Coarsening the grids of the B-splines representation (by decreasing the value of meshfactor 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 distrubte the B-spline coefficients table between the host and GPU memory, see option Spline_Size_Limit_MB in Sec. 7.2.1.

Chapter 4

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 5

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.

Chapter 6

Specifying the system to be simulated

6.1 Specifying the simulation cell

6.2 Specifying the particle set

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

6.2.1 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

6.2.2 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

`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

- `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[$\frac{1}{12}m_{^{12}\mathrm{C}} = 1$]).

6.2.3 Example use cases

Listing 6.1: 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 6.2: 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 6.3: 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 7

Trial wavefunction specification

7.1 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:

$$\Psi_T(\vec{r}) = \prod_k \Theta_k(\vec{r}), \tag{7.1}$$

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,

$$\Psi_T(\vec{r}) = A(\vec{r}) \prod_k J_k(\vec{r}), \tag{7.2}$$

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 7.3). 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.

7.2 Single-particle orbitals

7.2.1 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 seamless 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 [2]. The one-dimensional cubic B-spline is given by,

$$f(x) = \sum_{i'=i-1}^{i+2} b^{i',3}(x)\; p_{i'}, \tag{7.3}$$

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

$$\phi_n(x,y,z) = \sum_{i'=i-1}^{i+2} b_x^{i',3}(x) \sum_{j'=j-1}^{j+2} b_y^{j',3}(y) \sum_{k'=k-1}^{k+2} b_z^{k',3}(z)\; p_{i',j',k',n}. \tag{7.4}$$

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 7.1. A list of options is given in Table 7.1. `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 7.1: All electron Hamiltonian 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">
      </occupation>
    </determinant>
  </slaterdeterminant>
</determinantset>
```

`determinantset` element				
parent elements:	`wavefunction`			
child elements:	`slaterdeterminant`			
attribute :				
name	**datatype**	**values**	**default**	**description**
`type`	text	bspline		Type of `sposet`.
`href`	text			Path to the h5 file generated by pw2qmcpack
`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.
`Spline_Size_Limit_MB`	integer			Limit the size of B-spline coefficient table on
`source`	text	*any*	ion0	Particle set with the position of atom centers

Table 7.1: Options for the `determinantset` xml-block associated with B-spline single particle orbital sets.

Additional information:

- `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`.
- `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 penaty introduced by it is significant but allows large calculations to go.

7.2.2 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:

$$\phi_i(\vec{r}) = \sum_k C_{i,k}\ \eta_k(\vec{r}), \tag{7.5}$$

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. (While it is possible to use this `sposet` on calculations of periodic systems, this feature is not currently implemented in QMCPACK.) Examples include calculations of molecules using gaussian basis sets or slater-type basis functions. Even though this section is called "Gaussian basis set" (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 7.4. The basic structure for the input block of a single slater determinant is given in Listing 7.2. A list of options for `determinantset` associated with this `sposet` is given in Table 7.2.

Listing 7.2: 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>
```

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

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 7.3, while the corresponding lists of options are given in Tables 7.3 and 7.4.

Listing 7.3: 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 7.3: 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 7.4: 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 7.5: Options for the basisGroup xml-block.

Listing 7.4: 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"*:

-

`slaterdeterminant` attributes:

-

7.2.3 Plane-wave basis sets

7.2.4 Homogeneous electron gas

The interacting Fermi Liquid has its own special determinantset for filling up a Fermi surface. The shell number can be specified seperately 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 7.5: 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 7.6: 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>
```

7.3 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:

$$\Psi(\vec{R}) = \mathcal{A}(\vec{R}) \exp\left[J(\vec{R})\right] \tag{7.6}$$

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.

7.3.1 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 7.3.1 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

$$J1 = \sum_{I}^{ion0} \sum_{i}^{e} u_{ab}(|r_i - R_I|) \tag{7.7}$$

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 [1]. 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[3]. In this case, the derivative of the jastrow factor as the electron approaches the nucleus will be given by:

$$\left(\frac{\partial J}{\partial r_{iI}}\right)_{r_{iI}=0} = -Z \tag{7.8}$$

  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.7.9 is a good choice in such cases.

$$u_{ab}(r) = \frac{a * r + c * r^2}{1 + b * r} \tag{7.9}$$

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.7.9
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">
    <parameter id="LiA" name="A">  0.34 </parameter>
    <parameter id="LiB" name="B"> 12.78 </parameter>
    <parameter id="LiC" name="C">  1.62 </parameter>
  </correlation>
  <correlation elementType="H"">
```

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

7.3.2 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[3].

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 7.3.2 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

$$J2 = \sum_{i}^{e} \sum_{j>i}^{e} u_{ab}(|r_i - r_j|) \tag{7.10}$$

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 [1]. 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>
```

7.3.3 Three-body Jastrow functions

Explicit three body correlations can be included in the wavefunction via the three-body jastrow factor.

7.4 Multideterminant wavefunctions

7.5 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:

$$\mathbf{x}'_{i_\alpha} = \mathbf{x}_{i_\alpha} + \sum_{\alpha \leq \beta} \sum_{i_\alpha \neq j_\beta} \eta^{\alpha\beta}(|\mathbf{x}_{i_\alpha} - \mathbf{x}_{j_\beta}|)(\mathbf{x}_{i_\alpha} - \mathbf{x}_{j_\beta}) \tag{7.11}$$

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 we'll talk about 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, I will explain how to describe general terms like Eq. 7.11 in a QMCPACK XML file. For specificity, I 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".

7.5.1 Input Specifications

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

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.

Just like one and two-body jastrows, parameterization of the backflow transformations are specified within the `<transformation>` blocks by `<correlation>` blocks. Please refer to 7.3.1 for more information.

7.5.2 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's where discussed one and two-body terms are defined -->
         </transformation>
     </backflow>

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

```
</determinantset>
```

7.5.3 Additional Information

- **Optimization**: Optimizable backflow transformation parameters are notoriously nonlinear, and so optimizing backflow wavefunctions can sometimes be difficult. We direct the reader to our provided backflow tutorials for more information.

Chapter 8

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.

8.1 The Hamiltonian

The many-body Hamiltonian in Hartree units is given by

$$\hat{H} = -\sum_i \frac{1}{2m_i}\nabla_i^2 + \sum_i v^{ext}(r_i) + \sum_{i<j} v^{qq}(r_i,r_j) + \sum_{i\ell} v^{qc}(r_i,r_\ell) + \sum_{\ell<m} v^{cc}(r_\ell,r_m). \tag{8.1}$$

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 8.1: 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 8.2: 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>
```

8.2 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	
	numerical/*num*		Numerical radial potential	
	skpot		*Unknown*	
	vhxc		Exchange correlation potential (external)	
	jellium		Atom-centered spherical jellium potential	
	hardsphere		Hard sphere potential	
	gaussian		Gaussian potential	
	modpostel		Modified Poschl-Teller potential	
	huse		Huse quintic potential	
	modInsKE		Model insulator kinetic energy	
	oscillatory		*Unknown*	
	LJP_smoothed		Helium pair potential	
	HeSAPT_smoothed		Helium pair potential	
	HFDHE2_Moroni1995		Helium pair potential	
	HFDHE2		Helium pair potential	
	eHe		Helium-electron pair potential	
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, numerical, jellium, hardsphere, gaussian, huse, modpostel, oscillatory, skpot, vhxc, modInsKE, LJP_smoothed, HeSAPT_smoothed, HFDHE2_Moroni1995, eHe, HFDHE2`) are not covered.
- Available only if `QMC_BUILD_LEVEL>2` and `QMC_CUDA` is not defined: `hardsphere, gaussian, huse, modpostel, oscillatory, skpot`.

- Available only if `OHMMS_DIM==3`: `mpc, vhxc, pseudo`.
- Available only if `OHMMS_DIM==3` and `QMC_BUILD_LEVEL>2` and `QMC_CUDA` is not defined: `cpp, LJP_smoothed, HeSAPT_smoothed, HFDHE2_Moroni1995, eHe, jellium, HFDHE2, modInsKE`.

8.2.1 Coulomb potentials

The bare Coulomb potential is used in open boundary conditions:

$$V_c^{open} = \sum_{i<j} \frac{q_i q_j}{|r_i - r_j|} \tag{8.2}$$

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

$$V_c^{pbc} = \sum_{i<j} \frac{q_i q_j}{|r_i - r_j|} + \frac{1}{2} \sum_{L \neq 0} \sum_{i,j} \frac{q_i q_j}{|r_i - r_j + L|} \tag{8.3}$$

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. [4]) for details.

For information on how to set the boundary conditions, consult Sec. 6.1.

`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 8.3: XML element for Coulomb interaction between electrons.

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

Listing 8.4: XML element for Coulomb interaction between electrons and ions (all-electron only).

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

Listing 8.5: XML element for Coulomb interaction between ions.

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

8.2.2 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:

$$V^{PP} = \sum_{ij} \Big(V_{\bar{\ell}}(|r_i - \tilde{r}_j|) + \sum_{\ell \neq \bar{\ell}}^{\ell_{max}} \sum_{m=-\ell}^{\ell} |Y_{\ell m}\rangle \big[V_\ell(|r_i - \tilde{r}_j|) - V_{\bar{\ell}}(|r_i - \tilde{r}_j|) \big] \langle Y_{\ell m}| \Big) \tag{8.4}$$

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. [5] 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

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).

Listing 8.6: XML element for pseudopotential electron-ion interaction (psf files).

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

Listing 8.7: XML 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 8.8: XML element for pseudopotential of single ionic species.

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

8.2.3 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 8.9: Modified periodic coulomb for finite size post-correction.

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

8.3 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	
	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 8.5)	
	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	
	numberfluctuations		Spatial number fluctuations	
	HFDHE2		Helium pressure	
	NearestNeighbors		Trace nearest neighbor indices	
	Kinetic		*No current function*	
	Pressure		*No current function*	
	ZeroVarObs		*No current function*	
	DMCCorrection		*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

8.3.1 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. [6]. The total energy can be corrected for finite size effects by using this estimator in conjuction 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 8.10: "Chiesa" kinetic energy finite size post-correction.

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

8.3.2 Density estimator

The particle number density operator is given by

$$\hat{n}_r = \sum_i \delta(r - r_i) \tag{8.5}$$

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:

$$n_c = \int dR |\Psi|^2 \int_{\Omega_c} dr \sum_i \delta(r - r_i) \tag{8.6}$$

`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 $10\times10\times10$ 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 8.11: Density estimator (uniform grid).

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

8.3.3 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 stdout
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 8.12: Spin density estimator (uniform grid).

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

Listing 8.13: 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>
```

8.3.4 Pair correlation function, $g(r)$

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

$$g_{ss'}(r) = \frac{V}{4\pi r^2 N_s N_{s'}} \sum_{i_s=1}^{N_s} \sum_{j_{s'}=1}^{N_{s'}} \delta(r - |r_{i_s} - r_{j_{s'}}|). \tag{8.7}$$

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

$$\tilde{g}_{ss'}(r) = \frac{V}{4\pi r^2 N_s N_{s'}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_{s'}} \frac{1}{\delta r} \int_{r-\delta r/2}^{r+\delta r/2} dr' \delta(r' - |r_{si} - r_{s'j}|), \tag{8.8}$$

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 8.14: Pair correlation function estimator element.

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

Listing 8.15: Pair correlation function estimator element with additional electron-ion correlations.

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

8.3.5 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:

$$S(\mathbf{k}) = \frac{1}{N^e}\langle \rho^e_{-\mathbf{k}} \rho^e_{\mathbf{k}} \rangle \tag{8.9}$$

`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 8.16: Static structure factor estimator element.

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

8.3.6 Energy density estimator

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

$$\int dr \hat{\mathcal{E}}_r = \hat{H}, \tag{8.10}$$

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

$$\hat{\mathcal{E}}_r = \hat{\mathcal{T}}_r + \hat{\mathcal{V}}_r \tag{8.11}$$

with each component given by

$$\begin{aligned}
\hat{\mathcal{T}}_r &= \frac{1}{2}\sum_i \delta(r-r_i)\hat{p}_i^2 \\
\hat{\mathcal{V}}_r &= \sum_{i<j}\frac{\delta(r-r_i)+\delta(r-r_j)}{2}\hat{v}^{ee}(r_i,r_j) + \sum_{i\ell}\frac{\delta(r-r_i)+\delta(r-\tilde{r}_\ell)}{2}\hat{v}^{eI}(r_i,\tilde{r}_\ell) \\
&\quad + \sum_{\ell<m}\frac{\delta(r-\tilde{r}_\ell)+\delta(r-\tilde{r}_m)}{2}\hat{v}^{II}(\tilde{r}_\ell,\tilde{r}_m).
\end{aligned} \tag{8.12}$$

Here r_i and $\tilde{r}_\ell$ represent electon 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. [7].

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 8.17: 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 8.18: 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 8.19: 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 8.1) 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 8.1). 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 8.1: 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.

`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

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`.

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.

8.3.7 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:

$$\hat{n}_1 = \sum_n Tr_{R_n} |\Psi_T\rangle\langle\Psi_{FN}| \tag{8.13}$$

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

$$n_1(r, r') = \langle r|\hat{n}_1|r'\rangle = \sum_n \int dR_n \Psi_T(r, R_n)\Psi^*_{FN}(r', R_n) \tag{8.14}$$

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, $\{\phi_i\}$:

$$\begin{aligned} n_1(i, j) &= \langle\phi_i|\hat{n}_1|\phi_j\rangle \\ &= \int dR\Psi^*_{FN}(R)\Psi_T(R) \sum_n \int dr'_n \frac{\Psi_T(r'_n, R_n)}{\Psi_T(r_n, R_n)} \phi_i(r'_n)^*\phi_j(r_n) \end{aligned} \tag{8.15}$$

The integration over r' in Eq. 8.15 is inefficient when one is also interested in obtaining matrices involving energetic quantities, such as the energy density matrix of Ref. [8] or the related (and more well known) Generalized Fock matrix. For this reason, an approximation is introduced as follows:

$$n_1(i, j) \approx \int dR\Psi_{FN}(R)^*\Psi_T(R) \sum_n \int dr'_n \frac{\Psi_T(r'_n, R_n)^*}{\Psi_T(r_n, R_n)^*} \phi_i(r_n)^*\phi_j(r'_n) \tag{8.16}$$

For VMC, FN-DMC, FP-DMC, and RN-DMC the formula above represents an exact sampling of the 1RDM corresponding to $\hat{\rho}_N^\dagger$ (see appendix A of Ref. [8] 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	no	`basis` comes norm'ed
`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_g`
  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/dens`
- `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.
- `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. [8].

Listing 8.20: 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 8.21: 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 8.22: 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 8.23: 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 8.24: 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>
```

8.4 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

Additional information:

`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
`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

- **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 8.25: Forward walking estimator element.

```
<estimator name="fw" type="ForwardWalking">
    <Observable name="LocalPotential" max="300" frequency="1"/>
     <!--- Additional Observable blocks go here -->
 </estimator>
```

8.5 "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:

$$F_z = -Z \sum_{i=1}^{N_e} \frac{z_i}{r_i^3} [\theta(r_i - \mathcal{R}) + \theta(\mathcal{R} - r_i) \sum_{\ell=1}^{M} c_\ell r_i^\ell] \tag{8.17}$$

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,

$$\chi^2 = \int_0^{\mathcal{R}} dr\, r^m \, [f_z(r) - \tilde{f}_z(r)]^2 \tag{8.18}$$

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 estim
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 9

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	???
`checkpoints`	integer	-1, 0, n	-1	checkpoint frequency
`dumpconfig`	integer	n	-1	dump configuration every n step
`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 acception or rejection. The other method is the all-electron move, namely all the electrons are moved once for testing acception 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.

- `checkpoints`. If Checkpoint="-1" no checkpoint will be done (default setting). If Checkpoint="0" dump after the completion of a qmc section. When dumconfig="n" is present with Checkpoint="0", the configurations will be dumped at the end of the run (due to Checkpoint="0"), but also at every n block. If Checkpoint="n", configurations will be dump every n block.

All the dumped data will be written in a *.config.h5. The config.h5 file will contain the state of a population to continue a run including the random number sequences; the list of what is included in the .congig.h5 is: number of walkers, status of the run, branch mode, energy dataset, ratio to accepted moves, ratio to proposed moves, variance dataset, vParam{tau, taueff. E_trial, E_ref, Branch_Max, BranchCutOff, BranchFilter, Sigma, Accepted_Energy, Accepted_Samples}, IParamwarmumSteps, Energy_Update_Interval, Counter, targetwalkers, Maxwalkers, MinWalkers, Branching Interval, Walker coordinates, Random number size, Random number sequence, version of the code.

Listing 9.1: The following is an example of running a simulation that can be restarted .

```
<qmc method="dmc" move="pbyp"  checkpoint="0" dumpconfig="5">
  <parameter name="timestep">         0.004  </parameter>
  <parameter name="blocks">           100   </parameter>
  <parameter name="steps">            400    </parameter>
</qmc>
```

In this case, the there will be The flags checkpoint and dumpconfig 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 following before your VM/DMC block:

Listing 9.2: Restart (read wakers from previous run)

```
<mcwalkerset fileroot="BH.s002" version="0 6" collected="yes"/>
 <qmc method="dmc" move="pbyp"  checkpoint="0" dumpconfig="5">
   <parameter name="timestep">         0.004  </parameter>
   <parameter name="blocks">           100   </parameter>
   <parameter name="steps">            400    </parameter>
 </qmc>
```

where, 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 rewriting on it.

9.1 Variational Monte Carlo

Additional information:

- `walkers`. The initial default number of walkers is 1 but in the CPU branch this number will be overwritten as the number of OpenMP threads if the user requested number is smaller than the number of threads.

- `blocks`. This parameter is universal for all the method. At the end of each block, all the statistics accumulated in the block is dumped in to files, e.g. scalar.dat.

- `warmupsteps`. Warm-up steps are steps used only for equilibration. All the samples generated by warm-up steps are discarded. In practice, there's no need to use many walm-up steps because we can always discard more statistics when we perform the post-process.

`vmc` method				
parameters				
name	**datatype**	**values**	**default**	**description**
`walkers`	integer	> 0	dep.	number of walkers per node
`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	total number of samples
`stepsbetweensamples`	integer	> 0	1	period of the sample accumulation
`samplesperthread`	integer	≥ 0	0	number of samples per thread
`storeconfigs`	integer	all values	0	store configurations

- `substeps`. In a substep, each of the electrons is moved only once by either particle-by-particle or all-electron move. Because the local energy is evaluated not at each substep but at each step, increasing the number of substeps doesn't accumulate more samples. But in order to reduce the correlation between consecutive samples, increasing substeps is a very good option for its cheaper computational 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 timestep reduces the time correlation but might also reduces the accept ratio. Users are required to check the accept ratio of the calculation and make sure it's larger than 0.9 or between 0.2 and 0.8 with or without the drift.

- `stepsbetweensamples`. Due to the fact that samples generated by consecutive steps might be still correlated. Having stepsbetweensamples larger than 1 reduces that correlation. In practice, using larger substeps is cheaper than using stepsbetweensamples to decorrelate samples.

- `samples`. This is the total amount of samples generated in the current VMC session. This parameter is not important for VMC only calculation but necessary if optimization or DMC follows.
$$\text{samples} = \frac{\text{blocks} \cdot \text{steps} \cdot \text{walkers}}{\text{stepsbetweensamples}} \cdot \text{number of MPI tasks}$$

- `samplesperthread`. This is an alternative way to set the target amount of samples. More useful in the VMC session preparing the population for the following DMC calculation.
$$\text{samplesperthread} = \frac{\text{blocks} \cdot \text{steps}}{\text{stepsbetweensamples}}$$

- `storeconfigs`. If storeconfigs is set to a non-zero value, then electron configurations during the VMC run will be saved to the files.

The following is an example of VMC section.

```
<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>
```

9.2 Wavefunction Optimization

9.3 Diffusion Monte Carlo

dmc method

parameters

name	**datatype**	**values**	**default**	**description**
`targetwalkers`	integer	> 0	dep.	number of walkers per node
`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 updat
`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 t:qechnique
`branchInterval`	integer	≥ 0	1	branching interval
`substeps`	integer	≥ 0	1	branching interval
`nonlocalmoves`	string	yes/other	no	run with tmoves
`scaleweight`	string	yes/other	yes	scale weights (CUDA only)
`MaxAge`	double	≥ 0	10	kill persistent walkers
`MaxCopy`	double	≥ 0	2	limit population growth
`fastgrad`	text	yes/other	yes	fast gradients
`maxDisplSq`	real	all values	-1	maximum particle move
`storeconfigs`	integer	all values	0	store configurations

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 fluctutations 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`. DMC driver for running Hamiltonians with non-local moves. An typical usage is to simulate Hamitonians with non-local psuedopotentials with T-Moves. Setting this equal to false will impose the locality approximation.
- `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.

Listing 9.3: 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 9.4: The following is an example of running a simulation that can be restarted .

```
<qmc method="dmc" move="pbyp"  checkpoint="0" dumpconfig="5">
  <parameter name="timestep">         0.004  </parameter>
  <parameter name="blocks">           100   </parameter>
  <parameter name="steps">            400    </parameter>
</qmc>
```

The flags checkpoint and dumpconfig 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 9.5: Restart (read wakers from previous run)

```
<mcwalkerset fileroot="BH.s002" version="0 6" collected="yes"/>
```

where, 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 transfered between them.

Listing 9.6: 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>
```

9.4 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

vmc 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.

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?** For all observables appearing in the `scalar.dat` file in either VMC or DMC, RMC appends the suffix `_m` or `_p` for mixed and pure estimates respectively.

- **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.

Chapter 10

Analysing QMCPACK data

10.1 Using the qmca tool

10.2 Densities and spin-densities

10.3 Energy densities

Chapter 11

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.

11.1 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

11.2 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 12

Lab 1: Monte Carlo Statistical Analysis

12.1 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

12.2 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
```

12.3 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.

12.4 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:

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{12.1}$$

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) + (-0.46204733`
`)/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:

$$\sigma = \sqrt{\frac{1}{N(N-1)} \sum_{i=1}^{N} (x_i - \overline{x})^2} \tag{12.2}$$

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
+ (-0.46204733302-(-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:

$$\tilde{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left[(x^2)_i - (x_i)^2\right]} \tag{12.3}$$

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.

12.5 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).

12.6 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.

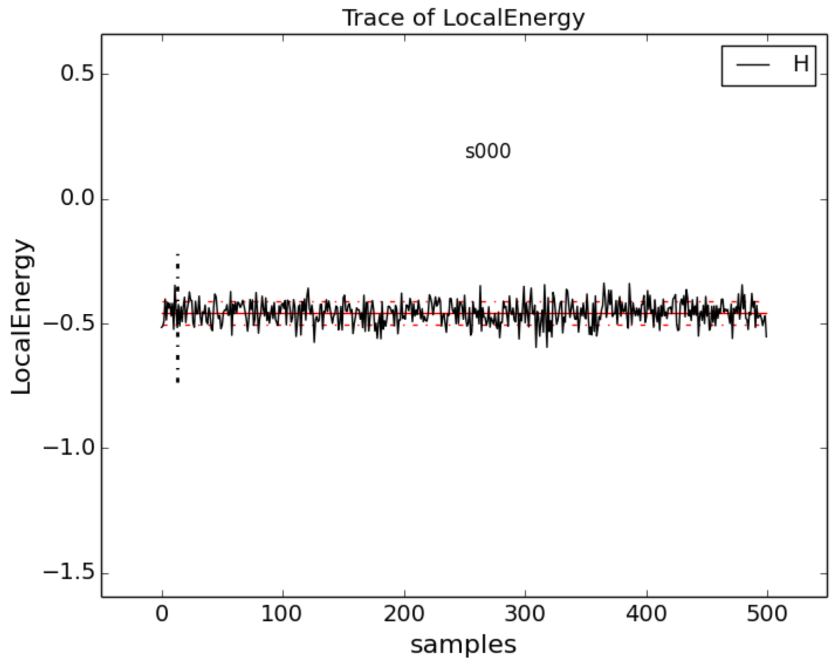
12.7 Evaluating MC simulation quality

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.

12.7.1 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:

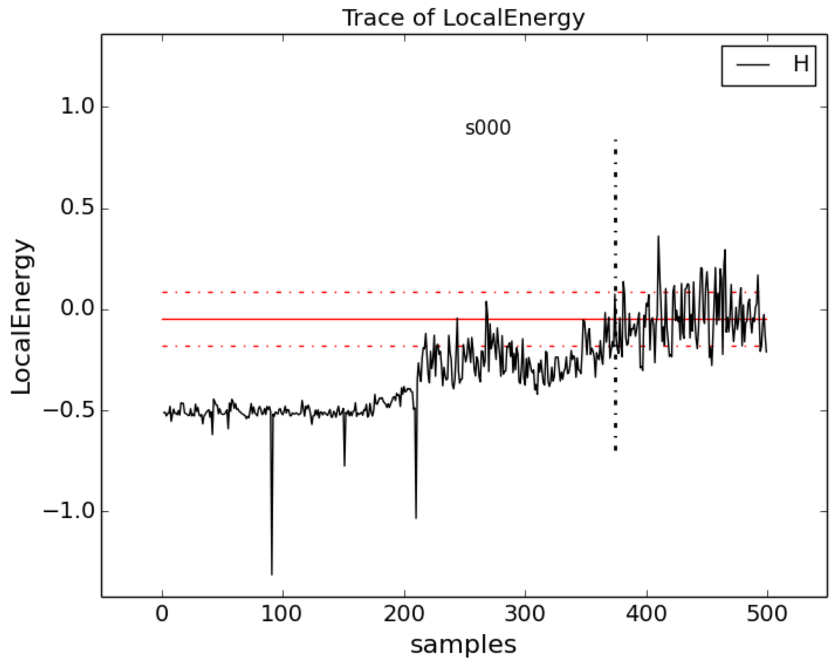


The solid black line connects the values of the local energy at each MC block (labeled "samples"). The average value is marked with a horizontal, solid red line. One standard deviation above and below the average are marked with horizontal, dashed red lines.

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.

12.7.2 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
```

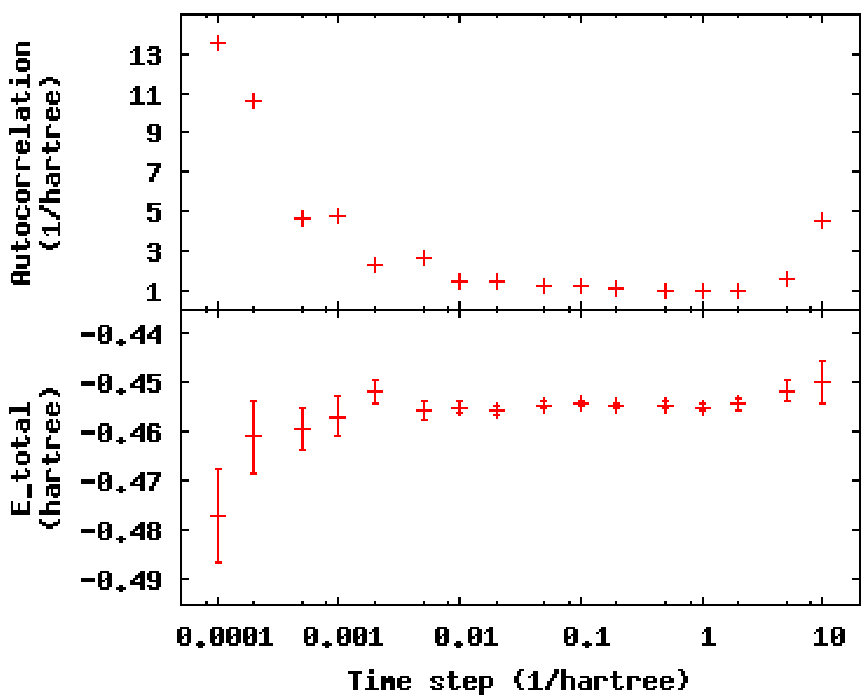
The value after the error bar on the quantity is the autocorrelation (1.0 in this case).

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**:

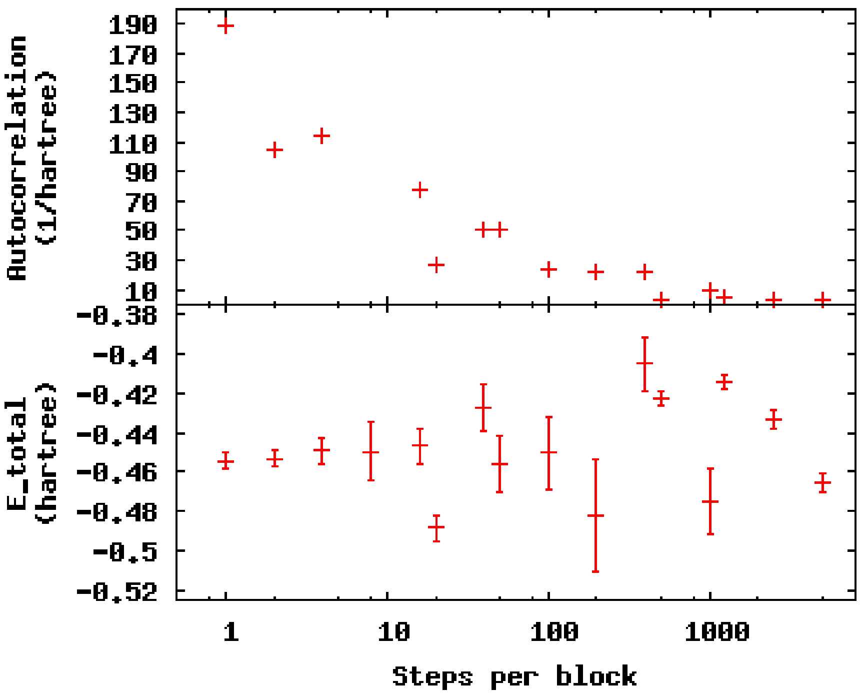


The error bar also increases with the autocorrelation.

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.

12.7.3 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).

12.7.4 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
```

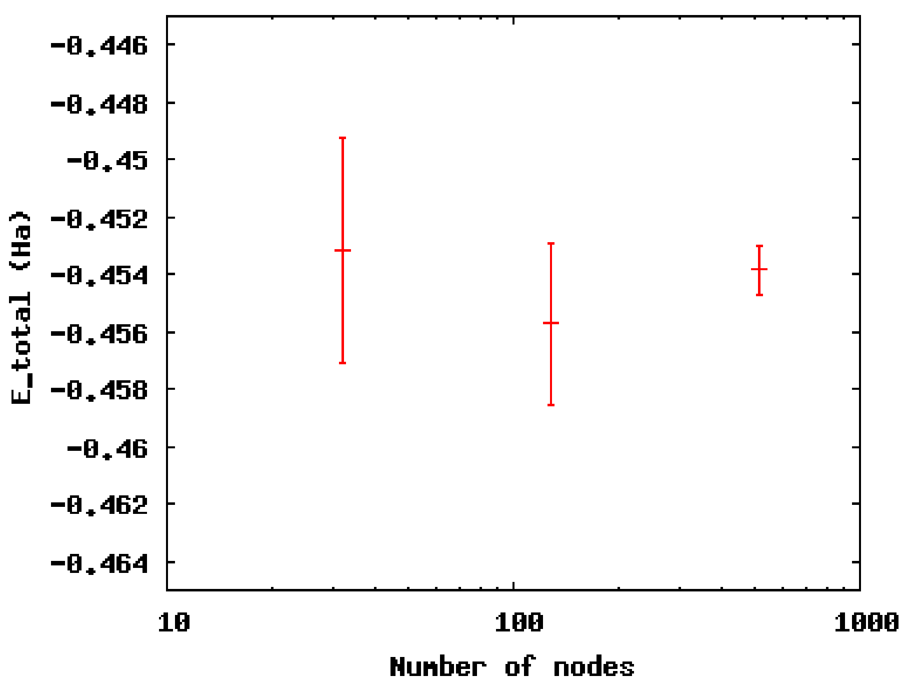
1.1651 is a very high ratio indicating the square of the fluctuations is on average larger than the value itself. In the next section, we will approach ways to improve the variance that subsequent labs will build upon.

12.8 Reducing statistical error bars

12.8.1 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).

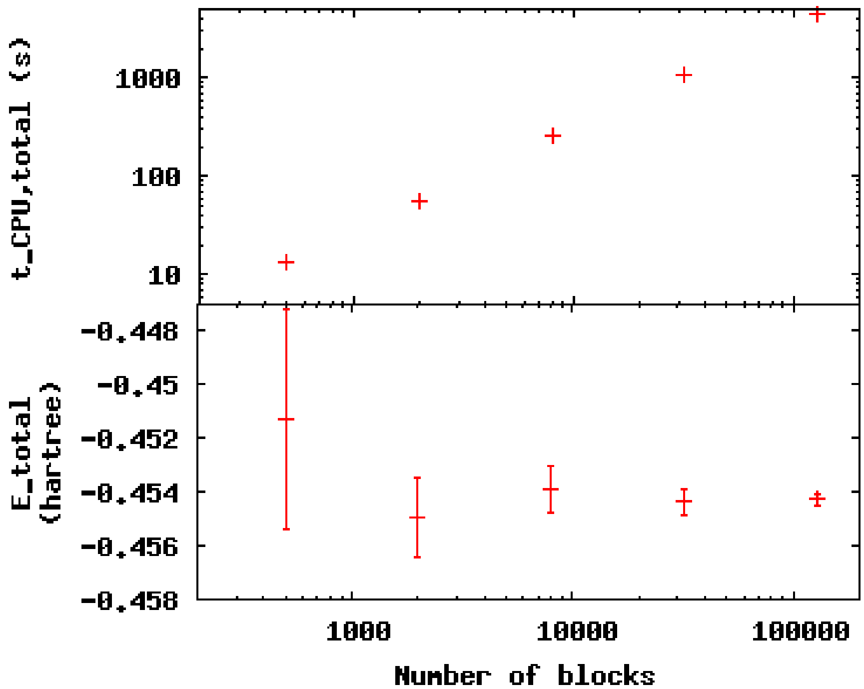
To see the effect of the running more samples in parallel, change to the directory `nodes`. The series here increases the number of nodes by factors of four from 32 to 128 to 512. Type **qmca -q ev *scalar.dat** and note the change in the error bar on the local energy as the number of nodes. Visualize 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.

12.8.2 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.

12.8.3 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.

12.9 Scaling to larger numbers of electrons

12.9.1 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:

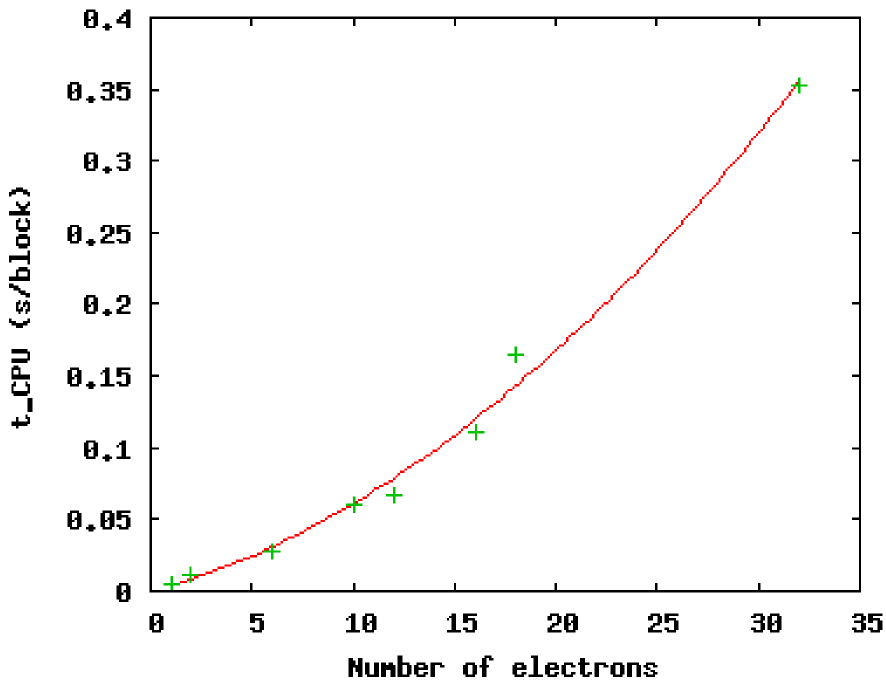
```
H2_STO___no_jastrow  series 0  Efficiency = 16698.725453 +/- 0.000000
H2_STO_with_jastrow  series 0  Efficiency = 52912.365609 +/- 0.000000
```

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).

12.9.2 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 13

Lab 2: QMC Basics

13.1 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

13.2 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

13.3 Lab directories and files

```
labs/lab2_qmc_basics/

 docs                  - documentation
    Lab_2_Slides.pdf     - slides presented during the lab

 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
```

13.4 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. 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:

```
jobrun_vesta 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:

```
jobrun_vesta 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.xml`.

13.5 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 13.1 below. Setting `wf_collect=.true.` instructs Quantum Espresso to write the orbitals to disk at the end of the run. Note that the plane-wave energy cutoff has been set to a reasonable value of 200 Ry here (`ecutwfc=200`). 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 13.1: 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          = 200
   ecutrho          = 800
   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       14.17294600      14.17294600      14.17294600

K_POINTS automatic
   1 1 1  0 0 0

CELL_PARAMETERS cubic
        28.34589199       0.00000000       0.00000000
         0.00000000      28.34589199       0.00000000
         0.00000000       0.00000000      28.34589199
```

Run Quantum Espresso by typing

```
jobrun_vesta pw.x O.q0.dft.in
```

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.output`". Once finished, you should check the LDA total energy in `O.q0.dft.output` by typing "`grep '!  ' O.q0.dft.output`". The result should be close to

```
!    total energy              =     -31.56730213 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 13.2 below.

Listing 13.2: `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:

```
jobrun_vesta pw2qmcpack.x O.q0.p2q.in
```

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.

13.6 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

$$\Psi_{Slater-Jastrow} = e^{-J}\Psi_{Slater} \tag{13.1}$$

$$J = \sum_{\sigma\sigma'}\sum_{i<j} u_2^{\sigma\sigma'}(|r_i - r_j|) + \sum_{\sigma}\sum_{iI} u_1^{\sigma I}(|r_i - r_I|) \tag{13.2}$$

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 (use multiples of 512 on Vesta).

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 no, evaluate non-local pseudopotential derivatives approximately during optimization. This saves time and often does not affect optimization results unless the non-local contribution to the energy is large.

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

```
jobrun_vesta qmcpack O.q0.opt.in.xml
```

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.output`. 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: 7.1637471800e-01 NewCost: 7.1635425970e-01 Delta Cost:-2.0458302174e-05
...
  <optVariables href="O.q0.opt.s010.opt.xml">
uu_0 -7.1521288964e-01 1 1  ON 0
uu_1 -9.2877240336e-01 1 1  ON 1
uu_2 -1.0467103153e+00 1 1  ON 2
uu_3 -1.0953972230e+00 1 1  ON 3
uu_4 -1.1070130734e+00 1 1  ON 4
uu_5 -1.0874728130e+00 1 1  ON 5
uu_6 -1.0258488995e+00 1 1  ON 6
uu_7 -1.4467669332e+00 1 1  ON 7
ud_0 2.6875014571e-01 1 1  ON 8
ud_1 -1.5766165503e-02 1 1  ON 9
ud_2 -1.4123427373e-01 1 1  ON 10
ud_3 -1.8739730188e-01 1 1  ON 11
ud_4 -1.8070740017e-01 1 1  ON 12
ud_5 -1.2069963316e-01 1 1  ON 13
ud_6 -1.7982920869e-01 1 1  ON 14
ud_7 -4.2474426156e-01 1 1  ON 15
eO_0 -1.0520981758e+00 1 1  ON 16
eO_1 -9.1814574396e-01 1 1  ON 17
```

```
eO_2 -6.9775892269e-01 1 1  ON 18
eO_3 -4.5411507428e-01 1 1  ON 19
eO_4 -2.5743489506e-01 1 1  ON 20
eO_5 -1.0879610780e-01 1 1  ON 21
eO_6 -2.4139449229e-02 1 1  ON 22
eO_7 2.1037749269e-02 1 1  ON 23
  </optVariables>
...
  QMC Execution time = 2.6048565362e+01 secs
```

The cost function should decrease during each linear optimization (`Delta cost < 0`). Try "`grep OldCost *opt.output`". You should see something like this:

```
 OldCost: 1.3134592963e+00 NewCost: 7.7188924330e-01 Delta Cost:-5.4157005302e-01
 OldCost: 7.4845000981e-01 NewCost: 7.2064323018e-01 Delta Cost:-2.7806779628e-02
 OldCost: 6.7191694349e-01 NewCost: 6.7190823880e-01 Delta Cost:-8.7046816816e-06
 OldCost: 7.1515211999e-01 NewCost: 7.1420521174e-01 Delta Cost:-9.4690825078e-04
 OldCost: 3.0887809677e+00 NewCost: 3.0318584021e+00 Delta Cost:-5.6922565616e-02
 OldCost: 9.9369152107e-01 NewCost: 9.5254988608e-01 Delta Cost:-4.1141634997e-02
 OldCost: 7.1198030212e-01 NewCost: 7.1163560440e-01 Delta Cost:-3.4469771549e-04
 OldCost: 7.4114649965e-01 NewCost: 7.4022000532e-01 Delta Cost:-9.2649432275e-04
 OldCost: 7.2092937088e-01 NewCost: 7.2054430351e-01 Delta Cost:-3.8506736720e-04
 OldCost: 7.5929893734e-01 NewCost: 7.5713646616e-01 Delta Cost:-2.1624711826e-03
 OldCost: 7.1637471800e-01 NewCost: 7.1635425970e-01 Delta Cost:-2.0458302174e-05
 OldCost: 7.4630768329e-01 NewCost: 7.3689350610e-01 Delta Cost:-9.4141771893e-03
```

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  -15.752691 +/- 0.006638   1.308990 +/- 0.020904   0.0831
O.q0.opt  series 1  -15.854499 +/- 0.007304   0.754957 +/- 0.017430   0.0476
O.q0.opt  series 2  -15.866341 +/- 0.007299   0.685069 +/- 0.011021   0.0432
O.q0.opt  series 3  -15.860209 +/- 0.006086   0.706933 +/- 0.013362   0.0446
O.q0.opt  series 4  -15.823484 +/- 0.008361   3.435846 +/- 1.077998   0.2171
O.q0.opt  series 5  -15.855915 +/- 0.006224   0.988390 +/- 0.036496   0.0623
O.q0.opt  series 6  -15.871419 +/- 0.004609   0.723626 +/- 0.019255   0.0456
O.q0.opt  series 7  -15.874430 +/- 0.009471   0.707508 +/- 0.022512   0.0446
```

```
...
```

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. Why is the mean and the error of the variance sometimes large? Consider using "qmca -t -q ev O.q0.opt*scalar*" to investigate.
4. 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?

13.7 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 `walkers` = 1 in the VMC block ensures there will be only one VMC walker per execution thread. There will be a total of 512 VMC walkers in this case (see `O.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` = 512 here):

$$\texttt{VBLOCKS} \times \texttt{VSTEPS} \geq \frac{\texttt{DWALKERS}}{\texttt{VWALKERS}} \frac{5\ \text{Ha}^{-1}}{\texttt{VTIMESTEP}} \tag{13.3}$$

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

$$\begin{aligned}
\texttt{timestep}_n &= \texttt{timestep}_{n-1}/2 \\
\texttt{warmupSteps}_n &= \texttt{warmupSteps}_{n-1} \times 2 \\
\texttt{blocks}_n &= \texttt{blocks}_{n-1} \\
\texttt{steps}_n &= \texttt{steps}_{n-1} \times 2
\end{aligned} \tag{13.4}$$

Each DMC run will require about twice as much computer time as the one preceeding it. Note that the number of blocks is kept fixed for uniform statistical analysis. `blocks` × `steps` × `timestep` $\sim$ 60 Ha^{-1} is sufficient for this system.

Choose an initial DMC timestep and create a sequence of N timesteps according to 13.4. 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 13.4. Start the DMC timestep extrapolation run by typing:

```
jobrun_vesta qmcpack 0.q0.dmc.in.xml
```

The run should take only a few minutes to complete.

QMCPACK will create files prefixed with `0.q0.dmc`. The log file is `0.q0.dmc.output`. 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 0.q0.dmc.in.xml 40`"). You should see a plot similar to fig. 13.1. 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               = -15.8911         +/- 0.000756     (0.004757%)
b               = -0.221687        +/- 0.03757      (16.95%)
...
```

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 0.q0.dmc*dmc.dat --noac`). Does the population seem to be stable?

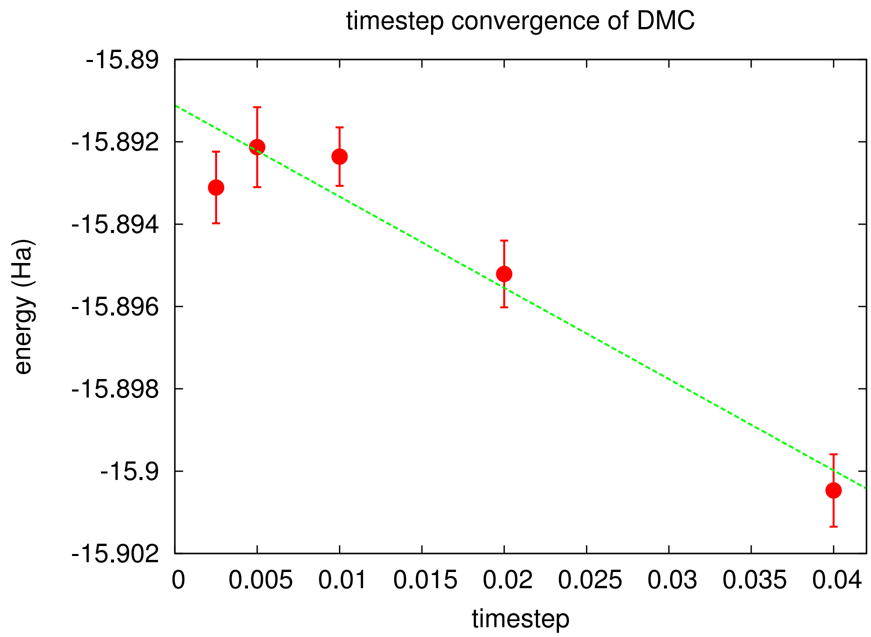


Figure 13.1: Linear fit to DMC timestep data from `PlotTstepConv.pl`.

5. (Optional) Study population control bias for the oxygen atom. Select a few population sizes (use multiples of 512 to fit cleanly on a single Vesta partition). Copy `O.q0.dmc.in.xml` to a new file and remove all but one DMC run (select a single timestep). Make one copy of the new file for each population, set "`samples`", and choose a unique `id` in `<project/>`. Run one job at a time to avoid crowding the lab allocation. 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?

13.8 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: `jobrun_vesta pw.x O.q1.dft.in`

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: `jobrun_vesta pw2qmcpack.x O.q1.p2q.in`

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) Perform the Jastrow optimization run: `jobrun_vesta qmcpack O.q1.opt.in.xml`

   (d) 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.opt.in.xml` to use the DFT prefix and the optimal Jastrow

   ```
   ...
   <project id="O.q1.opt" series="0">
   ...
   <include href="O.q1.ptcl.xml"/>
   ...
   <include href="[your opt.xml]"/>
   ...
   ```

   (c) Perform the DMC run: `jobrun_vesta qmcpack O.q1.dmc.in.xml`

(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 13.10 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. For comparison the LDA value is 12.25 eV.
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?

13.9 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 [9], 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 13.12, which gives a condensed summary of Python constructs. For more information about the functionality and effective use of Nexus, consult `docs/Nexus.pdf`. Details about the inner workings of Nexus can be found in the reference publication [9].

```
#! /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       = 'vesta',              # name of local machine
    account       = 'QMCPACK-Training'    # charge account for cpu time
    )

vesta = get_machine('vesta')          # allow max of one job at a time (lab only)
vesta.queue_size = 1

qmcjob = job(                         # specify job parameters
    nodes   = 32,                         # use 32 Vesta nodes
    threads = 16,                         # 16 OpenMP threads per node (32 MPI tasks)
    hours   = 1,                          # wallclock limit of 1 hour
                                          # use QMCPACK executable
    app     = '/soft/applications/qmcpack/Binaries/qmcpack'
    )

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
```

13.10 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 correponding 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.

Following the job parameters, inputs for the optimization method are given. The keywords should all be familiar from the QMCPACK XML input files you used previously:

```
linopt1 = linear(
    energy               = 0.0,
    unreweightedvariance = 1.0,
    reweightedvariance   = 0.0,
    timestep             = 0.4,
    samples              = 10240,
    warmupsteps          = 50,
    blocks               = 200,
    substeps             = 1,
    nonlocalpp           = True,
    usebuffer            = True,
    walkers              = 1,
    minwalkers           = 0.5,
    maxweight            = 1e9,
    usedrift             = True,
    minmethod            = 'quartic',
    beta                 = 0.025,
    exp0                 = -16,
    bigchange            = 15.0,
    alloweddifference    = 1e-4,
    stepsize             = 0.2,
    stabilizerscale      = 1.0,
```

```
    nstabilizers         = 3
    )
```

Requesting multiple loop's with different numbers of samples is more compact than in XML:

```
linopt1 = ...

linopt2 = linopt1.copy()
linopt2.samples = 61440 # opt w/ 61440 samples

opt_calcs = [loop(max=4,qmc=linopt1), # loops over opt's
             loop(max=8,qmc=linopt2)]
```

The VMC/DMC method inputs should also look familiar:

```
qmc_calcs = [
    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
        )
    ]
```

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       = 15.0,
    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',8,4.5),
                    ('J2','pade',0.5,0.5)],
    calculations = opt_calcs,
    dependencies = (p2q,'orbitals'),
    )
sims.append(opt)

qmc = generate_qmcpack(
    identifier   = 'qmc',
    ...
    jastrows     = [],
    calculations = qmc_calcs,
    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 4.5 Bohr and a 2-body Padé Jastrow with up-up and up-down "`B`" parameters set to 0.5 1/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 (we have explicitly prevented this for this lab by setting `queue_size=1` for Vesta). 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 `O_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 1
      Would have executed:  qsub --mode script --env BG_SHAREDMEMSIZE=32 dft.qsub.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 2
      Would have executed:  qsub --mode script --env BG_SHAREDMEMSIZE=32 p2q.qsub.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 3
      Would have executed:  qsub --mode script --env BG_SHAREDMEMSIZE=32 opt.qsub.in

  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 4
      Would have executed:  qsub --mode script --env BG_SHAREDMEMSIZE=32 qmc.qsub.in
  ...

  poll 17  memory 93.00 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
 dft.qsub.in
 O.BFD.upf
 O.BFD.xml
 opt.in.xml
 opt.qsub.in
 p2q.in
 p2q.qsub.in
```

```
 pwscf_output
 qmc.in.xml
 qmc.qsub.in
 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`) and submission (`dft.qsub.in`, `p2q.qsub.in`, `opt.qsub.in`, `qmc.qsub.in`) 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 15 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 `[1.00,0.90,0.925,0.95,0.975,1.05,1.075,1.10]`). 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?

13.11 (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 13.4 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 13.10 (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`, repectively. 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:

```
qsub --mode script --env BG_SHAREDMEMSIZE=32 scf.qsub.in

(wait until JOB DONE appears in scf.output)

qsub --mode script --env BG_SHAREDMEMSIZE=32 p2q.qsub.in
```

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:

```
qsub --mode script --env BG_SHAREDMEMSIZE=32 vmc.qsub.in
```

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.

13.12 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 14

Lab 3: Advanced Molecular Calculations

14.1 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, CASSCF, 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 rate of multi-determinant DMC convergence

14.2 Lab directories and files

```
labs/lab3_advanced_molecules/
  ...
```

14.3 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.

14.3.1 Generation of a HF wave-function with GAMESS

From the top directory, go to "GAMESS/DFT/HF". 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, 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}/GAMESS/DFT/HF
rungms H2O.HF.inp > H2O.HF.out
convert4qmc -gamessAscii H2O.HF.output -add3BodyJ
mv sample.Gaussian-G2.xml H2O.HF.wfs.xml
mv sample.Gaussian-G2.ptcl.xml H2O.ptcl.xml
```

Due to the particular requirements of Vesta these executions can not be performed on the login nodes, they must be performed in the compute nodes. In order to accomplish this, we must make a submission script with the appropriate commands and submit it to the batch system. Two submission scripts have been provided, one for the GAMESS execution called submit gamess.csh and one for the submission of convert4qmc, with a similar descriptive name. Study these input files. (In this and all other exercises, you will need to make all submission scripts executables with the command chmod u+x script.csh.) When you are ready, start by submitting the GAMESS execution to the batch system using "./script gamess.csh" (This script calls the GAMESS run script, which itself calls qsub. Do not attempt to submit this specific script with qsub). The HF energy of the system is "-16.9600590022". To search for the energy in the output file quickly, you can use grepTOTAL ENERGY = H2O.HF.output. When the calculation completes, submit the execution of the converter using "qsub submit convert.csh" (all QMCPACK execution scripts will be submitted with qsub). This is a good time to review section B, which contains a description on the use of the converter.

14.3.2 Optimize the wave-function

When the execution of the previous steps is completed, there should be 2 new files called H2O.HF.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 "QMCPACK/SJ/First Run/Optimize". 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) and a submission script (submit.csh). 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/ ¿ and ¡include href=wfs.xml/ ¿ (the particle set must be defined before the wavefunction). The name of the files should now be H2O.ptcl.xml and H2O.HF.wfs.xml. Study both

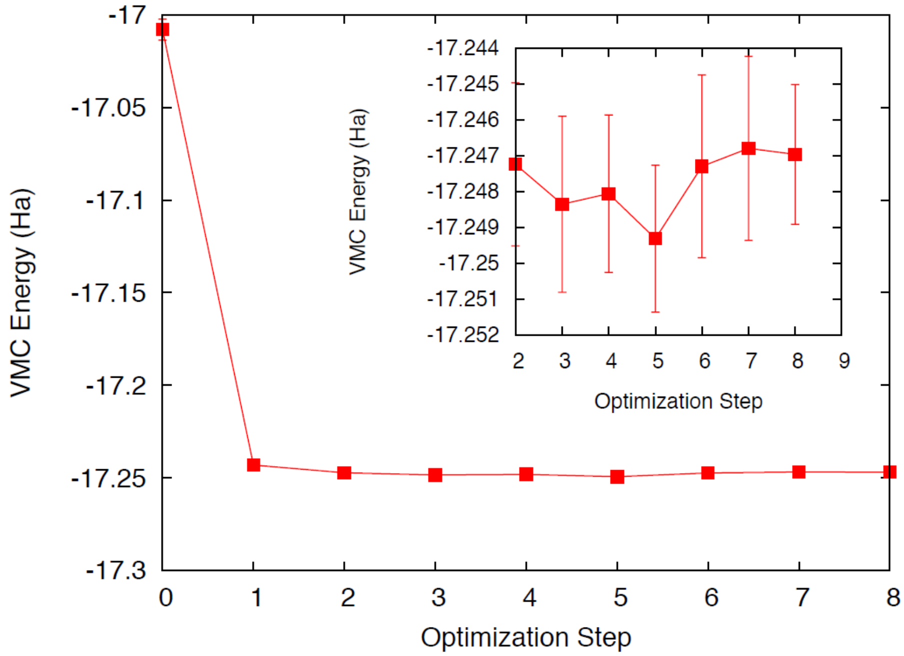


Figure 14.1: VMC energy as a function of optimization step.

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}/QMCPACK/SJ/First Run/Optimize
cp ${TRAINING TOP}/GAMESS/DFT/HF/H2O.*.xml .
qsub ./submit.csh
```

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. The resulting energy as a function of optimization step should look qualitatively similar to figure 14.1 below.

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.

14.3.3 Time-step Study

Now we will study the dependence of the DMC energy with time-step. From the top directory, go to QMCPACK/SJ/First Run/TimeStep. This folder contains a basic xml input file (vmc dmc.xml) that performs a short VMC calculation and three DMC calculations with varying time-steps (0.1, 0.05, 0.01). Copy the particle set and the last optimization file from the previous folder (the file called 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.HF.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. Using the submission script of the previous exercise as a base, create a submission script for this step and submit the run. Set the number of nodes to 32 (2 places must be changed), the number of threads to 16 and leave the number of tasks at 1.

The main steps needed to perform this exercise are:

```
cd ${TRAINING TOP}/QMCPACK/SJ/First Run/Timestep
cp ${TRAINING TOP}/QMCPACK/SJ/First Run/Optimize/H2O.HF.s007.xml
H2O.HF.opt.xml
cp ${TRAINING TOP}/QMCPACK/SJ/First Run/Optimize/H2O.ptcl.xml .
cp ${TRAINING TOP}/QMCPACK/SJ/First Run/Optimize/submit.csh .
vim vmc dmc.xml
vim submit.csh
qsub ./submit.csh
```

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 14.2, 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.

14.3.4 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.

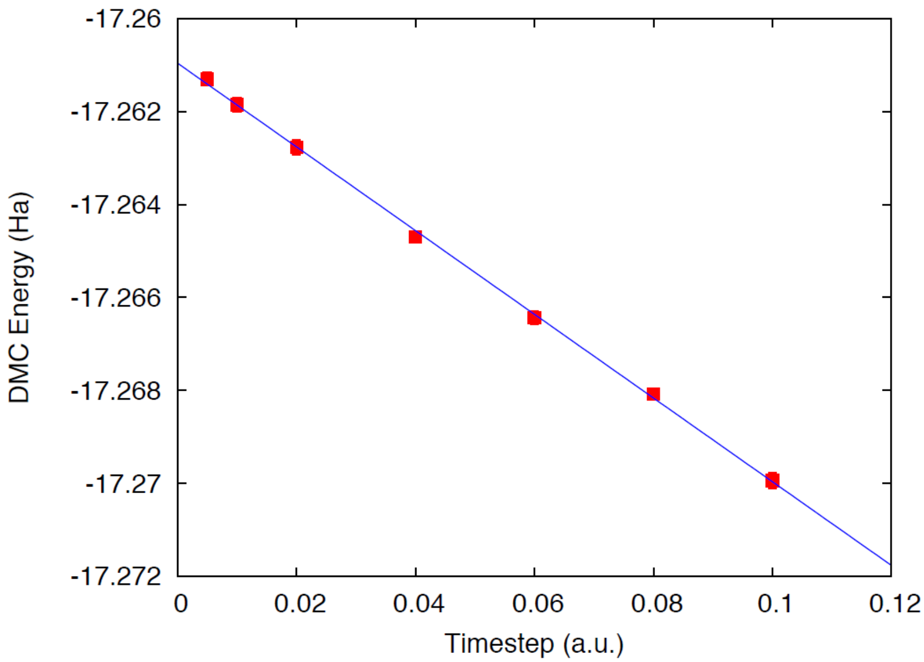


Figure 14.2: DMC energy as a function of timestep.

From the top directory, go to "QMCPACK/SJ/First Run/NumberWalkers". 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. Optimized wave-functions files can also be found in "REFERENCE/QMCPACK/SJ/First Run/Optimize". The directory contains a sample DMC input file and submission script. Make 3 directories named NWx, with x values 60,120,480 and copy the input file and submission script to each one. Go to "NW60", 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 "../H2O.HF.opt.xml" for example), change "targetWalkers" to 60, change the number of steps to 100, the time-step to 0.04 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. Modify the submission script to use 2 nodes and 16 threads and submit the run. We recommend setting ∼2*(#threads) walkers per node (slightly smaller than this value), which is the reason why we use 2 nodes when we want 60 walkers in total.

Repeat the same procedure in the other folders by setting (targetWalkers=120, steps=100, timestep=0.04, blocks=200,nodes=4,threads=16) in NW120 and (targetWalkers=480, steps=100, timestep=0.04, blocks=100,nodes=16,threads=16) 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 14.3 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.

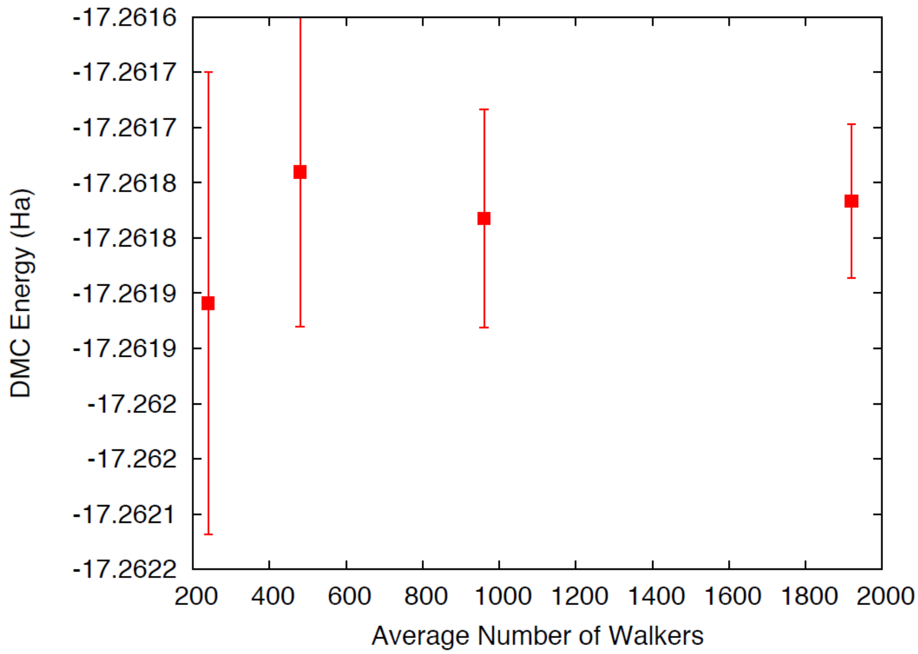


Figure 14.3: DMC energy as a function of the average number of walkers.

14.4 Exercise #2 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 generic versions provided in "QMCPACK-/Generic Files" and "GAMESS/Generic Files". 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.

14.4.1 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 "QMCPACK/SJ/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,

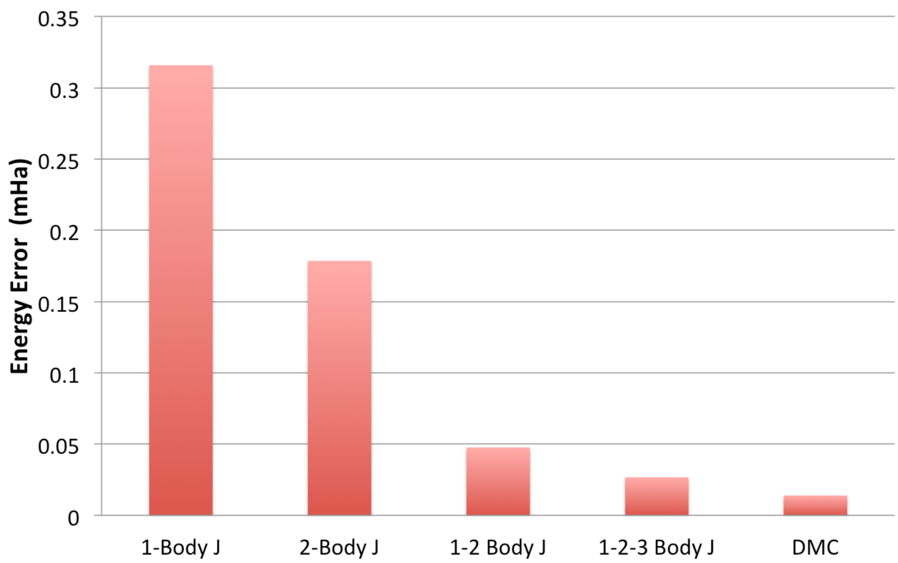


Figure 14.4: VMC energy as a function of Jastrow type.

1-2J and 1-2-3J. For both 2J and 1-2J (we have already optimized a wave-function for the 1-2-3J case, so the steps will be slightly different in this case), copy the file optm vmc.xml and the sample submission file from “QMCPACK/Generic Files” . This input file performs both wave-function optimization and a VMC calculation. Copy the un-optimized HF wave-function and particle set files from “GAMESS/DFT/HF”, if you followed the instructions in exercise #1 these should be named H2O.HF.wfs.xml and H2O.ptcl.xml. Otherwise, you can obtained them from the REFERENCE files. Modify the file H2O.HF.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 1-2J, 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 1-2J cases. For the 1-2-3J case, the wave-function has already been optimized in the previous exercise. Copy the optimized HF wave-function and the particle set from QMCPACK/SJ/First Runs/Optimize. Copy the input file and submission script 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 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 14.4 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.

14.4.2 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 "GAMESS/DFT". 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 generic GAMESS input file for a ROHF calculation from "GAMESS/Generic Files", a file named rohf.inp should exist. 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 generic the HF wave-function used in exercise #1. In order to perform a DFT calculation we only need to define "DFTTYP" in the `$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.

14.4.3 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 QMCPACK/SJ/SPOrbitals. 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 QMCPACK/Generic Files. A file named optm vmc dmc.xml should exist that contains all three execution blocks. 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 GAMESS/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?

14.4.4 Generation of a CISD wave-functions using GAMESS

In this section we will use GAMESS to generate a multi-determinant wave-function with 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.

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.Youll see several input and output files named H2O.XXX.inp and H2O.XXX.out, where XXX is one of the following multi-determinant methods: CISD, CASSCF, CASCI, SOCI. There will be time in the next step to study the GAMESS input files and the description in section A. In the next exercise we will use the CISD output, in the next exercise we will use the remaining files. Since the output is already provided, the only thing ended is to use the converter to generate the appropriate QMCPACK files. Copy a submission script from GAMESS/Generic Files and execute the converter for all the output files in the directory (with the exception of CASSCF, which is used to generate orbitals but it doesnt contain appropriate CI coefficients). In all cases we used PRTMO=.T. in the GUESS section, so you should read these orbitals from the output

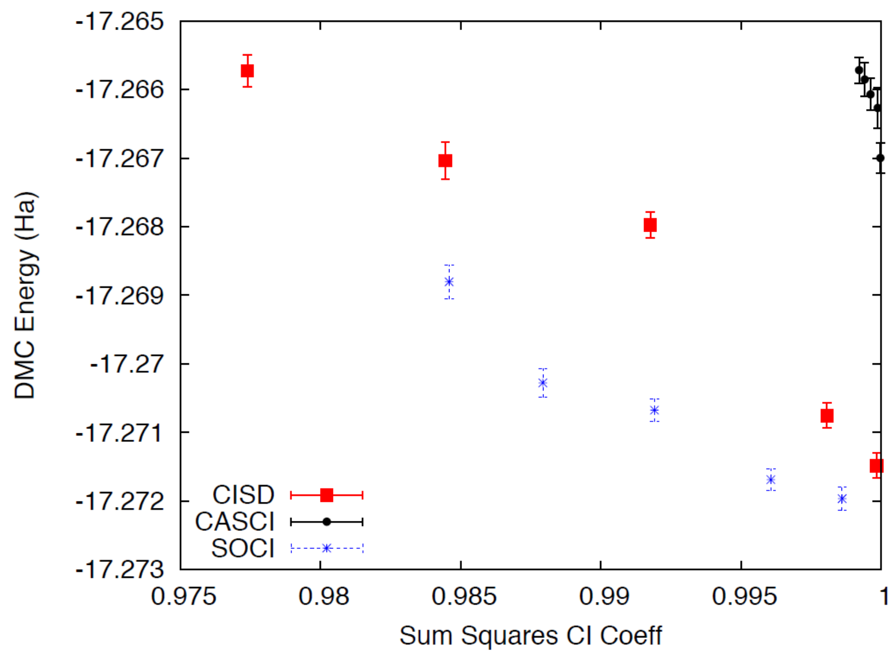


Figure 14.5: DMC energy as a function of the sum of the square of CI coefficients from CISD.

(-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.

14.4.5 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 QMCPACK/MSD/CISD and make a folder called thr0.01. 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.CISD.wfs.xml. Look for the multideterminant XML block and change the cutoff parameter in detlist to 0.01. Then 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-funtion optimization followed by VMC and DMC calculations. Submit the calculation.

While this run is completed, go to section A and review the GAMESS input file description. When the run is completed, go to the previous directory and make a new folder named thr0.0075. 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. Instead of starting from an un-optimized wave-function, start from optimized wave-function from thr0.01. 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. Do 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 14.5 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.

14.5 Exercise #3: Orbital Sets and Configurations in 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. Since the un-optimized wave-functions were generated in subsection Generation of a CISD wavefunctions using GAMESS of exercise #2, we can skip this section and go straight to the wave-function optimization. Go to QMCPACK/MSD and make folders for the remaining wave-function types: CASCI and SOCI. The exercise has already been performed with a CISD wave-function in exercise #2.

A CASCI wave-function is produced from a CI calculation that includes all the determinants in a 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 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 eave 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 14.5 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. 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?

14.6 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).

14.6.1 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`.

```
$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.

14.6.2 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.

14.6.3 Multi-Configuration Self-Consistent Field (MCSCF)

MCSCF calculations are performed by setting SCFTYP=MCSCF in the $CONTROLsection.Ifthisoptionisset, aM$ 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.

```
$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.

14.6.4 Configuration Interaction (CI)

Configuration interaction (full CI, truncated CI, CAS-CI, etc) calculations are performed by setting SCFTYP=NONE and CITYP=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.

14.6.5 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.

```
$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.

```
$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.

```
$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.

```
$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
```

14.6.6 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.

```
$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
```

14.7 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.

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.

14.8 Appendix C: Wave-function Optimization XML 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>
```

Figure 14.6: Sample XML optimization block.

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.

14.9 Appendix D: VMC and DMC XML block

```
<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>
```

Figure 14.7: Sample XML blocks for VMC and DMC calculations.

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 Casulas T-moves.

14.10 Appendix E: Wave-function XML block

```xml
<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>

    <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>
```

Figure 14.8: Basic framework for a single determinant determinantset XML block.

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. Figure 14.8 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. Figure 14.9 shows a section of a basisset block for an oxygen atom. The

```xml
<basisset name="LCAOBSet">
  <atomicBasisSet name="Gaussian-G2" angular="cartesian" type="Gaussian"
      elementType="O" normalized="no">
    <grid type="log" ri="1.e-6" rf="1.e2" npts="1001"/>
    <basisGroup rid="O00" n="0" l="0" type="Gaussian">
      <radfunc exponent="1.253460000000e-01" contraction="5.574095889400e-02"/>
      <radfunc exponent="2.680220000000e-01" contraction="3.048477751890e-01"/>
      <radfunc exponent="5.730980000000e-01" contraction="4.537516653790e-01"/>
      <radfunc exponent="1.225429000000e+00" contraction="2.959257817680e-01"/>
      <radfunc exponent="2.620277000000e+00" contraction="1.956698557000e-02"/>
      <radfunc exponent="5.602818000000e+00" contraction="-1.286269051440e-01"/>
      <radfunc exponent="1.198024500000e+01" contraction="1.202399113300e-02"/>
      <radfunc exponent="2.561680100000e+01" contraction="4.069997000000e-04"/>
      <radfunc exponent="5.477521600000e+01" contraction="-7.599994400000e-05"/>
    </basisGroup>
    <basisGroup rid="O10" n="1" l="0" type="Gaussian">
      <radfunc exponent="1.686633000000e+00" contraction="1.000000000000e+00"/>
    </basisGroup>
    <basisGroup rid="O20" n="2" l="0" type="Gaussian">
      <radfunc exponent="2.379970000000e-01" contraction="1.000000000000e+00"/>
    </basisGroup>
```

Figure 14.9: Sample XML block for an atomic orbital basis set.

structure of this block is rigid and should not be modified. Figure 14.10 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.

```xml
<slaterdeterminant>
  <determinant id="updet" size="4">
    <occupation mode="ground"/>
    <coefficient size="57" id="updetC">
 8.56319000000000e-01 -1.06750000000000e-02 -9.24550000000000e-02  0.00000000000000e+00
 0.00000000000000e+00  1.26393000000000e-01  0.00000000000000e+00  0.00000000000000e+00
-3.68840000000000e-02  0.00000000000000e+00  0.00000000000000e+00 -2.07730000000000e-02
-1.12000000000000e-04  9.06000000000000e-04 -7.94000000000000e-04  0.00000000000000e+00
 0.00000000000000e+00  0.00000000000000e+00 -3.51300000000000e-03  2.29100000000000e-03
 1.22100000000000e-03  0.00000000000000e+00  0.00000000000000e+00  0.00000000000000e+00
 0.00000000000000e+00  0.00000000000000e+00 -1.63400000000000e-03  0.00000000000000e+00
-1.81900000000000e-03  0.00000000000000e+00  4.01100000000000e-03  0.00000000000000e+00
 0.00000000000000e+00  0.00000000000000e+00  1.52778000000000e-01  3.87120000000000e-02
-8.84000000000000e-04  0.00000000000000e+00  2.80600000000000e-02 -1.68630000000000e-02
 0.00000000000000e+00  1.07410000000000e-02 -6.63300000000000e-03 -3.92300000000000e-03
 4.72700000000000e-03 -8.04000000000000e-04  0.00000000000000e+00  0.00000000000000e+00
-5.74000000000000e-03  1.52778000000000e-01  3.87120000000000e-02 -8.84000000000000e-04
 0.00000000000000e+00 -2.80600000000000e-02 -1.68630000000000e-02  0.00000000000000e+00
-1.07410000000000e-02 -6.63300000000000e-03 -3.92300000000000e-03  4.72700000000000e-03
-8.04000000000000e-04  0.00000000000000e+00  0.00000000000000e+00  5.74000000000000e-03
```

Figure 14.10: Sample XML block for the single slater determinant case.

Figure 14.11 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.

```
<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="0.01">
        <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>
```

Figure 14.11: Basic framework for a multi-determinant determinantset XML block.

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 14.12 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.

```
<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>
```

Figure 14.12: Sample Jastrow XML block.

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 15

Lab 4: Condensed Matter Calculations

15.1 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

15.2 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-loop-buffer.py   - VMC scan over orbital bspline buffer region size
 graphene-final.py         - DMC for final meshfactor and buffer region
 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 qmcpacks 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.

15.3 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:

$$\Psi(\vec{r}) = e^{i\vec{k}\cdot\vec{r}}u(\vec{r}) \tag{15.1}$$

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

$$\mathbf{a}_i^s = S_{ij}\mathbf{a}_j^p \tag{15.2}$$

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:

$$\vec{r}' = \vec{r} + x\mathbf{a}_1^p + y\mathbf{a}_2^p + z\mathbf{a}_3^p = \vec{r} + \vec{T} \tag{15.3}$$

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:

$$\Psi(\vec{r}) = \Psi(\vec{r}')e^{i\vec{T}\cdot\vec{k}} \tag{15.4}$$

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 necesary 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.

15.4 Total energy of BCC beryllium

As was discussed in this mornings 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:

$$E_{\text{inf}} = C + E_N/N \tag{15.5}$$

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 advantagous 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:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \tag{15.6}$$

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 direcotry 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 inefficent 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 executabe 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 teh 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 warrent 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.

15.5 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 scripts graphene-loop-buffer.py and graphene-loop-mesh.py. These examine within variational Monte Carlo two approaches 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. Similarly, the script graphene-loop-buffer.py uses a feature which generates two spline tables for the wavefunction. One will have half of the mesh spacing requested in the input file and will be valid everywhere. The second one will only be defined in the smallest parallelpiped that contains all of the atoms in the simulation cell with minimum distance given by the buffer size. Again, see what the smallest possible buffer size is that preserves the vmc energy and 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    = '---',
    truncate     = False,
    buffer       = 0.0,
```

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, a buffer table and single precision arithmetic.

15.6 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 16

Contributing to the Manual

This section briefly describes how to contribute to the manual. It 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.
- Place unformatted text targeted at developers 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.

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 8. 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 17

References

Bibliography

[1] 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.

[2] 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.

[3] T. Kato, "Fundamental properties of hamiltonian operators of the schrodinger type," *Transactions of the American Mathematical Society*, vol. 70, pp. 195–211, 1951.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. T. Krogel, "Nexus: A modular workflow management system for quantum simulation codes," *Computer Physics Communications*, vol. 198, pp. 154 – 168, 2016.