OPENMM WORKSHOP

John D. Chodera
MSKCC Computational Biology Program
http://www.choderalab.org

Slides available at http://www.choderalab.org

DISCLOSURES:
- Scientific Advisory Board, Schrödinger

CAUTION: EXPERIMENTAL!

SIMPLIFYING TESTING

openmmtools.testsystems

INSTALL VIA CONDA

```
conda install -c omnia openmmtools
```

INSTALL FROM SOURCE

```
git clone https://github.com/choderalab/openmmtools.git
cd openmmtools
python setup.py install
```

BROWSE GITHUB

http://github.com/choderalab/openmmtools

SIMPLIFYING TESTING

openmmtools.testsystems

```
conda install -c omnia openmmtools
```

- **SOME COMMON TEST SYSTEMS YOU CAN EASILY USE IN YOUR OWN CODE**
- **GOOD EXAMPLES OF HOW TO IMPLEMENT CUSTOM FORCES/SYSTEMS**

HarmonicOscillator - 3D harmonic oscillator
PowerOscillator - higher-order oscillators
Diatom - nitrogen (N_2) molecule
DiatomicFluid - liquid nitrogen
DipolarFluid - fluid of charged diatomic molecules
ConstraintCoupledHarmonicOscillator
HarmonicOscillatorArray
SodiumChlorideCrystal
LennardJonesCluster, LennardJonesFluid
LennardJonesGrid, CustomLennardJonesFluidMixture
WCAFluid
IdealGas, MolecularIdealGas

WaterBox, FourSiteWaterBox,
DischargedWaterBox, FlexibleWaterBox
AlanineDipeptideVacuum
AlanineDipeptideImplicit
DHFRExplicit
LysozymeImplicit
SrcImplicit, SrcExplicit, SrcExplicitReactionField
MethanolBox
CustomGBForceSystem
AMOEBAIonBox
LennardJonesPair
CustomExternalForcesTestSystem

openmmtools.testsystems: https://github.com/choderalab/openmmtools/blob/master/openmmtools/testsystems.py

SIMPLIFYING TESTING

openmmtools.testsystems

```
conda install -c omnia openmmtools
```

system, positions, topology:

```
>>> from openmmtools import testsystems
>>> testsystem = testsystems.HarmonicOscillator()
>>> system    = testsystem.system
>>> positions = testsystem.positions
>>> topology  = testsystem.topology
```

analytical and numerical expectations:

```
>>> print testsystem.analytical_properties
['potential_expectation', 'potential_standard_deviation']
>>> state = testsystems.ThermodynamicState(temperature=300.0*unit.kelvin)
>>> print testsystem.get_potential_expectation(state)
3741.51261205 J/mol
```

openmmtools.testsystems: https://github.com/choderalab/openmmtools/blob/master/openmmtools/testsystems.py

CUSTOM INTEGRATORS

WHAT CAN YOU DO WITH OPENMM?

* EXPERIMENT WITH INTEGRATOR FORMULATIONS
* FOCUS ON THE SCIENCE (INTEGRATOR EQUATIONS), NOT CODE
* RUN FAST ON GPUS
* CHANGE YOUR HAMILTONIAN ON THE FLY
* EASILY ACCUMULATE STATISTICS WHILE INTEGRATING ON THE GPU

User Guide: http://docs.openmm.org/6.3.0/userguide/theory.html#customintegrator
Python API: http://docs.openmm.org/6.3.0/api-python/classsimtk_1_1openmm_1_1openmm_1_1CustomIntegrator.html

CUSTOM INTEGRATORS

VELOCITY VERLET

$$v_{n+1/2} = v_n + 0.5\, f(x_n)/m \quad (1)$$

$$x_{n+1} = x_n + \Delta t\, v_{n+1/2} \quad (2)$$

$$v_{n+1} = v_{n+1/2} + 0.5\, f(x_{n+1})/m \quad (3)$$

```
# Create a velocity Verlet integrator
integrator = openmm.CustomIntegrator(timestep)


integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick (Eq 1)
integrator.addComputePerDof("x", "x+dt*v") # position full kick (Eq 2)


integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick (Eq 3)
```

Example code: https://github.com/choderalab/openmmtools/blob/master/openmmtools/integrators.py#L188

CUSTOM INTEGRATORS

VELOCITY VERLET

$$v_{n+1/2} = v_n + 0.5\, f(x_n)/m \quad (1)$$

$$x_{n+1} = x_n + \Delta t\, v_{n+1/2} \quad (2)$$

$$v_{n+1} = v_{n+1/2} + 0.5\, f(x_{n+1})/m \quad (3)$$

```
# Create a velocity Verlet integrator
integrator = openmm.CustomIntegrator(timestep)
integrator.addPerDofVariable("x1", 0) # holds positions prior to constraint correction
integrator.addUpdateContextState() # allow other forces (e.g. MonteCarloBarostat) to update system
integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick (Eq 1)
integrator.addComputePerDof("x", "x+dt*v") # position full kick (Eq 2)
integrator.addComputePerDof("x1", "x") # store positions before position constraints are applied
integrator.addConstrainPositions() # apply position constraints (CCMA)
integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick (Eq 3)
integrator.addComputePerDof("v", "v+(x-x1)/dt") # correct velocities for position constraints (RATTLE)
integrator.addConstrainVelocities() # constrain velocities (RATTLE)
```

Example code: https://github.com/choderalab/openmmtools/blob/master/openmmtools/integrators.py#L188

CUSTOM INTEGRATORS

ANDERSEN THERMOSTAT WITH WEAK COLLISIONS

```python
# Integrator initialization.
integrator = openmm.CustomIntegrator(timestep)
integrator.addGlobalVariable("kT", kT)  # thermal energy
integrator.addGlobalVariable("p_collision", timestep * collision_rate)  # per-particle collision probability per timestep
integrator.addPerDofVariable("sigma_v", 0)  # velocity distribution stddev for Maxwell-Boltzmann (computed later)
integrator.addPerDofVariable("collision", 0)  # 1 if collision has occured this timestep, 0 otherwise
integrator.addPerDofVariable("x1", 0)  # for constraints

# Andersen thermostat stochastic velocity update
integrator.addComputePerDof("sigma_v", "sqrt(kT/m)")
integrator.addComputePerDof("collision", "step(p_collision-uniform)")  # if collision has occured this timestep, 0 otherwise
integrator.addComputePerDof("v", "(1-collision)*v + collision*sigma_v*gaussian")  # randomize velocities of collided particles

# Standard Velocity Verlet step
integrator.addUpdateContextState() # allow other forces (e.g. MonteCarloBarostat) to update system
integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick
integrator.addComputePerDof("x", "x+dt*v") # position full kick
integrator.addComputePerDof("x1", "x") # store positions before position constraints are applied
integrator.addConstrainPositions() # apply position constraints (CCMA)
integrator.addComputePerDof("v", "v+0.5*dt*f/m") # velocity half kick
integrator.addComputePerDof("v", "v+(x-x1)/dt") # correct velocities for position constraints (RATTLE)
integrator.addConstrainVelocities() # constrain velocities (RATTLE)
```

Example code: https://github.com/choderalab/openmmtools/blob/master/openmmtools/integrators.py#L188

CUSTOM INTEGRATORS

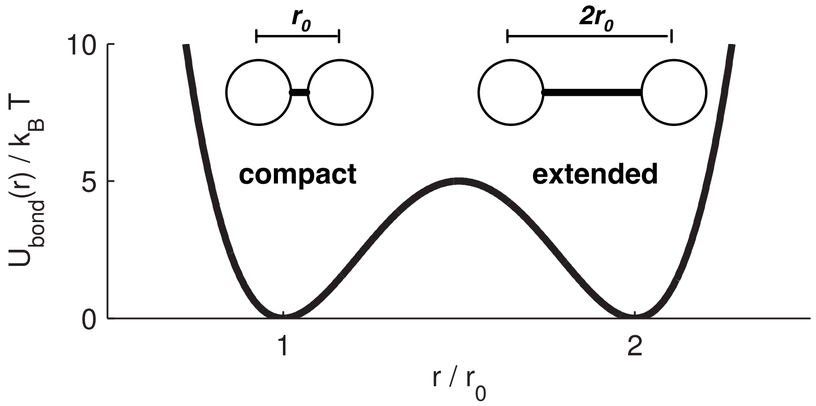
HYBRID MONTE CARLO

```
# Integrator initialization
integrator = openmm.CustomIntegrator(timestep)
integrator.addPerDofVariable("xold", 0) # old positions
integrator.addGlobalVariable("Eold", 0) # old total energy
integrator.addPerDofVariable("x1", 0) # positions prior to constraint correction
integrator.addPerDofVariable("sigma", 0) # Gaussian standard deviation for Maxwell-Boltzmann distribution
integrator.addGlobalVariable("accept", 0) # determines whether move is accepted or not
integrator.addGlobalVariable("kT", kT.value_in_unit_system(unit.md_unit_system)) # thermal energy

# Draw a new velocity from the Maxwell-Boltzmann distribution
integrator.addComputePerDof("v","sigma*gaussian; sigma = sqrt(kT/m)")
integrator.addConstrainVelocities() # constrain velocities (RATTLE)

integrator.addComputeSum("ke","0.5*m*v*v") # compute kinetic energy
integrator.addComputeGlobal("Eold","ke + energy") # compute old total energy
integrator.addComputePerDof("xold","x") # store old positions

# standard Velocity Verlet step goes here
. . .

# Accept/reject step
integrator.addComputeSum("ke","0.5*m*v*v") # compute kinetic energy
integrator.addComputeGlobal("accept","step(exp(-(Enew-Eold)/kT) - uniform); Enew = ke + energy")
integrator.addComputePerDof("x", "x*accept + xold*(1-accept)") # keep old or new configuration
```

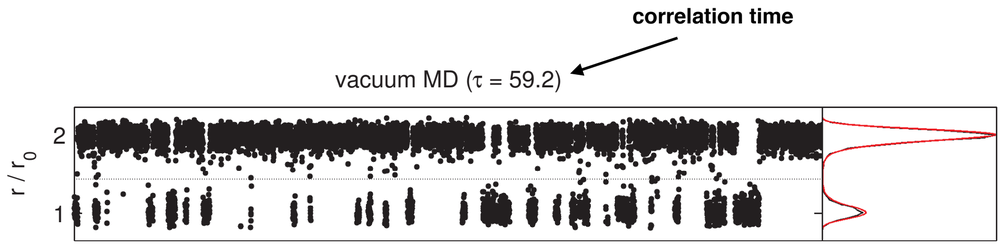
CUSTOM INTEGRATORS

NONEQUILIBRIUM INTEGRATORS

MIXING IN CLEVER MONTE CARLO MOVES CAN SPEED SAMPLING

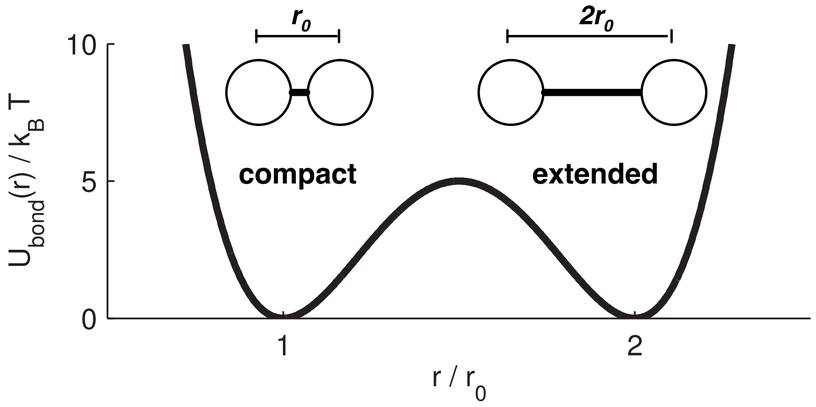


“Geisslerene”



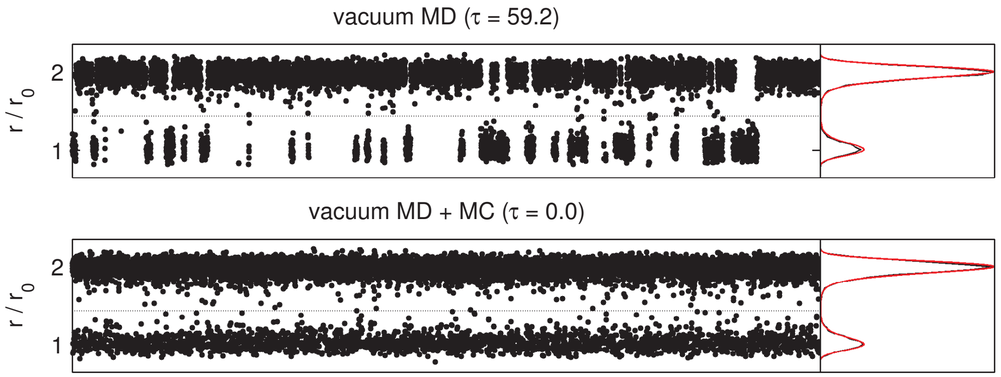
Straightforward molecular dynamics can be slow.

MONTE CARLO MOVES CAN SPEED BARRIER CROSSING IN VACUUM...

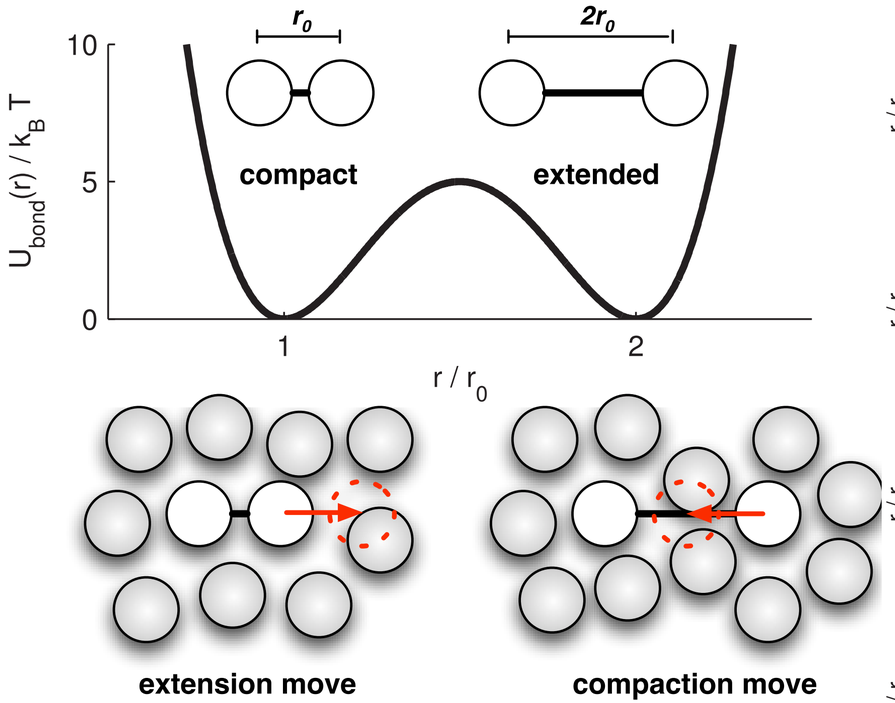


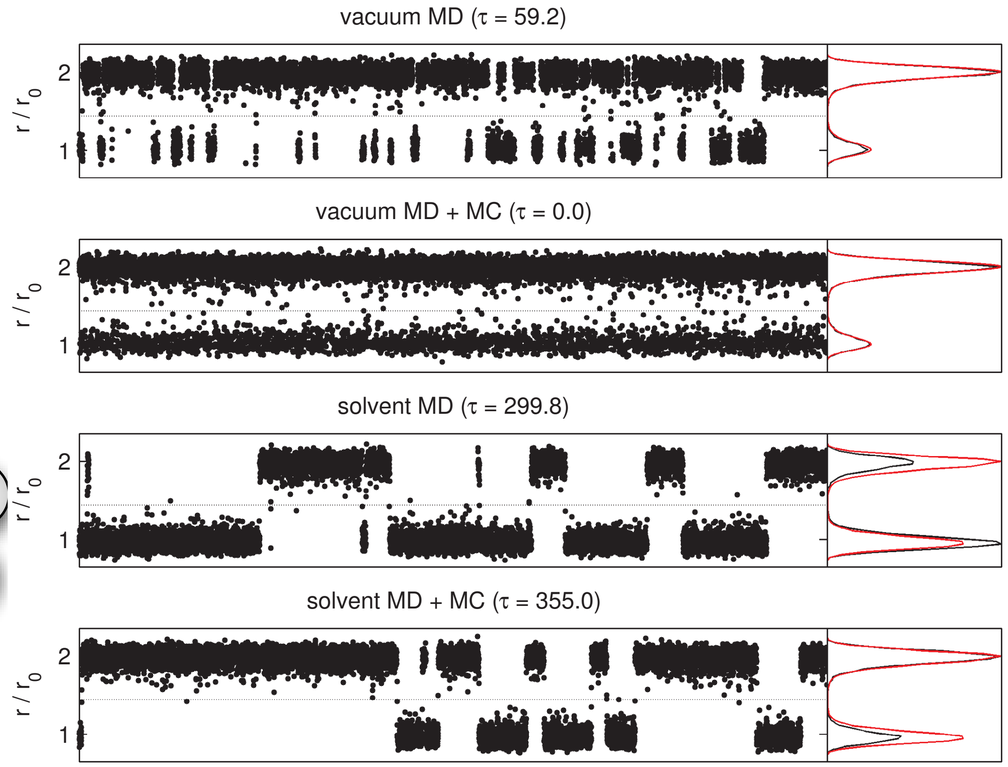
extension move

compaction move



...BUT MONTE CARLO MOVES ARE USELESS IN DENSE SOLVENT





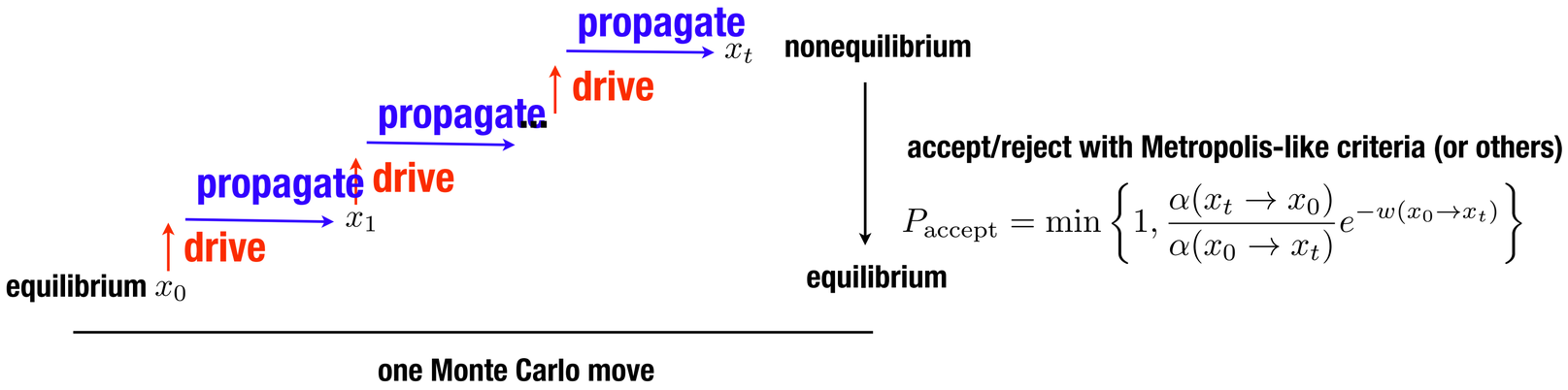
WHAT IF WE DRIVE SOME DEGREES OF FREEDOM, BUT PROPAGATE THE REST? CAN WE CORRECT FOR THIS AND GET BACK TO EQUILIBRIUM?

Algorithm:

Drive some degrees of freedom or thermodynamic parameters in small steps, accumulating work

In between driven steps, **propagate** others using Metropolis MC or molecular dynamics

Accept or reject final configuration with Metropolis-like criterion



Follows earlier ideas by Manuel Athenes (work-bias Monte Carlo), Harry Stern (constant pH simulation), Chris Jarzynski (switching replica temperatures), and Jerome Nilmeier (approximate).

EQUILIBRIUM MONTE CARLO WITH NONEQUILIBRIUM MOVES

$x_t \in \mathcal{X}$ **configurations in some state space**

$\alpha_t(x, y)$ **perturbation kernel** $\sum_y \alpha_t(x, y) = 1$

$K_t(x, y)$ **propagation kernel** $\sum_x \pi_t(x) K_t(x, y) = \pi_t(y)$

$\Lambda \equiv \{\alpha_0, K_0, \ldots, \alpha_T, K_T\}$ **switching protocol**

$X \equiv \{x_0, x_1^*, x_1, \ldots, x_T\}$ **trajectory**

$\tilde{x}$ **is momentum-reversed** x

$\alpha_t(x, y) > 0 \Leftrightarrow \alpha_t(y, x) > 0$

$K_t(x, y) > 0 \Leftrightarrow K_t(y, x) > 0$

$\tilde{\Lambda}$ **is time-reversed** Λ

$\tilde{X}$ **is time-reversed** X

forward process $x_0 \xrightarrow{\alpha_1} x_1^* \xrightarrow{K_1} x_1 \longrightarrow \cdots \longrightarrow x_{T-1} \xrightarrow{\alpha_T} x_T^* \xrightarrow{K_T} x_T.$

reverse process $\tilde{x}_T \xrightarrow{K_T} \tilde{x}_T^* \xrightarrow{\alpha_T} \tilde{x}_{T-1} \longrightarrow \cdots \longrightarrow \tilde{x}_1 \xrightarrow{K_1} \tilde{x}_1^* \xrightarrow{\alpha_1} \tilde{x}_0.$

$P(\tilde{\Lambda}|\tilde{x}_T) > 0 \Leftrightarrow P(\Lambda|x_0 > 0)$

both must be selected with nonzero probability!

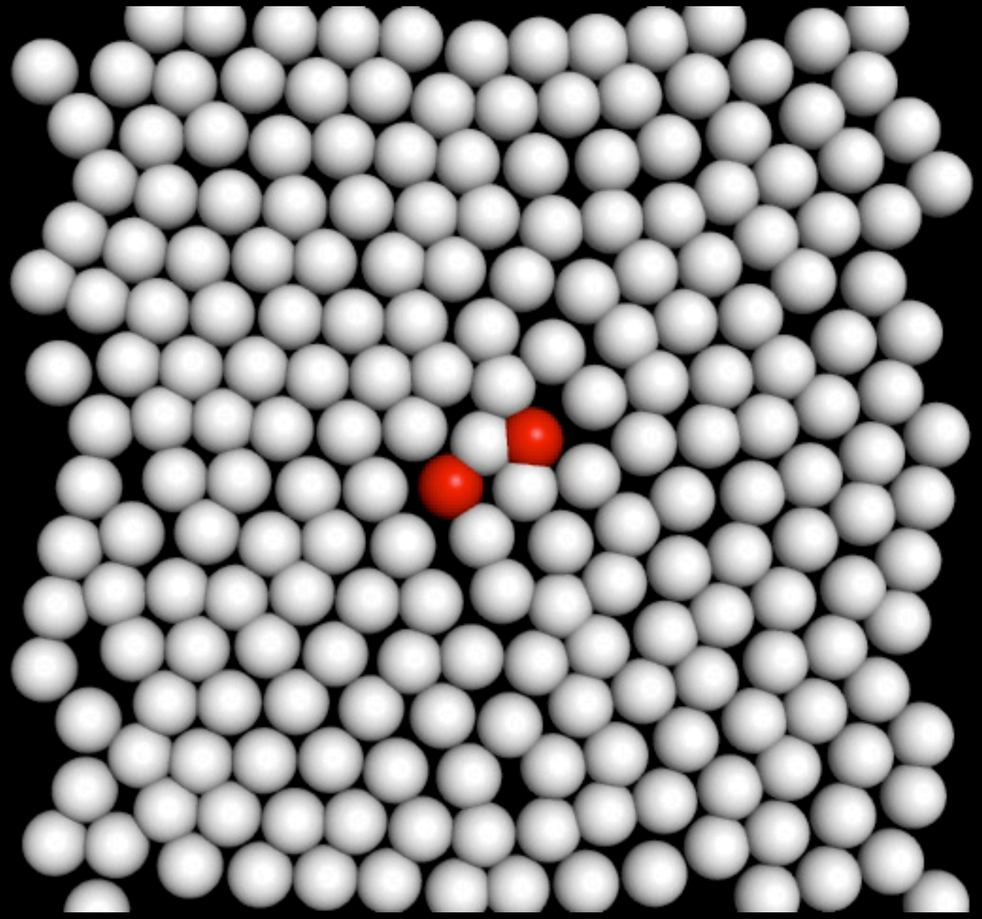
Enforce strict "pathwise" form of detailed balance (which also ensures detailed balance is satisfied):

$$\underbrace{A(X|\Lambda)}_{\text{acceptance probability}} \underbrace{\Pi(X|x_0, \Lambda)}_{\text{path generation probability}} \underbrace{P(\Lambda|x_0, \lambda_0)}_{\text{protocol selection}} \underbrace{\pi(x_0, \lambda_0)}_{\text{equilibrium}} = \underbrace{A(\tilde{X}|\tilde{\Lambda})}_{\text{acceptance probability}} \underbrace{\Pi(\tilde{X}|\tilde{x}_T, \tilde{\Lambda})}_{\text{path generation probability}} \underbrace{P(\tilde{\Lambda}|\tilde{x}_T, \lambda_T)}_{\text{protocol selection}} \underbrace{\pi(\tilde{x}_T, \lambda_T)}_{\text{equilibrium}}$$

Result is a general acceptance criteria for any nonequilibrium perturbation:

$$A(X|\Lambda) = \min\left\{1, \underbrace{\frac{\pi(x_T, \lambda_T)}{\pi(x_0, \lambda_0)}}_{\text{equilibrium}} \underbrace{\frac{P(\tilde{\Lambda}|\tilde{x}_T, \lambda_T)}{P(\Lambda|x_0, \lambda_0)}}_{\text{selection}} \underbrace{\frac{\tilde{\alpha}(\tilde{X})}{\alpha(X)}}_{\text{perturbation}} \underbrace{e^{-\Delta S(X)}}_{\text{action}}\right\}$$

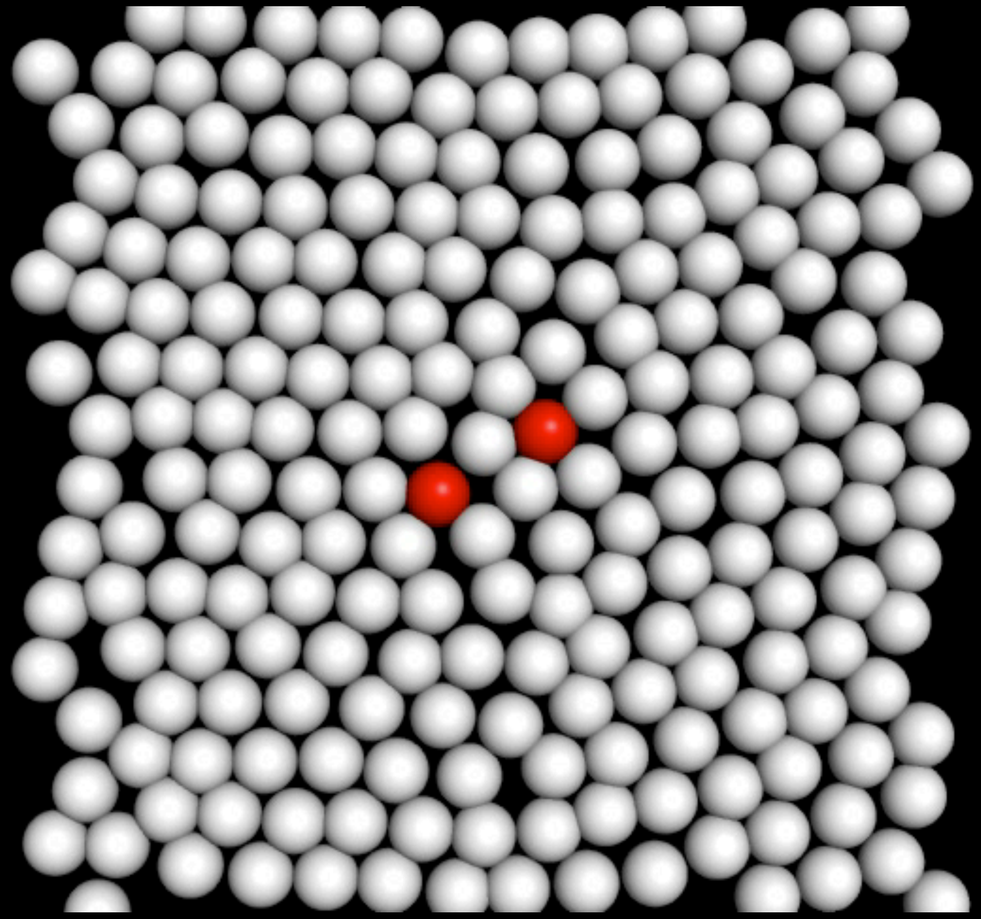
NONEQUILIBRIUM MC DIMER MOVES IN A 2D WCA FLUID



Bad solvent overlap:
Always rejected :(

Instantaneous Metropolis Monte Carlo

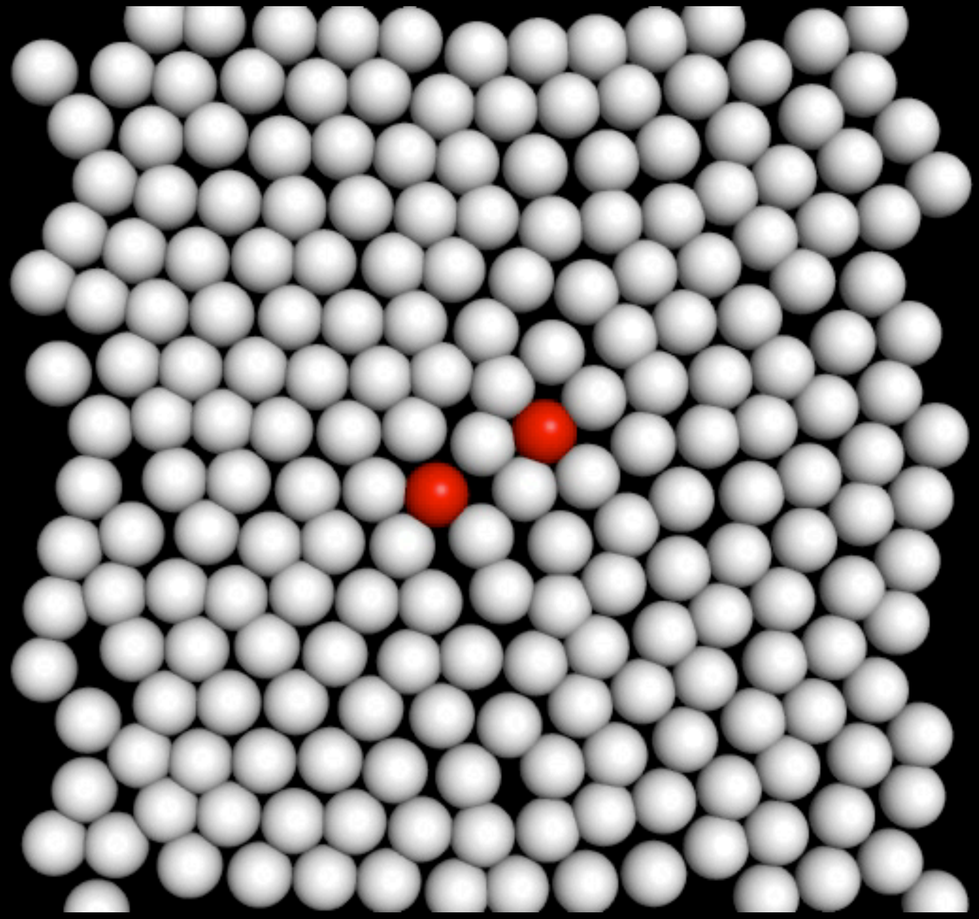
NONEQUILIBRIUM MC DIMER MOVES IN A 2D WCA FLUID



Solvent is “squeezed” out
but significantly perturbed :/

32 switching steps with velocity Verlet dynamics as “propagation kernel”

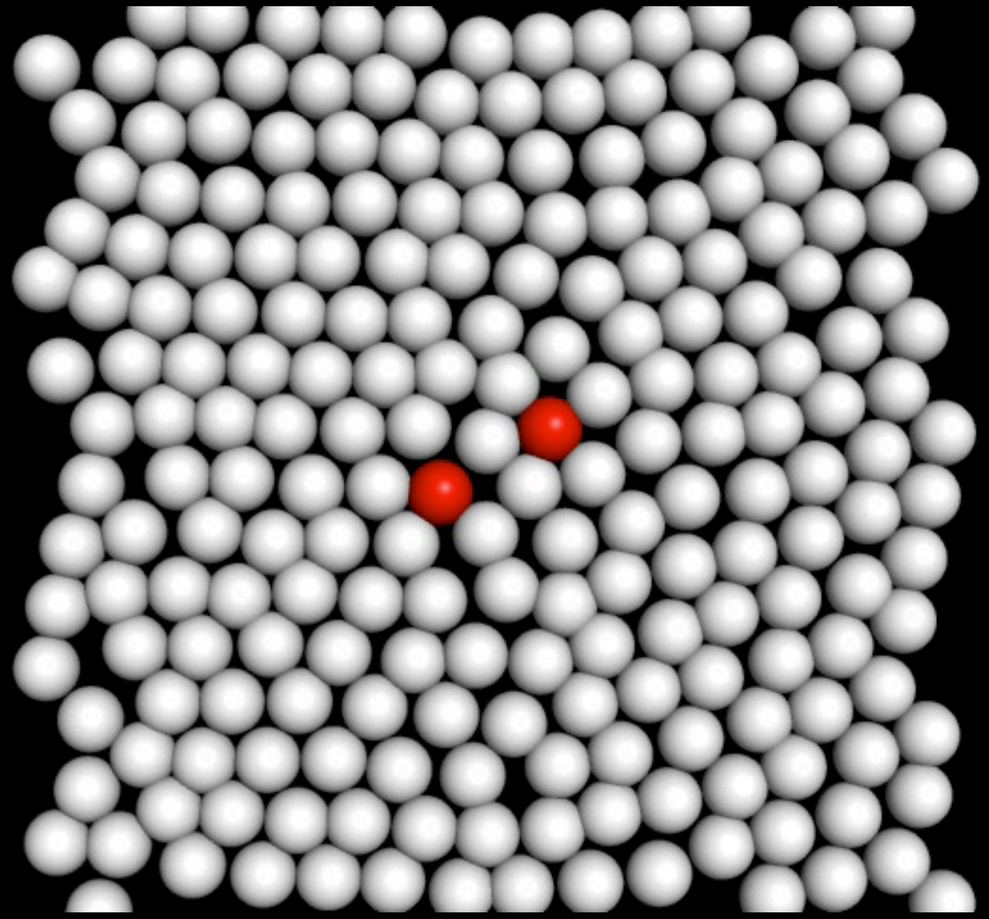
NONEQUILIBRIUM MC DIMER MOVES IN A 2D WCA FLUID



Solvent is “squeezed” out
with time for reorganization :)

64 switching steps with velocity Verlet dynamics as “propagation kernel”

NONEQUILIBRIUM MC DIMER MOVES IN A 2D WCA FLUID

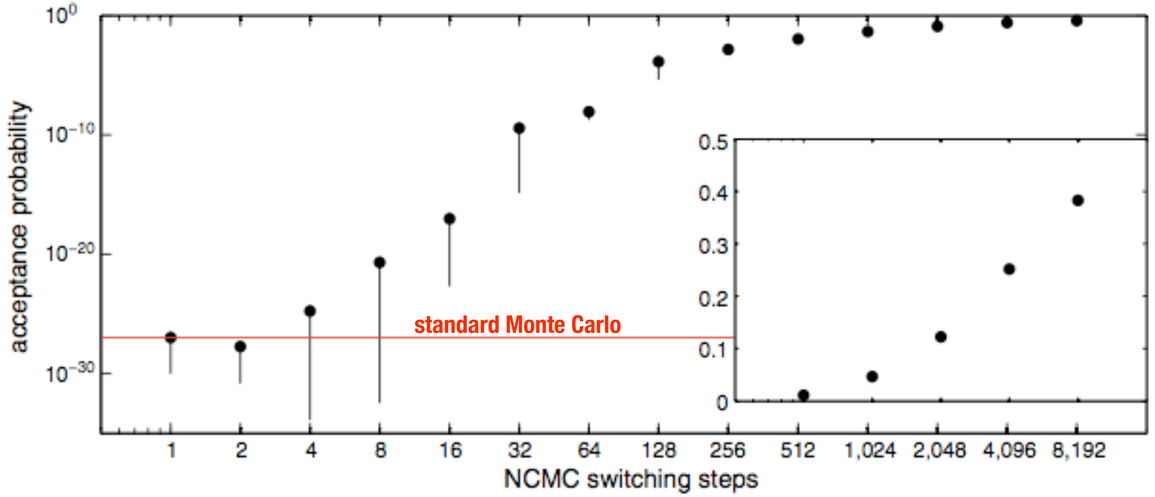


Solvent is “squeezed” out close to reversible limit; high acceptance rates! :D

256 switching steps with velocity Verlet dynamics as “propagation kernel”

ACCEPTANCE PROBABILITY CAN BE ASTRONOMICALLY BOOSTED

Acceptance probability can be increased from 10^{-27} to 10^{0} (38%)!



PNAS PLUS

Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation

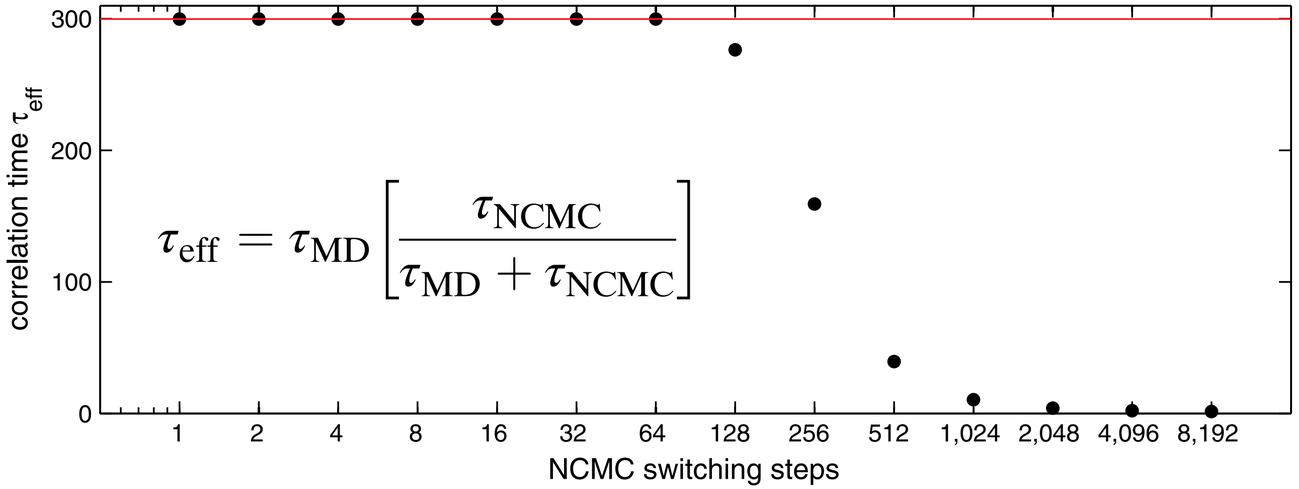
Jerome P. Nilmeier[a], Gavin E. Crooks[b], David D. L. Minh[c], and John D. Chodera[d,1]

[a]Biosciences and Biotechnology Division, Physical and Life Sciences Directorate, Lawrence Livermore National Laboratory, Livermore, CA 94550; [b]Physical Biosciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720; [c]Biosciences Division, Argonne National Laboratory, Argonne, IL 60439; and [d]California Institute for Quantitative Biosciences (QB3), University of California, Berkeley, CA 94720

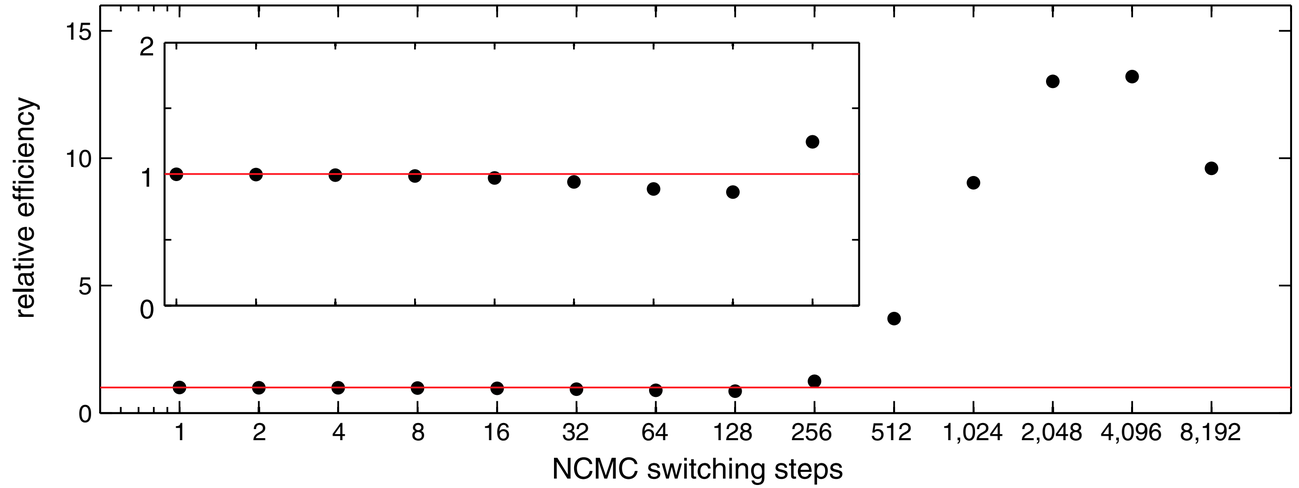
Full 3D system
216 WCA particles
Reduced density $\rho\sigma^3 = 0.96$
Reduced temperature $k_B T/\epsilon = 0.824$
5 kT barrier

OVERALL EFFICIENCY GAIN CAN STILL BE LARGE, EVEN WHEN THE EXTRA WORK REQUIRED FOR NCMC SWITCHING IS INCLUDED

measure of number of iterations to reduce correlation to 1/e



ratio of computer time required to generate one uncorrelated sample without and with NCMC



CUSTOM INTEGRATORS

NONEQUILIBRIUM INTEGRATORS

```
# Create an empty system object.
 system = openmm.System()

 # Add the particle to the system.
 system.addParticle(mass)

 # Add a restraining potential centered at the origin.
 energy_expression = '(K/2.0) * (x^2 + y^2 + z^2);'
 energy_expression += 'K = testsystems_HarmonicOscillator_K;'
 force = openmm.CustomExternalForce(energy_expression)
 force.addGlobalParameter('testsystems_HarmonicOscillator_K', \
                                  K.value_in_unit_system(unit.md_unit_system))
 force.addParticle(0, [])
 system.addForce(force)
```

CUSTOM INTEGRATORS

NONEQUILIBRIUM INTEGRATORS

```
from simtk import openmm, unit
from openmmtools import testsystems

timestep = 2.0 * unit.femtoseconds
temperature = 300.0 * unit.kelvin
nsteps = 100 # number of steps over which switching occurs
delta_K =  900.0 # change in spring constant K over nsteps (kJ/nm**2)

# Create a nonequilibrium integrator based on velocity Verlet.
integrator = openmm.CustomIntegrator(timestep)
integrator.addComputeGlobal("testsystems_HarmonicOscillator_K", "testsystems_HarmonicOscillator_K + %f" %
     (delta_K/float(nsteps))) # update spring constant
# Standard Velocity Verlet here
# …

# Run dynamics of a harmonic oscillator that changes spring constant dynamically
testsystem = testsystems.HarmonicOscillator()
context = openmm.Context(testsystem.system, integrator)
context.setPositions(testsystem.positions)
context.setVelocitiesToTemperature(temperature)
context.setParameter('testsystems_HarmonicOscillator_K', 100.0) # reset dynamic parameter that will be switched (kJ/nm**2)
state = context.getState(getEnergy=True)
initial_total_energy = state.getPotentialEnergy() + state.getKineticEnergy()
integrator.step(nsteps) # perform nonequilibrium switching
state = context.getState(getEnergy=True)
final_total_energy = state.getPotentialEnergy() + state.getKineticEnergy()
print "nonequilibrium work: %s" % str(final_total_energy - initial_total_energy)
```

CUSTOM INTEGRATORS

openmmtools.integrators

- **SOME COMMON INTEGRATORS YOU CAN EASILY USE IN YOUR OWN CODE**
- **GOOD EXAMPLES OF HOW TO IMPLEMENT COMPLEX THINGS**

MTSIntegrator - RESPA multiple time step integrator
GradientDescentMinimization - simple minimizer
VelocityVerletIntegrator - velocity Verlet
AndersenVelocityVerletIntegrator - Andersen thermostat with per-particle collisions
MetropolisMonteCarloIntegrator - example of simple Metropolis Monte Carlo for liquids
HMCIntegrator - hybrid Monte Carlo
GHMCIntegrator - generalized hybrid Monte Carlo (Metropolized Langevin)
VVVRIntegrator - velocity Verlet with velocity randomization (Langevin based on symplectic inner step)

```
conda install -c omnia openmmtools
```

WHAT OTHER INTEGRATORS WOULD BE USEFUL TO YOU?

openmmtools.integrators: https://github.com/choderalab/openmmtools/blob/master/openmmtools/integrators.py

CUSTOM OPENMM FORCES ALLOW EXPERIMENTATION WITH ALCHEMICAL DEFINITIONS

```
if isinstance(reference_force, openmm.NonbondedForce):
    # CustomNonbondedForce will handle softcore interactions with ligand.
    energy_expression = “4*epsilon*lambda*x*(x-1.0);" # softcore potential
    energy_expression += "x = 1.0/(alpha*(1.0-lambda) + (r/sigma)^6);"
    energy_expression += "epsilon = sqrt(epsilon1*epsilon2);” # Lorentz-Berthelot combining rules
    energy_expression += "sigma = 0.5*(sigma1 + sigma2);” # Lorentz-Berthelot combining rules
    energy_expression += "lambda = lambda1*lambda2;" # alchemical combining rule

    force = openmm.CustomNonbondedForce(energy_expression)
    alpha = 0.5 # softcore parameter
    force.addGlobalParameter("alpha", alpha);
    force.addPerParticleParameter("sigma")
    force.addPerParticleParameter("epsilon")
    force.addPerParticleParameter("lambda");
    for particle_index in range(reference_force.getNumParticles()):
        # Retrieve parameters.
        [charge, sigma, epsilon] = reference_force.getParticleParameters(particle_index)
        # Alchemically modify parameters.
        if particle_index in ligand_atoms:
            force.addParticle([sigma, epsilon, vdw_lambda])
        else:
            force.addParticle([sigma, epsilon, 1.0])
    for exception_index in range(reference_force.getNumExceptions()):
        # Retrieve parameters.
        [iatom, jatom, chargeprod, sigma, epsilon] = reference_force.getExceptionParameters(exception_index)
        # All exceptions are handled by NonbondedForce, so we exclude all these here.
        force.addExclusion(iatom, jatom)
    force.setNonbondedMethod( reference_force.getNonbondedMethod() )
    force.setCutoffDistance( reference_force.getCutoffDistance() )
    system.addForce(force)
```

CUSTOM OPENMM FORCES ALLOW EXPERIMENTATION WITH ALCHEMICAL DEFINITIONS

```
conda install -c omnia alchemy
```

```python
from alchemy import AbsoluteAlchemicalFactory, AlchemicalState
from openmmtools import testsystems

# Create a reference system.
waterbox = testsystems.WaterBox()
[reference_system, positions] = [waterbox.system, waterbox.positions]

# Create a factory to produce alchemical intermediates.
factory = AbsoluteAlchemicalFactory(reference_system, ligand_atoms=[0, 1, 2])

# Create a perturbed systems using this protocol.
alchemical_state = AlchemicalState()
alchemical_system = factory.createPerturbedSystem(alchemical_state)

# Perturb this system.
alchemical_state = AlchemicalState(lambda_sterics=0.90, lambda_electrostatics=0.90)
factory.perturbSystem(alchemical_system, alchemical_state)
```

CUSTOM OPENMM FORCES ALLOW EXPERIMENTATION WITH ALCHEMICAL DEFINITIONS

```
conda install -c omnia alchemy
```

```python
from alchemy import AbsoluteAlchemicalFactory
from openmmtools import testsystems

# Create a reference system.
complex = testsystems.LysozymeImplicit()
[reference_system, positions] = [complex.system, complex.positions]

# Create a factory to produce alchemical intermediates.
receptor_atoms = range(0,2603) # T4 lysozyme L99A
ligand_atoms = range(2603,2621) # p-xylene
factory = AbsoluteAlchemicalFactory(reference_system, ligand_atoms=ligand_atoms)

# Get the default protocol for 'denihilating' in complex in explicit solvent.
protocol = factory.defaultComplexProtocolImplicit()

# Create the perturbed systems using this protocol.
systems = factory.createPerturbedSystems(protocol)
```

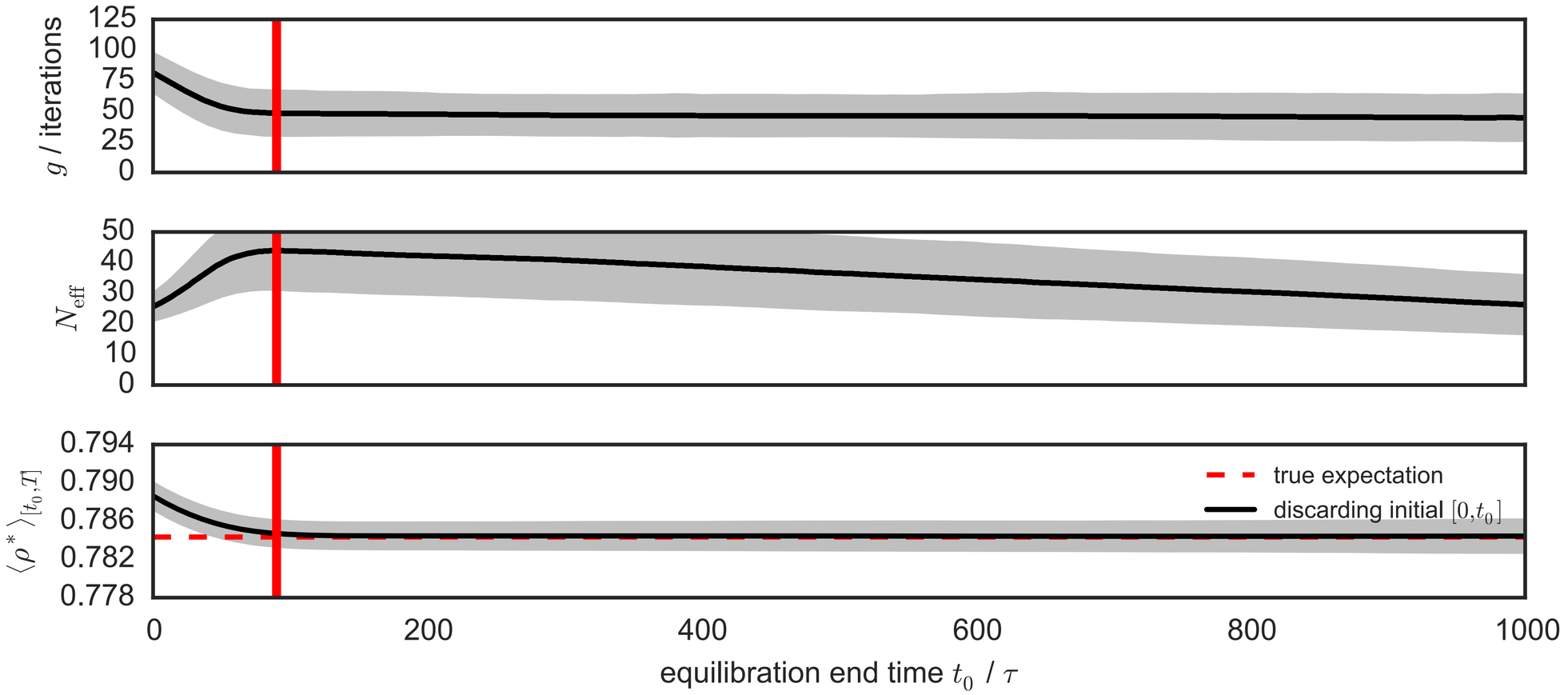
Omnia

empowering molecular simulation

OPEN SOURCE, HIGH PERFORMANCE, HIGH USABILITY TOOLKITS FOR PREDICTIVE BIOMOLECULAR SIMULATION.

http://omnia.md

OMNIA ENABLES REPRODUCIBLE SCIENCE



bioRχiv beta
THE PREPRINT SERVER FOR BIOLOGY

A simple method for automated equilibration detection in molecular simulations

John D Chodera

doi: http://dx.doi.org/10.1101/021659

OMNIA ENABLES REPRODUCIBLE SCIENCE

```
#!/bin/bash

# Create conda environment with exact versions of tools needed to reproduce paper.
if [ ! -d conda-env ]; then
   conda config --add channels http://conda.binstar.org/omnia
   conda create --yes --quiet -p conda-env python=2.7 openmmtools=0.7.0 openmm=6.2 matplotlib=1.4 \
       pymbar=3.0.0.beta2 netCDF4
fi
source activate ./conda-env

# Run simulations.
python simulate.py

# Analyze simulation data to generate figures.
if [ ! -e figures ]; then
    mkdir figures
fi
python analyze-1.py
python analyze-2.py

# Deactivate conda environment.
source deactivate
```

bioRχiv beta
THE PREPRINT SERVER FOR BIOLOGY

A simple method for automated equilibration detection in molecular simulations

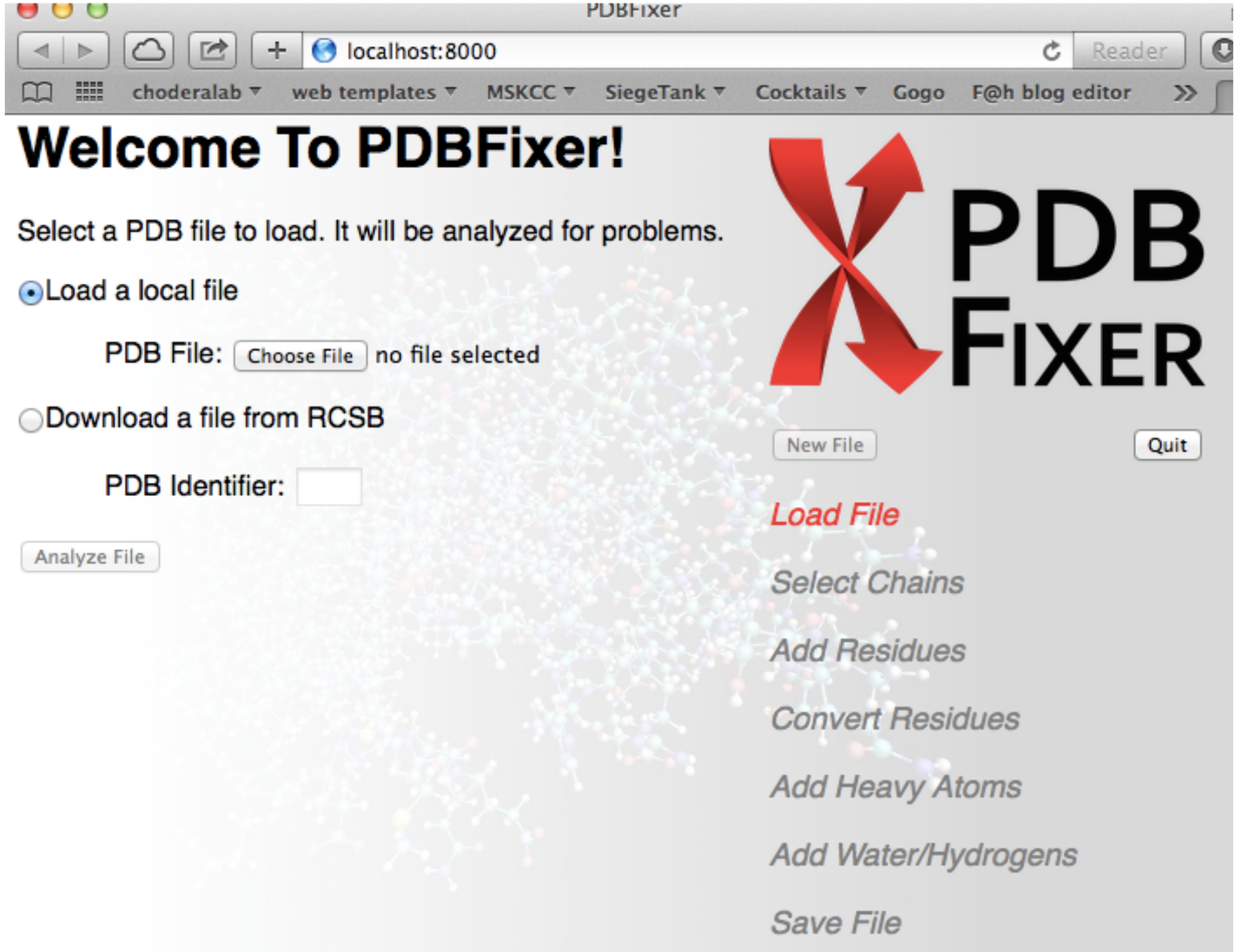
John D Chodera
doi: http://dx.doi.org/10.1101/021659


```
from pdbfixer import PDBFixer
from simtk.openmm.app import PDBFile
fixer = PDBFixer(filename='myfile.pdb')
fixer.findMissingResidues()
fixer.findNonstandardResidues()
fixer.replaceNonstandardResidues()
fixer.findMissingAtoms()
fixer.addMissingAtoms()
fixer.removeHeterogens(True)
fixer.addMissingHydrogens(7.0)
fixer.addSolvent(fixer.topology.getUnitCellDimensions())
PDBFile.writeFile(fixer.topology, fixer.positions, open('output.pdb', 'w'))
```

http://github.com/pandegroup/pdbfixer

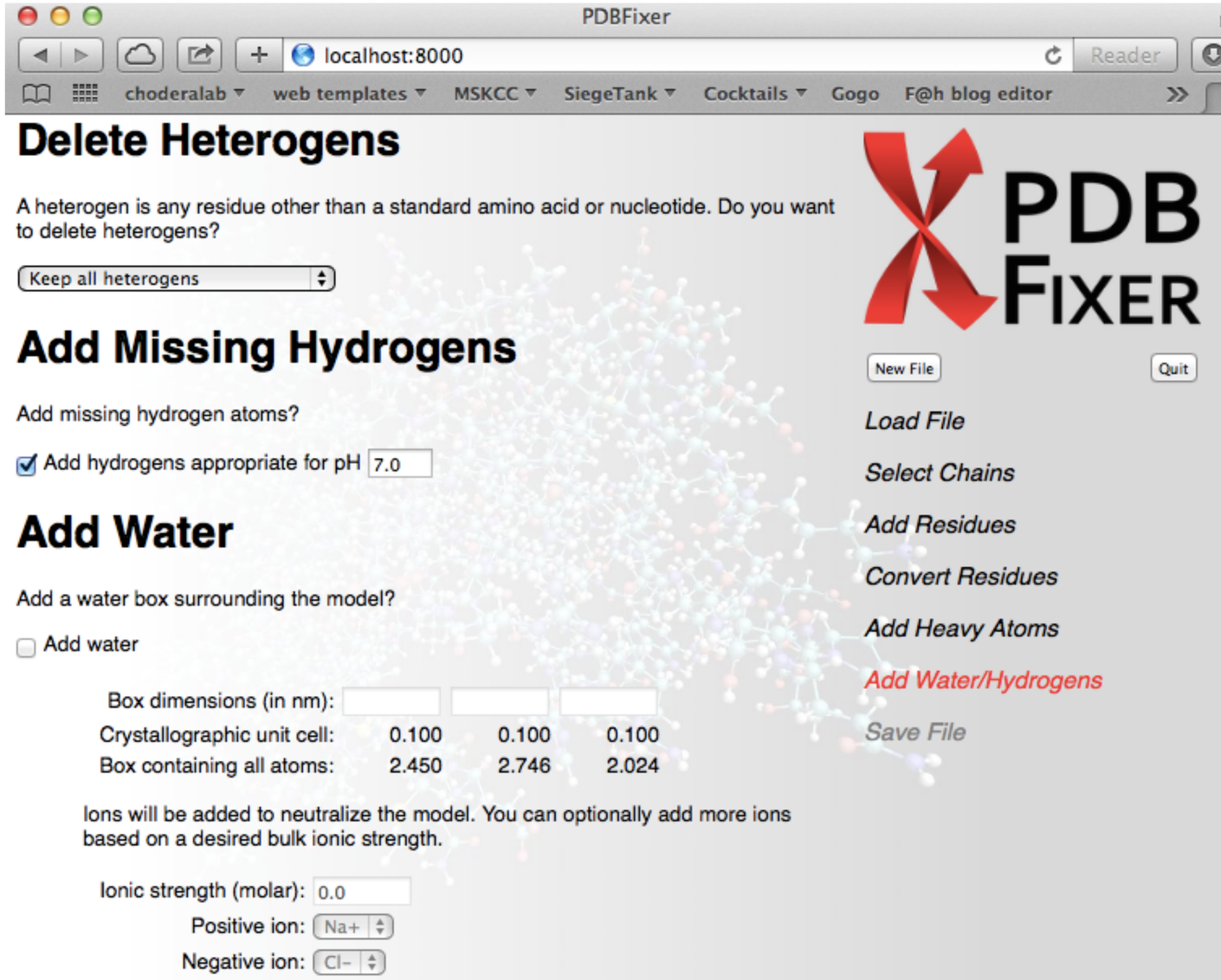

```
Usage: pdbfixer
       pdbfixer [options] filename

When run with no arguments, it launches the user interface.  If any arguments are specified, it runs in command line mode.

Options:
  -h, --help            show this help message and exit
  --pdbid=PDBID         PDB id to retrieve from RCSB [default: None]
  --url=URL             URL to retrieve PDB from [default: None]
  --output=FILENAME     output pdb file [default: output.pdb]
  --add-atoms=ATOMS     which missing atoms to add: all, heavy, hydrogen, or
                        none [default: all]
  --keep-heterogens=OPTION
                        which heterogens to keep: all, water, or none
                        [default: all]
  --replace-nonstandard
                        replace nonstandard residues with standard equivalents
  --add-residues        add missing residues
  --water-box=X Y Z     add a water box. The value is the box dimensions in nm
                        [example: --water-box=2.5 2.4 3.0]
  --ph=PH               the pH to use for adding missing hydrogens [default:
                        7.0]
  --positive-ion=ION    positive ion to include in the water box: Cs+, K+,
                        Li+, Na+, or Rb+ [default: Na+]
  --negative-ion=ION    negative ion to include in the water box: Cl-, Br-,
                        F-, or I- [default: Cl-]
  --ionic-strength=STRENGTH
                        molar concentration of ions to add to the water box
                        [default: 0.0]
```


```
[LSKI1497:~] choderaj% pdbfixer
PDBFixer running: http://localhost:8000
```

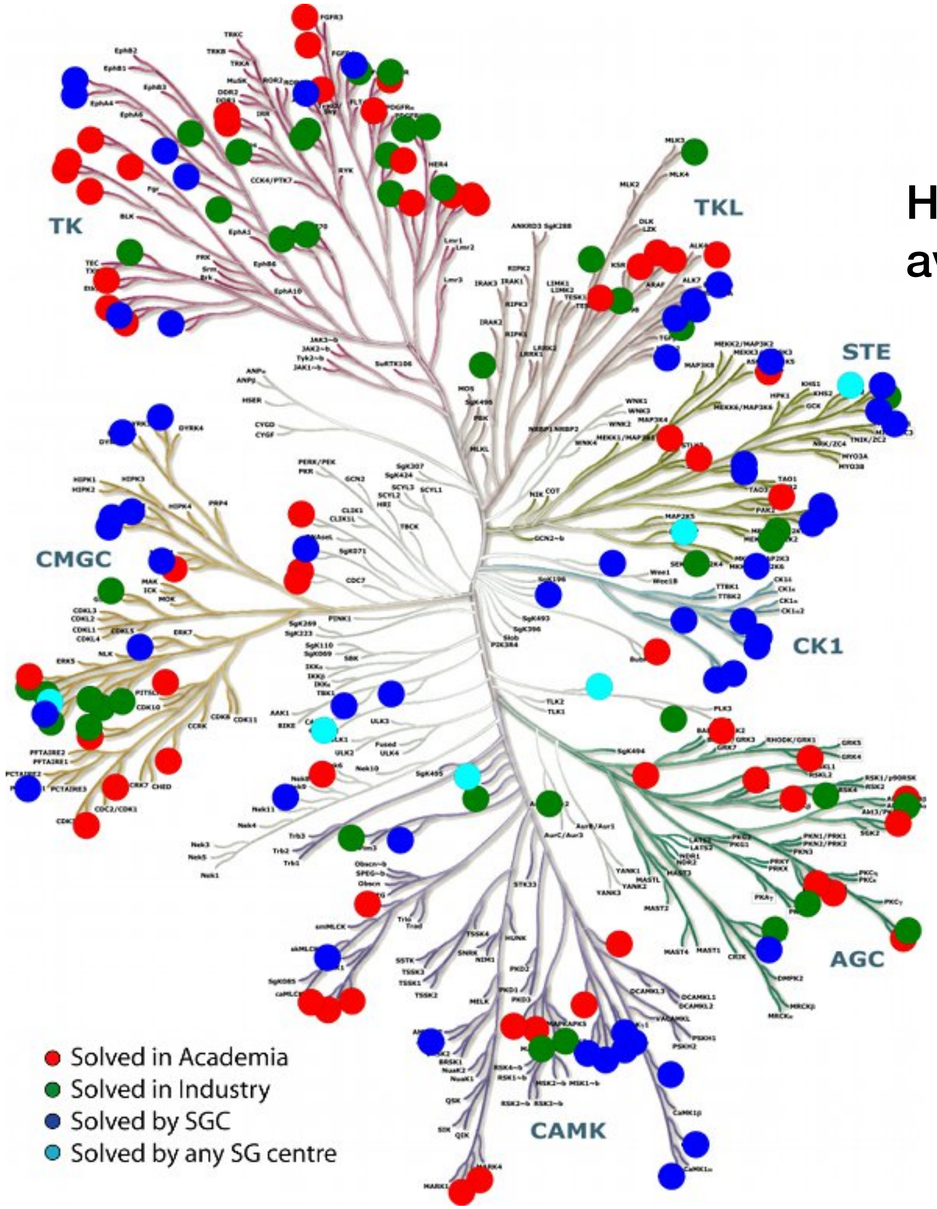


PDB FIXER

```
[LSKI1497:~] choderaj% pdbfixer
PDBFixer running: http://localhost:8000
```



STRUCTURAL DATA ON HUMAN KINASES IS INCOMPLETE



Human kinases with available structural data

http://www.sgc.ox.ac.uk/research/kinases/

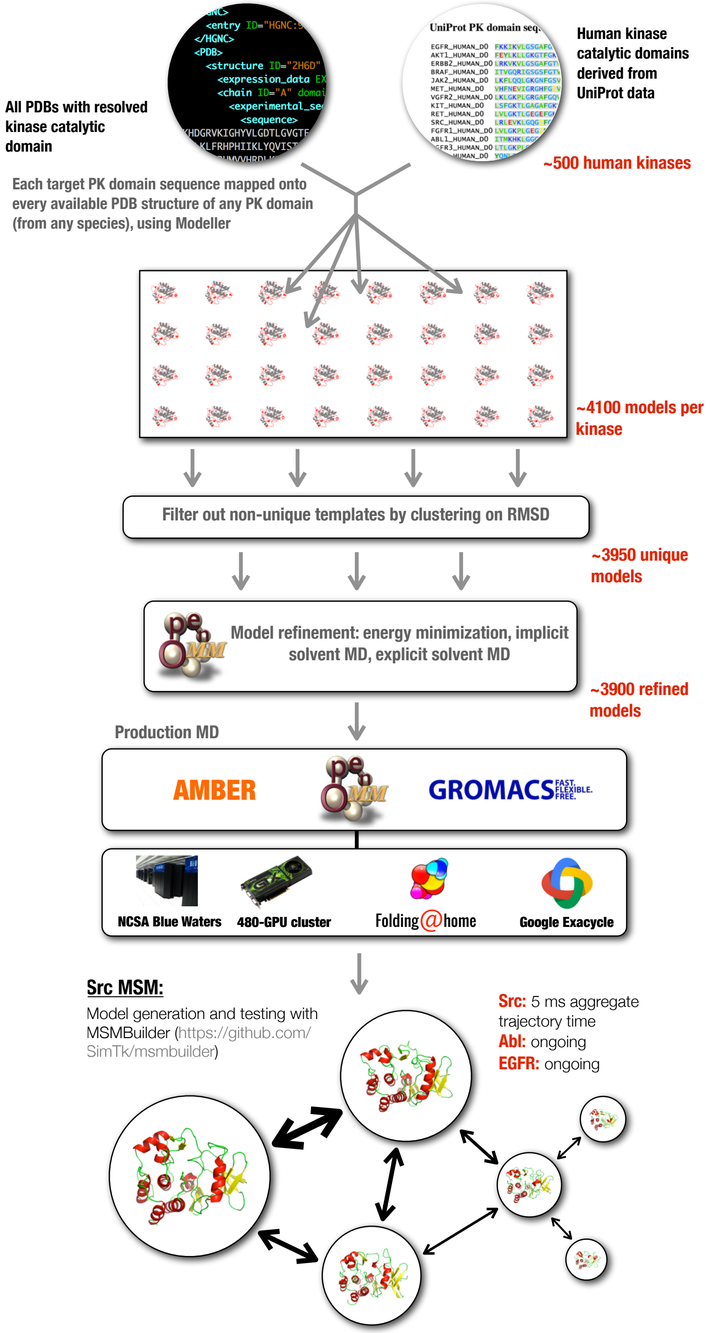
BUILDING AN ATLAS OF KINASE CONFORMATIONS AND ENERGETICS

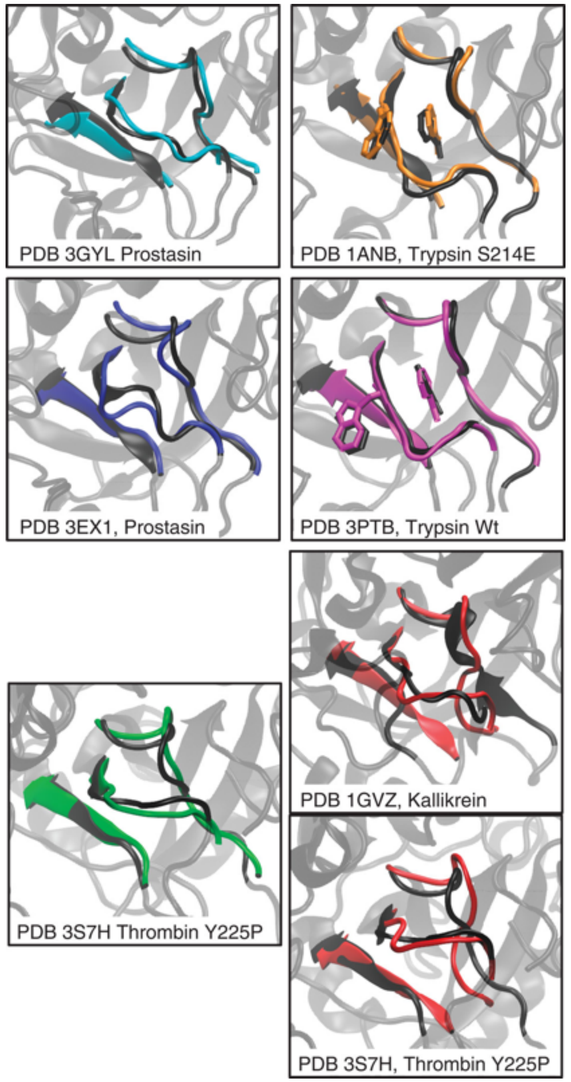
Daniel Parton
Postdoc

Patrick Grinaway
PBSB student

Kyle Beauchamp
Postdoc

Jan-Hendrik Prinz
Postdoc





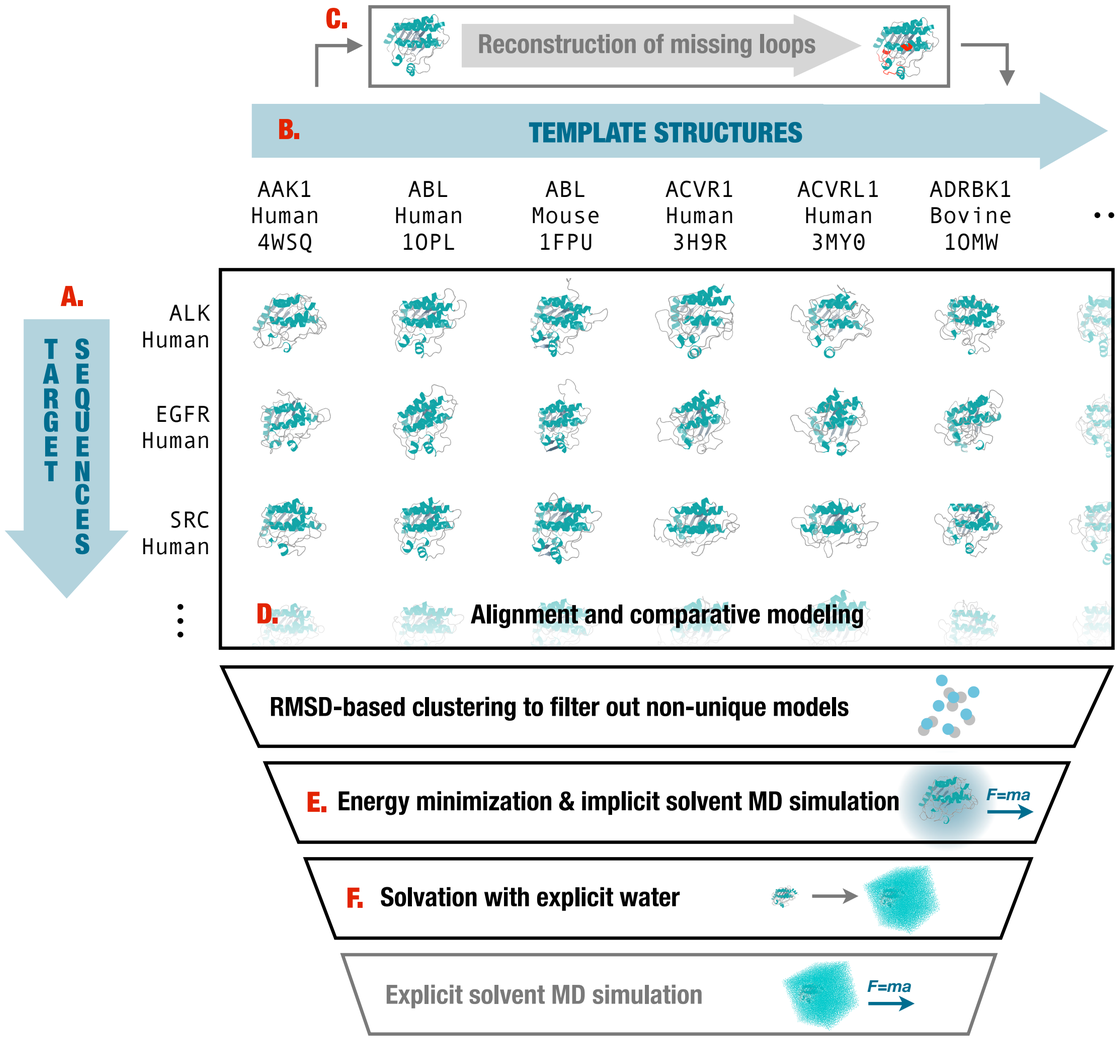
Plattner and Noé. Nature Comm. 6:7653, 2015.

http://github.com/choderalab/ensembler

ENSEMBLER

AUTOMATED MODELING AND PREPARATION FOR SUPERFAMILY-SCALE SIMULATIONS

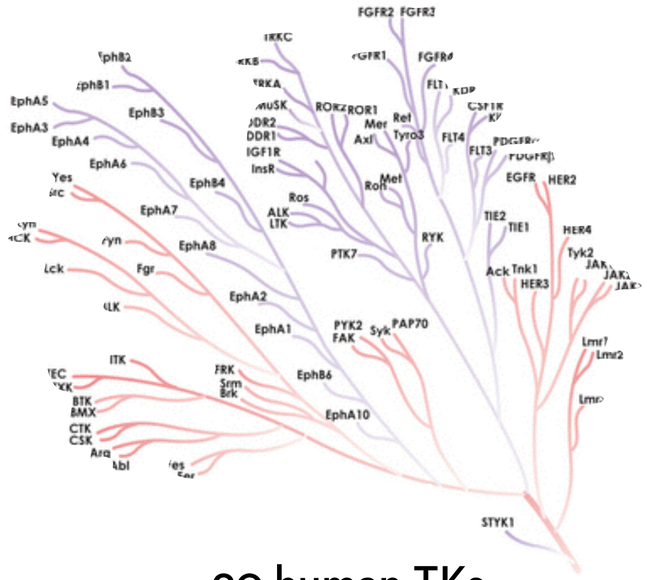
DANIEL PARTON, SONYA HANSON, PATRICK GRINAWAY



	Number of models			Timing
	All TKs	SRC	ABL	
				3h/template (loop reconstruction)
	412,269	4433	4433	
	397,621	4271	4431	89s/model 9863 CPU-h total
	394,040	4135	4289	0.3s/model 32 CPU-h total
	389,067	4066	4289	118s/model 12,418 GPU-h total
		4003	4280	9s/model 947 CPU-h total

bioRxiv preprint:
http://dx.doi.org/10.1101/018036

MODELING ALL 90 HUMAN TYROSINE KINASES ONTO ALL KINASE CATALYTIC DOMAIN PDBS



90 human TKs

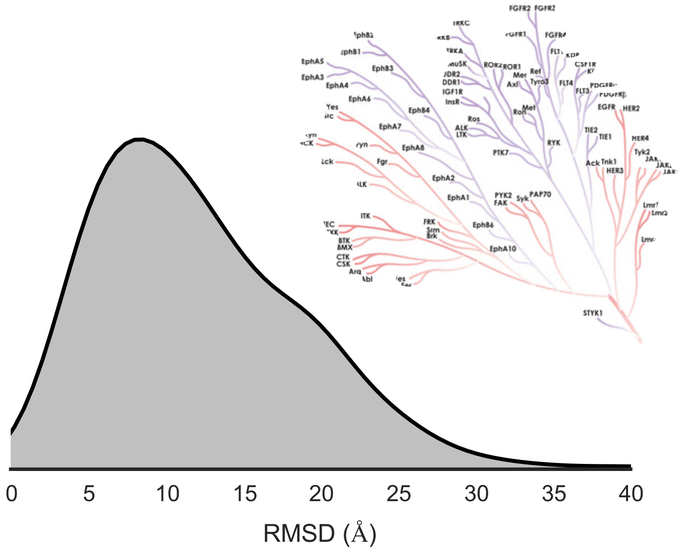
x

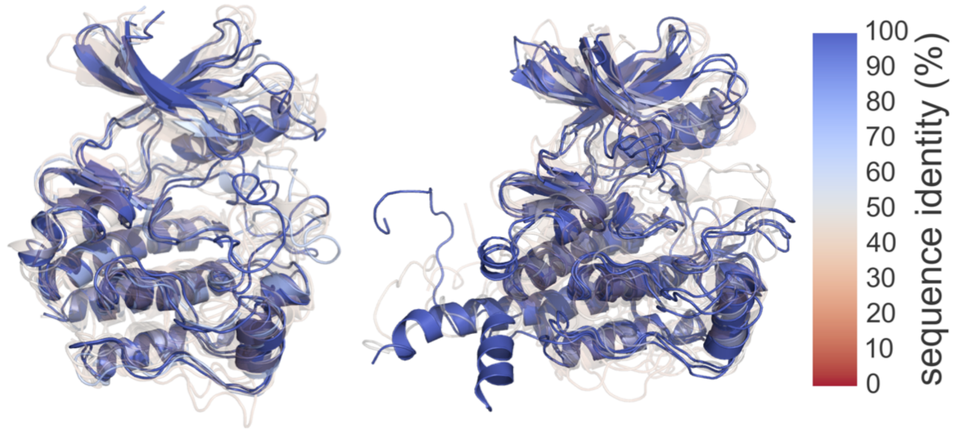
RCSB PDB
PROTEIN DATA BANK

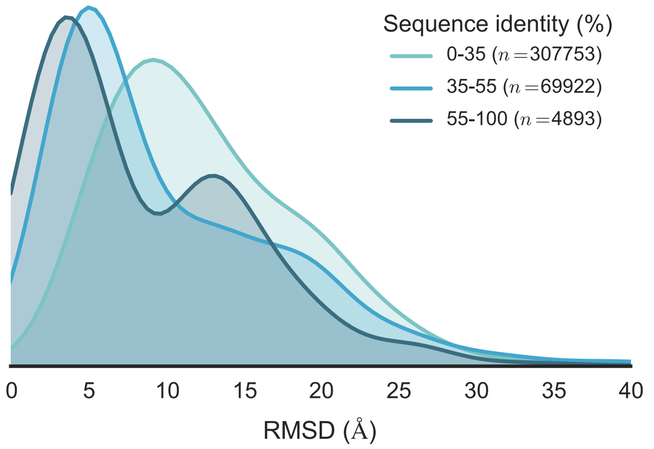
4433 PDB structures of kinase catalytic domains

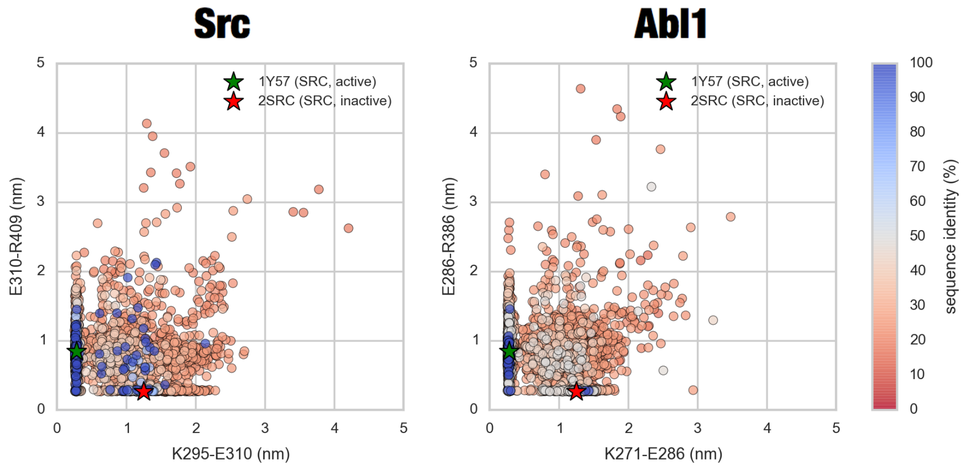
```
#!/bin/bash

conda create -c https://conda.binstar.org/omnia -n ensembler1.0 python=2.7 ensembler=1.0 --yes
source activate ensembler1.0

ensembler init
ensembler gather_targets --query 'family:"tyr protein kinase family" AND organism:"homo sapiens" AND reviewed:yes' \
   --uniprot_domain '^Protein kinase(?!; truncated)(?!; inactive)'
ensembler gather_templates --gather_from uniprot --query 'domain:"Protein kinase" AND reviewed:yes' \
   --uniprot_domain_regex '^Protein kinase(?!; truncated)(?!; inactive)'
ensembler loopmodel
ensembler align
ensembler build_models
ensembler cluster
ensembler refine_implicit
```

MODELING ALL 90 HUMAN TYROSINE KINASES ONTO ALL KINASE CATALYTIC DOMAIN PDBS










```
$ conda config --add channels http://conda.binstar.org/omnia
$ conda install ensembler
```

bioRxiv preprint:
http://dx.doi.org/10.1101/018036

YANK

A free, open source, extensible platform for GPU-accelerated binding free energy calculations

PRIMARY MAINTAINERS
John D. Chodera, Patrick Grinaway, Bas Rustenburg
CONTRIBUTORS
Kim Branson, Kyle A. Beauchamp, Peter M. Eastman, Mark Friedrichs, Imran S. Haque, Christoph Klein, Levi Naden, Daniel Parton, Randy Radmer, Andrea Rizzi, Michael Shirts, Kai Wong

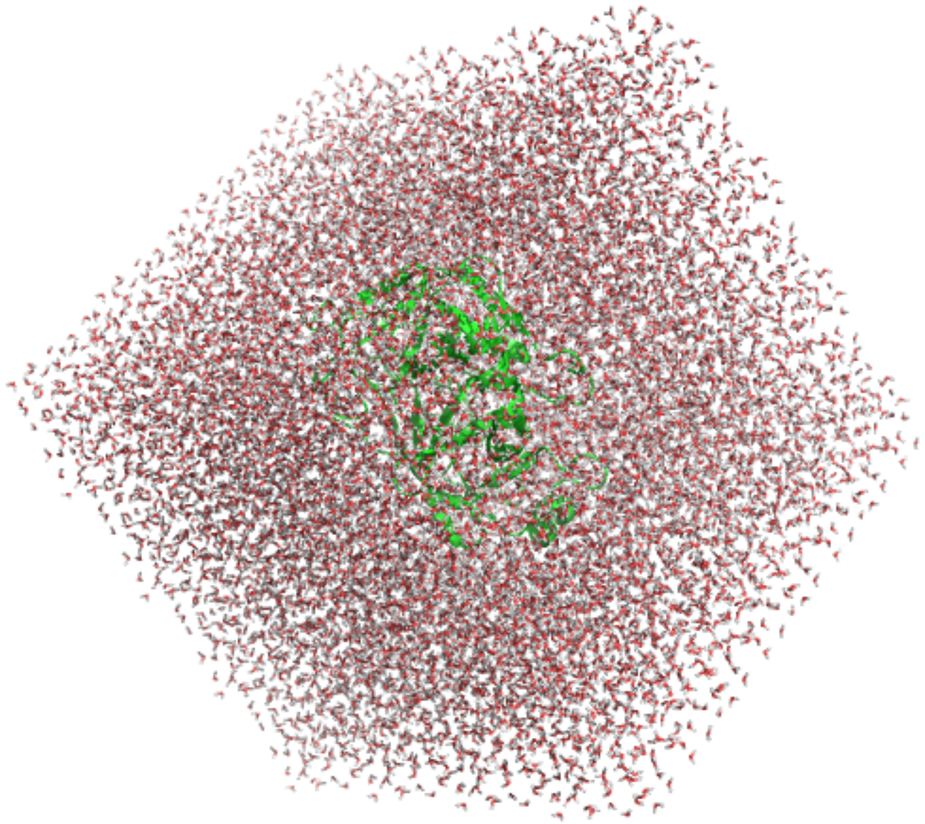
John D. Chodera
MSKCC Computational Biology Program
http://www.choderalab.org

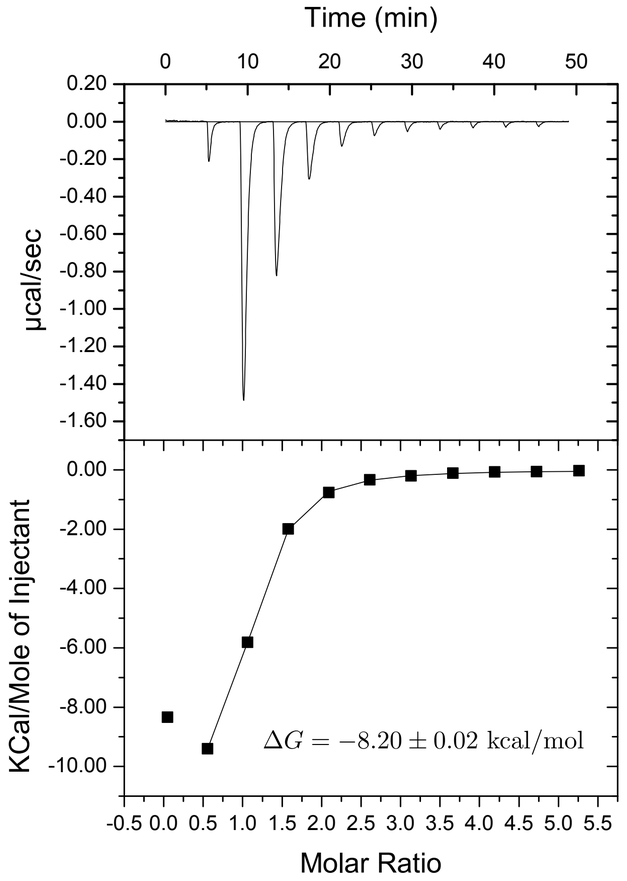
DISCLOSURES:
- Scientific Advisory Board, Schrödinger

OBLIGATORY HAZARD STATEMENT

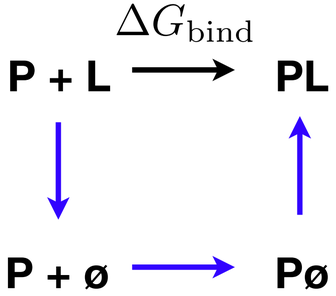
YANK 0.7 is research software and still under active development!
There is a suite of unit tests, but we have not yet verified every capability works as expected.
Use at your own risk.

YANK COMPUTES
ABSOLUTE BINDING FREE ENERGIES
TO COMPARE DIRECTLY WITH EXPERIMENT

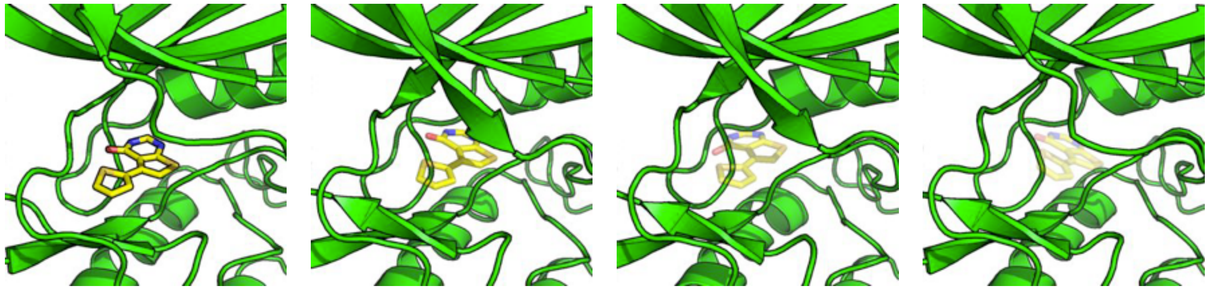




ALCHEMICAL FREE ENERGY CALCULATIONS PROVIDE A RIGOROUS WAY TO EFFICIENTLY COMPUTE BINDING AFFINITIES



multiple simulations of alchemical intermediates



restraint imposition | discharging | steric decoupling | noninteracting

Requires orders of magnitude less effort than simulating direct association process, but still includes all enthalpic/entropic contributions to binding free energy.

$$\Delta F_{1\to N} = -\beta^{-1}\ln\frac{Z_N}{Z_1} = -\beta^{-1}\ln\frac{Z_2}{Z_1}\cdot\frac{Z_3}{Z_2}\cdots\frac{Z_N}{Z_{N-1}} = \sum_{n=1}^{N-1}\Delta F_{n\to n+1} \qquad Z_n = \int d\mathbf{x}\, e^{-\beta U(\mathbf{x})}$$

Pioneering work from many: McCammon, van Gunsteren, Kollman, Jorgensen, Chipot, Roux, Boresch, Fujitani, Pande, Shirts, Swope, Christ, Mobley, and many more
Recent review: Chodera, Mobley, Shirts, Dixon, Branson, Pande. Curr Opin Struct Biol 21:150, 2011.

THE MULTISTATE BENNETT ACCEPTANCE RATIO (MBAR) ESTIMATOR EXTRACTS ALL INFORMATION FROM THE DATA

Statistically optimal analysis of samples from multiple equilibrium states

Michael R. Shirts[1,a)] and John D. Chodera[2,b)]

– HIDE AFFILIATIONS

[1] Department of Chemical Engineering, University of Virginia, Charlottesville, Virginia 22904, USA

[2] Department of Chemistry, Stanford University, Stanford, California 94305, USA

a) Electronic mail: michael.shirts@virginia.edu.

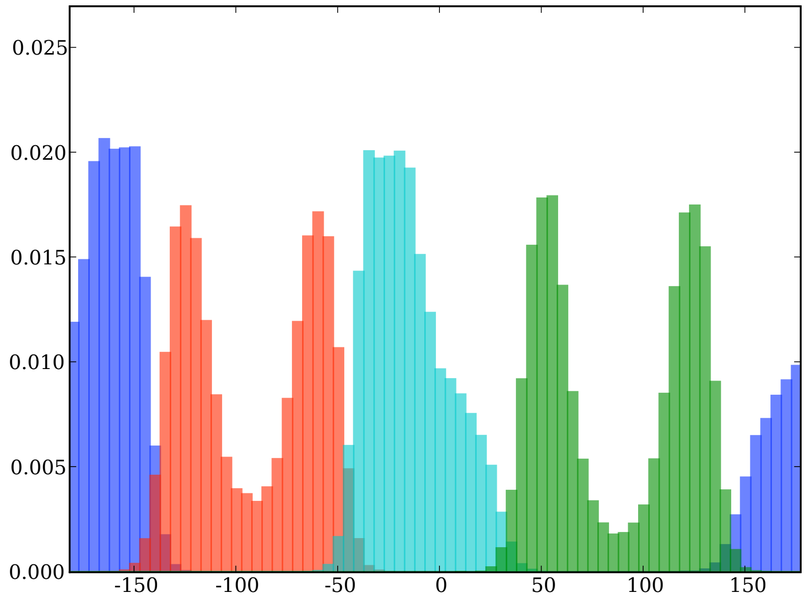
b) Electronic mail: jchodera@gmail.com

J. Chem. Phys. **129**, 124105 (2008); http://dx.doi.org/10.1063/1.2978177

$$\hat{f}_i = -\ln \sum_{j=1}^{K} \sum_{n=1}^{N_j} \frac{\exp[-u_i(\boldsymbol{x}_{jn})]}{\sum_{k=1}^{K} N_k \exp[\hat{f}_k - u_k(\boldsymbol{x}_{jn})]}$$

$$\delta^2 \Delta \hat{f}_{ij} = \hat{\Theta}_{ii} - 2\hat{\Theta}_{ij} + \hat{\Theta}_{jj}$$

$$\hat{\boldsymbol{\Theta}} = \mathbf{W}^{\mathrm{T}}(\mathbf{I}_N - \mathbf{W}\mathbf{N}\mathbf{W}^{\mathrm{T}})^{+}\mathbf{W}$$

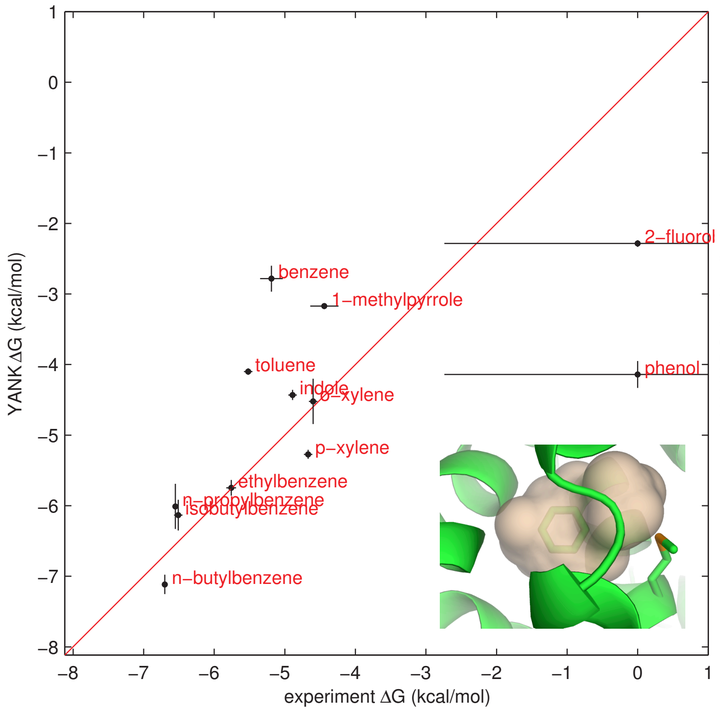


- asymptotically optimal estimator for free energy differences from equilibrium data
- robust estimates of uncertainties
- combine data from multiple temperatures, pressures, bias potentials
- freely-available Python implementation installable via **conda**
- batteries included: comes with tools to subsample correlated data to extract independent data

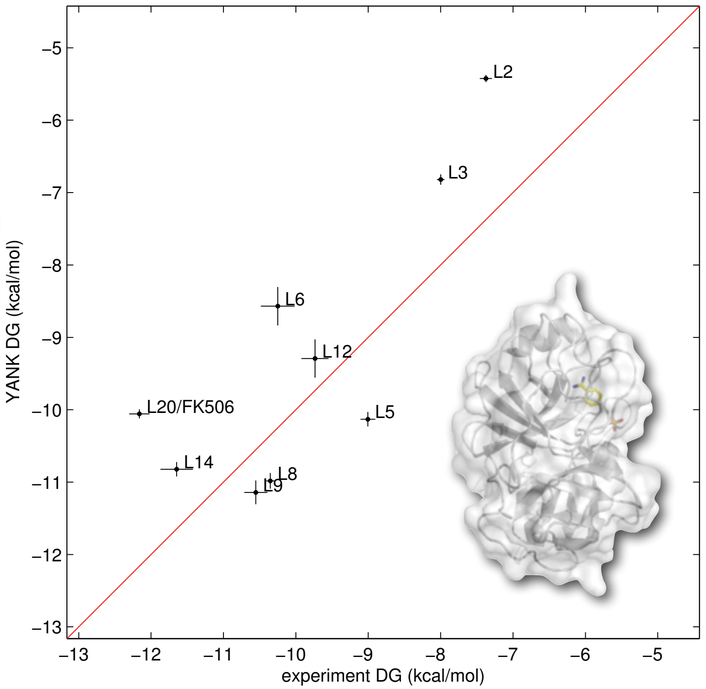
http://github.org/choderalab/pymbar

FREE ENERGIES WITH IMPLICIT MODELS OF SOLVENT ARE PROMISING: COULD PLAY A ROLE IN RAPID AFFINITY PREDICTION

T4 LYSOZYME L99A



FKBP12



AMBER ff96 + OBC GBSA (no cutoff) + GAFF/AM1-BCC
12 h on 2 GPUs

Chodera and Shirts. JCP 135:194110, 2011
Wang, Chodera, Yang, and Shirts. JCAMD 27:989, 2013.
http://github.org/choderalab/yank

INSTALLING YANK

```
conda install -c omnia yank
```

SETTING UP A YANK CALCULATION

Using the command-line:

```
#!/bin/bash

# Set up calculation.
echo "Setting up binding free energy calculation..."
yank prepare binding amber --setupdir=setup --ligand="resname MOL" --store=output --iterations=1000 \
    --–nbmethod=CutoffPeriodic --temperature="300*kelvin" --pressure="1*atmosphere" --minimize --verbose

# Run the simulation with verbose output:
echo "Running simulation..."
yank run --store=output --verbose

# Analyze the data (can be done asynchronously)
echo "Analyzing data..."
yank analyze --store=output
```

SETTING UP A YANK CALCULATION

Using the Python API:

```
from yank.yank import Yank

# Initialize YANK object.
yank = Yank(store_dir)

# Set some options.
options = dict()
options['number_of_iterations'] = 1000

# Create reference thermodynamic state.
from yank.repex import ThermodynamicState
thermodynamic_state = ThermodynamicState(temperature=temperature, pressure=pressure)

# Create new simulation.
yank.create(phases, systems, positions, atom_indices, thermodynamic_state, options=options)

# Run the simulation
yank.create(phases, systems, positions, atom_indices, thermodynamic_state, options=options)

# Analyze the simulation
results = yank.analyze()
```

REPLICA-EXCHANGE WITH OPENMM

Current “experimental” replica-exchange: **http://github.com/choderalab/repex**

Features:

* generic replica exchange (supports any System objects that have same number of particles)
* convenience subclasses for parallel tempering and Hamiltonian exchange
* easy to subclass to add other kinds of dynamics or specialized exchanges
* supports multiple thermodynamic ensembles (NVT, NPT, etc.)
* supports implicit and explicit solvent
* optional MPI support through **mpi4py** to use multiple GPUs on same or different nodes
* caches multiple Context objects to minimize overhead
* defaults to efficient **Gibbs sampling** for mixing replicas
* writes portable and compact NetCDF files that make resuming and analyzing simulation easy
* writes energies of all replicas at all states needed for MBAR
* provides estimated time of completion
* Langevin dynamics by default

REPLICA-EXCHANGE WITH OPENMM

```
# Parallel tempering simulation of alanine dipeptide in implicit solvent (replica exchange among temperatures)

import math

# OpenMM imports
import simtk.unit as units
import simtk.openmm as openmm
from repex import ThermodynamicState, ReplicaExchange

# Create test system (alanine dipeptide in OBC GBSA implicit solvent).
from openmmtools import testsystems
[system, coordinates] = testsystems.AlanineDipeptideImplicit()

# Create thermodynamic states for parallel tempering using exponentially-spaced schedule.
import math
nreplicas = 5 # number of temperature replicas
T_min = 298.0 * units.kelvin # minimum temperature
T_max = 600.0 * units.kelvin # maximum temperature
T_i = [ T_min + (T_max - T_min) * (math.exp(float(i) / float(nreplicas-1)) - 1.0) / (math.e - 1.0) for i in range(nreplicas) ] # temperature schedule
states = [ ThermodynamicState(system=system, temperature=T_i[i]) for i in range(nreplicas) ] # thermodynamic states

# Create simulation.
filename = 'output.nc' # define output filename for NetCDF file
simulation = ReplicaExchange(states, coordinates, filename) # initialize the replica-exchange simulation
simulation.verbose = True # verbose output
simulation.minimize = False # don't minimize before simulation
simulation.number_of_iterations = 20 # number of iterations
simulation.timestep = 2.0 * units.femtoseconds # timestep for integration
simulation.nsteps_per_iteration = 500 # number of timesteps per iteration
simulation.run() # run the simulation
```

Example provided as replica-exchange-example.py

REPLICA-EXCHANGE WITH OPENMM

```
Iteration 19 / 20
Mixing replicas...
Will attempt to swap all pairs of replicas using weave-accelerated code, using a total of 3125 attempts.
Accepted 2978 / 6250 attempted swaps (47.6 %)
Mixing of replicas took 0.001 s
Propagating all replicas for 1.000 ps...
Replica 0/5: integrator elapsed time 0.490 s (positions 0.003 s | velocities 0.004 s | integrate+getstate 0.482 s).
Replica 1/5: integrator elapsed time 0.492 s (positions 0.003 s | velocities 0.004 s | integrate+getstate 0.484 s).
Replica 2/5: integrator elapsed time 0.497 s (positions 0.003 s | velocities 0.004 s | integrate+getstate 0.490 s).
Replica 3/5: integrator elapsed time 0.490 s (positions 0.003 s | velocities 0.004 s | integrate+getstate 0.482 s).
Replica 4/5: integrator elapsed time 0.491 s (positions 0.003 s | velocities 0.002 s | integrate+getstate 0.486 s).
Time to propagate all replicas: 2.461 s (0.492 per replica, 175.550 ns/day).
Computing energies...
Time to compute all energies 0.103 s (0.004 per energy calculation).
reduced potential (kT)      current state  state   0  state   1  state   2  state   3  state   4
replica 0                                2      -37.7      -32.3      -27.3      -22.7      -18.7
replica 1                                4      -24.0      -20.5      -17.3      -14.5      -11.9
replica 2                                1      -42.1      -36.1      -30.5      -25.4      -20.9
replica 3                                3      -32.0      -27.4      -23.1      -19.3      -15.9
replica 4                                0      -45.9      -39.3      -33.2      -27.7      -22.8
Writing data to NetCDF file took 0.031 s (0.000 s for sync)
Cumulative symmetrized state mixing transition matrix:
             0      1      2      3      4
0        0.444  0.361  0.194
1        0.361  0.444  0.083  0.111
2        0.194  0.083  0.500  0.222
3               0.111  0.222  0.556  0.111
4                      0.111  0.889
Perron eigenvalue is   0.89300; state equilibration timescale is ~ 9.3 iterations
Time to compute mixing statistics 0.124 s
Iteration took 2.724 s.
Estimated completion in 0:00:02.721196, at Mon Mar 12 17:23:29 2012 (consuming total wall clock time 0:00:54.423911).
```

NETCDF OUTPUT IS SELF-DESCRIBING

Output from NetCDF tool ‘ncdump -h’:

```
netcdf output {
dimensions:
   iteration = UNLIMITED ; // (50 currently)
   replica = 5 ;
   atom = 648 ;
   spatial = 3 ;
   scalar = 1 ;
variables:
   float positions(iteration, replica, atom, spatial) ;
     positions:units = "nm" ;
     positions:long_name = "positions[iteration][replica][atom][spatial] is position of coordinate \'spatial\' of atom
\'atom\' from replica \'replica\' for iteration \'iteration\'." ;
   int states(iteration, replica) ;
     states:units = "none" ;
     states:long_name = "states[iteration][replica] is the state index (0..nstates-1) of replica \'replica\' of iteratio
\'iteration\'." ;
   float energies(iteration, replica, replica) ;
     energies:units = "kT" ;
     energies:long_name = "energies[iteration][replica][state] is the reduced (unitless) energy of replica \'replica\'
from iteration \'iteration\' evaluated at state \'state\'." ;
   int64 proposed(iteration, replica, replica) ;
     proposed:units = "none" ;
     proposed:long_name = "proposed[iteration][i][j] is the number of proposed transitions between states i and j from
iteration \'iteration-1\'." ;
   int64 accepted(iteration, replica, replica) ;
     accepted:units = "none" ;
     accepted:long_name = "accepted[iteration][i][j] is the number of proposed transitions between states i and j from
iteration \'iteration-1\'." ;
   float box_vectors(iteration, replica, spatial, spatial) ;
```

ANALYSIS OF REPLICA-EXCHANGE OUTPUT

* Sample analysis script provided with example code

* NetCDF provides easy access to data:

```
# Get current dimensions.
niterations = ncfile.variables['energies'].shape[0]
nstates = ncfile.variables['energies'].shape[1]

# Extract energies.
print "Reading energies..."
energies = ncfile.variables['energies']
u_kln_replica = numpy.zeros([nstates, nstates, niterations], numpy.float64)
for n in range(niterations):
    u_kln_replica[:,:,n] = energies[n,:,:]
print "Done."
```

ParallelTempering SUBCLASS AUTOMATES INITIALIZATION OF STATES

```
# Parallel tempering simulation of alanine dipeptide in implicit solvent (replica exchange among temperatures)

import math

# OpenMM imports
import simtk.unit as units
import simtk.openmm as openmm
from repex import ThermodynamicState, ParallelTempering

# Create test system (alanine dipeptide in GBSA OBC implicit solvent).
from openmmtools import testsystems
[system, coordinates] = testsystems.AlanineDipeptideImplicit()

# Create thermodynamic states for parallel tempering with exponentially-spaced schedule.
import math
nreplicas = 5 # number of temperature replicas
Tmin = 298.0 * units.kelvin # minimum temperature
Tmax = 600.0 * units.kelvin # maximum temperature

# Create simulation.
filename = 'output.nc' # define output filename for NetCDF file
simulation = ParallelTempering(system, coordinates, filename, Tmin=Tmin, Tmax=Tmax, ntemps=nreplicas) # initialize parallel tempering
simulation
simulation.verbose = True # verbose output
simulation.minimize = False # don't minimize before simulation
simulation.number_of_iterations = 20 # set number of iterations
simulation.timestep = 2.0 * units.femtoseconds # set the timestep for integration
simulation.nsteps_per_iteration = 500 # set number of timesteps per iteration
simulation.run() # run the simulation
```

Example provided as parallel-tempering-example.py

MPI ALLOWS EXECUTION TO BE PARALLELIZED OVER GPUS/NODES

```
# Parallel tempering simulation of alanine dipeptide in implicit solvent (replica exchange among
temperatures)

from mpi4py import MPI # use MPI
import math

# OpenMM imports
import simtk.unit as units
import simtk.openmm as openmm
from repex import ThermodynamicState, ParallelTempering

# Create test system (alanine dipeptide in GBSA OBC implicit solvent).
from openmmtools import testsystems
[system, coordinates] = testsystems.AlanineDipeptideImplicit()

# Create thermodynamic states for parallel tempering with exponentially-spaced schedule.
import math
nreplicas = 5 # number of temperature replicas
Tmin = 298.0 * units.kelvin # minimum temperature
Tmax = 600.0 * units.kelvin # maximum temperature

# Create simulation.
filename = 'output.nc' # define output filename for NetCDF file
simulation = ParallelTempering(system, coordinates, filename, Tmin=Tmin, Tmax=Tmax, ntemps=nreplicas,
mpicomm=MPI.COMM_WORLD) # initialize with MPI
simulation.verbose = True # verbose output
simulation.minimize = False # don't minimize before simulation
simulation.number_of_iterations = 20 # set number of iterations
simulation.timestep = 2.0 * units.femtoseconds # set the timestep for integration
simulation.nsteps_per_iteration = 500 # set number of timesteps per iteration
simulation.run() # run the simulation
```

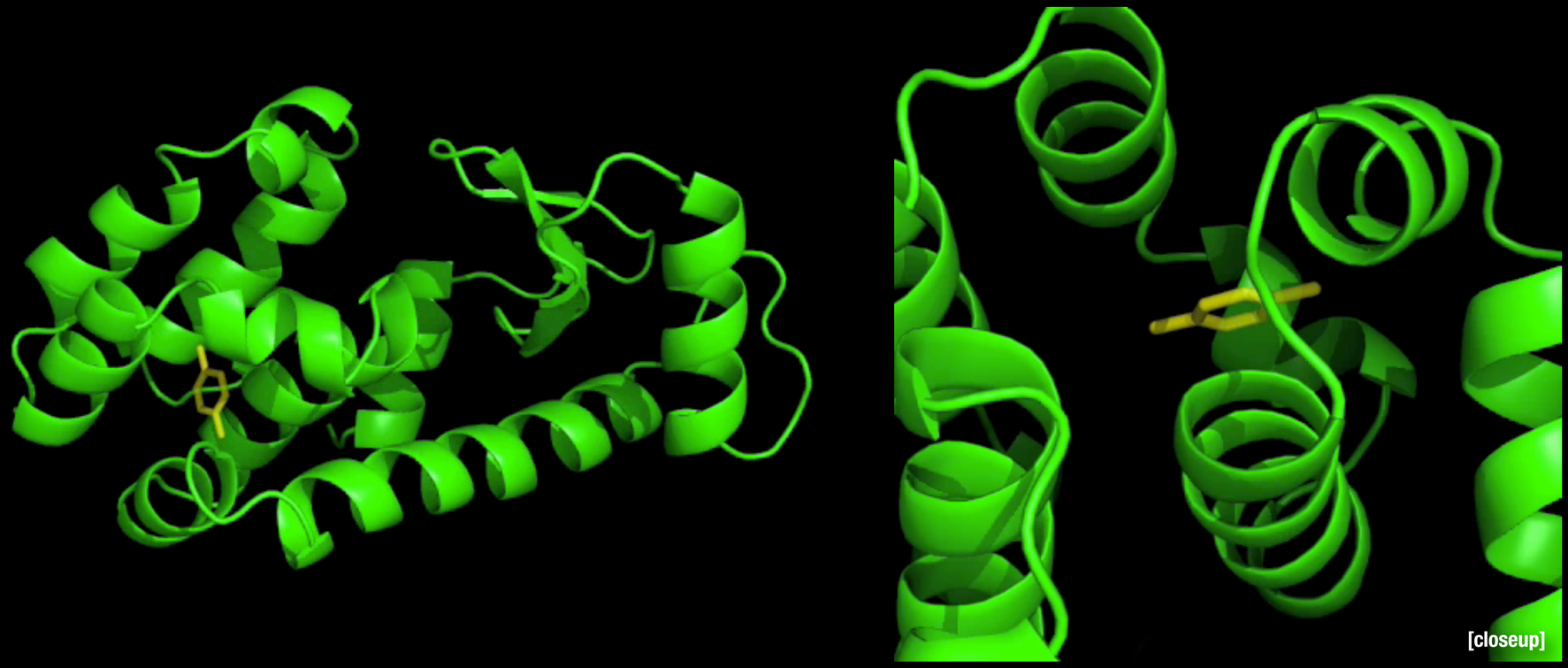
Example provided as parallel-tempering-mpi-example.py

HAMILTONIAN EXCHANGE EXAMPLE

```python
# Hamiltonian exchange example of discharging water in box of water.

# OpenMM imports
import simtk.unit as units
import simtk.openmm as openmm
from repex import ThermodynamicState, HamiltonianExchange

# Create test system.
from openmmtools import testsystems
[reference_system, coordinates] = testsystems.WaterBox()

# Create reference thermodynamic state.
reference_state = ThermodynamicState(reference_system, temperature=298.0*units.kelvin)

# Create thermodynamic states for parallel tempering with exponentially-spaced schedule.
systems = list()
ndischarge = 5 # number of intermediate states for discharging water
for state_index in range(ndischarge):
   # Make a deep copy of the reference system (so we can modify it).
   import copy
   system = copy.deepcopy(reference_system)
   # Find the NonbondedForce object for this system.
   forces = [ system.getForce(force_index) for force_index in range(system.getNumForces()) ] # collect Force objects
   forces = [ force for force in forces if isinstance(force, openmm.NonbondedForce) ] # filter to keep NonbondedForce objects only
   force = forces[0] # keep the first one
   # Modify charges of first water.
   for atom_index in range(3):
      [charge, sigma, epsilon] = force.getParticleParameters(atom_index) # get nonbonded parameters
      charge *= (1.0-state_index/float(ndischarge)) # modify charge
      force.setParticleParameters(atom_index, charge, sigma, epsilon) # replace nonbonded parameters
   # Add this system to the list.
   systems.append(system)

# Create simulation.
filename = 'output.nc' # define output filename for NetCDF file
simulation = HamiltonianExchange(reference_state, systems, coordinates, filename) # initialize Hamiltonian exchange
simulation.verbose = True
simulation.minimize = False # don't minimize before simulation
simulation.number_of_iterations = 50 # set number of iterations
simulation.timestep = 1.0 * units.femtoseconds # set the timestep for integration
simulation.nsteps_per_iteration = 100 # set number of timesteps per iteration
simulation.run() # run the simulation
```

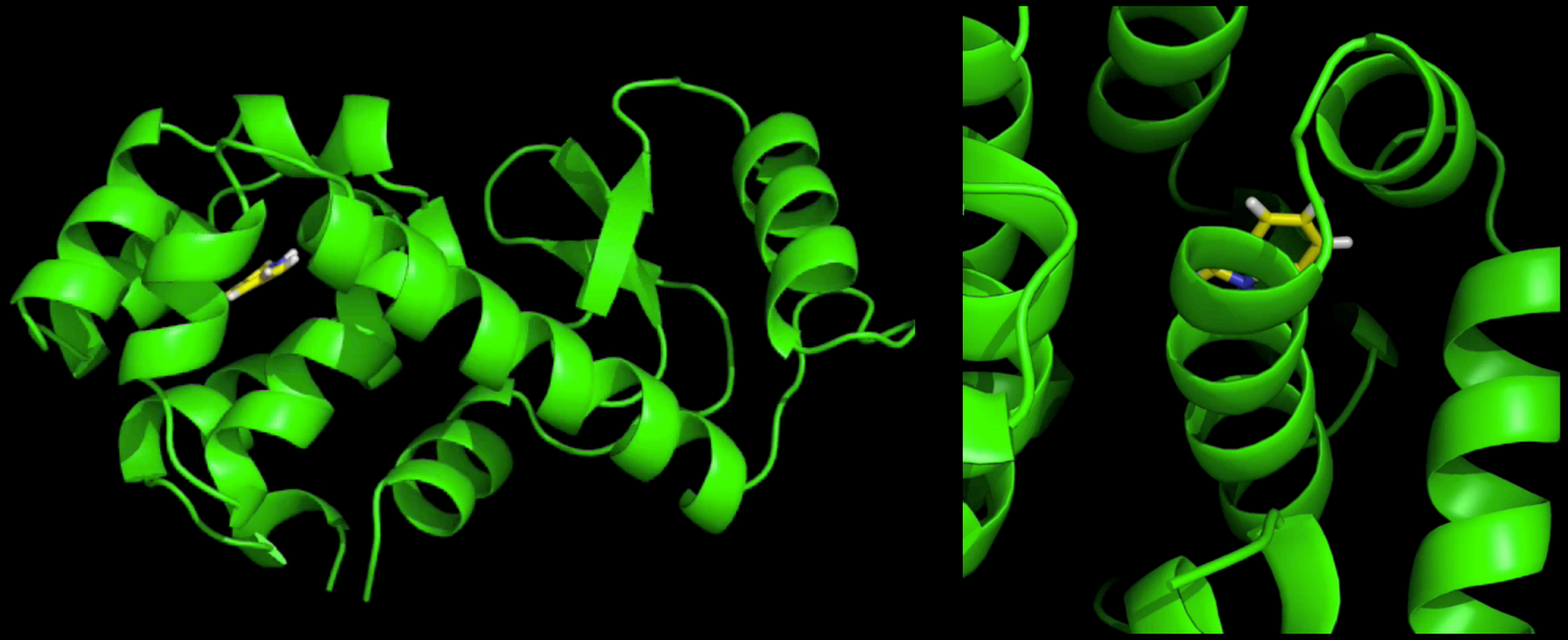
Example provided as hamiltonian-exchange-example.py

ENHANCED SAMPLING ALGORITHMS ALLOW FOR REPEATED BINDING/UNBINDING EVENTS AND



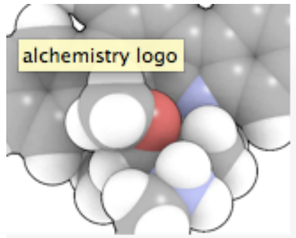
p-xylene bound to T4 lysozyme L99A

ENHANCED SAMPLING ALGORITHMS ALLOW FOR REPEATED BINDING/UNBINDING EVENTS AND



indole bound to T4 lysozyme L99A

VISIT ALCHEMISTRY.ORG FOR MORE INFORMATION ON ALCHEMICAL FREE ENERGY CALCULATIONS



Page | Discussion | Read | View source | View history | Search | Go | Search

wiki navigation

- Main page
- Recent changes
- Random page
- Help

Alchemistry.org Content

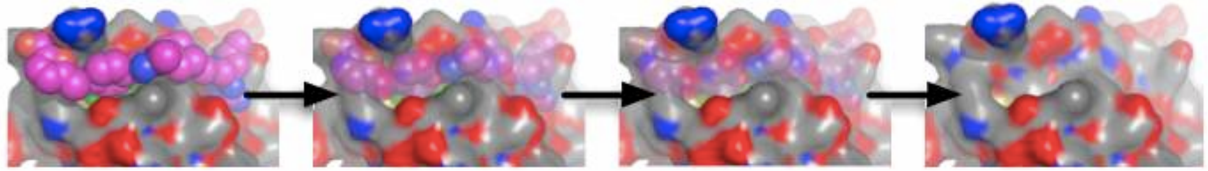
- Free Energy Fundamentals
- Best Practices
- Tutorials
- Free Energy References (external)
- Test System Repository
- Events

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

Alchemistry.org

Modern alchemy, done computationally, can turn protein structural information into the "gold" of binding free energies and other information. In these pages, learn how it works and how to do it yourself.



What is Alchemistry?

Alchemical free energy calculations employ unphysical ("alchemical") intermediates to estimate the free energies of various physical processes. Such process include ligand binding to a protein receptor, the transfer of a small molecule from gas to water, or the free energy of a mutation of a side chain. This approach provides not only a quantitative and rigorous method for computing free energies, but often provides insight into the dominant interactions driving binding.

Getting Started with Free Energy Methods

This site is designed to both give novices best practices information about how to perform free energy calculations, and to provide an ongoing reference for the current state of research into methods for calculating free energies.

Free Energy Fundamentals

Example Free Energy Calculations

A look into practical examples without software-specific details. We outline the steps needed for realistic free energy problem, detailing problems you can expect to encounter, and how to analyze the data to get a robust free energy difference.

Tutorials and How-to's