Sandia National Laboratories

Exceptional service in the national interest

Molecular Calculations driven by Nexus

Amanda Dumi, aedumi@sandia.gov
QMCPack Summer School 2025

2025-07-10

U.S. DEPARTMENT OF ENERGY NNSA

Goal of this Tutorial

- How to drive the calculations using Nexus
- How to define a trial wave function
- How to optimize Jastrow factors
- Considerations for DMC calculations

Funding: U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, as part of the Computational Materials Sciences Program and Center for Predictive Simulations of Functional Materials

Outline

Beryllium dimer

```
- Using Nexus to drive pyscf and QMCPack
    - define the trial wave function
    - optimize the trial
    - explore parameters for production DMC
```

Oxygen dimer

```
- a fully contained example
```

Additional Considerations

```
- How to get started on your own calculations
- choice of trial wave functions
```

Beryllium Dimer

Our Test Case: Beryllium dimer

Quantum Monte Carlo calculation of the binding energy of the beryllium dimer

- Michael J. Deible, Melody Kessler, Kevin E. Gasperich, Kenneth D. Jordan, J. Chem. Phys. 143, 084116 (2015)

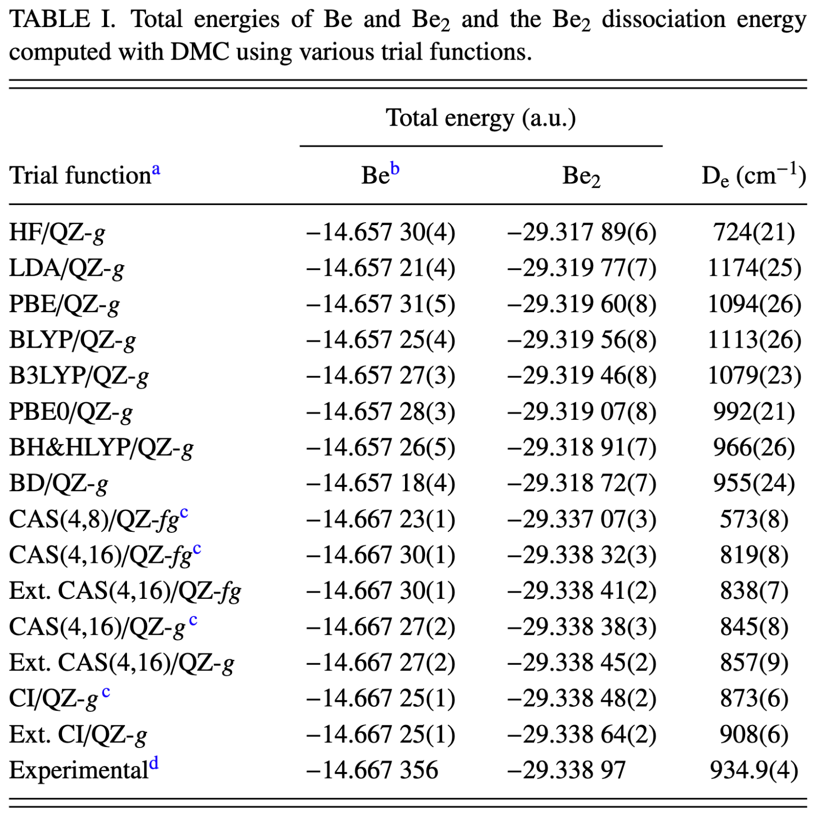
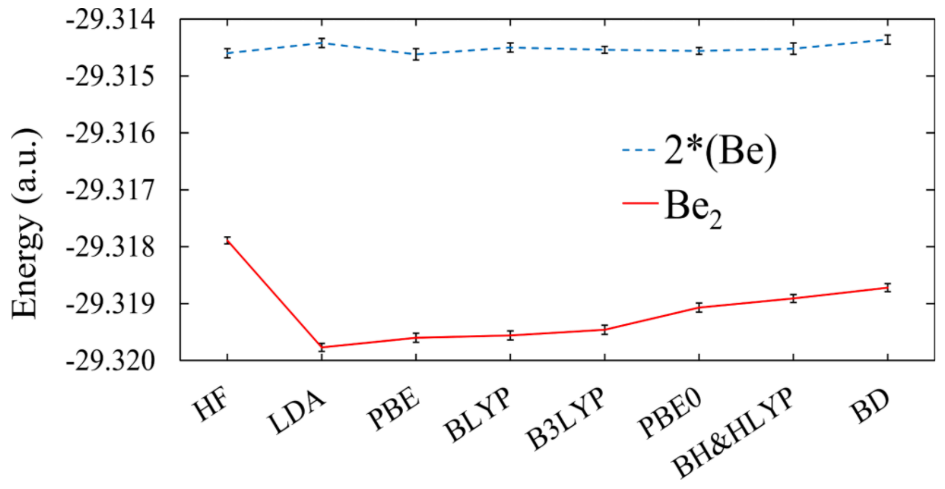


TABLE I. Total energies of Be and Be_2 and the Be_2 dissociation energy computed with DMC using various trial functions.

	Total energy (a.u.)		
Trial function[a]	Be[b]	Be_2	D_e (cm^{-1})
HF/QZ-*g*	−14.657 30(4)	−29.317 89(6)	724(21)
LDA/QZ-*g*	−14.657 21(4)	−29.319 77(7)	1174(25)
PBE/QZ-*g*	−14.657 31(5)	−29.319 60(8)	1094(26)
BLYP/QZ-*g*	−14.657 25(4)	−29.319 56(8)	1113(26)
B3LYP/QZ-*g*	−14.657 27(3)	−29.319 46(8)	1079(23)
PBE0/QZ-*g*	−14.657 28(3)	−29.319 07(8)	992(21)
BH&HLYP/QZ-*g*	−14.657 26(5)	−29.318 91(7)	966(26)
BD/QZ-*g*	−14.657 18(4)	−29.318 72(7)	955(24)
CAS(4,8)/QZ-*fg*[c]	−14.667 23(1)	−29.337 07(3)	573(8)
CAS(4,16)/QZ-*fg*[c]	−14.667 30(1)	−29.338 32(3)	819(8)
Ext. CAS(4,16)/QZ-*fg*	−14.667 30(1)	−29.338 41(2)	838(7)
CAS(4,16)/QZ-*g*[c]	−14.667 27(2)	−29.338 38(3)	845(8)
Ext. CAS(4,16)/QZ-*g*	−14.667 27(2)	−29.338 45(2)	857(9)
CI/QZ-*g*[c]	−14.667 25(1)	−29.338 48(2)	873(6)
Ext. CI/QZ-*g*	−14.667 25(1)	−29.338 64(2)	908(6)
Experimental[d]	−14.667 356	−29.338 97	934.9(4)

DMC energies for various trial wave functions



Pyscf

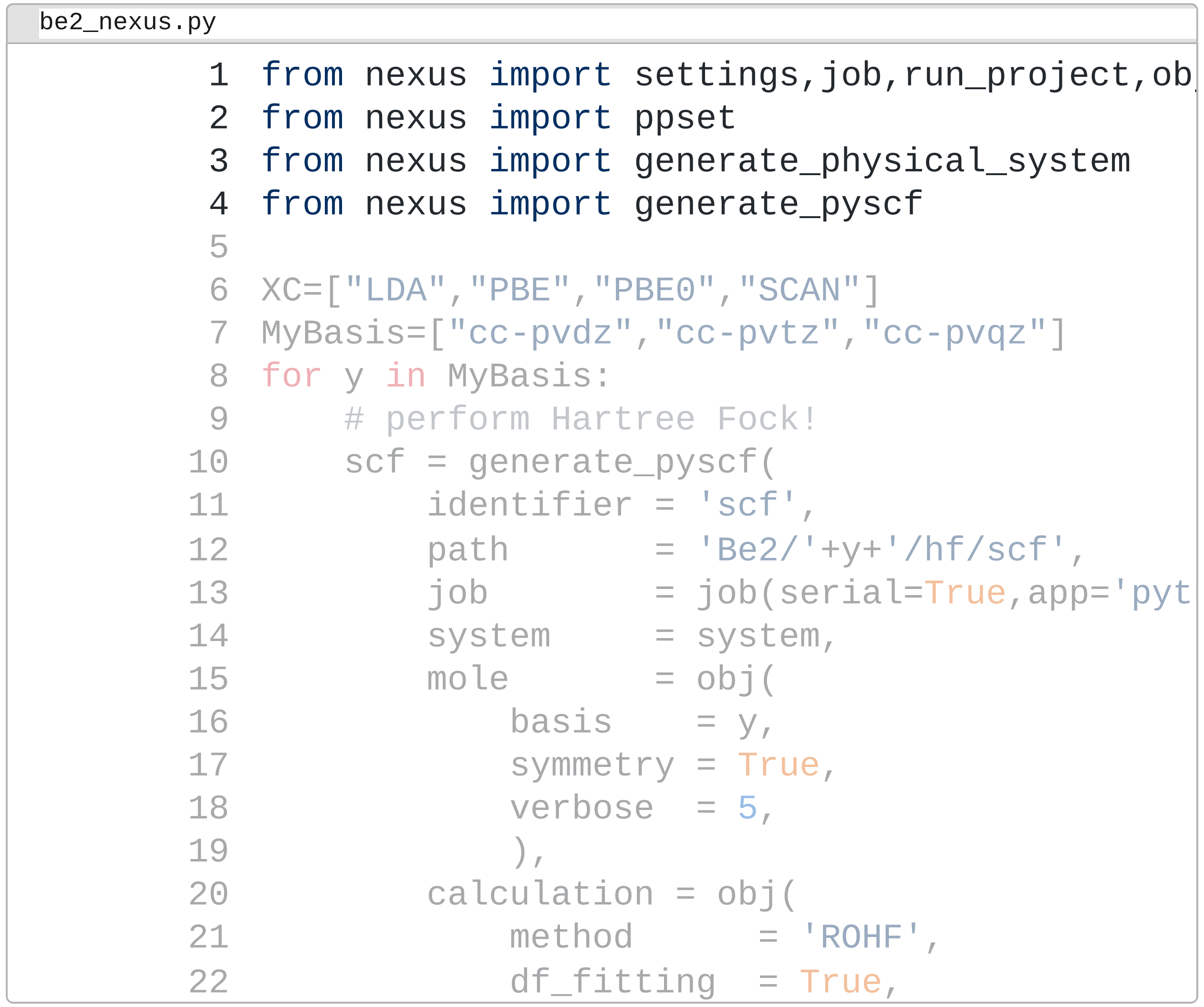
```python
import numpy as np
from pyscf import df, scf, dft

from pyscf import gto as gto_loc
mol = gto_loc.Mole()
mol.verbose  = 1
mol.atom     = '''
               Be   0.00000000   0.00000000   0.00000000
               Be   2.45360300   0.00000000   0.00000000
               '''
mol.basis    = 'cc-pvtz'
mol.unit     = 'A'
mol.charge   = 0
mol.spin     = 0
mol.symmetry = True
mol.build()

mf = scf.ROHF(mol).density_fit()
mf.max_cycle=200
mf.level_shift=0.0
mf.tol          = '1e-10'
e_scf = mf.kernel()                                                    ①
```

① e_scf = -29.133800914375186

Nexus to drive Pyscf

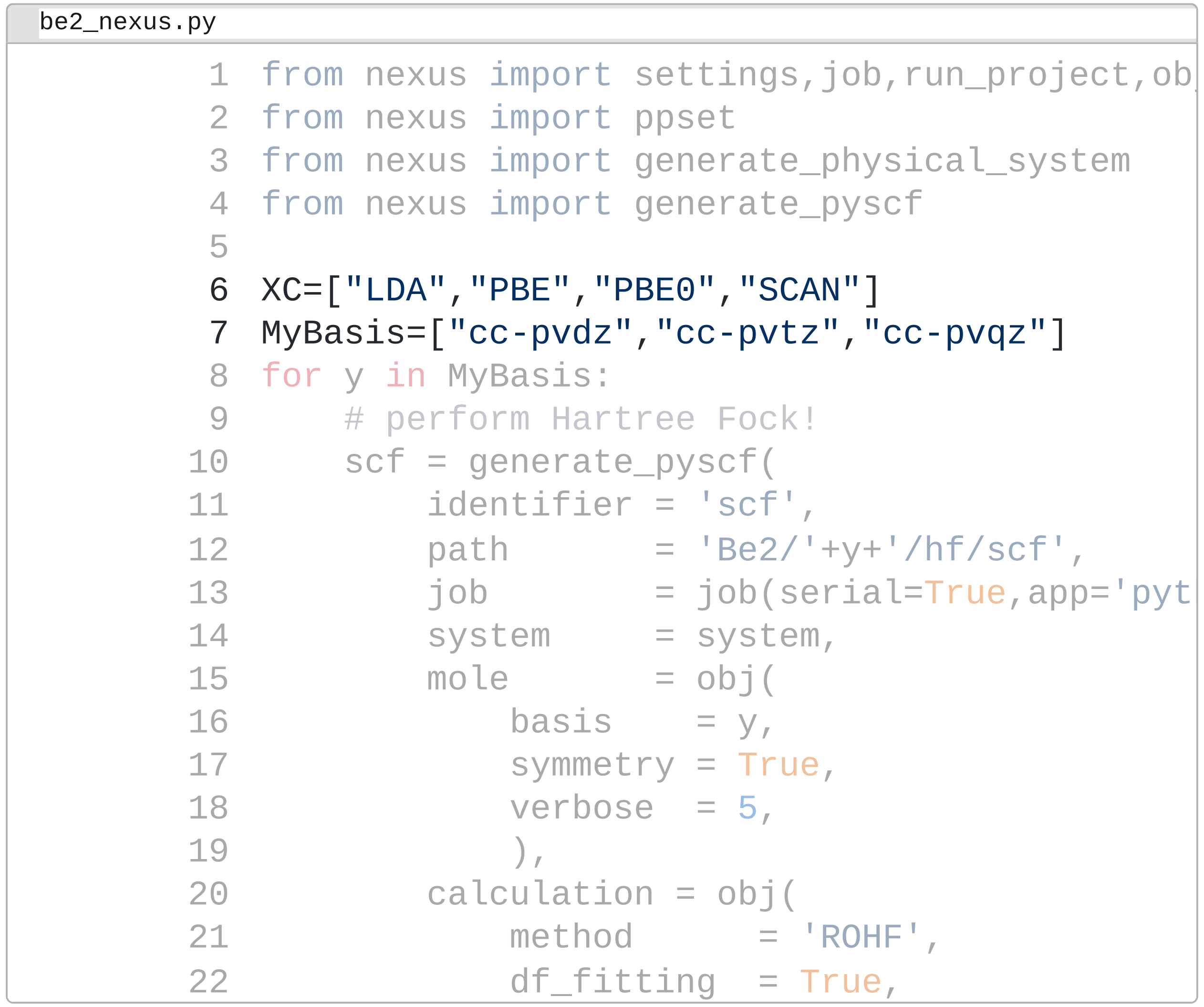


be2_nexus.py

```
from nexus import settings,job,run_project,ob
from nexus import ppset
from nexus import generate_physical_system
from nexus import generate_pyscf

XC=["LDA","PBE","PBE0","SCAN"]
MyBasis=["cc-pvdz","cc-pvtz","cc-pvqz"]
for y in MyBasis:
    # perform Hartree Fock!
    scf = generate_pyscf(
        identifier = 'scf',
        path       = 'Be2/'+y+'/hf/scf',
        job        = job(serial=True,app='pyt
        system     = system,
        mole       = obj(
            basis    = y,
            symmetry = True,
            verbose  = 5,
            ),
        calculation = obj(
            method      = 'ROHF',
            df_fitting  = True,
```

Nexus to drive Pyscf

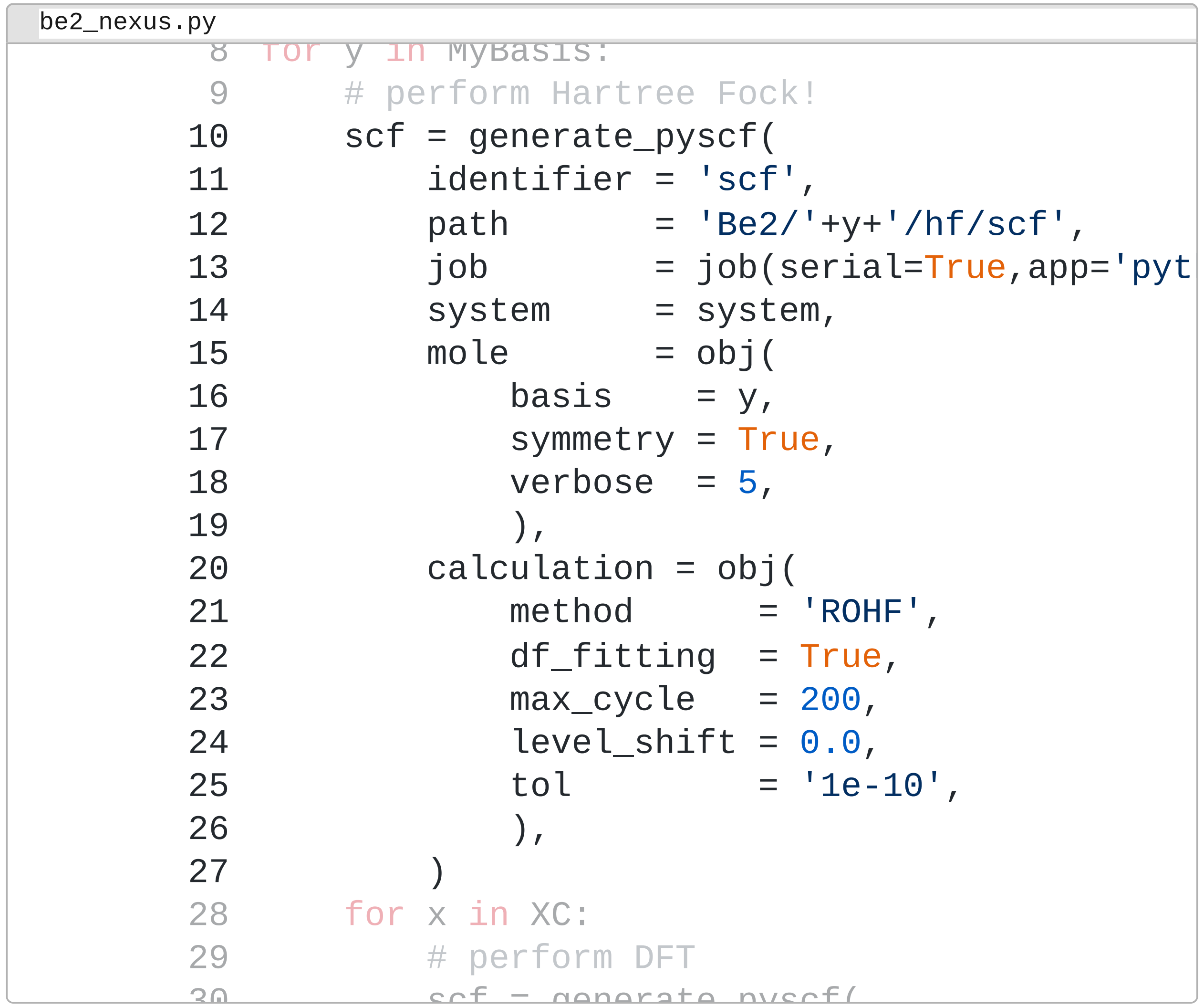


be2_nexus.py

```python
from nexus import settings,job,run_project,ob
from nexus import ppset
from nexus import generate_physical_system
from nexus import generate_pyscf

XC=["LDA","PBE","PBE0","SCAN"]
MyBasis=["cc-pvdz","cc-pvtz","cc-pvqz"]
for y in MyBasis:
    # perform Hartree Fock!
    scf = generate_pyscf(
        identifier = 'scf',
        path       = 'Be2/'+y+'/hf/scf',
        job        = job(serial=True,app='pyt
        system     = system,
        mole       = obj(
            basis    = y,
            symmetry = True,
            verbose  = 5,
            ),
        calculation = obj(
            method      = 'ROHF',
            df_fitting  = True,
```

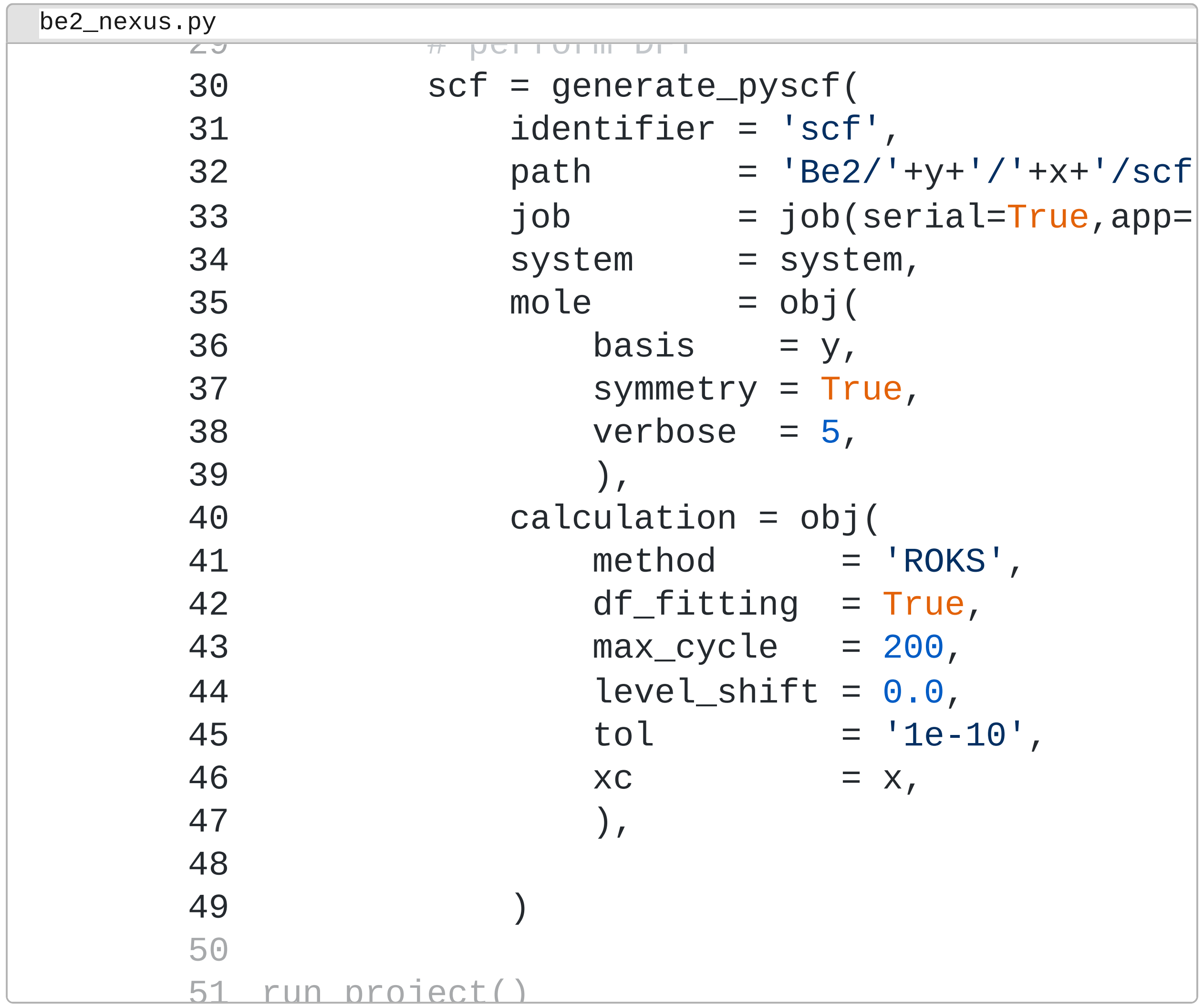
Nexus to drive Pyscf



be2_nexus.py

```
for y in MyBasis:
    # perform Hartree Fock!
    scf = generate_pyscf(
        identifier = 'scf',
        path       = 'Be2/'+y+'/hf/scf',
        job        = job(serial=True,app='pyt
        system     = system,
        mole       = obj(
            basis    = y,
            symmetry = True,
            verbose  = 5,
            ),
        calculation = obj(
            method      = 'ROHF',
            df_fitting  = True,
            max_cycle   = 200,
            level_shift = 0.0,
            tol         = '1e-10',
            ),
        )
    for x in XC:
        # perform DFT
        scf = generate_pyscf(
```

Nexus to drive Pyscf

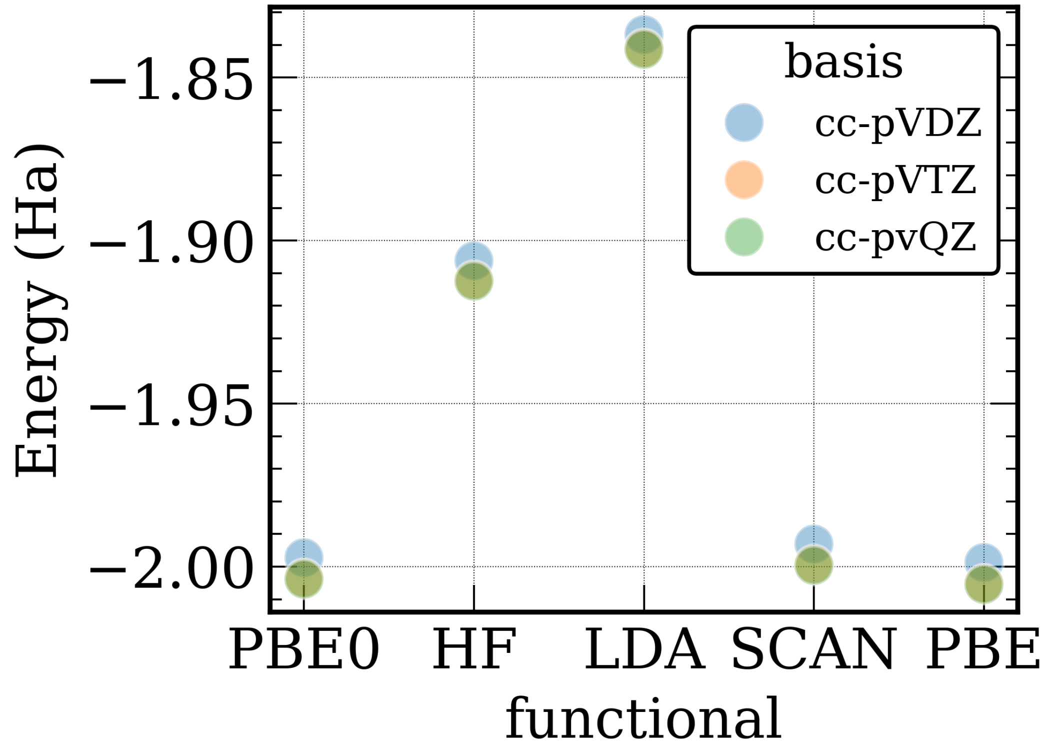

be2_nexus.py

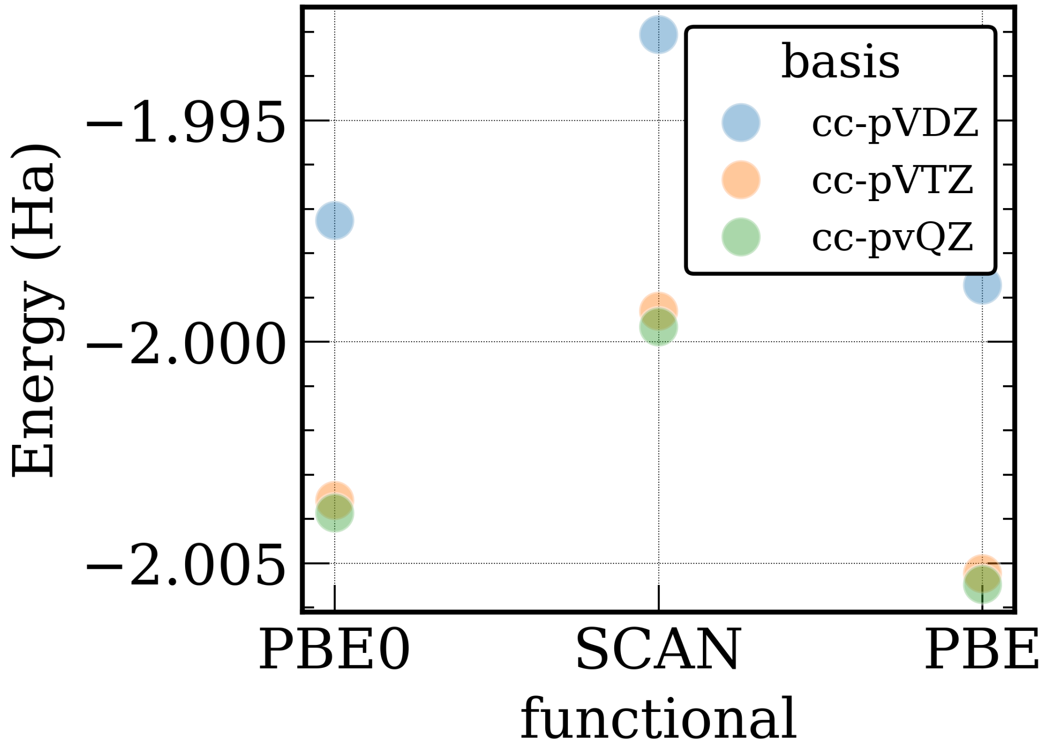
```
        # perform DFT
        scf = generate_pyscf(
            identifier = 'scf',
            path       = 'Be2/'+y+'/'+x+'/scf
            job        = job(serial=True,app=
            system     = system,
            mole       = obj(
                basis    = y,
                symmetry = True,
                verbose  = 5,
                ),
            calculation = obj(
                method      = 'ROKS',
                df_fitting  = True,
                max_cycle   = 200,
                level_shift = 0.0,
                tol         = '1e-10',
                xc          = x,
                ),

            )

run_project()
```

DFT results


basis
cc-pVDZ
cc-pVTZ
cc-pvQZ
Energy (Ha)
−1.85
−1.90
−1.95
−2.00
PBE0
HF
LDA
SCAN
PBE
functional


basis
cc-pVDZ
cc-pVTZ
cc-pvQZ
Energy (Ha)
−1.995
−2.000
−2.005
PBE0
SCAN
PBE
functional

Our wave function

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

1. Antisymmetric portion: Slater determinant
2. Symmetric portion: the Jastrow factor

Our wave function

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

1. Antisymmetric portion: Slater determinant
2. Symmetric portion: the Jastrow factor

$$\Psi_{AS} = \sum_k^M C_k D_k^{\uparrow}(\phi) D_k^{\downarrow}(\phi)$$

$$D(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{r}_1) & \chi_1(\mathbf{r}_2) & \cdots & \chi_1(\mathbf{r}_N) \\ \chi_2(\mathbf{r}_1) & \chi_2(\mathbf{r}_2) & \cdots & \chi_2(\mathbf{r}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_N(\mathbf{r}_1) & \chi_N(\mathbf{r}_2) & \cdots & \chi_N(\mathbf{r}_N) \end{vmatrix}$$

Our wave function

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

1. Antisymmetric portion: Slater determinant
2. Symmetric portion: the Jastrow factor

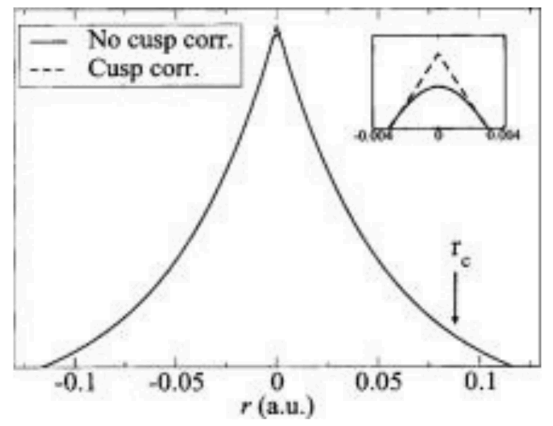
$$\Psi_{AS} = \sum_k^M C_k D_k^{\uparrow}(\phi) D_k^{\downarrow}(\phi)$$

$$D(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{r}_1) & \chi_1(\mathbf{r}_2) & \cdots & \chi_1(\mathbf{r}_N) \\ \chi_2(\mathbf{r}_1) & \chi_2(\mathbf{r}_2) & \cdots & \chi_2(\mathbf{r}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_N(\mathbf{r}_1) & \chi_N(\mathbf{r}_2) & \cdots & \chi_N(\mathbf{r}_N) \end{vmatrix}$$

Linear Combination of Atomic Orbitals

$$\chi(\mathbf{r}) = \sum_{i=1}^N c_i \phi_i(\mathbf{r})$$

For all electron calculations, employ a correction for electron nuclear cusp



QMC Workflow Overview

1. Wave function definition for QMCPack

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

2. Wave function optimization
3. Production parameter choices

Considerations	Files to explore
cusp correction	session4_molecules/01_Be2_dimer/run0_vmc_noj.py
optimize Ψ_T	session4_molecules/01_Be2_dimer/run1_qmc_wfopt.py
timestep error	session4_molecules/01_Be2_dimer/run2_dmc_timestep.py
error bar control	session4_molecules/01_Be2_dimer/run3_dmc_errorbars.py
population bias	session4_molecules/01_Be2_dimer/run4_dmc_population.py
production run	session4_molecules/01_Be2_dimer/run5_dmc_production.py

Calculating the antisymmetric portion of trial

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

run0_vmc_noJ.py

```
scf = generate_pyscf(
    identifier = 'scf',             # log ou
    ...
    calculation = obj(
        method      = 'ROKS',      # Restric
        df_fitting  = True,        # Density
        ...
        xc          = 'pbe',      # Exchange
        ),
    save_qmc    = True ,           # Save the
    )
```

Run a restricted DFT calculation using PBE functional.

`save_qmc` option outputs orbitals in a usable way

Calculating the antisymmetric portion of trial

$$\Psi_T(R) = J(R)\Psi_{AS}(R)$$

run0_vmc_noJ.py

```
scf = generate_pyscf(
    identifier = 'scf',              # log out
    ...
    calculation = obj(
        method      = 'ROKS',       # Restrict
        df_fitting  = True,         # Density
        ...
        xc          = 'pbe',       # Exchange
        ),
    save_qmc   = True ,             # Save the
    )
```

```
# convert orbitals to QMCPACK format
c4q = generate_convert4qmc(
  identifier   = 'c4q',
  path         = 'Be2/'+y+'/'+x+'/SCF',
  job          = job(cores=1),
  dependencies = (scf,'orbitals'),
  )
```

Run a restricted DFT calculation using PBE functional.

`save_qmc` option outputs orbitals in a usable way

use `convert4qmc` to generate input files for QMCPack

- supports a number of codes

Results from convert4qmc

output files: c4q.orbs.h5, c4q.qmc.in-wfj.xml, c4q.structure.xml, c4q.wfj.xml

c4q.qmc.in-wfj.xml

```xml
<!--Example QMCPACK input file produced by convert4qmc
-->
  <!--Name and Series number of the project.-->
  <project id="c4q" series="0"/>
  <!--Link to the location of the Atomic Coordinates and the location of the Wavefunction.-->
  <include href="c4q.structure.xml"/>
  <include href="c4q.wfj.xml"/>
  <!--Hamiltonian of the system.-->
  <hamiltonian name="h0" type="generic" target="e">
    <pairpot name="ElecElec" type="coulomb" source="e" target="e" physical="true"/>
    <pairpot name="IonIon" type="coulomb" source="ion0" target="ion0"/>
    <pairpot name="IonElec" type="coulomb" source="ion0" target="e"/>
  </hamiltonian>
  <!--Example initial VMC to measure initial energy and variance -->
  <qmc method="vmc" move="pbyp" checkpoint="-1">
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="warmupSteps">100</parameter>
    <parameter name="blocks">20</parameter>
    <parameter name="steps">50</parameter>
    <parameter name="substeps">8</parameter>
    <parameter name="timestep">0.5</parameter>
    <parameter name="usedrift">no</parameter>
```

Adding symmetric portion of wave function

$$\Psi_T(R) \; = \; J(R)\Psi_{AS}(R)$$

$$J(R) = e^{J_1 + J_2 + \ldots}$$

Adding symmetric portion of wave function

$$\Psi_T(R) \;=\; J(R)\Psi_{AS}(R)$$

$$J(R) = e^{J_1+J_2+\ldots}$$

$$J_1 = \int_I \sum_i^e u_{ab}(|r_i - R_I|)$$

$$J_2 = \sum_i^e \sum_{j<i}^e u_{ab}(|r_i - r_j|)$$

- We will optimize the J_1 and J_2

```
# optimize 2-body Jastrow
optJ2 = generate_qmcpack(
  identifier        = 'opt',
  path              = 'Be2/'+y+'/'+x+'/optJ2
  job               = job(cores=cores),
  system            = system,
  J2                = True,          # 2-body
  J1_rcut           = 6.0,           # 6 Bohr
  J2_rcut           = 8.0,           # 8 Bohr
  seed              = 42,            # Fix the
  qmc               = 'opt',         # Wavefu
  minmethod         = 'oneshift',    # Energy
  init_cycles       = 4,             # 4 itera
  cycles            = 8,             # 8 produ
  warmupsteps       = 10,
  blocks            = 20,
  steps             = 3,
  timestep          = 0.1,
  init_minwalkers   = 0.1,
  minwalkers        = 0.5,
  samples           = 25600,         # VMC sam
  dependencies      = orbdeps,
```

Adding symmetric portion of wave function

$$\Psi_T(R) \;=\; J(R)\Psi_{AS}(R)$$

$$J(R) = e^{J_1+J_2+\ldots}$$

$$J_1 = \int_I \sum_i^e u_{ab}(|r_i - R_I|)$$

$$J_2 = \sum_i^e \sum_{j<i}^e u_{ab}(|r_i - r_j|)$$

- We will optimize the J_1 and J_2

```
# optimize 2-body Jastrow
optJ2 = generate_qmcpack(
  identifier        = 'opt',
  path              = 'Be2/'+y+'/'+x+'/optJ2
  job               = job(cores=cores),
  system            = system,
  J2                = True,         # 2-body
  J1_rcut           = 6.0,          # 6 Bohr
  J2_rcut           = 8.0,          # 8 Bohr
  seed              = 42,           # Fix the
  qmc               = 'opt',        # Wavefu
  minmethod         = 'oneshift',   # Energy
  init_cycles       = 4,            # 4 itera
  cycles            = 8,            # 8 produ
  warmupsteps       = 10,
  blocks            = 20,
  steps             = 3,
  timestep          = 0.1,
  init_minwalkers   = 0.1,
  minwalkers        = 0.5,
  samples           = 25600,        # VMC sa
  dependencies      = orbdeps,
```

Adding symmetric portion of wave function

$$\Psi_T(R) \ = \ J(R)\Psi_{AS}(R)$$

$$J(R) = e^{J_1+J_2+\ldots}$$

$$J_1 = \int_I \sum_i^e u_{ab}(|r_i - R_I|)$$

$$J_2 = \sum_i^e \sum_{j<i}^e u_{ab}(|r_i - r_j|)$$

- We will optimize the J_1 and J_2

```
optJ2 = generate_qmcpack(
  identifier         = 'opt',
  path               = 'Be2/'+y+'/'+x+'/optJ2
  job                = job(cores=cores),
  system             = system,
  J2                 = True,              # 2-body
  J1_rcut            = 6.0,               # 6 Bohr
  J2_rcut            = 8.0,               # 8 Bohr
  seed               = 42,                # Fix the
  qmc                = 'opt',             # Wavefu
  minmethod          = 'oneshift',        # Energy
  init_cycles        = 4,                 # 4 itera
  cycles             = 8,                 # 8 produ
  warmupsteps        = 10,
  blocks             = 20,
  steps              = 3,
  timestep           = 0.1,
  init_minwalkers    = 0.1,
  minwalkers         = 0.5,
  samples            = 25600,             # VMC sa
  dependencies       = orbdeps,
  )
```

Adding symmetric portion of wave function

The three-body Jastrow form:

$$u_{\sigma\sigma' I}(r_{\sigma I}, r_{\sigma' I}, r_{\sigma\sigma'}) = \sum_{\ell=0}^{M_{eI}} \sum_{m=0}^{M_{eI}} \sum_{n=0}^{M_{ee}} \gamma_{\ell m n} r_{\sigma I}^{\ell} r_{\sigma' I}^{m} r_{\sigma\sigma'}^{n}$$
$$\times \left(r_{\sigma I} - \frac{r_c}{2}\right)^3 \Theta\left(r_{\sigma I} - \frac{r_c}{2}\right)$$
$$\times \left(r_{\sigma' I} - \frac{r_c}{2}\right)^3 \Theta\left(r_{\sigma' I} - \frac{r_c}{2}\right)$$

- correlation is only a function of the interparticle distances
- correlations are set to zero beyond a distance
- J3 will impact J1 and J2, so optimizing them all together is important

```
# optimize 3-body Jastrow
optJ3 = generate_qmcpack(
    identifier         = 'opt',
    path               = 'Be2/'+y+'/'+x+'/optJ3
    job                = job(cores=cores),
    system             = system,
    J3                 = True,             # 3-body
    seed               = 42,               # Fix the
    qmc                = 'opt',            # Wavefu
    minmethod          = 'oneshift',       # Energy
    init_cycles        = 4,                # 4 itera
    cycles             = 8,                # 8 produ
    warmupsteps        = 10,
    blocks             = 20,
    steps              =  5,
    timestep           = 0.1,
    init_minwalkers    = 0.1,
    minwalkers         = 0.5,
    samples            = 25600,            # VMC sa
    dependencies       = orbdeps+[(optJ2,'jastr
    )
```

Adding symmetric portion of wave function

The three-body Jastrow form:

$$u_{\sigma\sigma' I}(r_{\sigma I}, r_{\sigma' I}, r_{\sigma\sigma'}) = \sum_{\ell=0}^{M_{eI}} \sum_{m=0}^{M_{eI}} \sum_{n=0}^{M_{ee}} \gamma_{\ell mn} r_{\sigma I}^{\ell} r_{\sigma' I}^{m} r_{\sigma\sigma'}^{n} \\ \times \left(r_{\sigma I} - \frac{r_c}{2}\right)^3 \Theta\left(r_{\sigma I} - \frac{r_c}{2}\right) \\ \times \left(r_{\sigma' I} - \frac{r_c}{2}\right)^3 \Theta\left(r_{\sigma' I} - \frac{r_c}{2}\right)$$

- correlation is only a function of the interparticle distances
- correlations are set to zero beyond a distance
- J3 will impact J1 and J2, so optimizing them all together is important

```
# optimize 3-body Jastrow
optJ3 = generate_qmcpack(
  identifier         = 'opt',
  path               = 'Be2/'+y+'/'+x+'/optJ3
  job                = job(cores=cores),
  system             = system,
  J3                 = True,         # 3-body
  seed               = 42,           # Fix the
  qmc                = 'opt',        # Wavefu
  minmethod          = 'oneshift',   # Energy
  init_cycles        = 4,            # 4 itera
  cycles             = 8,            # 8 produ
  warmupsteps        = 10,
  blocks             = 20,
  steps              =  5,
  timestep           = 0.1,
  init_minwalkers    = 0.1,
  minwalkers         = 0.5,
  samples            = 25600,        # VMC sar
  dependencies       = orbdeps+[(optJ2,'jastro
  )
```

Adding symmetric portion of wave function

The three-body Jastrow form:

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

- correlation is only a function of the interparticle distances
- correlations are set to zero beyond a distance
- J3 will impact J1 and J2, so optimizing them all together is important

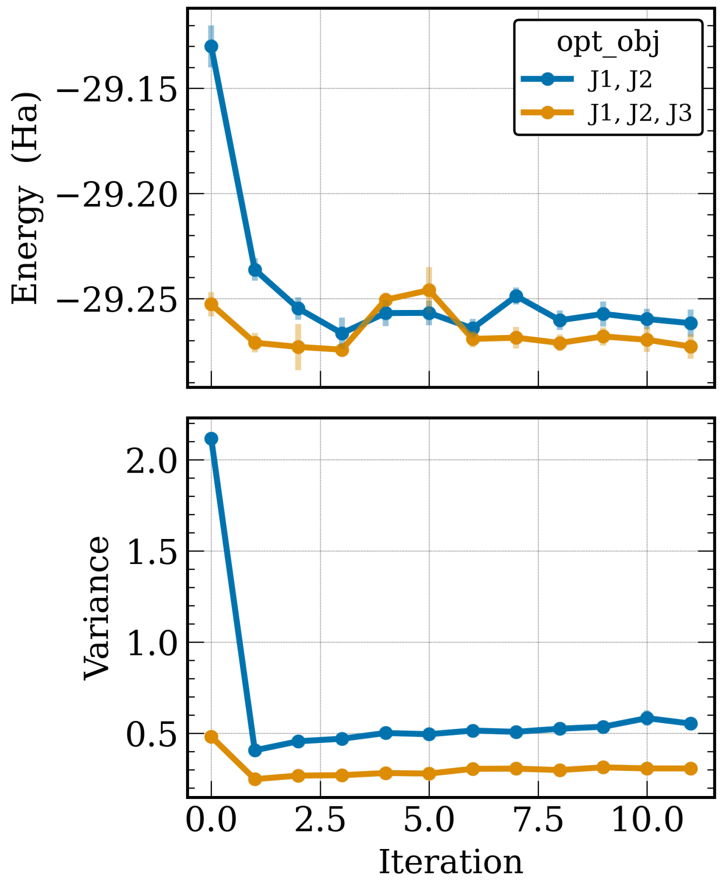
```
# optimize 3-body Jastrow
optJ3 = generate_qmcpack(
  identifier         = 'opt',
  path               = 'Be2/'+y+'/'+x+'/optJ3
  job                = job(cores=cores),
  system             = system,
  J3                 = True,          # 3-body
  seed               = 42,            # Fix the
  qmc                = 'opt',         # Wavefu
  minmethod          = 'oneshift',    # Energy
  init_cycles        = 4,             # 4 itera
  cycles             = 8,             # 8 produ
  warmupsteps        = 10,
  blocks             = 20,
  steps              =  5,
  timestep           = 0.1,
  init_minwalkers    = 0.1,
  minwalkers         = 0.5,
  samples            = 25600,         # VMC sa
  dependencies       = orbdeps+[(optJ2,'jastr
  )
```

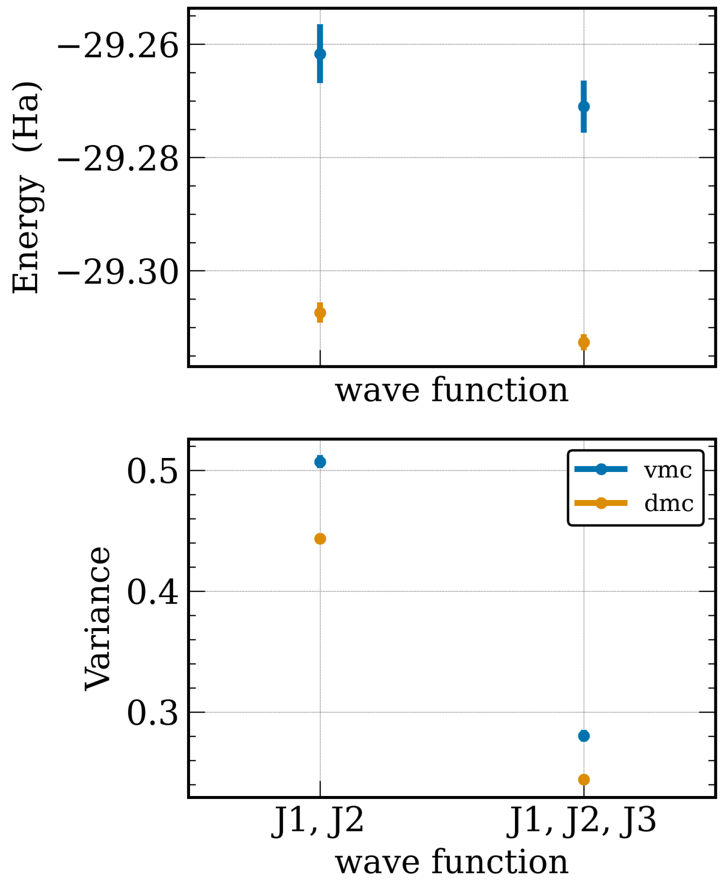
Analyzing a wave fuction optimization run

```
$ qmca -qev *.scalar.dat

                   LocalEnergy               Variance           ratio
opt  series 0  -29.130060 +/- 0.010435   2.117672 +/- 0.018035   0.0727
opt  series 1  -29.236196 +/- 0.005300   0.406483 +/- 0.006288   0.0139
opt  series 2  -29.254653 +/- 0.005361   0.456244 +/- 0.004171   0.0156
opt  series 3  -29.266499 +/- 0.007434   0.470493 +/- 0.010077   0.0161
opt  series 4  -29.256925 +/- 0.006055   0.501486 +/- 0.005856   0.0171
opt  series 5  -29.256848 +/- 0.005754   0.494502 +/- 0.023800   0.0169
opt  series 6  -29.264160 +/- 0.004457   0.514852 +/- 0.006401   0.0176
opt  series 7  -29.248796 +/- 0.004104   0.508019 +/- 0.005036   0.0174
opt  series 8  -29.260340 +/- 0.004567   0.524970 +/- 0.006886   0.0179
opt  series 9  -29.257258 +/- 0.005917   0.535870 +/- 0.006333   0.0183
opt  series 10  -29.259730 +/- 0.004787   0.583399 +/- 0.041853   0.0199
opt  series 11  -29.261702 +/- 0.006473   0.553353 +/- 0.004991   0.0189
```

Results of wave function optimization





Including three-body terms decreases energy and variance for VMC

Jastrow function improves significantly the variance, but eventually DMC will recover the missing correlation

Diffusion Monte Carlo

Considerations	Files to explore
timestep error	session4_molecules/01_Be2_dimer/run1_dmc_timestep.py
reducing error bars	session4_molecules/01_Be2_dimer/run1_dmc_errorbars.py
population bias	session4_molecules/01_Be2_dimer/run4_dmc_population.py
production run	session4_molecules/01_Be2_dimer/run5_dmc_production.py

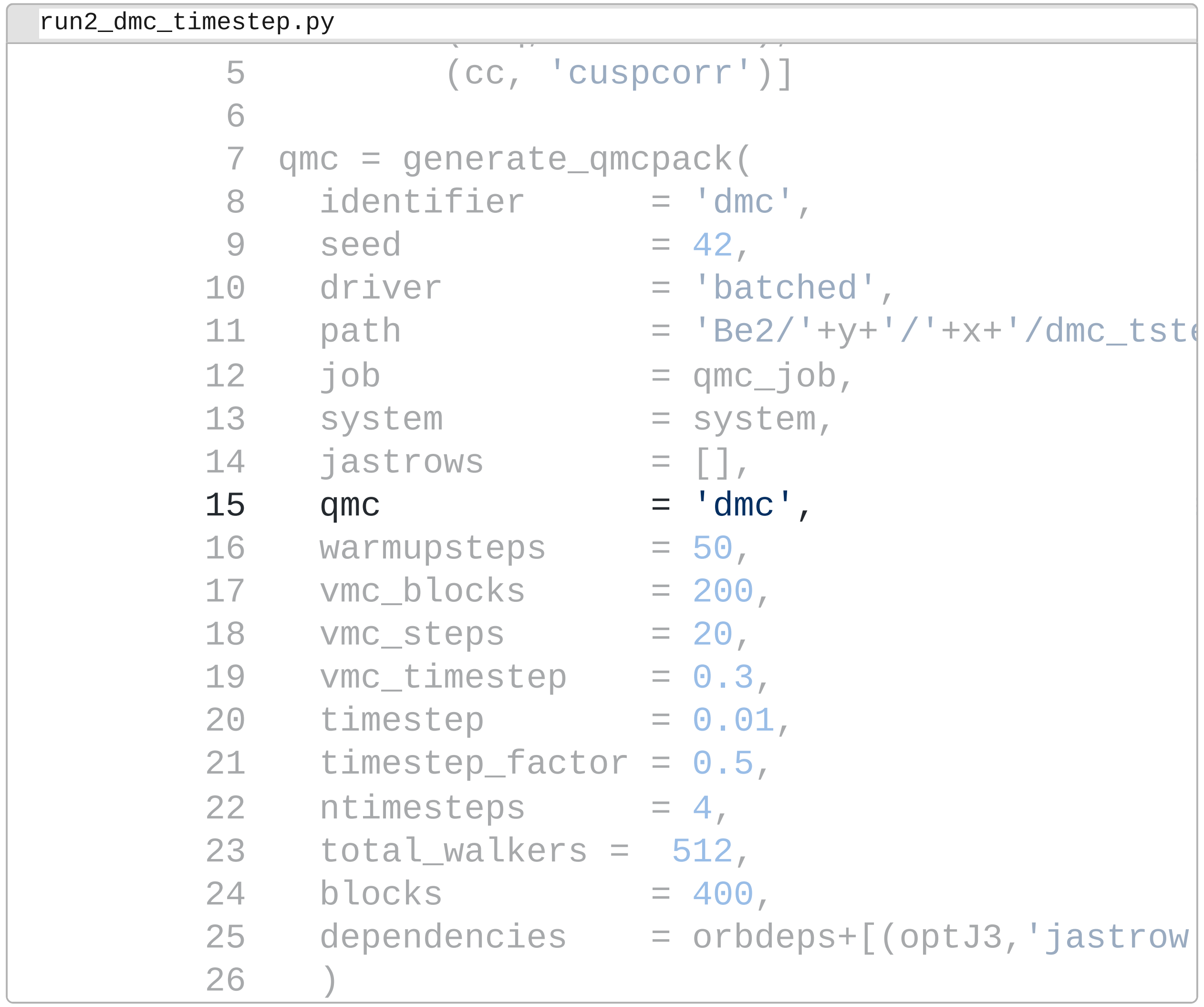
DMC: timestep

run2_dmc_timestep.py

```
x="PBE"
y="cc-pvtz"
orbdeps = [(c4q,'particles'), # pyscf change
           (c4q,'orbitals'),
           (cc, 'cuspcorr')]

qmc = generate_qmcpack(
  identifier      = 'dmc',
  seed            = 42,
  driver          = 'batched',
  path            = 'Be2/'+y+'/'+x+'/dmc_tst
  job             = qmc_job,
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  warmupsteps     = 50,
  vmc_blocks      = 200,
  vmc_steps       = 20,
  vmc_timestep    = 0.3,
  timestep        = 0.01,
  timestep_factor = 0.5,
  ntimesteps      = 4,
```

DMC: timestep



run2_dmc_timestep.py

```
        (cc, 'cuspcorr')]

qmc = generate_qmcpack(
  identifier      = 'dmc',
  seed            = 42,
  driver          = 'batched',
  path            = 'Be2/'+y+'/'+x+'/dmc_tste
  job             = qmc_job,
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  warmupsteps     = 50,
  vmc_blocks      = 200,
  vmc_steps       = 20,
  vmc_timestep    = 0.3,
  timestep        = 0.01,
  timestep_factor = 0.5,
  ntimesteps      = 4,
  total_walkers =  512,
  blocks          = 400,
  dependencies    = orbdeps+[(optJ3,'jastrow
  )
```

DMC: timestep

run2_dmc_timestep.py

```

qmc = generate_qmcpack(
  identifier      = 'dmc',
  seed            = 42,
  driver          = 'batched',
  path            = 'Be2/'+y+'/'+x+'/dmc_tste
  job             = qmc_job,
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  warmupsteps     = 50,
  vmc_blocks      = 200,
  vmc_steps       = 20,
  vmc_timestep    = 0.3,
  timestep        = 0.01,
  timestep_factor = 0.5,
  ntimesteps      = 4,
  total_walkers =  512,
  blocks          = 400,
  dependencies    = orbdeps+[(optJ3,'jastrow
  )

```

DMC: timestep

run2_dmc_timestep.py

```

qmc = generate_qmcpack(
  identifier       = 'dmc',
  seed             = 42,
  driver           = 'batched',
  path             = 'Be2/'+y+'/'+x+'/dmc_tste
  job              = qmc_job,
  system           = system,
  jastrows         = [],
  qmc              = 'dmc',
  warmupsteps      = 50,
  vmc_blocks       = 200,
  vmc_steps        = 20,
  vmc_timestep     = 0.3,
  timestep         = 0.01,
  timestep_factor  = 0.5,
  ntimesteps       = 4,
  total_walkers =  512,
  blocks           = 400,
  dependencies     = orbdeps+[(optJ3,'jastrow
  )

```

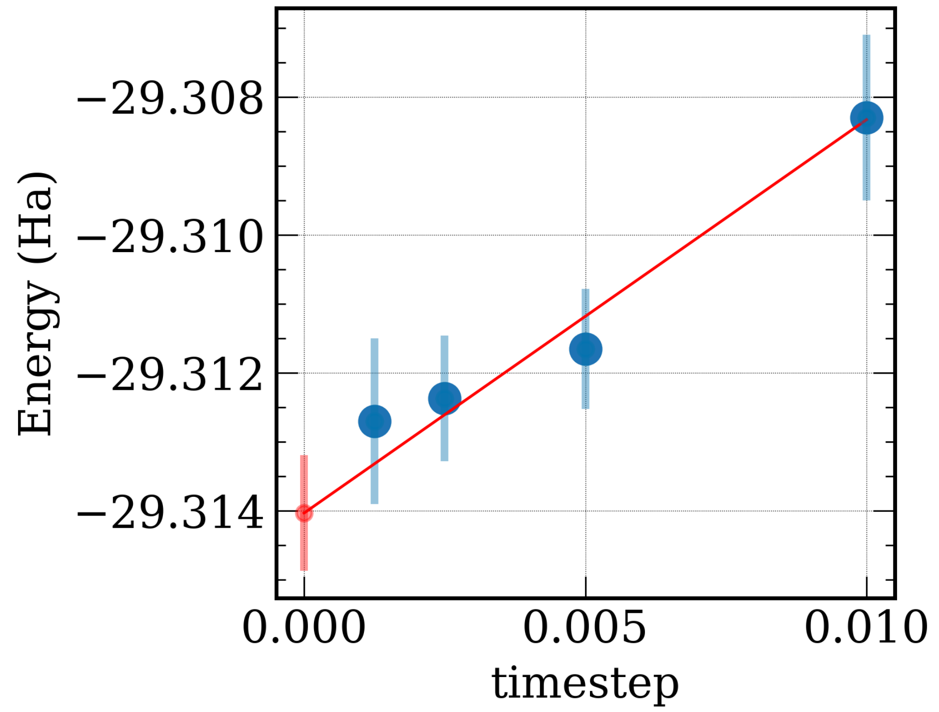
DMC: timestep

run2_dmc_timestep.py

```
x="PBE"
y="cc-pvtz"
orbdeps = [(c4q,'particles'), # pyscf changes
           (c4q,'orbitals'),
           (cc, 'cuspcorr')]

qmc = generate_qmcpack(
  identifier      = 'dmc',
  seed            = 42,
  driver          = 'batched',
  path            = 'Be2/'+y+'/'+x+'/dmc_tste
  job             = qmc_job,
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  warmupsteps     = 50,
  vmc_blocks      = 200,
  vmc_steps       = 20,
  vmc_timestep    = 0.3,
  timestep        = 0.01,
  timestep_factor = 0.5,
  ntimesteps      = 4,
```

```
qmc-fit ts -e 20 '2 4 4' -t '0.01 0.005 0.0025'
*s00{1,2,3}.scalar.dat
```



DMC: reducing error bars

To reduce the error bar by a factor N, multiply the population or the number of blocks by N^2

run3_dmc_errorbars.py

```python
init_blocks=100
for i in range(1,6):
 myblocks=init_blocks*i*i
 qmc = generate_qmcpack(
  identifier      = 'dmc_error'+str(i),
  seed            = 42,
  path            = 'Be2/'+y+'/'+x+'/dmc_erro
  job             = job(cores=cores),
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  vmc_samples     = 1024,
  warmupsteps     = 50,
  vmc_blocks      = 100,
  vmc_steps       = 10,
  vmc_timestep    = 0.1,
  timestep        = 0.00250,
  steps           = 80,
  blocks          = myblocks,
  dependencies    = orbdeps+[(optJ3,'jastrow
  )
```

DMC: reducing error bars

To reduce the error bar by a factor N, multiply the population or the number of blocks by N^2

run3_dmc_errorbars.py

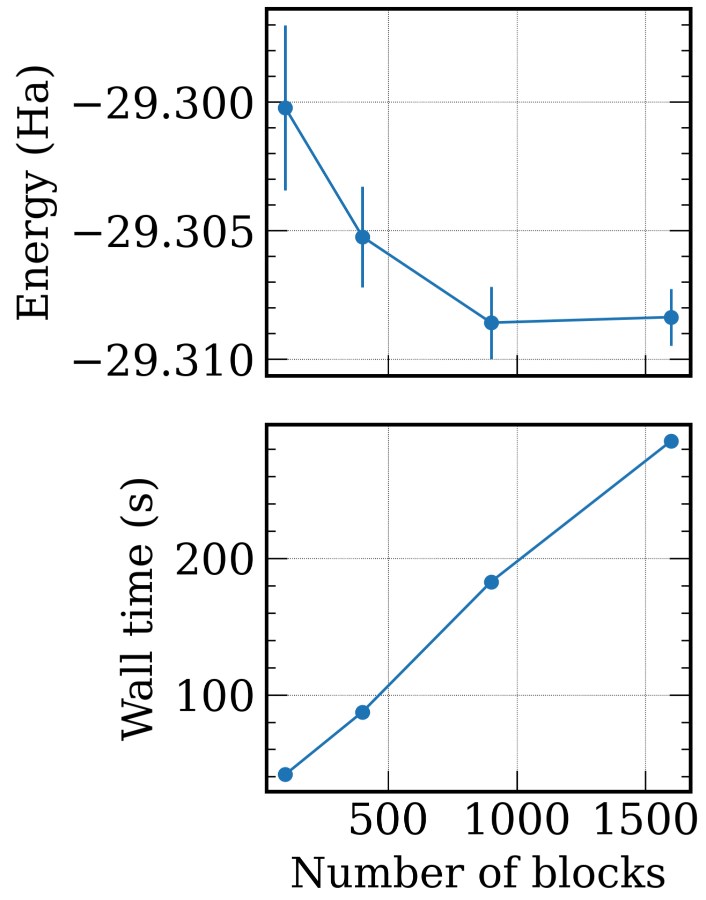
```
init_blocks=100
for i in range(1,6):
 myblocks=init_blocks*i*i
 qmc = generate_qmcpack(
  identifier      = 'dmc_error'+str(i),
  seed            = 42,
  path            = 'Be2/'+y+'/'+x+'/dmc_err
  job             = job(cores=cores),
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  vmc_samples     = 1024,
  warmupsteps     = 50,
  vmc_blocks      = 100,
  vmc_steps       = 10,
  vmc_timestep    = 0.1,
  timestep        = 0.00250,
  steps           = 80,
  blocks          = myblocks,
  dependencies    = orbdeps+[(optJ3,'jastrow
  )
```

DMC: reducing error bars

To reduce the error bar by a factor N, multiply the population or the number of blocks by N^2

run3_dmc_errorbars.py

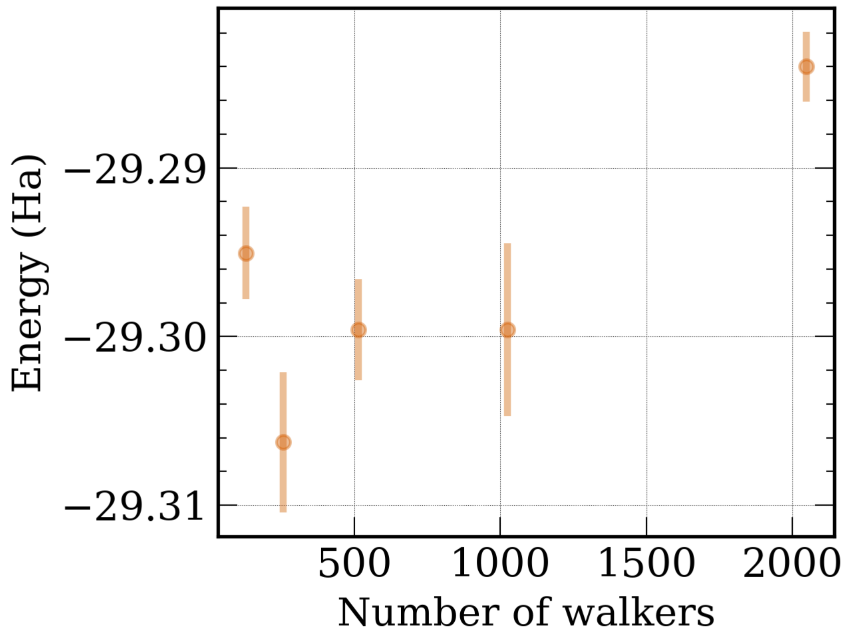
```
init_blocks=100
for i in range(1,6):
 myblocks=init_blocks*i*i
 qmc = generate_qmcpack(
  identifier      = 'dmc_error'+str(i),
  seed            = 42,
  path            = 'Be2/'+y+'/'+x+'/dmc_erro
  job             = job(cores=cores),
  system          = system,
  jastrows        = [],
  qmc             = 'dmc',
  vmc_samples     = 1024,
  warmupsteps     = 50,
  vmc_blocks      = 100,
  vmc_steps       = 10,
  vmc_timestep    = 0.1,
  timestep        = 0.00250,
  steps           = 80,
  blocks          = myblocks,
  dependencies    = orbdeps+[(optJ3,'jastrow
  )
```



DMC: reducing population bias

run4_dmc_population.py

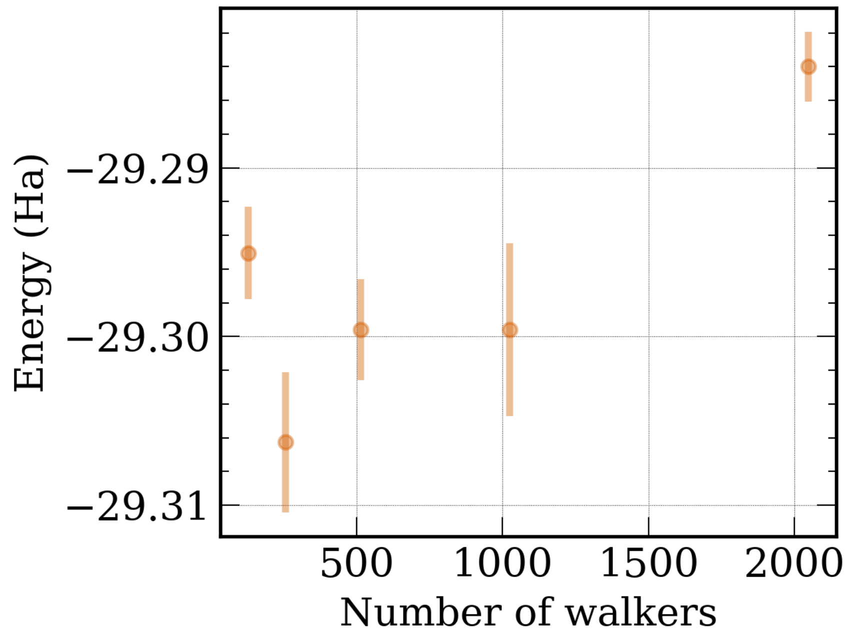
```python
pop=64
while pop<2049:
   qmc = generate_qmcpack(
     identifier      = 'dmc_pop'+str(pop),
     seed            = 42,
     path            = 'Be2/'+y+'/'+x+'/dmc_
     job             = qmc_job,
     system          = system,
     jastrows        = [],
     qmc             = 'dmc',
     total_walkers   = pop,
     warmupsteps     = 0,
     vmc_blocks      = 10,
     vmc_steps       = 1,
     vmc_timestep    = 0.3,
     timestep        = 0.01,
     steps           = 20480//pop,
     blocks          = 10,
     dependencies    = orbdeps+[(optJ3,'jast
     )
   pop=pop*2
```



DMC: reducing population bias

run4_dmc_population.py

```
pop=64
while pop<2049:
   qmc = generate_qmcpack(
     identifier       = 'dmc_pop'+str(pop),
     seed             = 42,
     path             = 'Be2/'+y+'/'+x+'/dmc_
     job              = qmc_job,
     system           = system,
     jastrows         = [],
     qmc              = 'dmc',
     total_walkers    = pop,
     warmupsteps      = 0,
     vmc_blocks       = 10,
     vmc_steps        = 1,
     vmc_timestep     = 0.3,
     timestep         = 0.01,
     steps            = 20480//pop,
     blocks           = 10,
     dependencies     = orbdeps+[(optJ3,'jast
     )
   pop=pop*2
```



Oxygen dimer

Oxygen dimer: using pseudopotentials

Perform SCF with Pseudopoential

```
system = generate_physical_system(
    units         = 'A',
    elem         = ['O','O'],
    pos          = [[0.000000, 0.000000, 0.000000],
                   [0.000000, 0.00000, 1.208]],
    O=6,
    )
settings(
    machine    = 'ws4',
    pseudo_dir  = './pseudos/')
```

Oxygen dimer: using pseudopotentials

Perform SCF with Pseudopoential

```
system = generate_physical_system(
    units         = 'A',
    elem         = ['O','O'],
    pos          = [[0.000000, 0.000000, 0.000000],
                    [0.000000, 0.00000, 1.208]],
    O=6,
    )
settings(
    machine     = 'ws4',
    pseudo_dir  = './pseudos/')
```

Create pseudopotentials set

```
ppset(
    label   = 'ccecp',
    qmcpack = ['O.ccECP.xml'],
    )
```

Oxygen dimer: using pseudopotentials

Perform SCF with Pseudopoential

```
system = generate_physical_system(
    units         = 'A',
    elem         = ['O','O'],
    pos          = [[0.000000, 0.000000, 0.000000],
                    [0.000000, 0.00000, 1.208]],
    O=6,
    )
settings(
    machine     = 'ws4',
    pseudo_dir  = './pseudos/')
```

Create pseudopotentials set

```
ppset(
    label    = 'ccecp',
    qmcpack = ['O.ccECP.xml'],
    )
```

Point programs towards pseudopotentials

```
scf = generate_pyscf(
    ...
    mole        = obj(
        basis     = 'ccecp-ccpvtz',
        ecp       = 'ccecp',
```

```
qmc = generate_qmcpack(
    identifier    = 'dmc',
    ...
    system        = system,
    pseudos       = 'ccecp',
```

Oxygen dimer: workflow

Calibration runs(`run1_calibration.py`)

- determine appropriate timestep, walker count, acceptable error

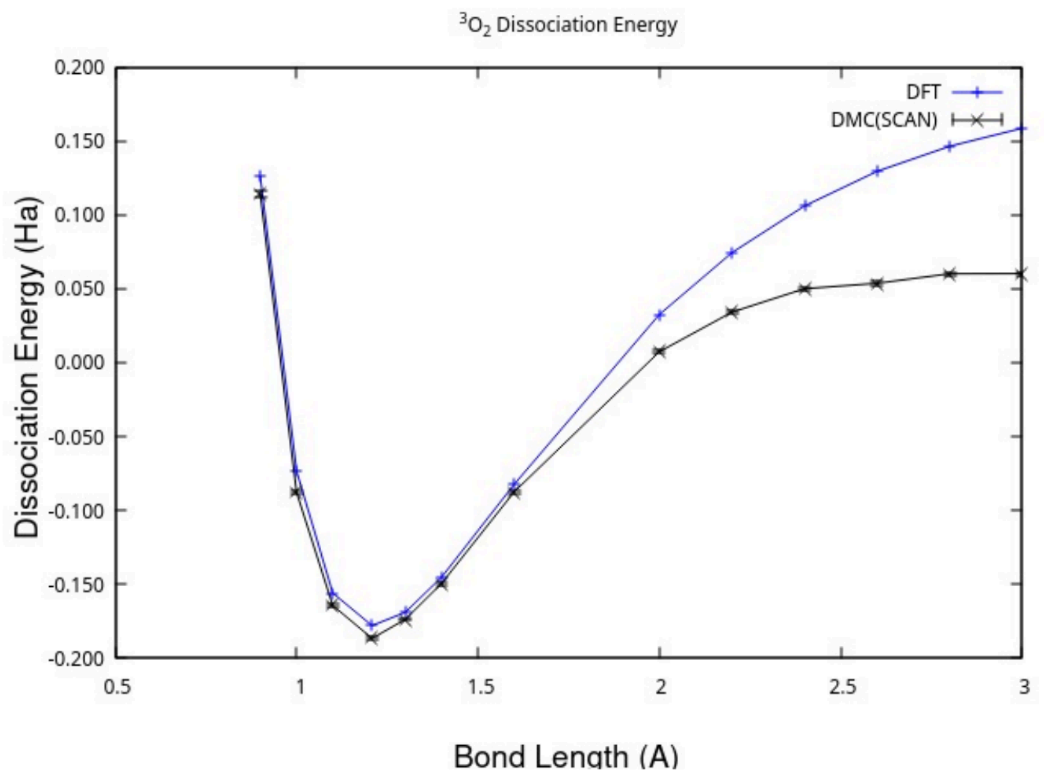
Production (`run2_dissociation_curve.py`)

run2_dissociation_curve.py

```
oo_dists=[0.9, 1.0, 1.1, 1.208, 1.3, 1.4, 1.
for i in oo_dists:
    system = generate_physical_system(
        units        = 'A',
        elem        = ['O','O'],
        pos         = [[0.000000, 0.000000, 
                       [0.000000, 0.00000, i]]
        O=6,
    )
    scf = generate_pyscf(...)
    c4q = generate_convert4qmc(...)

    orbdeps = [(c4q,'particles'),
             (c4q,'orbitals' )]

   optJ2 = generate_qmcpack(...)
   optJ3 = generate_qmcpack(...)
   qmc = generate_qmcpack(...)
```



Additional Considerations

How to start your own system

The materials of this workshop are a great starting place.

- while the scale of these problem are small, the workflow files give a great framework to start from
- Tune parameters to make the most of your computational resources
- We’ve provided the start of a water molecule example in the materials for you to try and set up your own workflow. `session4_molecules/03_water_molecule/`

Additional Considerations

Choices of trial wave functions

Choice of DFT functional

Multideterminant expansions

- link: previous workshop discussing selected CI
- link: qmcpack manual

Orbital optimization

- link: previous workshop with examples

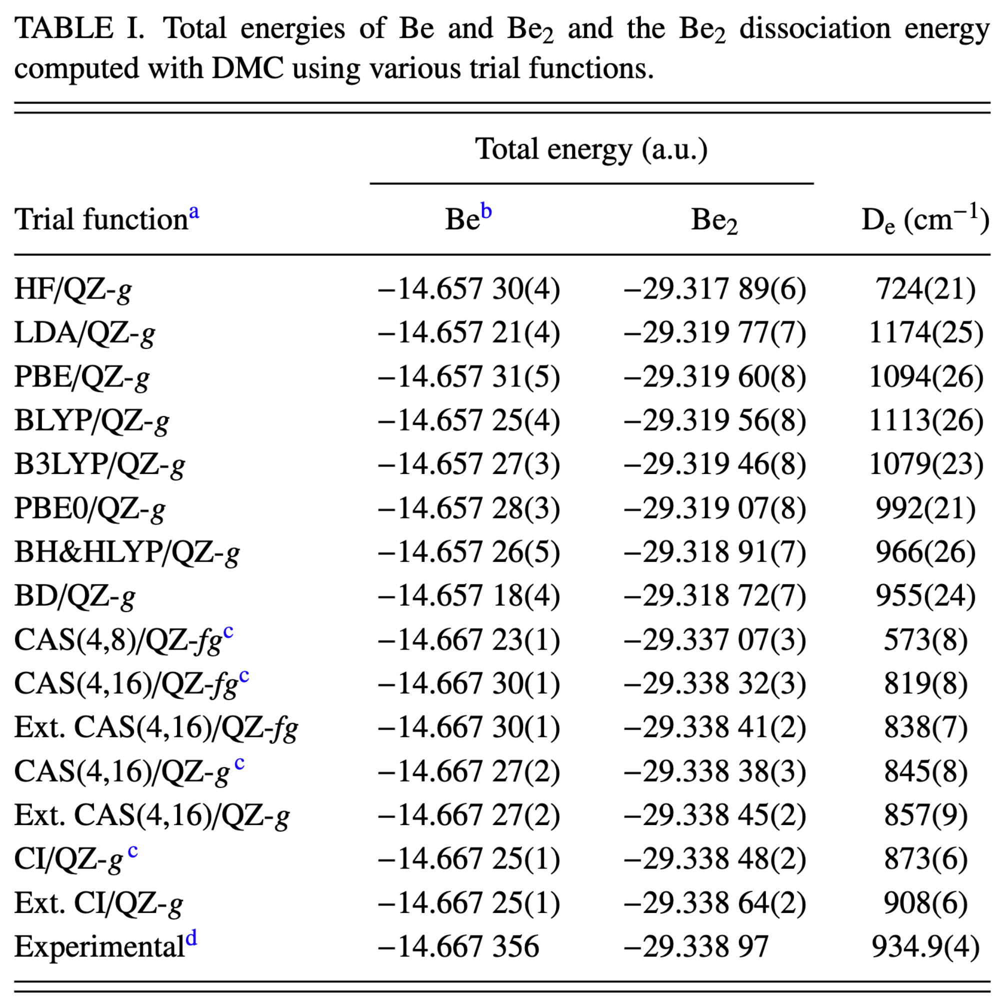


TABLE I. Total energies of Be and Be_2 and the Be_2 dissociation energy computed with DMC using various trial functions.

	Total energy (a.u.)		
Trial function[a]	Be[b]	Be_2	D_e (cm^{-1})
HF/QZ-*g*	−14.657 30(4)	−29.317 89(6)	724(21)
LDA/QZ-*g*	−14.657 21(4)	−29.319 77(7)	1174(25)
PBE/QZ-*g*	−14.657 31(5)	−29.319 60(8)	1094(26)
BLYP/QZ-*g*	−14.657 25(4)	−29.319 56(8)	1113(26)
B3LYP/QZ-*g*	−14.657 27(3)	−29.319 46(8)	1079(23)
PBE0/QZ-*g*	−14.657 28(3)	−29.319 07(8)	992(21)
BH&HLYP/QZ-*g*	−14.657 26(5)	−29.318 91(7)	966(26)
BD/QZ-*g*	−14.657 18(4)	−29.318 72(7)	955(24)
CAS(4,8)/QZ-*fg*[c]	−14.667 23(1)	−29.337 07(3)	573(8)
CAS(4,16)/QZ-*fg*[c]	−14.667 30(1)	−29.338 32(3)	819(8)
Ext. CAS(4,16)/QZ-*fg*	−14.667 30(1)	−29.338 41(2)	838(7)
CAS(4,16)/QZ-*g*[c]	−14.667 27(2)	−29.338 38(3)	845(8)
Ext. CAS(4,16)/QZ-*g*	−14.667 27(2)	−29.338 45(2)	857(9)
CI/QZ-*g*[c]	−14.667 25(1)	−29.338 48(2)	873(6)
Ext. CI/QZ-*g*	−14.667 25(1)	−29.338 64(2)	908(6)
Experimental[d]	−14.667 356	−29.338 97	934.9(4)

Summary

The goals were to learn:

How to drive the calculations using Nexus

```
- examples of beryllium dimer, and oxygen binding curve
```

How to define a trial wave function

```
- convert4qmc, defining a jastrow factor
```

How to optimize Ψ_T

```
- convergence in energy and variance and the ratio between the two
- nexus does have a way to guide this decision
```

Considerations for our DMC calculations

```
- timestep, desired error bars, population bias
```

Next session: Solid-State calculations Tuesday, July 15th 11AM EST US time