The Nexus User Guide

By Jaron T. Krogel

22 May 2013

Contents

Contents ii

1 Using this document 1

2 Overview of Nexus 2
- 2.1 What Nexus is . . . 2
- 2.2 What Nexus can do . . . 2
- 2.3 How Nexus is used . . . 2

3 Nexus Installation 4

4 Complete Examples 6
- 4.1 Simple QMC Calculations . . . 7

5 Nexus User Reference 23
- 5.1 Reading what you wrote . . . 23
- 5.2 Nexus settings: global state and user-specific information . . . 23

6 QMC Practice in a Nutshell 25
- 6.1 VMC and DMC in the abstract . . . 25
- 6.2 From expectation values to random walks . . . 26
- 6.3 Quality orbitals: planewaves, cutoffs, splines, and meshes . . . 26
- 6.4 Quality Jastrows: less variance = more efficient . . . 27
- 6.5 Finite size effects: k-points, supercells, and corrections . . . 28
- 6.6 Imaginary time discretization: the DMC timestep . . . 29
- 6.7 Population control bias: safety in numbers . . . 29
- 6.8 The fixed node/phase approximation: varying the nodes/phase . . . 30
- 6.9 Pseudopotentials: theoretical dissonance, the locality approximation, and T-moves . . . 31
- 6.10 Other approximations: what else is missing? . . . 32

7 Recommended Reading 33
- 7.1 Helpful Links for Installing Python Modules . . . 33
- 7.2 Helpful Links for Installing Electronic Structure Codes . . . 33
- 7.3 Brushing Up On Python . . . 34

7.4 Quantum Monte Carlo: Theory and Practice 35

Index **36**

1 Using this document

The Nexus User Guide provides an overview of Nexus (2), instructions on how to install it (3), complete examples of electronic structure calculations using it (4), a complete reference section (5), a brief overview of Quantum Monte Carlo (QMC) from an applied perspective (6), and directions on where to go to learn more (7). If you are new to QMC, consider reading the "QMC Practice in a Nutshell" section (6) and the review articles and online resources listed under "Quantum Monte Carlo: Theory and Practice" (7.4) before proceeding to the overview (2) and the examples (4). For those more experienced in QMC, or the impatient, quickly visit "Nexus Installation" (3) and see the examples section (4) for template calculations to begin using Nexus immediately. For fine-grained information about Nexus's many features, consult the "Nexus User Reference" (5). If you cannot find what you need in this document, contact the main developer of Nexus (Jaron Krogel), at krogeljt@ornl.gov (but please make a thorough search first!).

2 Overview of Nexus

2.1 What Nexus is

Nexus is a collection of tools, written in Python, to perform complex electronic structure calculations and analyze the results. The main focus is currently on performing arbitrary Quantum Monte Carlo (QMC) calculations with QMCPACK. A single QMC calculation typically requires several previous calculations with other codes to produce a starting guess for the many-body wavefunction and convert it into a form that QMCPACK understands. Managing the resulting array of calculations, and the flow of information between them, quickly becomes unweildy to the researcher, demands a great deal of human time, and increases the potential for human error. Nexus reduces both the human time required and potential for error by automating the total simulation process.

2.2 What Nexus can do

The capabilities of Nexus currently include crystal structure generation, standalone Density Functional Theory (DFT) calculations with PWSCF, Hartree-Fock (HF) calculations of atoms with the SQD code (packaged with QMCPACK), complete QMC calculations with QMCPACK (including wavefunction optimization, Variational Monte Carlo (VMC), and Diffusion Monte Carlo (DMC) in periodic or open boundary conditions), automated job management on workstations (by acting as a virtual queue) and clusters/supercomputers (such as OIC5, Edison, Blue Waters, with Titan coming) including handling of dependencies between calculations and bundling of jobs, and extraction of results from completed calculations for analysis. The integration of these capabilities permits the user to focus on the high-level tasks of problem formulation and interpretation of the results without (in principle) becoming too involved in the time-consuming, lower level details.

2.3 How Nexus is used

Use of Nexus currently involves writing a short Python script describing the calculations to be performed. This small script formed by the user closely resembles an input file for electronic structure codes. A key difference is that this "input file" represents executable code, and so variables are easily defined for use in expressions and more complicated simulation workflows (*e.g.* an equation of state) can be constructed with if/else logic and for loops. Knowledge of the Python programming language is helpful to perform complex cal-

culations, but not essential for use of Nexus. Starting from working "input files" such as those covered in the "Complete Examples" section (4) is a good way to proceed.

3 Nexus Installation

Installation of Nexus can be accomplished by a single download with Subversion (SVN) and setting a single environment variable provided a working python environment exists. Follow the example below to download Nexus:

```
cd /your_download_path
svn co https://subversion.assembla.com/svn/qmcdev/trunk/nexus
```

If you do not have access to the Assembla SVN repository, please make an account on assembla.com and email the lead developer of QMCPACK (Jeongnim Kim) at jnkim@ornl.gov to obtain access.

To make your Python installation (must be Python 2.x as 3.x is not supported) aware of Nexus, simply set the PYTHONPATH environment variable. For example, in bash this would look like:

```
export PYTHONPATH=/your_download_path/nexus/library
```

If you want to use the command line tools, add them to your path:

```
export PATH=/your_download_path/nexus/executables:$PATH
```

Add these to *e.g.* your .bashrc file to make Nexus available to future sessions.

In addition to the standard Python installation, the `numpy` module must be installed for Nexus to function at a basic level. To realize the full range of functionality available, it is recommended that the `scipy`, `matplotlib`, and `h5py` modules be installed as well. Many of these packages are already available in various supercomputing environments. On a debian-based Linux system, such as Ubuntu, installation of these python modules is easily accomplished by invoking the following at the command line:

```
sudo apt-get install python-numpy
sudo apt-get install python-scipy python-matplotlib python-h5py
```

For installing the Python modules on other platforms, please see "Helpful Links for Installing Python Modules" (section 7.1).

Of course, to run full calculations, the simulation codes and converters involved must be installed as well. These include a modified version of Quantum Espresso (`pw.x`, `pw2qmcpack.x`, optionally `pw2casino.x`), QMCPACK (`qmcapp`, `qmcapp_complex`), SQD (`sqd`, packaged with QMCPACK), and, optionally, `wfconvert`. Complete coverage of this task is beyond the scope of the current document, but please see "Helpful Links for Installing Electronic Structure Codes" (section 7.2).

4 Complete Examples

Disclaimer: Please note that the examples given here do not generally qualify as production calculations because the supercell size, optimization process, DMC timestep and other key parameters may not be converged. Pseudopotentials are provided "as is" and should not be trusted without explicit validation.

Complete examples of calculations performed with Nexus are provided in the following sections. These examples are intended to highlight basic features of Nexus and act as templates for future calculations. A complete description of the available features can be found in "Nexus User Reference" (section 5). If there is an example you would like to contribute, or if you feel an example on a particular topic is needed, please contact the developer at krogeljt@ornl.gov to discuss the possibilities.

To perform the example calculations yourself, consult the `examples` directory in your Nexus installation:

```
/your_download_path/nexus/examples
```

The examples assume that you have working versions of `pw.x`, `pw2qmcpack.x`, `qmcapp` (real version), and `qmcapp_complex` (complex version) installed and in your `PATH`. A brief description of each example is given below.

Graphene Sheet DMC

A representative bulk calculation. The total DMC energy of a graphene "sheet" consisting of 8 atoms is computed. DFT is performed with PWSCF on the primitive cell followed by Jastrow optimization by QMCPACK and finally a supercell VMC+DMC calculation by QMCPACK.

C 20 Molecule DMC

A representative molecular calculation. The total DMC energy of an ideal C 20 molecule is computed. DFT is performed with PWSCF on a periodic cell with some vacuum surrounding the molecule. QMCPACK optimization and VMC+DMC follow on the system with open boundary conditions.

(Note that without the crystal field splitting afforded by the initial artificial periodicity, the Kohn-Sham HOMO would be degenerate, and so a production calculation would likely require more care in appropriately setting up the wavefunction.)

4.1 Simple QMC Calculations

The simplest QMC calculations that can be performed with Nexus involve five main stages:

Configure Nexus settings
The `settings` function allows you to specify where pseudopotentials are located, whether to generate input files without running jobs, details of the machine you are on, and how often to have Nexus check on the status of running jobs.

Describe the physical system
Generate a crystal structure with the `generate_physical_system` convenience function (see "Graphene Sheet DMC" 4.1), or load an XYZ file into a `Structure` object (see "C 20 Molecule DMC" 4.1). This is where information about k-points, net system charge, or net system spin are recorded.

Describe the calculations
Use the `standard_qmc` (see "Graphene Sheet DMC" 4.1) or `basic_qmc` (see "C 20 Molecule DMC" 4.1) convenience functions to select specific pseudopotentials, specify PWSCF and QMCPACK input parameters, and job details, like how many nodes/cores/threads to use.

Run the simulations
Pass simulation objects created by `standard_qmc/basic_qmc` to the `ProjectManager` and run the jobs by calling the `run_project` function.

Collect simulation results
Load results of simulation objects using the `load_analyzer_image` function. This gives you access to several/most (PWSCF/QMCPACK) physical results produced by the simulation with statistical analysis already performed (though the responsibility is still yours to verify absolute correctness).

For more information about the functions/objects mentioned above, consider the examples in the following sections or consult "Nexus User Reference" (section 5).

Example: Graphene Sheet DMC

The files for this example are found in:

```
/your_download_path/nexus/examples/simple_qmc/graphene_example
```

Take a moment to study the "input file" script (`graphene_example.py`) and the attendant comments (prefixed with #). The five stages listed in section 4.1 should be apparent.

```
#! /usr/bin/env python

from nexus import settings,ProjectManager,Job
from nexus import generate_physical_system
from nexus import loop,linear,vmc,dmc
from qmcpack_calculations import standard_qmc


#general settings for Nexus
settings(
    pseudo_dir    = './pseudopotentials',# directory with all pseudopotentials
    sleep         = 3,                   # check on runs every 'sleep' seconds
    generate_only = 0,                   # only make input files
    status_only   = 0,                   # only show status of runs
    machine       = 'node16',            # local machine is 16 core workstation
    )



#generate the graphene physical system
graphene = generate_physical_system(
    structure = 'graphite_aa',  # graphite keyword
    cell      = 'hex',          # hexagonal cell shape
    tiling    = (2,2,1),        # tiling of primitive cell
    constants = (2.462,10.0),   # a,c constants
    units     = 'A',            # in Angstrom
    kgrid     = (1,1,1),        # Monkhorst-Pack grid
    kshift    = (.5,.5,.5),     # and shift
    C         = 4               # C has 4 valence electrons
    )


#generate the simulations for the qmc workflow
```

```
qsims = standard_qmc(
    # subdirectory of runs
    directory       = 'graphene_test',
    # description of the physical system
    system          = graphene,
    pseudos         = ['C.BFD.upf',  # pwscf PP file
                       'C.BFD.xml'], # qmcpack PP file
    # job parameters
    scfjob          = Job(cores=16), # cores to run scf
    nscfjob         = Job(cores=16), # cores to run non-scf
    optjob          = Job(cores=16), # cores for optimization
    qmcjob          = Job(cores=16), # cores for qmc
    # dft parameters (pwscf)
    functional      = 'lda',         # dft functional
    ecut            =  150 ,         # planewave energy cutoff (Ry)
    conv_thr        =  1e-6,         # scf convergence threshold (Ry)
    mixing_beta     =    .7,         # charge mixing factor
    scf_kgrid       = (8,8,8),       # MP grid of primitive cell
    scf_kshift      = (1,1,1),       #  to converge charge density
    # qmc wavefunction parameters (qmcpack)
    meshfactor      = 1.0,           # bspline grid spacing, larger is finer
    jastrows        = [
        dict(type     = 'J1',        # 1-body
             function = 'bspline',   # bspline jastrow
             size     = 8),          # with 8 knots
        dict(type     = 'J2',        # 2-body
             function = 'bspline',   # bspline jastrow
             size     = 8)           # with 8 knots
        ],
    # opt parameters (qmcpack)
    perform_opt     = True,     # produce optimal jastrows
    block_opt       = False,    # if true, ignores opt and qmc
    skip_submit_opt = False,    # if true, writes input files, does not run opt
    opt_kpoint      = 'L',      # supercell k-point for the optimization
    opt_calcs       = [         # qmcpack input parameters for opt
        loop(max = 4,                       # No. of loop iterations
             qmc = linear(                  # linearized optimization method
                energy               =  0.0, # cost function
                unreweightedvariance =  1.0, #   is all unreweighted variance
                reweightedvariance   =  0.0, #   no energy or r.w. var.
                timestep             =  0.5, # vmc timestep (1/Ha)
                warmupsteps          =  100, # MC steps before data collected
                samples              = 16000,# samples used for cost function
                stepsbetweensamples  =   10, # steps between uncorr. samples
                blocks               =   10, # ignore this
                minwalkers           =   0.1,#  and this
```

```
            bigchange           =  15.0,#  and this
            alloweddifference   =  1e-4 #  and this, for now
            )
        ),
    loop(max = 4,
        qmc = linear(                   # same as above, except
            energy              =  0.5, # cost function
            unreweightedvariance =  0.0, #   is 50/50 energy and r.w. var.
            reweightedvariance  =  0.5,
            timestep            =  0.5,
            warmupsteps         =  100,
            samples             = 64000,# and there are more samples
            stepsbetweensamples =   10,
            blocks              =   10,
            minwalkers          =   0.1,
            bigchange           =  15.0,
            alloweddifference   =  1.0e-4
            )
        )
    ],
# qmc parameters (qmcpack)
block_qmc       = False,    # if true, ignores qmc
skip_submit_qmc = False,    # if true, writes input file, does not run qmc
qmc_calcs       = [         # qmcpack input parameters for qmc
    vmc(                      # vmc parameters
        timestep      = 0.5,  # vmc timestep (1/Ha)
        warmupsteps   = 100,  # No. of MC steps before data is collected
        blocks        = 200,  # No. of data blocks recorded in scalar.dat
        steps         =  10,  # No. of steps per block
        substeps      =   3,  # MC steps taken w/o computing E_local
        samplesperthread = 40 # No. of dmc walkers per thread
        ),
    dmc(                      # dmc parameters
        timestep      = 0.01, # dmc timestep (1/Ha)
        warmupsteps   =  50,  # No. of MC steps before data is collected
        blocks        = 400,  # No. of data blocks recorded in scalar.dat
        steps         =   5,  # No. of steps per block
        nonlocalmoves = True  # use Casula's T-moves
        ),                    #  (retains variational principle for NLPP's)
    ],
# return a list or object containing simulations
return_list = False
)


#the project manager monitors all runs
```

```
pm = ProjectManager()

# give it the simulation objects
pm.add_simulations(qsims.list())

# run all the simulations
pm.run_project()



# print out the total energy
performed_runs = not settings.generate_only and not settings.status_only
if performed_runs:
    # get the qmcpack analyzer object
    # it contains all of the statistically analyzed data from the run
    qa = qsims.qmc.load_analyzer_image()
    # get the local energy from dmc.dat
    le = qa.dmc[1].dmc.LocalEnergy  # dmc series 1, dmc.dat, local energy
    #  print the total energy for the 8 atom system
    print 'The DMC ground state energy for graphene is:'
    print '    {0} +/- {1} Ha'.format(le.mean,le.error)
#end if
```

To run the example, navigate to the example directory and type

```
./graphene_example.py
```

or, alternatively,

```
python ./graphene_example.py
```

You should see output like this (without the added # comments):

```
  Pseudopotentials   # reading pseudopotential files
    reading pp:  ./pseudopotentials/C.BFD.upf
    reading pp:  ./pseudopotentials/C.BFD.xml

Project starting
```

```
checking for file collisions  # ensure created files don't overlap
loading cascade images        # load previous simulation state
  cascade 0 checking in
checking cascade dependencies # ensure sim.'s have needed dep.'s
  all simulation dependencies satisfied

starting runs:                # start submitting jobs
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
poll 0  memory 56.28 MB
  Entering ./runs/graphene_test/scf 0 # scf job
    writing input files  0 scf        # input file written
  Entering ./runs/graphene_test/scf 0
    sending required files  0 scf     # PP files copied
    submitting job  0 scf             # job is in virtual queue
  Entering ./runs/graphene_test/scf 0
    Executing:                        # job executed on workstation
      export OMP_NUM_THREADS=1
      mpirun -np 16 pw.x -input scf.in

poll 1  memory 56.30 MB               # waiting for job to finish
poll 2  memory 56.30 MB
poll 3  memory 56.30 MB
poll 4  memory 56.30 MB
  Entering ./runs/graphene_test/scf 0
    copying results  0 scf            # job is finished, copy results
  Entering ./runs/graphene_test/scf 0
    analyzing  0 scf                  # analyze output data

                                      # now do the same for
                                      # nscf job for Jastrow opt
                                      #   single k-point
                                      # nscf job for VMC/DMC
                                      #   multiple k-points

poll 5  memory 56.31 MB
  Entering ./runs/graphene_test/nscf 1 # nscf dmc
    writing input files  1 nscf
    ...
  Entering ./runs/graphene_test/nscfopt 4 # nscf opt
    writing input files  4 nscf
    ...

                                      # now convert KS orbitals
                                      # from planewave to bspline
```

```
                                    # with pw2qmcpack.x
                                    # for nscf opt & nscf dmc

poll 7  memory 56.32 MB
  Entering ./runs/graphene_test/nscf 2 # convert dmc orbitals
    sending required files  2 p2q
    ...
  Entering ./runs/graphene_test/nscfopt 4 # convert opt orbitals
    copying results  4 nscf
    ...

poll 10  memory 56.32 MB
  Entering ./runs/graphene_test/opt 6 # submit jastrow opt
    writing input files  6 opt        # write input file
  Entering ./runs/graphene_test/opt 6
    sending required files  6 opt     # copy PP files
    submitting job  6 opt             # job is in virtual queue
  Entering ./runs/graphene_test/opt 6
    Executing:                        # run qmcpack
      export OMP_NUM_THREADS=1        # w/ complex arithmetic
      mpirun -np 16 qmcapp_complex opt.in.xml

poll 11  memory 56.32 MB
poll 12  memory 56.32 MB
poll 13  memory 56.32 MB
...
...
...
poll 793  memory 56.32 MB   # qmcpack opt finishes
poll 794  memory 56.32 MB   # nearly an hour later
poll 795  memory 56.32 MB
  Entering ./runs/graphene_test/opt 6
    copying results  6 opt            # copy output files
  Entering ./runs/graphene_test/opt 6
    analyzing  6 opt                  # analyze the results

poll 796  memory 56.41 MB
  Entering ./runs/graphene_test/qmc 3 # submit dmc
    writing input files  3 qmc        # write input file
  Entering ./runs/graphene_test/qmc 3
    sending required files  3 qmc     # copy PP files
    submitting job  3 qmc             # job is in virtual queue
  Entering ./runs/graphene_test/qmc 3
    Executing:                        # run qmcpack
```

```
        export OMP_NUM_THREADS=1
        mpirun -np 16 qmcapp_complex qmc.in.xml

  poll 797  memory 57.31 MB
  poll 798  memory 57.31 MB
  poll 799  memory 57.31 MB
  ...
  ...
  ...
  poll 1041  memory 57.31 MB   # qmcpack dmc finishes
  poll 1042  memory 57.31 MB   # about 15 minutes later
  poll 1043  memory 57.31 MB
    Entering ./runs/graphene_test/qmc 3
      copying results  3 qmc            # copy output files
    Entering ./runs/graphene_test/qmc 3
      analyzing  3 qmc                  # analyze the results

Project finished                        # all jobs are finished

The DMC ground state energy for graphene is:
    -45.824960552 +/- 0.00498990689364 Ha    # one value from
                                             # qmcpack analyzer
```

If successful, you have just performed a start-to-finish QMC calculation. The total energy quoted above probably will not match the one you produce due to different compilation environments and the probabilistic nature of QMC. They should not, however differ by three sigma.

Take some time to inspect the input files generated by Nexus and the output files from PWSCF and QMCPACK. The runs were performed in sub-directories of the `runs` directory. The order of execution of the simulations is roughly `scf`, `nscf`, `nscfopt`, `opt`, then `qmc`.

```
runs
 graphene_test
     nscf
        nscf.in
        nscf.out
     nscfopt
        nscf.in
        nscf.out
     opt
        opt.in.xml
        opt.out
```

```
     qmc
        qmc.in.xml
        qmc.out
     scf
         scf.in
         scf.out
```

The directories above contain all the files generated by the simulations. Often one only wants to save the files with the most important data, which are generally small. These are copied to the `results` directory which mirrors the structure of `runs`.

```
results
 runs
     graphene_test
         nscf
            nscf.in
            nscf.out
         nscfopt
            nscf.in
            nscf.out
         opt
            opt.in.xml
            opt.out
         qmc
            qmc.in.xml
            qmc.out
         scf
             scf.in
             scf.out
```

Although this QMC run was performed at a single k-point, a twist-averaged run could be performed simply by changing `kgrid` in `generate_physical_system` from `(1,1,1)` to `(4,4,1)`, or similar.

Example: C 20 Molecule DMC

The files for this example are found in:

```
/your_download_path/nexus/examples/simple_qmc/c20_example
```

Take a moment to study the "input file" script (`c20_example.py`) and the attendant comments (prefixed with #). The relevant differences from the graphene example mostly involve how the structure is procured (it is read from an XYZ file rather than being generated), the boundary conditions (open BC's, see `bconds` in the QMCPACK input parameters), and the workflow involved (as opposed to `standard_qmc`, `basic_qmc` does not perform non-self-consistent DFT calculations).

```
#! /usr/bin/env python

from nexus import settings,ProjectManager,Job
from nexus import Structure,PhysicalSystem
from nexus import loop,linear,vmc,dmc
from qmcpack_calculations import basic_qmc


#general settings for Nexus
settings(
    pseudo_dir    = './pseudopotentials',# directory with all pseudopotentials
    sleep         = 3,                   # check on runs every 'sleep' seconds
    generate_only = 0,                   # only make input files
    status_only   = 0,                   # only show status of runs
    machine       = 'node16',            # local machine is 16 core workstation
    )



#generate the C20 physical system
# specify the xyz file
structure_file = 'c20.cage.xyz'
# make an empty structure object
structure = Structure()
# read in the xyz file
structure.read_xyz(structure_file)
# place a bounding box around the structure
structure.bounding_box(
    box   = 'cubic',         # cube shaped cell
```

```
    scale = 1.5              # 50% extra space
    )
# make it a gamma point cell
structure.add_kmesh(
    kgrid      = (1,1,1),    # Monkhorst-Pack grid
    kshift     = (0,0,0)     # and shift
    )
# add electronic information
c20 = PhysicalSystem(
    structure = structure,   # C20 structure
    net_charge = 0,          # net charge in units of e
    net_spin   = 0,          # net spin in units of e-spin
    C          = 4           # C has 4 valence electrons
    )


#generate the simulations for the qmc workflow
qsims = basic_qmc(
    # subdirectory of runs
    directory       = 'c20_test',
    # description of the physical system
    system          = c20,
    pseudos         = ['C.BFD.upf',  # pwscf PP file
                       'C.BFD.xml'], # qmcpack PP file
    # job parameters
    scfjob          = Job(cores=16), # cores to run scf
    optjob          = Job(cores    = 16,        # cores for optimization
                          app_name = 'qmcapp'), # use real-valued qmcpack
    qmcjob          = Job(cores    = 16,        # cores for qmc
                          app_name = 'qmcapp'), # use real-valued qmcpack
    # dft parameters (pwscf)
    functional      = 'lda',         # dft functional
    ecut            =  150 ,         # planewave energy cutoff (Ry)
    conv_thr        =  1e-6,         # scf convergence threshold (Ry)
    mixing_beta     =    .7,         # charge mixing factor
    # qmc wavefunction parameters (qmcpack)
    bconds          = 'nnn',         # open boundary conditions
    meshfactor      = 1.0,           # bspline grid spacing, larger is finer
    jastrows        = [
        dict(type     = 'J1',        # 1-body
             function = 'bspline',   # bspline jastrow
             size     = 8,           # with 8 knots
             rcut     = 6.0),        # and a radial cutoff of 6 bohr
        dict(type     = 'J2',        # 2-body
             function = 'bspline',   # bspline jastrow
             size     = 8,           # with 8 knots
```

```
          rcut     = 8.0),        # and a radial cutoff of 8 bohr
    ],
# opt parameters (qmcpack)
perform_opt     = True,     # produce optimal jastrows
block_opt       = False,    # if true, ignores opt and qmc
skip_submit_opt = False,    # if true, writes input files, does not run opt
opt_calcs       = [         # qmcpack input parameters for opt
    loop(max = 4,                       # No. of loop iterations
         qmc = linear(                  # linearized optimization method
            energy               =  0.0, # cost function
            unreweightedvariance =  1.0, #   is all unreweighted variance
            reweightedvariance   =  0.0, #   no energy or r.w. var.
            timestep             =  0.5, # vmc timestep (1/Ha)
            warmupsteps          =  100, # MC steps before data collected
            samples              = 16000,# samples used for cost function
            stepsbetweensamples  =   10, # steps between uncorr. samples
            blocks               =   10, # ignore this
            minwalkers           =   0.1,#  and this
            bigchange            =  15.0,#  and this
            alloweddifference    =  1e-4 #  and this, for now
            )
         )
    ],
# qmc parameters (qmcpack)
block_qmc       = False,    # if true, ignores qmc
skip_submit_qmc = False,    # if true, writes input file, does not run qmc
qmc_calcs       = [         # qmcpack input parameters for qmc
    vmc(                      # vmc parameters
        timestep      = 0.5,  # vmc timestep (1/Ha)
        warmupsteps   = 100,  # No. of MC steps before data is collected
        blocks        = 200,  # No. of data blocks recorded in scalar.dat
        steps         =  10,  # No. of steps per block
        substeps      =   3,  # MC steps taken w/o computing E_local
        samplesperthread = 40 # No. of dmc walkers per thread
        ),
    dmc(                      # dmc parameters
        timestep      = 0.01, # dmc timestep (1/Ha)
        warmupsteps   =  50,  # No. of MC steps before data is collected
        blocks        = 400,  # No. of data blocks recorded in scalar.dat
        steps         =   5,  # No. of steps per block
        nonlocalmoves = True  # use Casula's T-moves
        ),                    #  (retains variational principle for NLPP's)
    ],
# return a list or object containing simulations
return_list = False
)
```

```
#the project manager monitors all runs
pm = ProjectManager()

# give it the simulation objects
pm.add_simulations(qsims.list())

# run all the simulations
pm.run_project()



# print out the total energy
performed_runs = not settings.generate_only and not settings.status_only
if performed_runs:
    # get the qmcpack analyzer object
    # it contains all of the statistically analyzed data from the run
    qa = qsims.qmc.load_analyzer_image()
    # get the local energy from dmc.dat
    le = qa.dmc[1].dmc.LocalEnergy  # dmc series 1, dmc.dat, local energy
    #  print the total energy for the 20 atom system
    print 'The DMC ground state energy for C20 is:'
    print '    {0} +/- {1} Ha'.format(le.mean,le.error)
#end if
```

To run the example, navigate to the example directory and type

```
./c20_example.py
```

or, alternatively,

```
python ./c20_example.py
```

You should see output like this (without the added # comments):

```
  Pseudopotentials   # reading pseudopotential files
    reading pp:  ./pseudopotentials/C.BFD.upf
    reading pp:  ./pseudopotentials/C.BFD.xml

Project starting
  checking for file collisions  # ensure created files don't overlap
  loading cascade images        # load previous simulation state
    cascade 0 checking in
  checking cascade dependencies # ensure sim.'s have needed dep.'s
    all simulation dependencies satisfied

  starting runs:                # start submitting jobs
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  poll 0  memory 56.21 MB
    Entering ./runs/c20_test/scf 0  # scf job
      writing input files  0 scf    # input file written
    Entering ./runs/c20_test/scf 0
      sending required files  0 scf # PP files copied
      submitting job  0 scf         # job is in the virtual queue
    Entering ./runs/c20_test/scf 0
      Executing:                    # job executed on workstation
        export OMP_NUM_THREADS=1
        mpirun -np 16 pw.x -input scf.in

  poll 1  memory 56.23 MB           # waiting for job to finish
  poll 2  memory 56.23 MB
  poll 3  memory 56.23 MB
  poll 4  memory 56.23 MB
  poll 5  memory 56.23 MB
  poll 6  memory 56.23 MB
  poll 7  memory 56.23 MB
  poll 8  memory 56.23 MB
    Entering ./runs/c20_test/scf 0
      copying results  0 scf        # job is finished, copy results
    Entering ./runs/c20_test/scf 0
      analyzing  0 scf              # analyze output data

  poll 9  memory 56.23 MB           # now convert KS orbitals
    Entering ./runs/c20_test/scf 1  # from planewave to bspline
      writing input files  1 p2q    # with pw2qmcpack.x
      ...

  poll 12  memory 56.23 MB
    Entering ./runs/c20_test/opt 3  # submit jastrow opt
```

```
    writing input files  3 opt    # write input file
  Entering ./runs/c20_test/opt 3
    sending required files  3 opt # copy PP files
    submitting job  3 opt         # job is in virtual queue
  Entering ./runs/c20_test/opt 3
    Executing:                    # run qmcpack
      export OMP_NUM_THREADS=1    # w/ real arithmetic
      mpirun -np 16 qmcapp opt.in.xml

poll 13  memory 56.24 MB
poll 14  memory 56.24 MB
poll 15  memory 56.24 MB
...
...
...
poll 204  memory 56.24 MB   # qmcpack opt finishes
poll 205  memory 56.24 MB   # about 10 minutes later
poll 206  memory 56.24 MB
  Entering ./runs/c20_test/opt 3
    copying results  3 opt        # copy output files
  Entering ./runs/c20_test/opt 3
    analyzing  3 opt              # analyze the results

poll 207  memory 56.27 MB
  Entering ./runs/c20_test/qmc 2  # submit dmc
    writing input files  2 qmc    # write input file
  Entering ./runs/c20_test/qmc 2
    sending required files  2 qmc # copy PP files
    submitting job  2 qmc         # job is in virtual queue
  Entering ./runs/c20_test/qmc 2
    Executing:                    # run qmcpack
      export OMP_NUM_THREADS=1
      mpirun -np 16 qmcapp qmc.in.xml

poll 208  memory 56.49 MB
poll 209  memory 56.49 MB
poll 210  memory 56.49 MB
...
...
...
poll 598  memory 56.49 MB   # qmcpack dmc finishes
poll 599  memory 56.49 MB   # about 20 minutes later
poll 600  memory 56.49 MB
  Entering ./runs/c20_test/qmc 2
```

```
      copying results  2 qmc        # copy output files
    Entering ./runs/c20_test/qmc 2
      analyzing  2 qmc              # analyze the results

Project finished                    # all jobs are finished

The DMC ground state energy for C20 is:
    -112.890695404 +/- 0.0151688786226 Ha  # one value from
                                           # qmcpack analyzer
```

Again, the total energy quoted above probably will not match the one you produce due to different compilation environments and the probabilistic nature of QMC. The results should still be statistically comparable.

The directory trees generated by Nexus for C 20 have a similar structure to the graphene example. Note the absence of the `nscf` runs. The order of execution of the simulations is `scf`, `opt`, then `qmc`.

```
runs
 c20_test
     opt
        opt.in.xml
        opt.out
     qmc
        qmc.in.xml
        qmc.out
     scf
         scf.in
         scf.out
results
 runs
     c20_test
         opt
            opt.in.xml
            opt.out
         qmc
            qmc.in.xml
            qmc.out
         scf
             scf.in
             scf.out
```

5 Nexus User Reference

Pending.

5.1 Reading what you wrote

5.2 Nexus settings: global state and user-specific information

The first section of a project script is often dedicated to providing information regarding the local machine, the location of various files, and the desired behavior of the `ProjectManager`. This information is communicated to Nexus through the `settings` function. The settings function is available in the `project` module. To make `settings` available in your project script, use the following import statement:

```
from nexus import settings
```

How to use the `settings` function

In most cases, it is sufficient to supply only four pieces of information through the `settings` function: whether to run all jobs or just create the input files, how often to check jobs for completion, the location of pseudopotential files, and a description of the local machine.

```
settings(
    generate_only = True,                # only write input files, do not run
    sleep         = 3,                   # check on jobs every 3 seconds
    pseudo_dir    = './pseudopotentials', # path to PP file collection
    machine       = 'node8'              # local machine is an 8 core workstation
    )
```

A few additional parameters are available in `settings` to control where runs are performed, where output data is gathered, and whether to print job status information.

More detailed information about both local and target machines can be provided, such as allocation account numbers, filesystem structure, and where executables are located.

```
settings(
    generate_only = True,                # only write input files, do not run
    sleep         = 3,                   # check on jobs every 3 seconds
    pseudo_dir    = './pseudopotentials', # path to PP file collection
    machine       = 'node8'              # local machine is an 8 core workstation
    )
```

Accessing settings data

6 QMC Practice in a Nutshell

The aim of this section is to provide a very brief overview of the essential concepts undergirding Quantum Monte Carlo calculations of electronic structure with a particular focus on the key approximations and quantities to converge to achieve high accuracy. The discussion here is not intended to be comprehensive. For deeper perspectives on QMC, please see the review articles listed in the "Recommended Reading" section (7.4).

6.1 VMC and DMC in the abstract

Ground state QMC methods, such as Variational (VMC) and Diffusion (DMC) Monte Carlo, attempt to obtain the ground state energy of a many-body quantum system.

$$E_0 = \frac{\langle \Psi_0 | \hat{H} | \Psi_0 \rangle}{\langle \Psi_0 | \Psi_0 \rangle} \tag{6.1.1}$$

The VMC method obtains an upper bound on the ground state energy (guaranteed by the Variational Principle) by introducing a guess at the ground state wavefunction, known as the trial wavefunction Ψ_T:

$$E_{VMC} = \frac{\langle \Psi_T | \hat{H} | \Psi_T \rangle}{\langle \Psi_T | \Psi_T \rangle} \geq E_0 \tag{6.1.2}$$

The DMC method improves on this variational bound by projecting out component eigenstates of the trial wavefunction lying higher in energy than the ground state. The operator that acts as a projector is the imaginary time, or thermodynamic, density matrix:

$$\begin{aligned} |\Psi_t\rangle &= e^{-t\hat{H}} |\Psi_T\rangle \\ &= e^{-tE_0} \left(|\Psi_0\rangle + \sum_{n>0} e^{-t(E_n - E_0)} |\Psi_n\rangle \right) \\ &\xrightarrow[t\to\infty]{} e^{-tE_0} |\Psi_0\rangle \end{aligned} \tag{6.1.3}$$

The DMC energy approaches the ground state energy from above as the imaginary time becomes large.

$$E_{DMC} = \lim_{t\to\infty} \frac{\langle \Psi_t | \hat{H} | \Psi_t \rangle}{\langle \Psi_t | \Psi_t \rangle} = E_0 \tag{6.1.4}$$

However from the equations above, one can already anticipate that the DMC method will struggle in the face of degeneracy or near-degeneracy.

In principle, the DMC method is exact for the ground state, but further complications arise for systems that are extended, comprised of fermions, or contain heavy nuclei, pseudized or otherwise. Approximations arising from the numerical implementation of the method also require care to keep under control.

6.2 From expectation values to random walks

Evaluating expectation values of a many-body system involves performing high dimensional integrals (the dimensionality is at least the dimensions of the physical space times the number of particles). In VMC, for example, the expectation value of the total energy is represented succinctly as:

$$E_{VMC} = \int dR |\Psi_T|^2 E_L \tag{6.2.1}$$

where E_L is the local energy $E_L = \Psi_T^{-1} \hat{H} \Psi_T$. The other factor in the integral $|\Psi_T|^2$ can clearly be thought of as a probability distribution and can therefore be sampled by Monte Carlo methods (such as the Metropolis algorithm) to evaluate the integral exactly.

The sampling procedure takes the form of random walks. A "walker" is just a set of particle positions, along with a weight, that evolves (or moves) to new positions according to a set of statisical rules. In VMC as few walkers are used as possible to reduce the equilibration time (the number of steps or moves required to lose a memory of the potentially poor starting guess for particle posistions). In DMC, the walker population is a dynamic feature of the calculation and must be large enough to avoid introducing bias in expectation values.

The tradeoff of moving to a the sampling procedure for the integration is that it introduces statistical error into the calculation which diminishes slowly with the number of samples (it falls off like $1/(\#ofsamples)$ by the Central Limit Theorem). The good news for ground state QMC is that this error can be reduced more rapidly through the discovery of better guesses at the detailed nature of the many-body wavefunction.

6.3 Quality orbitals: planewaves, cutoffs, splines, and meshes

Acting on an understanding of perturbation theory, the zeroth order representation of the wavefunction of an interacting system takes the form of a Slater determinant of single particle orbitals. In practice, QMC calculations often obtain a starting guess at these orbitals from Hartree-Fock or Density Functional Theory calculations (which already contain non-perturbative contributions from correlation). An important factor in the generation and use of these orbitals is to ensure that they are described to high accuracy within the parent theory.

For example, when taking orbitals from a planewave DFT calculation, one must take care to converge the planewave energy cutoff to a sufficient level of accuracy (usually far

beyond what is required to obtain an accurate DFT energy). One criterion to use it to converge the kinetic energy of the Kohn-Sham wavefunction with respect to the planewave energy cutoff until it is accurate to the energy scale you care about in your production QMC calcuation. For systems with a small number of valence electrons, a cutoff of around 200 Ry is often sufficient. To obtain the kinetic energy from a PWSCF calculation the `pw2casino.x` post-processing tool can be used. In Nexus one has the option to compute the kinetic energy by setting the `kinetic_E` flag in the `standard_qmc` or `basic_qmc` convenience functions.

For efficiency reasons, QMC codes often use a real-space representation of the wavefunction. It is common to represent the orbitals in terms of B-splines which have control points, or knots, that fall on a regular 3-D mesh. Analogous to the planewave cutoff, the fineness of the B-spline mesh controls the quality of the represented orbitals. To verify that the quality of the orbitals has not been compromised during the conversion process from planewave to B-spline, one often performs a VMC calculation with the B-spline Slater determinant wavefunction to obtain the kinetic energy. This value should agree with the kinetic energy of the planewave representation within the energy scale of interest.

In QMCPACK, the B-spline mesh is controlled with the `meshfactor` keyword. Larger values correspond to finer meshes. A value of 1.0 usually gives a similar quality representation as the original planewave calculation. Control of this parameter is made available in Nexus through the `meshfactor` keyword in the `standard_qmc` or `basic_qmc` convenience functions.

6.4 Quality Jastrows: less variance = more efficient

Taking a further que from perturbation theory, the first order correction to the Slater determinant wavefunction is the Jastrow correlation prefactor.

$$\Psi_T \approx e^{-J}\Psi_{Slater\,Det.} \tag{6.4.1}$$

In a quantum liquid, an appropriate form for the Jastrow factor is:

$$J = \sum_{i<j} u_{ij}(|r_i - r_j|) \tag{6.4.2}$$

This form is often used without modification in electronic structure calculations. Note that the correlation factors u_{ij} can be different for particles of differing species, or, if one of the particles in the pair is classical (such as a heavy atomic nucleus), the local electronic environment varies across the system.

The primary role of the Jastrow factor is to increase the efficiency of the QMC calculation. The variance of the local energy across all samples of the random walk is directly related to the statistical error of the final results:

$$v_{\Psi_T} = \frac{1}{N_{samples}} \sum_{s\in samples} E_L(s)^2 - \left[\frac{1}{N_{samples}} \sum_{s\in samples} E_L(s)\right]^2 \tag{6.4.3}$$

$$\sigma_{error} \approx \sqrt{\frac{v_{\Psi_T}}{N_{samples}}} \tag{6.4.4}$$

The variance of local energy is usually minimized by performing a statistical optimization of the Jastrow factor with QMC.

In addition to selecting a good form for the pair correlation functions u_{ij} (which are represented in QMCPACK as 1-D B-spline functions with a finite cutoff radius), the (iterative) optimization procedure must be performed with a sufficient number of samples to converge all the free parameters. Starting with a small number of samples ($\approx 20,000$) is usually preferable for early iterations, followed by a larger number for later iterations. This larger number is something close to $100,000 \times (\#\ of\ free\ parameters)^2$. For B-spline functions, the number of free parameters is the number of control points, or knots.

The number of samples is controlled with the `samples` keyword in QMCPACK. Control of this parameter is made available in Nexus through the `samples` keyword in the `linear` or `cslinear` convenience functions (Which are often used in conjunction with `standard_qmc` or `basic_qmc`). For a B-spline correlation factor, the number of free parameters/knots is indicated by the `size` keyword in either QMCPACK or Nexus.

6.5 Finite size effects: k-points, supercells, and corrections

For extended systems, finite size errors are a key consideration. In addition to the finite size effects that are typically seen in DFT (k-points related). Correlated, many-body methods such as QMC also must contend with correlation-related finite size effects. Both types of finite-size effects are reduced by simply using larger supercells. The complete elimination of finite size effects using this approach can be prohibitively costly since the finite size error typically falls off like $1/\Omega_C$, where Ω_C is the volume of the supercell. A more sophisticated approach involves a combination of the supercell size, k-point grid, and additional estimated corrections for correlation finite size effects.

Although there is no firm rule on the selection of these three elements, adhering to some general guidelines is usually helpful. For a production calculation of an extended system, the minimum supercell size is around 50 atoms. The size of the supercell k-point grid can then be determined by proxy with a DFT calculation (converge the energy down to the scale of interest). Note that although the cost of a DFT calculation scales linearly with the number of k-points, the cost of the corresponding QMC calculation is hardly increased due to the statistical averaging of the results (the QMC calculation at each separate supercell k-point is simply performed with fewer samples so that the total number of samples remains fixed w.r.t. the number of k-points). Finally, corrections for correlation-related finite size effects are computed during the QMC run and added to the result by hand in post-processing the data.

In Nexus, the supercell size is controlled through the `tiling` parameter in the `generate_physical_system`, `generate_structure`, `Structure`, or `Crystal` convenience functions. Supercells can also be constructed by tiling exising structures through the `tile` member function of `Structure` or `PhysicalSystem` objects. The k-point grid is controlled through the `kgrid` parameter in the `generate_physical_system`, `generate_structure`, `Structure`, or `Crystal` convenience functions. K-point grids can also be added to existing structures through the `add_kmesh` member function of `Structure` or `PhysicalSystem` objects.

6.6 Imaginary time discretization: the DMC timestep

An analytic form for the imaginary time projection operator is not known, but real-space approximations to it can be obtained in the small time limit. With importance sampling included (not covered here), the short-time projector splits into two parts, known as the drift-diffusion and branching factors (shown below in atomic units):

$$\rho(R', R; t) = \langle R'|\hat{\Psi}_T e^{-t\hat{H}} \hat{\Psi}_T{}^{-1}|R\rangle \tag{6.6.1}$$

$$= G_d(R', R; t) G_b(R', R, t) + \mathcal{O}(t^2) \tag{6.6.2}$$

$$G_d(R', R; t) \equiv \exp\left(-\tfrac{1}{2t}\left[R' - R - t\nabla_R \log \Psi_T(R)\right]^2\right) \tag{6.6.3}$$

$$G_b(R', R; t) \equiv \exp\left(\tfrac{1}{2}\left[E_L(R') + E_L(R)\right]\right) \tag{6.6.4}$$

The long-time projector is found as the product of many approximate short-time solutions, which takes the form of a many-body path integral in real space:

$$\rho(R_M, R_0; M\tau) = \int dR_1 dR_{M-1} \ldots \prod_{m=0}^{M-1} \rho(R_{m+1}, R_m; \tau) \tag{6.6.5}$$

The short-time parameter τ is known as the DMC timestep and accurate quantities are obtained only in the limit as τ approaches zero.

Ensuring that the timestep error is sufficiently small usually involves performing many DMC calculations over a range of timesteps (sometimes on a smaller supercell than the production calculation). The largest timestep is chosen that produces a bias smaller than the energy scale of interest. For very high accuracy, one uses the total energy as a function of timestep to extrapolate to the zero time limit.

The DMC timestep is made available in Nexus through the `timestep` parameter of the `dmc` convenience function (which is often used in conjuction with the `standard_qmc`, `basic_qmc`, `generate_qmcpack`, or `Qmcpack` functions).

6.7 Population control bias: safety in numbers

While the drift-diffusion factor $G_d(R', R; \tau)$ can be sampled exactly using Gaussian distributed random numbers (this generates the DMC random walk), the branching factor $G_b(R', R; \tau)$ is handled a different way for efficiency. The product of branching factors over an imaginary time trajectory (random walk) serves as a statistical weight for each walker. The fluctuations in this weight rapidly become quite large as the random walk progresses (because it approaches an infinite product of real numbers). As its name suggests, this weight factor is used to "branch" walkers every few steps. If the weight is small the walker is deleted, but if the weight is large the walker is copied many times ("branched") with each copy carrying a weight close to unity. This is more efficient because more walkers are created (and thus more statistics are gathered) in the high weight regions of phase space that contribute most to the integral.

The branching process in DMC naturally leads to a fluctuating population of walkers. The fluctuations in the walker population, if left to its own dynamics, are unbounded. This means that the walker population can grow very large, or even become zero. To prevent

collapse of the walker population, population control techniques (not covered here) are added to the algorithm. The practical upshot of population control is that it introduces a systematic bias in the DMC results that scales like $1/(\#ofwalkers)$ (Although note that another route to reduce the population control bias is to improve the trial wavefunction, since the fluctuations in the branching weights will become zero for the exact ground state).

For many production calculations, population control bias is not much of an issue because the simulations are performed on supercomputers with thousands of cores per run, and thus tens of thousands of walkers. As a rule of thumb, the walker population should at least number in the thousands. One should occasionally explicitly check the magnitude of the population control bias for the system under study since predictions have been made that it will eventually diverge exponentially with the number of particles in the system.

The DMC walker population can be directly controlled in QMCPACK or Nexus through the `samples` (total walker population) or `samplesperthread` (walkers per OpenMP thread) keywords in the VMC block directly proceeding DMC (`vmc` convenience function in Nexus). If you opt to use the `samples` keyword, check that each thread in the calculation will have at least a few walkers.

6.8 The fixed node/phase approximation: varying the nodes/phase

For every fermionic system, the bosonic ground state lies lower in energy than the fermionic ground state. This means that projection methods like DMC will approach the bosonic ground state exponentially fast in imaginary time if unconstrained (this would show up as an exponentially diverging statistical error). In order to guarantee that the projected wavefunction remains in the space of fermionic functions (and consequently that the projected energy remains an upper bound to the fermionic ground state energy), the projected wavefunction is constrained to share the nodes (if it is real-valued) or the phase (if it is complex-valued) of the trial wavefunction. The fixed node/phase approximation represents one of the two most important approximations for electronic structure calculations (the other is the pseudopotential approximation covered in the next section).

The fixed node/phase error can be reduced, but it cannot be completely eliminated unless the exact nodes/phase is known. A common approach to reduce the fixed node/phase error is to perform several DMC calculations (sometimes on a smaller supercell) with different sets of orbitals (perhaps generated with different functionals). Another, more expensive approach, is to include the backflow transformation (this is the second order correction to the wavefunction; it is not covered in any detail here) to get a lower bound on how large the fixed node error is in standard Slater-Jastrow calculations.

To perform a calculation of this type (scanning over orbitals from different functionals) with Nexus, the DFT functional can be selected with the `functional` keyword in the `standard_qmc` or `basic_qmc` convenience functions. If you are using pseudopotentials generated for use in DFT, you should maintain consistency between the functional and pseudopotential. Even if such consistency is maintained, the impact of using DFT pseudopotentials (or those made with many other theories) in QMC can be significant.

6.9 Pseudopotentials: theoretical dissonance, the locality approximation, and T-moves

The accurate use of pseudopotentials in electronic structure QMC calculations remains one of the largest challenges in current practice. The necessity for pseudopotentials arises from the rapidly increasing computational cost with increasing nuclear charge (it scales like Z^6, compared with the $N^3_{electrons}$ scaling with Z fixed). The challenge in using pseudopotentials in QMC is that practically no pseudopotentials exist that have been generated self-consistently with QMC. In other words, QMC is currently reliant on other theories to provide the pseudopotentials, which can be a critical source of error.

The current state-of-the-art is not without rigor, however. One source of Dirac-Fock based pseudopotentials, the Burkatzki-Filippi-Dolg database (see http://www.burkatzki.com/pseudos/index.2.html), has been explicitly vetted against quantum chemistry calculations of atoms (a higher-fidelity proxy for QMC calculations of small systems). It must be stressed that these pseudopotentials should still be validated for use in a particular target system. Another collection of Dirac-Fock pseudopotentials that have been created for use in QMC can be found in the Trail-Needs database (see http://www.tcm.phy.cam.ac.uk/~mdt26/casino2_pseudopotentials.html). Many current calculations also use the OPIUM package (see http://opium.sourceforge.net/) to generate DFT pseudopotentials and then port them directly to QMC.

Whatever the source of pseudopotentials (but perhaps especially so for those derived from DFT), testing and validation remains an important step preceeding production calculations. One option is to perform parallel pseudopotential and all-electron DMC calculations of atoms with varying electron count (*i.e.* ionization potential/electron affinity calculations). As with any electronic structure calculation, it is also advisable to devise a test in or close to the target host environment. Validating pseudopotentials remains a difficult task, and while the suggestions presented here may be of some help, they do not amount to a panacea for the issue.

Beyond the central approximation of using a pseudopotential at all, two approximations unique to pseudopotential use in DMC merit discussion. The direct use of non-local pseudopotentials in DMC leads to a second sign-problem (akin to the fixed-node issue) in the imaginary time projector. One solution, devised first, is known as the locality approximation. In the locality approximation, the non-local pseudopotential is replaced by a "localized" form: $V_{NLPP} \rightarrow \Psi_T^{-1} V_{NLPP} \Psi_T$. This approximation becomes exact as the trial wavefunction approaches the pseudo ground state, however the Variational Principle of the pseudo-system is lost (though it should be acknowledged that a non-variational portion of the energy has been discarded by using pseudopotentials at all). The Variational Principle for the pseudo-system can be restored with an advanced sampling technique known as T-moves (although the first incarnation of the technique reduces to the locality approximation as the system becomes larger than several atoms, the second version fixes this oversight).

One can select whether to use the locality approximation or T-moves (version 1!) in QMCPACK from within Nexus by setting the parameter `nonlocalmoves` to True or False in the `dmc` convenience function.

6.10 Other approximations: what else is missing?

Though a few points could be selected for mention at this point, only one additional approximation will be highlighted here. In most modern QMC calculations of electronic structure, relativistic effects have been neglected entirely (there have been a few exceptions) or simply assumed to be covered by the pseudopotential. Clearly this will become an issue for systems with large effective core charges. At present, relativistic corrections are not available within QMCPACK.

7 Recommended Reading

The sections below contain information, or at least links to information, that should be helpful for anyone who wants to use Nexus, but who is not an expert in one of the following areas: installing python and related modules, installing PWSCF and QMCPACK, the Python programming language, and the theory and practice of Quantum Monte Carlo.

7.1 Helpful Links for Installing Python Modules

Python itself

Download: http://www.python.org/download/
Be sure to get Python 2.x, not 3.x.

Numpy and Scipy

Download and installation: http://www.scipy.org/Installing_SciPy.

Matplotlib

Download: http://matplotlib.org/downloads.html
Installation: http://matplotlib.org/users/installing.html

H5py

Download and installation: http://www.h5py.org/

7.2 Helpful Links for Installing Electronic Structure Codes

PWSCF: pw.x, pw2qmcpack.x, pw2casino.x

Publicly available version

Download: svn co http://qmctools.googlecode.com/svn/dft/espresso-4.2
See also: http://qmcpack.cmscc.org/

Developer's version:

Download: svn co https://subversion.assembla.com/svn/qmcdev/qe4.3.2
(QE 5.0 is not currently supported in Nexus)

Installation instructions: See section 2 of the User Guide (user_guide.pdf) found in the Doc directory of the distribution.

Wfconvert: wfconvert

Download: svn co http://qmctools.googlecode.com/svn/trunk/wfconvert
See also: http://qmcpack.cmscc.org/

QMCPACK: sqd, qmcapp, qmcapp_complex

Install: http://users.nccs.gov/~jnkim/qmcpack/ug/a00001.html
See also: http://qmcpack.cmscc.org/getting-started

7.3 Brushing Up On Python

Python

Python is a flexible, multi-paradigm, interpreted programming language with powerful intrinsic datatypes and a large library of modules that greatly expand its functionality. A good way to learn the language is through the extensive Documentation provided on the python.org website. If you have never worked with Python before, be sure to go through the Tutorial. To learn more about the intrinsic data types and standard libraries look at Library Reference.

Documentation	http://docs.python.org/2/
Tutorial	http://docs.python.org/2/tutorial/index.html
Library Reference	http://docs.python.org/2/library/index.html

NumPy

Other than the Python Standard Library, the main library/module Nexus makes heavy use of is NumPy. NumPy provides a convenient and fairly fast implementation of multi-dimensional arrays and related functions, much like MATLAB. If you want to learn about NumPy arrays, the NumPy Tutorial is recommended. For more detailed information, see the NumPy User Guide and the NumPy Reference Manual. If MATLAB is one of your native languages, check out NumPy for MATLAB Users.

Tutorial	http://www.scipy.org/Tentative_NumPy_Tutorial
User Guide	http://docs.scipy.org/doc/numpy/user/index.html#user
Reference	http://docs.scipy.org/doc/numpy/reference/
MATLAB	http://www.scipy.org/NumPy_for_Matlab_Users

Matplotlib

Plotting in Nexus is currently handled by Matplotlib. If you want to learn more about plotting with Matplotlib, the Pyplot Tutorial is a good place to start. More detailed information is in the User's Guide. Sometimes Examples provide the fastest way to learn.

Tutorial	http://matplotlib.org/users/pyplot_tutorial.html
User's Guide	http://matplotlib.org/users/index.html
Examples	http://matplotlib.org/examples/

Scipy and H5Py

Nexus also occasionally uses functionality from SciPy and H5Py. Learning more about them is unlikely to help you interact with Nexus. However, they are quite valuable on their own. SciPy provides access to special functions, numerical integration, optimization, interpolation, fourier transforms, eigenvalue solvers, and statistical analysis. To get an overview, try the SciPy Tutorial. More detailed material is found in the Scipy Reference. H5Py provides a NumPy-like interface to HDF5 data files, which QMCPACK creates. To learn more about interacting with HDF5 files through H5Py, try the Quick Start Guide. For more information, see the General Documentation.

SciPy Tutorial	http://docs.scipy.org/doc/scipy/reference/tutorial/index.html
SciPy Reference	http://docs.scipy.org/doc/scipy/reference/
H5Py Quick Guide	http://www.h5py.org/docs/intro/quick.html#quick
H5Py General Docs	http://www.h5py.org/docs/

7.4 Quantum Monte Carlo: Theory and Practice

Currently, review articles may be the best way to get an overview of Quantum Monte Carlo methods and practices. The review article by Foulkes, *et al.* from 2001 remains quite relevant and is lucidly written. Other review articles also provide a broader perspective on QMC, including more recent developments. Another resource that can be useful for newcomers (and needs to be updated) is the QMC Wiki. If you are aware of resources that fill a gap in the information presented here (almost a certainty), please contact the developer at krogeljt@ornl.gov to add your contribution.

QMC Review Articles	
Foulkes, 2001	http://rmp.aps.org/abstract/RMP/v73/i1/p33_1
Bajdich, 2009	http://www.physics.sk/aps/pub.php?y=2009&pub=aps-09-02
Needs, 2010	http://iopscience.iop.org/0953-8984/22/2/023201/
Kolorenc, 2011	http://iopscience.iop.org/0034-4885/74/2/026502/
Online Resources	
QMCWiki	www.qmcwiki.org
QMC Summer School 2012	http://www.mcc.uiuc.edu/summerschool/2012/program.html

Index

basic_qmc, 8

generate_physical_system, 8

load_analyzer_image, 8

run_project, 8

settings, 8
standard_qmc, 8
Structure, 8