JDFT calculations in practice with JDFTx

The BEAST collaboration

3rd Annual BEAST Workshop, 2024

August 22, 2024

U.S. DEPARTMENT OF ENERGY

Office of Science

Award # DE-SC0022247

JDFTx features

- JDFTx is a fully-featured plane-wave DFT code
- Focus here: capabilities for electrochemistry
- We'll cover the underlined: small fraction of what JDFTx can do!
- Many more tutorials on https://jdftx.org

Electronic

- Exchange-correlation: semilocal, meta-GGA, EXX-hybrids, DFT+U, DFT-D2, LibXC
- Pseudopotentials: norm-conserving and ultrasoft
- Noncollinear magnetism / spin-orbit coupling
- Algorithms: variational minimization, SCF
- Grand canonical (fixed potential) for electrochemistry
- Truncated Coulomb for 0D, 1D, 2D or 3D periodicity
- Custom external potentials, electric fields
- Charged-defect corrections: bulk and interfacial
- Ion/lattice optimization with constraints
- *Ab initio* molecular dynamics
- Vibrational modes, phonons and free energies

Fluid

- Linear solvation: GLSSA13, SCCS, CANDLE
- Nonlinear solvation: GLSSA13
- Nonlocal solvation: SALSA
- JDFT with classical DFT fluids

Outputs (selected)

- DOS, optical matrix elements, polarizability etc.
- Wannier functions and *ab initio* tight-binding
- Electron-electron and electron-phonon scattering

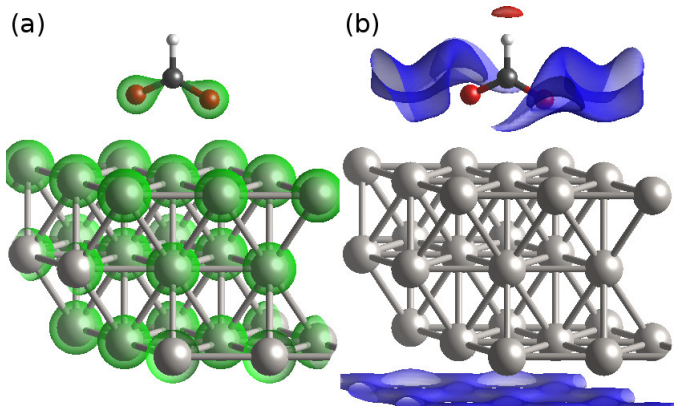
Interfaces

- Solvated QMC with CASINO
- Atomistic Simulation Environment (NEB, MD etc.)
- Visualization: VESTA, XCrySDen, PyMOL

The JDFTx input file

```
#—— Pseudopotentials ——
ion-species GBRV/$ID_pbe.uspp          #GBRV family
elec-cutoff 20 100              #Ecuts for psi and rho
#—— Geometry ——
lattice Hexagonal 10.53 30.0     #a and c in bohrs
coulomb-interaction Slab 001  #Make z nonperiodic
coulomb-truncation-embed 0 0 0     #Specify center
coords-type Lattice          #fractional coordinates
ion Pt   0.33333  -0.33333  -0.288    0
ion Pt   0.33333  -0.83333  -0.288    0
ion Pt   0.83333  -0.83333  -0.288    0
ion Pt   0.83333  -0.33333  -0.288    0
ion Pt   0.16667  -0.16667  -0.144    0
ion Pt   0.16667  -0.66667  -0.144    0
ion Pt   0.66667  -0.16667  -0.144    0
ion Pt   0.66667  -0.66667  -0.144    0
ion Pt   0.000     0.000     0.000    0
ion Pt   0.000    -0.500     0.000    0
ion Pt   0.500     0.000     0.000    0
ion Pt   0.500    -0.500     0.000    0    #0 => fixed
ion O    0.152    -0.079     0.155    1    #1 =>  free
ion O   -0.152    +0.079     0.155    1
ion C    0.000     0.000     0.190    1  Planar 0 0 1
ion H    0.000     0.000     0.260    1
ionic-minimize nIterations 10  #Optimize geometry
#—— Electronic ——
kpoint-folding 6 6 1          #Gamma-centered k-mesh
elec-smearing Cold 0.01       #Select cold smearing
target-mu -0.160                #Fix echem potential
#—— Fluid ——
fluid LinearPCM             #Class of solvation model
pcm-variant CANDLE    #Specific model within class
fluid-solvent H2O                 #Aqueous electrolyte
fluid-cation Na+ 1.                #1 mol/L Na+ cation
fluid-anion   F- 1.                #1 mol/L  F-  anion
#—— Outputs ——
dump Ionic IonicPositions ElecDensity BoundCharge
dump-name test.$VAR        #Output filename pattern
```

- Free-format input file with one command per line
- Command order does not matter (except the order of ions)
- Each documented extensively at
  https://jdftx.org/Commands.html
- Sensible defaults: input can be brief
- Hartree atomic units throughout
- Full example here: formate ion adsorbed on 2x2, 3-layer biased, solvated Pt(111)



Geometry

```
lattice Hexagonal 10.53 30.0    #a and c in bohrs
coords-type Lattice             #fractional coordinates
ion Pt  0.33333 -0.33333 -0.288   0
ion Pt  0.33333 -0.83333 -0.288   0
ion Pt  0.83333 -0.83333 -0.288   0
ion Pt  0.83333 -0.33333 -0.288   0
```

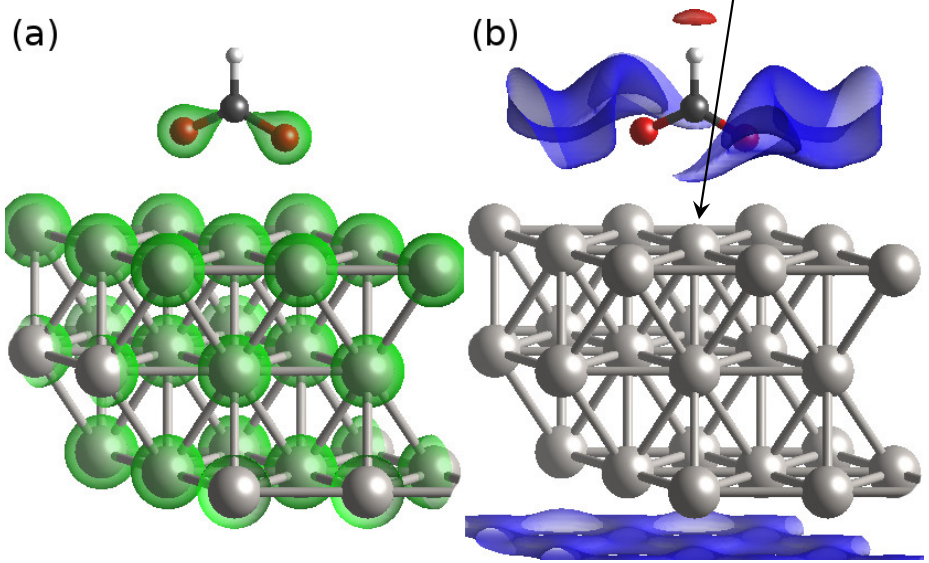
⋮

- Specify lattice system, or manually specify lattice vectors
- Specify ionic positions in fractional or Cartesian coordinates

Non-periodic geometries

```
coulomb-interaction Slab 001  #Make z nonperiodic
coulomb-truncation-embed 0 0 0    #Specify center
```

- Plane-wave DFT is intrinsically periodic
- Emulate non-periodic geometries by truncated Coulomb interactions
- Important: wave functions are still expanded in Fourier coefficients
- Leave just enough margin to let wave functions decay to zero
- JDFTx supports slab, wire or isolated geometries (2D, 1D or 0D periodic)



Electronic DFT parameters

```
ion-species GBRV/$ID_pbe.uspp    #Pseudopotentials
elec-ex-corr gga-PBE                      #Default XC

elec-cutoff 20 100                    #KE Cutoff on PWs
kpoint-folding 6 6 1          #Gamma-centered k-mesh

elec-smearing Cold 0.01  #To sample Fermi surface
```

- Must specify pseudopotials, individually or as a set
- Exchange-correlation (XC) is PBE GGA by default
- Basis set controlled by plane-wave kinetic energy cutoff (E_h)
- Brillouin zone sampling specified by k-mesh size
- Several smearing options for metals (Fermi, Gauss, Cold, MP1)

Actions

```
electronic-minimize energyDiffThreshold 1E-7 #or:
electronic-SCF energyDiffThreshold 1E-7

ionic-minimize nIterations 10  #Optimize geometry
```

- Kohn-Sham DFT solvable by two independent approaches:
  - Variational minimization (default, more robust)
  - Self-Consistent Field iteration (can be faster)
- Prefer the more-stable minimization for grand-canonical DFT
- Geometry optimization of ions and lattice (not on by default)

Outputs and continuation

```
initial-state test.$VAR     #Start from checkpoint

dump-name test.$VAR         #Output filename pattern
dump End BoundCharge      #For Visualization at end
dump Ionic State        #Checkpoint every ionic step
```

- Full control over what, when and how to name outputs
- Example:
  - Load from checkpoints saved in test.*
  - Save outputs in test.*
  - Write solvent charge response at end (will be test.nbound)
  - Write checkpoint every ionic step (will be test.ionpos, test.wfns, test.fillings, test.eigenvals)
- See documentation of dump command for full list of options

Solvation

```
fluid LinearPCM                #Class of solvation model
pcm-variant CANDLE          #Specific model within class
fluid-solvent H2O                    #Aqueous electrolyte
fluid-cation Na+ 1.                   #1 mol/L Na+ cation
fluid-anion  F- 1.                    #1 mol/L  F-  anion
```

- Specify type of fluid: none, a few implicit options, classical DFT
- For implicit solvent model, select variant (here: CANDLE)
- Select solvent, and optionally electrolyte
- CANDLE supports H2O and CH3CN (acetonitrile)
- Implicit electrolyte is always non-adsorbing, recommend always use NaF

Grand-canonical DFT

```
target_mu -0.160                    #Fix echem potential
```

- Specify absolute electron chemical potential in Hartrees: that's it!
- Need to convert potential U relative to reference electrode to absolute scale
- Essentially, $\mu = -(U + V_{\text{ref}})/27.21$, where V_{ref} is absolute potential of reference electrode below vacuum level
- For Standard Hydrogen electrode, $V_{\text{SHE}} = 4.66$ eV calibrated for the CANDLE solvation model
- Important: MUST specify electrolyte for GC-DFT to be sensible!

Solvation and electrochemistry workflow

1. Converge vacuum calculation (electronic and geometry)
2. Solvate at fixed charge / neutral
3. Apply bias if needed

Note: JDFTx will automatically run vacuum calculations where needed to get a reasonable starting point. We will use this in the tutorials, but recommend converging vacuum separately in production calculations.

Parallelization

- JDFTx is a hybrid MPI-threads code and particularly shines on GPUs
- On perlmutter, we will often use the 4 A100 GPUs on each node as:

  `srun -n 4 --gpus 4 jdftx_gpu -i in | tee out`

  which means
  - Run 4 processes using 4 GPUs total (one each)
  - Take input from file 'in' and mirror output to terminal and file 'out'

Parallelization

- JDFTx is a hybrid MPI-threads code and particularly shines on GPUs
- On perlmutter, we will often use the 4 A100 GPUs on each node as:

  `srun -n 4 --gpus 4 jdftx_gpu -i in | tee out`

  which means
  - Run 4 processes using 4 GPUs total (one each)
  - Take input from file 'in' and mirror output to terminal and file 'out'
- MPI parallelization in JDFTx is over *k-points and spin* only
- Look for 'nStates' in output file using a dry run 'jdftx -ni in'
- Determine number of MPI processes based on 'nStates'

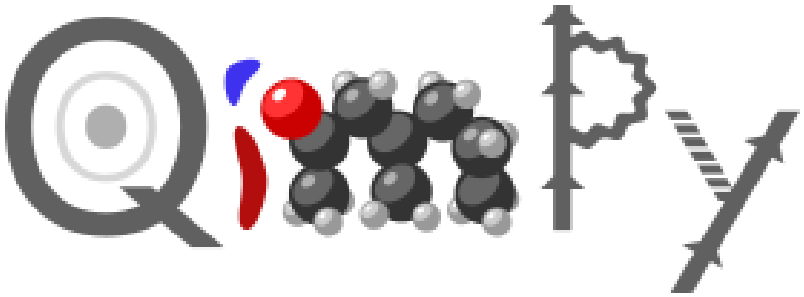
Parallelization

- JDFTx is a hybrid MPI-threads code and particularly shines on GPUs
- On perlmutter, we will often use the 4 A100 GPUs on each node as:

  `srun -n 4 --gpus 4 jdftx_gpu -i in | tee out`

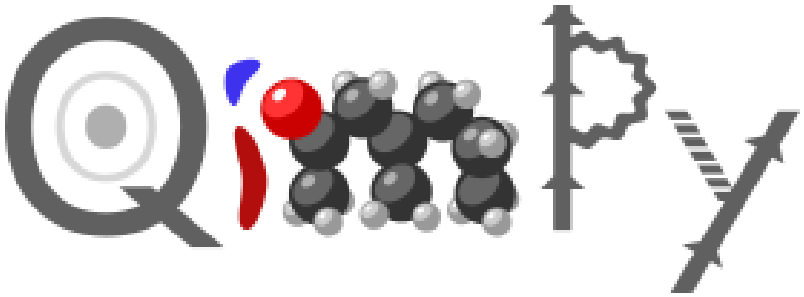
  which means
  - Run 4 processes using 4 GPUs total (one each)
  - Take input from file 'in' and mirror output to terminal and file 'out'
- MPI parallelization in JDFTx is over *k-points and spin* only
- Look for 'nStates' in output file using a dry run 'jdftx -ni in'
- Determine number of MPI processes based on 'nStates'
- In the tutorials, to demonstrate best practices, we will use 1 perlmutter node as:
  - Single process with one GPU (-n 1 –gpus 1) for molecule / ion calculations with nStates = 1
  - One process per GPU (-n 4 –gpus 4) for solvated / biased surface calculations with intermediate values of nStates
    (Our examples with nStates = 10 could be run over three nodes with -N 3 -n 10 –gpus 10, but we will keep to one node for the tutorial.)

QimPy: Quantum-Integrated Multi-PhYsics



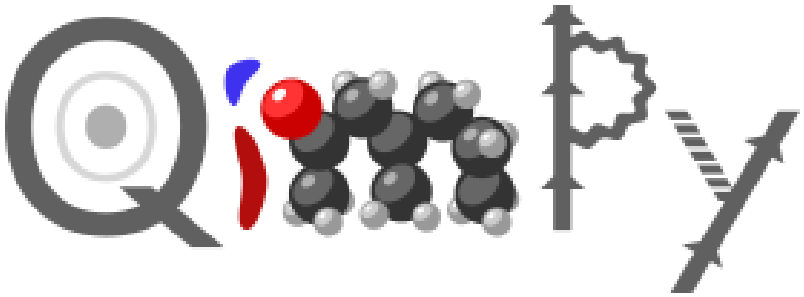
- Successor to JDFTx in Python using PyTorch as HAL for CPU/GPU/TPU
- Preview tomorrow: plane-wave DFT and AIMD with norm-conserving PS (Solvation, ultrasoft, PAW, DFT+U etc. coming soon)
- Efficient multi-level parallelization over replicas, k-points and bands to scale to many more CPUs and GPUs

QimPy: Quantum-Integrated Multi-PhYsics



- Successor to JDFTx in Python using PyTorch as HAL for CPU/GPU/TPU
- Preview tomorrow: plane-wave DFT and AIMD with norm-conserving PS (Solvation, ultrasoft, PAW, DFT+U etc. coming soon)
- Efficient multi-level parallelization over replicas, k-points and bands to scale to many more CPUs and GPUs
- Goal: rapid development in Python *without sacrificing performance*
- Common heirarchy in YAML inputs, HDF5 checkpoints and code structure
- Learn to use QimPy and you know where things are in the code!

QimPy: Quantum-Integrated Multi-PhYsics



- Successor to JDFTx in Python using PyTorch as HAL for CPU/GPU/TPU
- Preview tomorrow: plane-wave DFT and AIMD with norm-conserving PS (Solvation, ultrasoft, PAW, DFT+U etc. coming soon)
- Efficient multi-level parallelization over replicas, k-points and bands to scale to many more CPUs and GPUs
- Goal: rapid development in Python *without sacrificing performance*
- Common heirarchy in YAML inputs, HDF5 checkpoints and code structure
- Learn to use QimPy and you know where things are in the code!
- Looking for developers, maintainers and documenters: community effort
- Tomorrow's session will provide glimpse into development as well