OAK RIDGE National Laboratory

07/17/2025

Session 6: Real world calculations of 2D Materials

QMCPACK Summer School, 2025

Kayahan Saritas

saritask@ornl.gov

U.S. DEPARTMENT of ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

Goals of this tutorial

- How to reach publication quality results
- Calculate the binding energy of bilayer hBN
- How to write a reusable script in Nexus
- Finite size extrapolation in 2D systems

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

DMC Convergence for publication quality results

Statistical convergence

- Equilibration period
- Sampling length
- Autocorrelation

Session 3 (**S3**) has in-depth information. **Should always be controlled.** Mainly post-processing and can be **system dependent**. Nexus is very helpful.

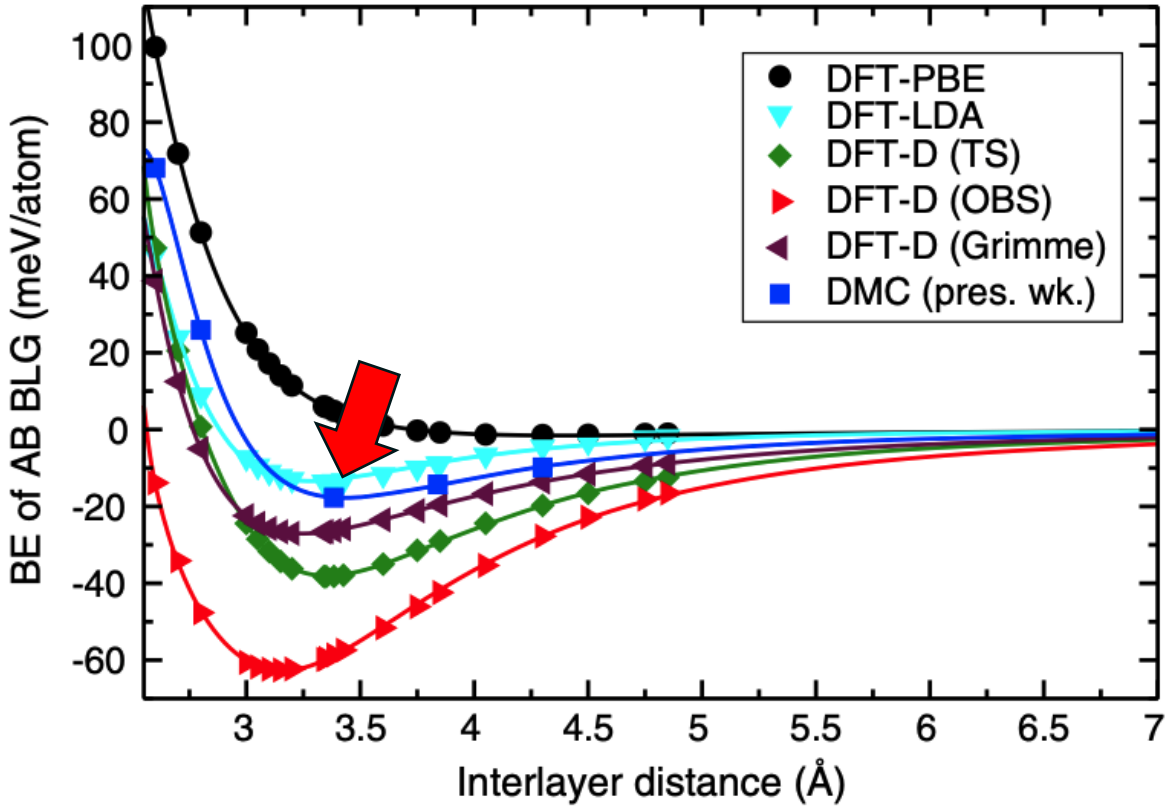
Systematic convergence

- Time step convergence (**S3**)
- Population control (walker count) (**S4**)
- Pseudopotentials (**S3**)
- Trial wavefunction quality and representation (**S3**, **S4, S5**)
- Localization approximation (**S3**)
- Finite-size effects (1 and 2 body) (**S5**)

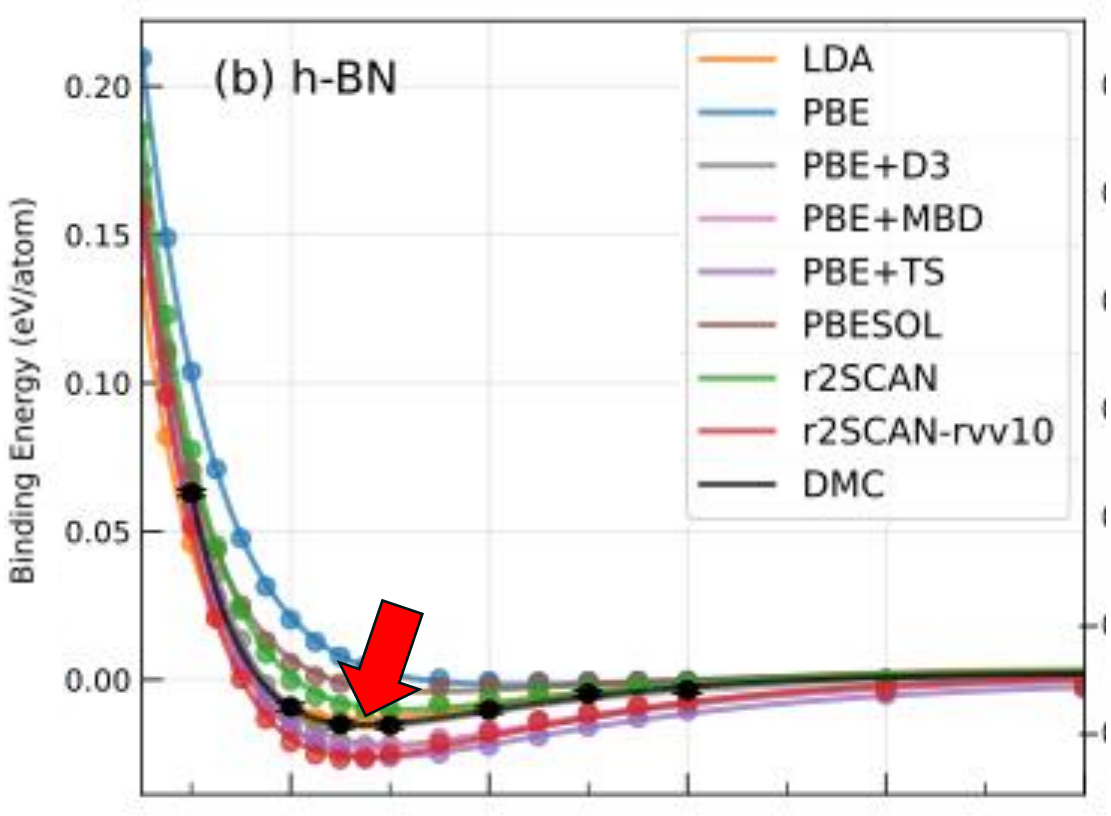
Requires an intuition and understanding of how DMC works.

For workshop files: https://github.com/QMCPACK/qmc_summer_school_2025

Accuracy of interlayer binding energy depends on the DFT functional

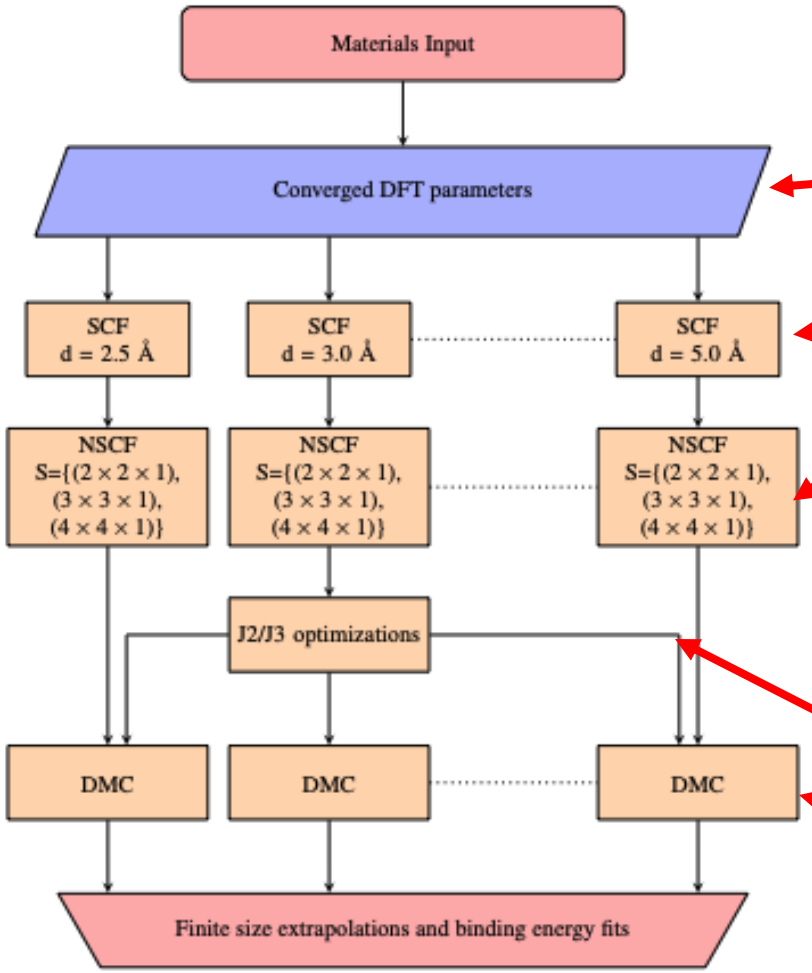


Mostaani et al. Phys. Rev. Lett. **115**, 115501 (2015)



How to write reusable scripts in Nexus

run.py



run_library.py

```python
#!/usr/bin/env python
from run_library import get_dft_settings, get_qmc_settings
from nexus import obj

dft_shared = obj(
    kgrid    = (8,8,1),  # K-point grid for DFT calculations
    ecutwfc  = 400,      # Plane-wave cutoff energy in Rydberg
    pseudos  = 'B.ccECP.upf N.ccECP.upf'.split()  # DFT PP files for Boron and Nitrogen
)
# SCF, NSCF and PW2QMCPACK settings
scf_shared, nscf_shared, conv_shared = get_dft_settings(**dft_shared)
```

```python
qmc_shared = obj(
      # Hybrid representation cutoff radius for Boron and Nitrogen in atomic units (a.u.)
      hybrid_rcut  = obj(B=1.1, N=1.1),
      # Maximum angular momentum for hybrid representation for Boron and Nitrogen
      hybrid_lmax  = obj(B=5, N=5),
      # Blip-spline Mesh factor for QMC calculations
      meshfactor   = 0.5,
      # Pseudopotential files for QMC calculations
      pseudos      = 'B.ccECP.xml  N.ccECP.xml'.split()
  )

j2_settings, j3_settings, dmc_settings  = get_qmc_settings(system = tiled_system,
                                                           **qmc_shared)
```

Contents of tutorial directory

```
/BN_tutorial
├── pseudos/
│   ├── B.ccECP.upf
│   ├── B.ccECP.xml
│   ├── N.ccECP.upf
│   └── N.ccECP.xml
├── README
├── run_library.py
├── run.py
└── structures/
    ├── hBN_d_2500.xsf
    ├── hBN_d_3000.xsf
    ├── hBN_d_3250.xsf
    ├── hBN_d_3500.xsf
    ├── hBN_d_4000.xsf
    ├── hBN_d_4500.xsf
    ├── hBN_d_5000.xsf
    └── hBN_mono.xsf
```

Supercell vectors

Corresponding twist grids

run.py

```
#!/usr/bin/env python
# user library imports
from run_library import get_dft_settings, get_qmc_settings
# nexus imports
from nexus import run_project, read_structure, obj
from nexus import generate_physical_system
from nexus import generate_pwscf
from nexus import generate_pw2qmcpack
from nexus import generate_qmcpack

# structure files and interlayer separations in Angstroms
structures = {3.0:  'structures/hBN_d_3000.xsf',
              2.5:  'structures/hBN_d_2500.xsf',
              3.25: 'structures/hBN_d_3250.xsf',
              3.5:  'structures/hBN_d_3500.xsf',
              4.0:  'structures/hBN_d_4000.xsf',
              4.5:  'structures/hBN_d_4500.xsf',
              5.0:  'structures/hBN_d_5000.xsf',
              'mono' : 'structures/hBN_mono.xsf'}
interlayer_separations  = list(structures.keys())

# Supercell tiling vectors and respective kgrids
tiling_vectors          = [(2,2,1), (3,3,1), (4,4,1)]
tiling_kgrids           = {(2,2,1):(4,4,1),
                           (3,3,1):(2,2,1),
                           (4,4,1):(2,2,1)}

# DFT and QMC settings shared across all calculations
system_shared = obj(
    B        = 3,        # Boron PP valency
    N        = 5,        # Nitrogen PP valency
    net_spin = 0         # Net spin of the system
)
```

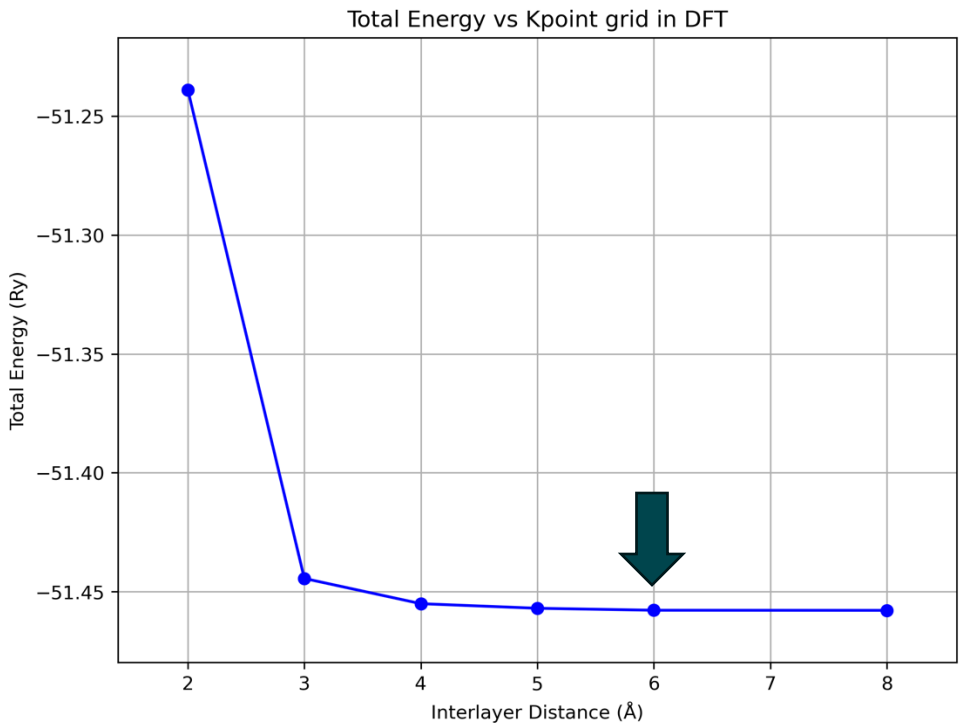
DFT convergence guides twist grid

run.py

```
#!/usr/bin/env python
# user library imports
from run_library import get_dft_settings, get_qmc_settings
# nexus imports
from nexus import run_project, read_structure, obj
from nexus import generate_physical_system
from nexus import generate_pwscf
from nexus import generate_pw2qmcpack
from nexus import generate_qmcpack

# structure files and interlayer separations in Angstroms
structures = {3.0:  'structures/hBN_d_3000.xsf',
              2.5:  'structures/hBN_d_2500.xsf',
              3.25: 'structures/hBN_d_3250.xsf',
              3.5:  'structures/hBN_d_3500.xsf',
              4.0:  'structures/hBN_d_4000.xsf',
              4.5:  'structures/hBN_d_4500.xsf',
              5.0:  'structures/hBN_d_5000.xsf',
              'mono' : 'structures/hBN_mono.xsf'}
interlayer_separations  = list(structures.keys())

# Supercell tiling vectors and respective kgrids
tiling_vectors          = [(2,2,1), (3,3,1), (4,4,1)]
tiling_kgrids           = {(2,2,1):(4,4,1),
                           (3,3,1):(2,2,1),
                           (4,4,1):(2,2,1)}

# DFT and QMC settings shared across all calculations
system_shared = obj(
    B        = 3,        # Boron PP valency
    N        = 5,        # Nitrogen PP valency
    net_spin = 0         # Net spin of the system
)
```

Supercell vectors

Corresponding twist grids



Settings common to all DFT and QMC calculations

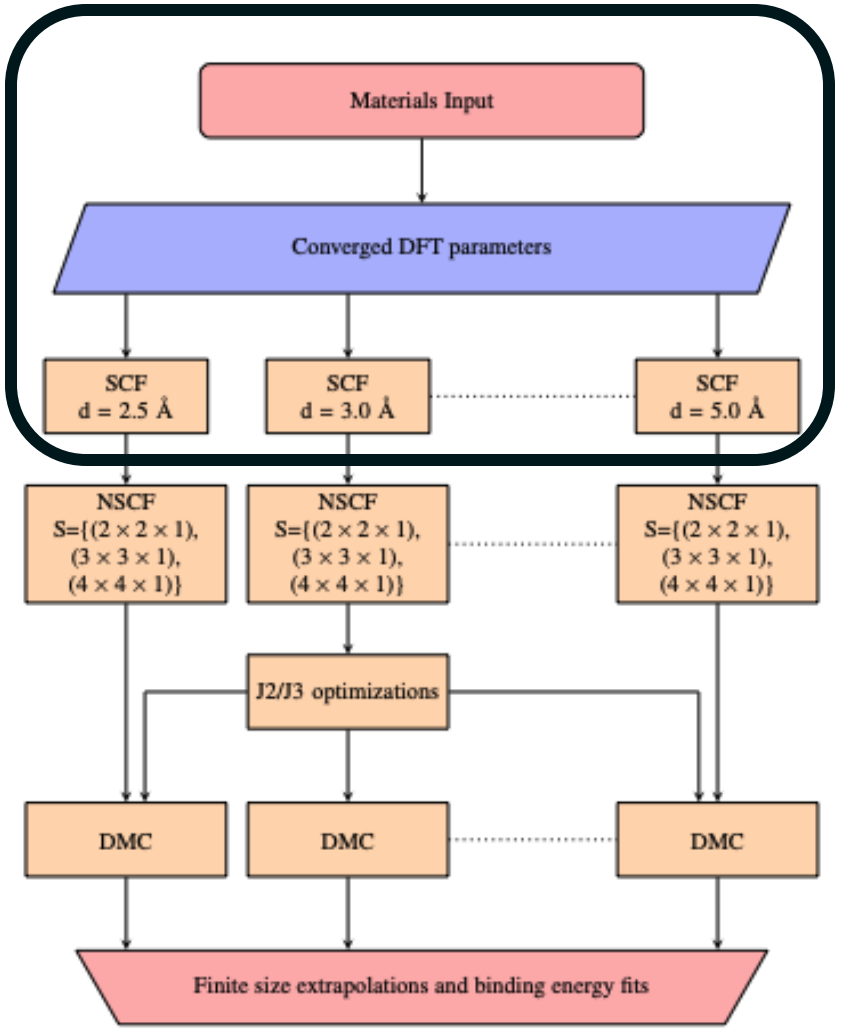
run.py

```
dft_shared = obj(
    kgrid    = (6,6,1),  # K-point grid for DFT calculations
    ecutwfc  = 400,      # Plane-wave cutoff energy in Rydberg
    pseudos  = 'B.ccECP.upf N.ccECP.upf'.split()  # DFT PP files for Boron and Nitrogen
)

qmc_shared = obj(
    # Hybrid representation cutoff radius for Boron and Nitrogen in atomic units (a.u.)
    hybrid_rcut  = obj(B=1.1, N=1.1),
    # Maximum angular momentum for hybrid representation for Boron and Nitrogen
    hybrid_lmax  = obj(B=5, N=5),
    # Blip-spline Mesh factor for QMC calculations
    meshfactor   = 0.5,
    # Pseudopotential files for QMC calculations
    pseudos      = 'B.ccECP.xml  N.ccECP.xml'.split()
)
```

SCF calculations

run.py

```
52  # SCF, NSCF and PW2QMCPACK settings
53  scf_shared, nscf_shared, conv_shared = get_dft_settings(**dft_shared)
54
55  # Binding energy workflow start
56  for d in interlayer_separations:
57      # Convert interlayer separation to an int for file naming
58      if isinstance(d, (int, float)):
59          d_name = int(d*1000)
60      else:
61          d_name = d
62
63      scf_path = 'scf_{}'.format(d_name)
64
65      # Generate the primitive cell system
66      prim_system = generate_physical_system(
67          structure = read_structure(structures[d]),
68          **system_shared
69          )
70      # SCF calculation
71      scf_run = generate_pwscf(
72          system = prim_system,
73          path = scf_path,
74          **scf_shared
75          )
```

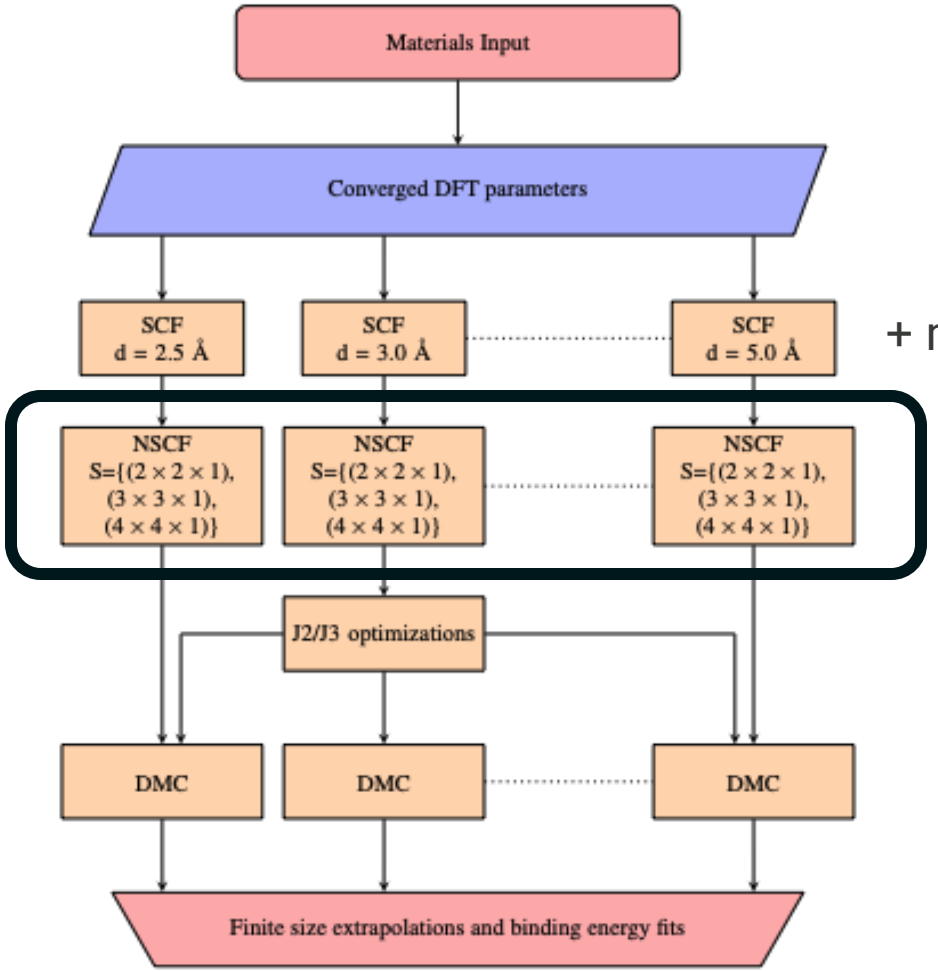


+ mono

NSCF/CONV calculations

run.py

```
76      for t in tiling_vectors:
77          # Directory for the NSCF calculation
78          nscf_path = 'nscf_{}_{}'.format(d_name, t[0])
79
80          # Generate the supercell system
81          tiled_system = generate_physical_system(
82              structure = read_structure(structures[d]),
83              tiling    = t,
84              kgrid     = tiling_kgrids[t],
85              **system_shared
86          )
87          # NSCF calculation
88          nscf_run = generate_pwscf(
89              system = tiled_system,
90              path = nscf_path,
91              **nscf_shared
92          )
93          # PW2QMCPACK conversion calculation
94          conv_run = generate_pw2qmcpack(
95              path         = nscf_path,    # Use the same path as the NSCF calculation
96              dependencies = (nscf_run, 'orbitals'),
97              **conv_shared
98          )
```

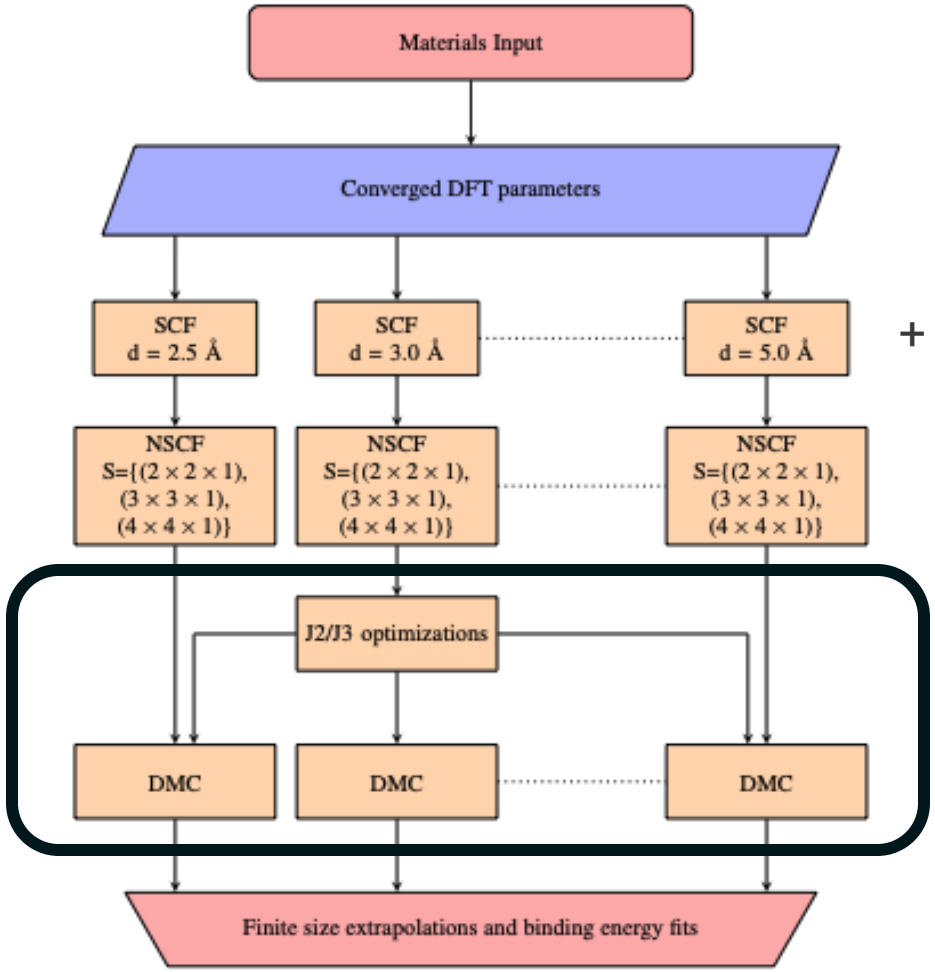


+ mono

VMC/DMC calculations

run.py

```
99
100         dmc_path = 'dmc_{}_{}'.format(d_name, t[0])
101
102         # Optimize jastrows using the first structure listed in interlayer_separations
103         # In this example, this is d == 3.0 since dictionary keys are always ordered in Python 3.7+
104         j2_settings, j3_settings, dmc_settings  = get_qmc_settings(system = tiled_system,
105                                                                    **qmc_shared)
106         if d == interlayer_separations[0]:
107             j2_path = 'j2_{}_{}'.format(d_name, t[0])
108             j3_path = 'j3_{}_{}'.format(d_name, t[0])
109             # J2, J3 optimizations and DMC calculation settings
110             # Here each "settings" object is specific to the system size
111
112
113             # J2 optimization calculation
114             j2_run = generate_qmcpack(path = j2_path,
115                                       dependencies = (conv_run, 'orbitals'),
116                                       **j2_settings)
117             # J3 optimization calculation
118             j3_run = generate_qmcpack(path = j3_path,
119                                       dependencies = [(j2_run, 'jastrow'), (conv_run, 'orbitals')],
120                                       **j3_settings)
121         # DMC calculation
122         dmc_run = generate_qmcpack(path = dmc_path,
123                                       dependencies = [(j3_run, 'jastrow'),(conv_run, 'orbitals')],
124                                       **dmc_settings)
125 run_project()
```



+ mono

DFT settings

run_library.py

```python
#!/usr/bin/env python
# nexus imports
from nexus import Job, obj
from nexus import settings
from nexus import linear, loop, vmc, dmc
from qmcpack_input import spindensity
# general settings for nexus
settings(
    pseudo_dir    = './pseudos',
    status_only   = 0,                  # only show status of runs
    generate_only = 0,                  # only make input files
    sleep         = 3,                  # check on runs every 3 secondsa
    machine       = 'perlmutter',       # Perlmutter NERSC machine
    account       = '<account_name>',   # User account name
)

def get_dft_settings(kgrid     = None,
                     ecutwfc   = None,
                     pseudos   = None,
                     start_mag = None,
                     hubbard   = None,
                     tot_magnetization = None):


    if settings.machine == 'perlmutter':
        qe_modules = '' # Modules used to build QE
        qe_bin     = '' # QE build directory
        dft_job = Job(cores  = 4,
                      threads= 1,
                      hours  = 12,
                      app    = qe_bin+'/pw.x',
                      constraints = 'cpu', # Default
                      presub= qe_modules)
        conv_job = Job(cores=1,
                       hours=1,
                       app='/pw2qmcpack.x',
                       constraints = 'cpu',
                       presub=qe_modules)
    else:
        print('Error: Unknown computer for DFT, using {}'.format(settings.machine))
        exit()
```

```python

    qe_shared = obj(
        job          = dft_job,
        input_type   = 'generic',
        ecutwfc      = ecutwfc,          # DFT planewave energy cutoff
        input_DFT    = 'PBE',            # DFT functional
        conv_thr     = 1e-8,             # SCF convergence threshold
        wf_collect   = True,             # write orbitals
        pseudos      = pseudos,          # QE Pseudopotentials
        start_mag    = start_mag,        # Starting magnetization
        hubbard      = hubbard,          # Hubbard U parameters
        occupations  = 'smearing',       # Occupation scheme
        smearing     = 'gauss',          # Smearing type
        degauss      = 0.001,            # Smearing widt
        tot_magnetization = tot_magnetization
    )

    scf_shared = obj(
        nosym        = False,            # use symmetry
        identifier   = 'scf',            # identifier/file prefix
        calculation  = 'scf',            # perform scf calculation
        kgrid        = kgrid,            # Converged DFT k-grid
        **qe_shared
    )

    nscf_shared = obj(
        nosym        = True,             # don't use symmetry
        identifier   = 'nscf',           # identifier/file prefix
        calculation  = 'nscf',           # perform nscf calculation
        diagonalization = 'cg',          # Diagonalization method
        **qe_shared
    )

    conv_shared = obj(
            identifier   = 'conv',           # identifier/file prefix
            job          = conv_job,         # Job object for PW2QMCPACK
            write_psir   = False,            # Don't write psir
    )

    return scf_shared, nscf_shared, conv_shared
```

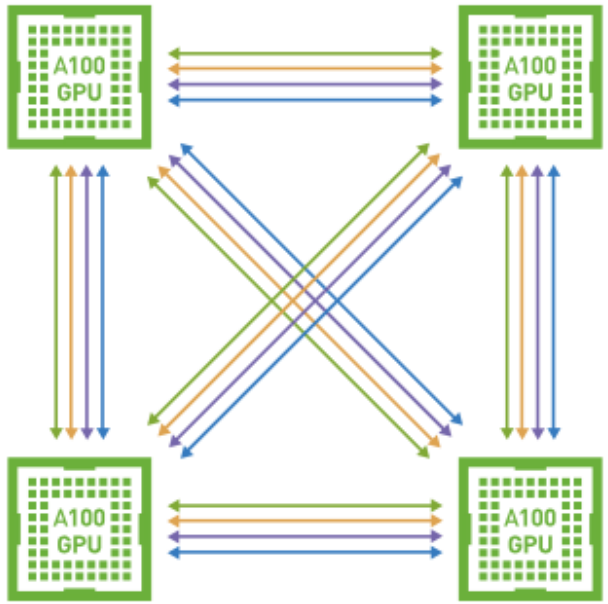
DMC is more sensitive to phase factor than DFT (**S5**)

QMC settings

run_library.py

```
83  def get_qmc_settings(system      = None,
84                       hybrid_rcut = None,
85                       hybrid_lmax = None,
86                       meshfactor  = 1.0,
87                       pseudos     = None):
88
89      if settings.machine == 'perlmutter':
90          qmcpack_modules = ''    # Modules used to build QMCPACK
91          qmcpack_bin     = ''    # QMCPACK build directory
92          qmcpack_exec    = qmcpack_bin+'/qmcpack_complex'
93          opt_job = Job(nodes    = 12,
94                        threads = 16,
95                        hours    = 12,
96                        constraint = 'gpu',
97                        app      = qmcpack_exec,
98                        presub   = qmcpack_modules)
99          dmc_job = Job(nodes    = 24,
100                       threads = 16,
101                       hours    = 12,
102                       constraint = 'gpu',
103                       app      = qmcpack_exec,
104                       presub   = qmcpack_modules)
105     else:
106         print('Error: Unknown computer for QMC, using {}'.format(settings.machine))
107         exit()
```

Perlmutter architecture



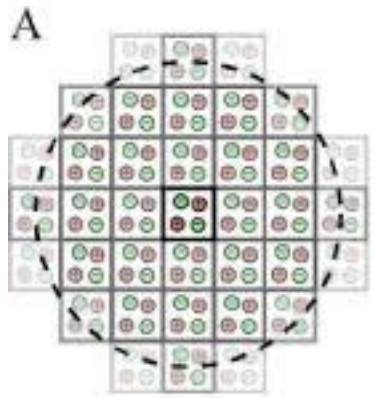
- Single AMD EPYC 7763 (Milan) CPU
- 64 cores per CPU
- Four NVIDIA A100 (Ampere) GPUs

https://docs.nersc.gov/systems/perlmutter/architecture/

QMC settings

run_library.py

```
109     system.structure.change_units('B')
110     rwigner = system.structure.rwigner()
111
112     qmc_settings = obj(
113         system          = system,        # PhysicalSystem object containing structural info
114         input_type      = 'basic',       # Simple input format for QMCPACK
115         pseudos         = pseudos,       # Pseudopotential files for QMC
116         driver          = 'batched',     # Use batched driver in QMCPACK
117         hybrid_rcut     = hybrid_rcut,   # Cutoff radius for hybrid orbital representation
118         hybrid_lmax     = hybrid_lmax,   # Max angular momentum for hybrid orbitals
119         meshfactor      = meshfactor,    # Controls fineness of real-space (Spline) grid
120         lr_handler      = 'ewald',       # Use Ewald summation for long-range interactions
121         lr_dim_cutoff   = 30,            # Cutoff for long-range Ewald sums
122         spin_polarized  = True,          # Enable spin-polarized calculations
123     )
124
125     opt_parameters = obj(
126         num_varmin_j2   = 12,     # Number of variance minimization iterations for 2-body Jastrow
127         num_emin_j2     = 8,      # Number of energy minimization iterations for 2-body Jastrow
128         num_emin_j3     = 6,      # Number of energy minimization iterations for 3-body Jastrow
129         j2_init         = "rpa",  # Initialize 2-body Jastrow with Random Phase Approximation
130         num_j1_jastrows = 10,     # Number of 1-body Jastrow parameters to optimize
131         num_j2_jastrows = 10,     # Number of 2-body Jastrow parameters to optimize
132         num_j3_jastrows = 3,      # Number of 3-body Jastrow parameters to optimize
133         j3_rcut         = 4.0 if rwigner > 4.0 else rwigner,  # 3-body Jastrow cutoff radius (min of 4.0 or Wigner radius)
134         timestep        = 1.0     # VMC timestep for optimization
135     )
136
137     opt_settings = obj(
138         job             = opt_job,
139         twistnum        = 0,     # Twist index
140     )
141     opt_settings = opt_settings.set(qmc_settings)
```



J. Chem. Phys. 124, 234104 (2006)

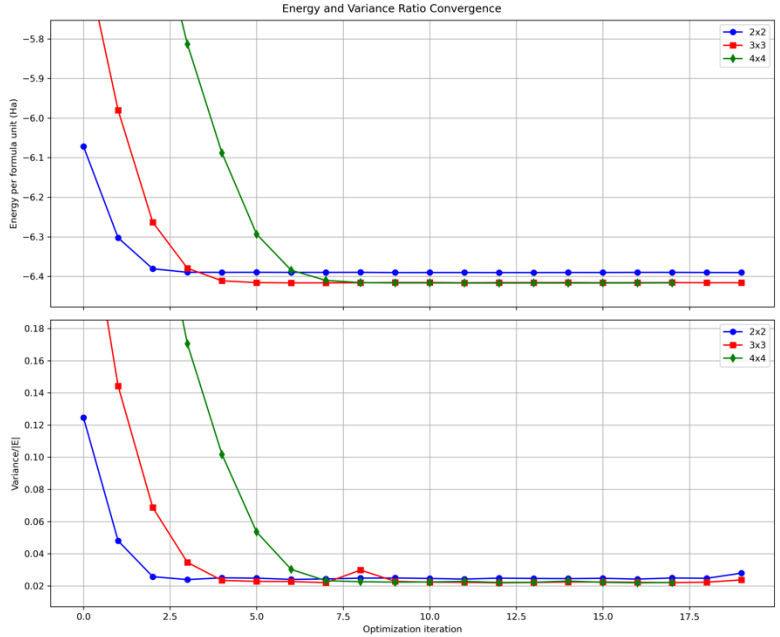
1. J2-varmin
2. J2-emin
3. J2+J3-emin

QMC settings

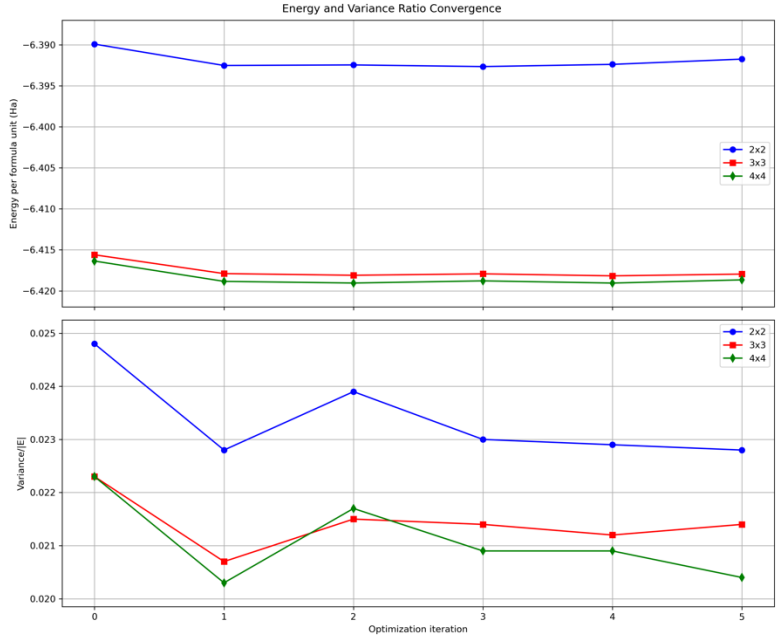
run_library.py

```
    # Variance minimization settings
    varmin = linear(
        energy               = 0.0,              # Weight for energy minimization (0 = pure variance min)
        unreweightedvariance = 1.0,              # Weight for unreweighted variance minimization
        reweightedvariance   = 0.0,              # Weight for reweighted variance minimization
        minwalkers           = 1e-4,             # Lower bound of the effective walker weight
        shift_i              = 0.05,             # (OneShiftOnly Optimizer) Direct stabilizer shift
        shift_s              = 1.0,              # (OneShiftOnly Optimizer) Stabilizer shift based on overlap matrix
        warmupsteps          = 200,              # Number of steps before measurements begin
        blocks               = 100,              # Number of statistical measurement blocks
        steps                = 1,                # Steps per block
        timestep             = 1.0,              # VMC timestep
        minmethod            = "OneShiftOnly",   # Minimization algorithm to use
        substeps             = 10,               # Number of MC steps between parameter updates
    )

    # Energy minimization settings
    emin = varmin.copy() # Copy from varmin
    emin.minwalkers             = 0.5  # Use larger minwalkers, since varmin provides a better starting point
    emin.energy                 = 0.95 # Mixed cost function 0.95 energy / 0.05 variance
    emin.unreweightedvariance   = 0.0
    emin.reweightedvariance     = 0.05
    emin.shift_i                = 0.01 # Reduced shift_i, since we are closer to the minimum

    j2_settings     = obj(
        calculations = [loop(max=opt_parameters.num_varmin_j2, qmc=varmin),
                        loop(max=opt_parameters.num_emin_j2,   qmc=emin)],
        J1_size = opt_parameters.num_j1_jastrows,
        J2_size = opt_parameters.num_j2_jastrows,
        J1_rcut = rwigner,
        J2_rcut = rwigner,
        J2_init = opt_parameters.j2_init,
        **opt_settings
    )

    j3_settings     = obj(
        calculations = [loop(max=opt_parameters.num_emin_j3, qmc=emin)],
        J3=True,
        J3_isize = opt_parameters.num_j3_jastrows,
        J3_esize = opt_parameters.num_j3_jastrows,
        J3_rcut  = opt_parameters.j3_rcut,
        **opt_settings
    )
```

J2-opt



J3-opt

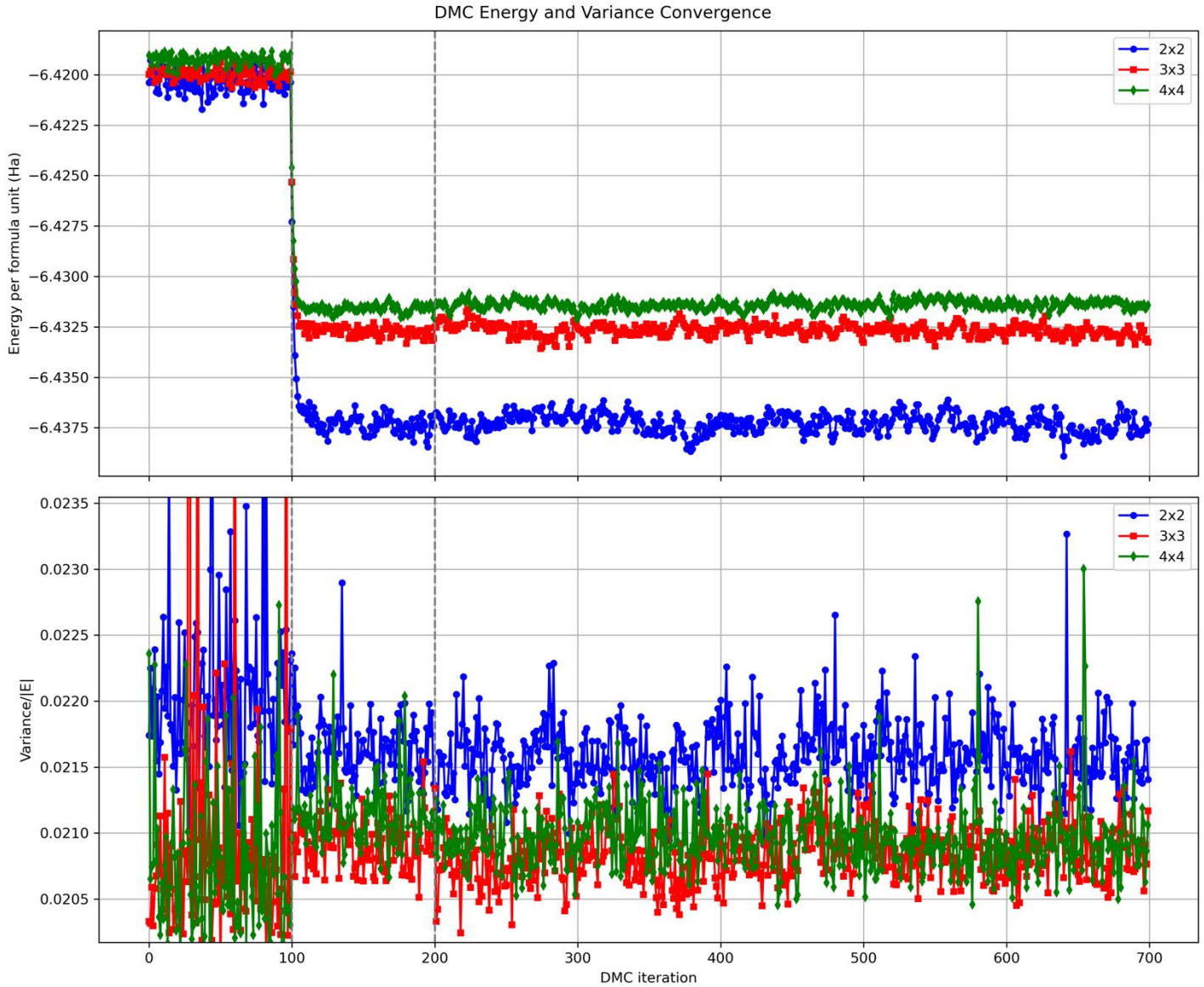


QMC settings

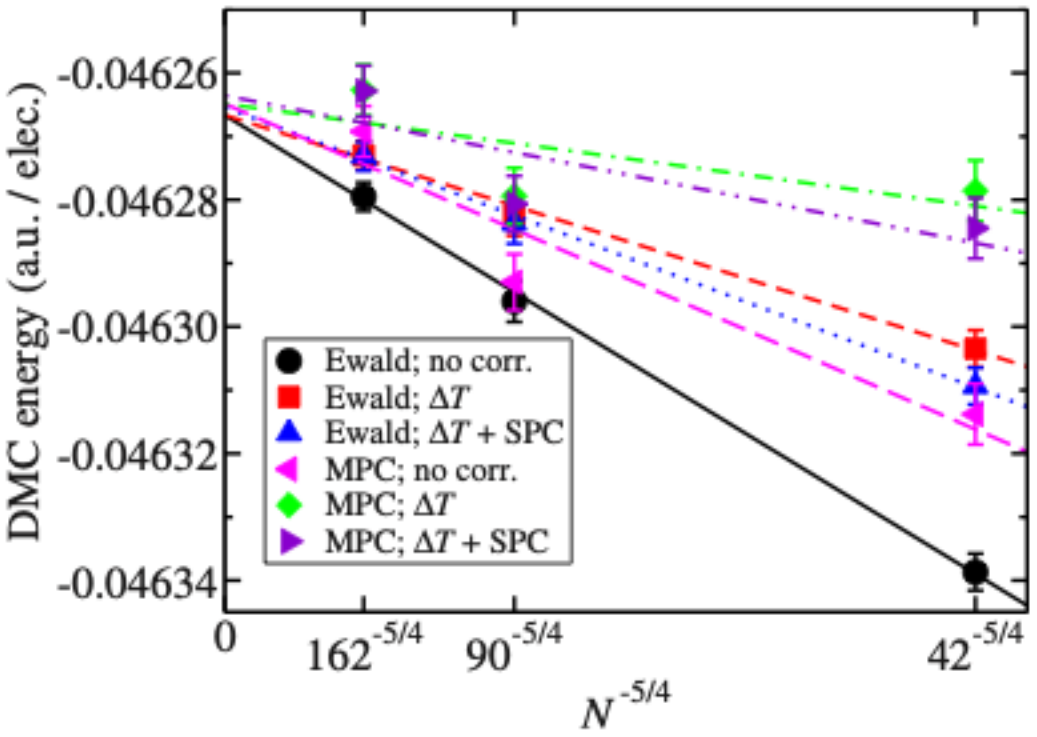
run_library.py

```
    dmc_parameters = obj(
        vmcdt                = 0.3,     # VMC timestep in atomic units
        vmcwarmup            = 25,      # Number of VMC blocks to equilibrate
        vmcblocks            = 100,     # Number of VMC measurement blocks
        vmcsubsteps          = 4,       # VMC steps between measurements
        dmc_eq_dt            = 0.02,    # DMC equilibration timestep
        dmc_eq_blocks        = 100,     # Number of DMC equilibration blocks
        dmcdt                = 0.005,   # DMC production timestep
        dmcblocks            = 500,     # Number of DMC production blocks
        dmcwarmup            = 100,     # Number of DMC blocks to equilibrate
        dmcsteps             = 10,      # Steps per DMC block
        vmc_walkers_per_rank = 240,     # Number of VMC walkers per MPI rank
        dmc_walkers_per_rank = 240,     # Number of DMC walkers per MPI rank
        nonlocalmoves        = False,   # Use T-moves for non-local pseudopotentials
    )
    vmc_dmc = obj(
        warmupsteps = dmc_parameters.vmcwarmup,
        blocks      = dmc_parameters.vmcblocks,
        steps       = 1,
        timestep    = dmc_parameters.
        substeps    = dmc_parameters.vmcsubsteps,
        walkers_per_rank = dmc_parameters.vmc_walkers_per_rank
    )
    dmc_eq  = obj(
        warmupsteps = dmc_parameters.dmcwarmup,
        blocks      = dmc_parameters.dmc_eq_blocks,
        steps       = dmc_parameters.dmcsteps,
        timestep    = dmc_parameters.dmc_eq_dt,
        walkers_per_rank = dmc_parameters.dmc_walkers_per_rank,
        nonlocalmoves = dmc_parameters.nonlocalmoves,
    )
    dmc_stat = obj(
        warmupsteps = dmc_parameters.dmcwarmup,
        blocks      = dmc_parameters.dmcblocks,
        steps       = dmc_parameters.dmcsteps,
        timestep    = dmc_parameters.dmcdt,
        walkers_per_rank = dmc_parameters.dmc_walkers_per_rank,
        nonlocalmoves = dmc_parameters.nonlocalmoves,
    )

    dmc_settings = obj(
        job           = dmc_job,
        calculations  = [vmc(**vmc_dmc), dmc(**dmc_eq), dmc(**dmc_stat)],
        estimators    = [spindensity(dr=3*[0.3])],
        **qmc_settings
    )

    return j2_settings, j3_settings, dmc_settings
```

(function) vmcsubsteps: Any

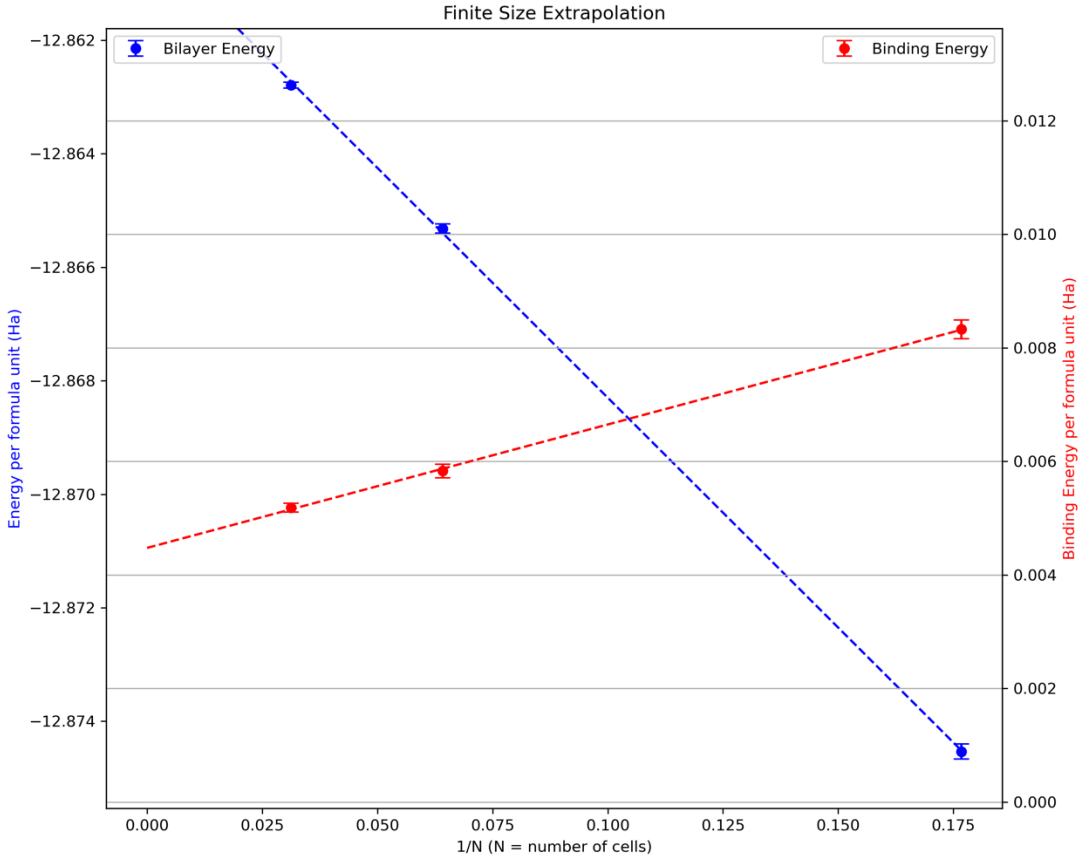


Finite size extrapolation in 2D



Drummond et al, PRB **78**, 125106 (2008)

hBN d = 2.5 A



Next session: Running on GPUs and Surrogate Hessian Geom. Opt.