LOOS: A Tool for Analyzing Molecular Dynamics Simulations

Tod D. Romo and Alan Grossfield
University of Rochester Medical School, Rochester, NY, USA

Analyzing Molecular Simulations

- Molecular simulations have unmatched resolution in time and space
- Need better analysis tools to extract maximum value
  - Most projects require custom code
  - Data analysis is an iterative process
  ---- Rapid development is key

LOOS Design Goals

- Package-agnostic
  - Read all common file formats
    - NAMD, Amber (netcdf and mdcrd), GROMACS, TINKER, OpenMM, LAMMPS
  - Programs don't care where files came from
  - Reduce duplicated effort
  - Improve reproducibility
- Easy to use
  - Powerful tools
  - Unique functionality
  - Convenient atom selection facility
  - Highly scriptable
  - Detailed documentation
  - High performance
    - 1-2 orders of magnitude faster than VMD, mdanalysis
    - Comparable to cpptraj
- Easy to develop
  - C++ core
    - Good object design makes code expressive
    - 4 key classes means easy to learn
    - No memory management
    - Atom selection identical to tool level
  - Python interface
    - Rapid application development
    - Easy code reuse
    - Interoperable with numpy, scipy
    - Python 3 support
  - Extensive documentation
    - Code level via doxygen
    - Github wiki
- Available everywhere
  - Tested on ~15 recent linux distributions
  - Builds under Conda (recipe coming soon)
  - GPLv3 license

Using LOOS

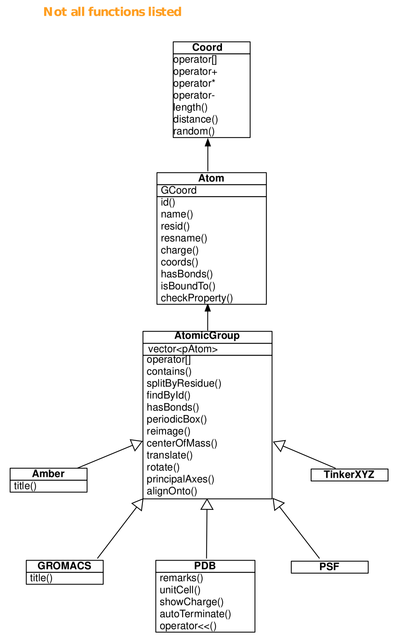
Example Tools

Macromolecules	
rmsds	All-to-all structure comparison
rmsf	Molecular fluctuations
svd	Principal component analysis
rdf	Radial distribution function
Membranes and membrane proteins	
order_parameters	Chain tilt in membranes
density-dist	Distribution along membrane normal
xy_rdf	Radial distribution in membrane plane
membrane_map	Lipid properties around membrane
mops	Molecular order parameters for chains
dibmops	mops vs. distance from macromolecule
Trajectory manipulation	
merge-traj	Rapidly merge trajectories
subsetter	Merge, reimage, and subset trajectories
Convergence Package	
block_average	Block average of time-series data
decorr_time	Decorrelation time of structural histograms
bcom, boot_bcom	Block covariance overlap method
Voronoi Package	
area_per_molecule.py	Area distribution for a membrane slice
area_profile.py	Voronoi area for protein along normal
Elastic Network Model Package	
anm	Anisotropic network model
enmovie	Visualize ENM modes via a trajectory
vsa	Vibrational subsystem analysis
Gridded Density Package	
water-hist	3D density histogram for atoms
near_blobs	Find residues near density peaks
grid2xplor	Convert density grid to X-plor format
Optimal Membrane Generator Package	
omg.py	Build membrane systems
solvate.py	Build water around soluble molecules

Developing with LOOS

Structure Classes

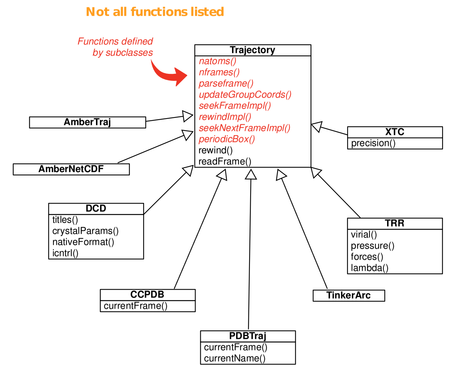
Not all functions listed



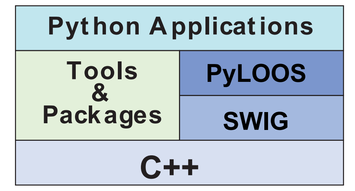
Classes map to key concepts
- Makes code expressive
- AtomicGroup is workhorse
  - File formats are subclasses
  - Common tasks are methods
- Trajectory formats are subclasses
- Code with parent classes is format-agnostic

Trajectories

Not all functions listed



Application Layers



Most new tools are python
- Easier development
- Great libraries
  - Interoperate via numpy, scipy
  - Voronoi package, clustering
- Performance gap is small because most work is done in the library

Example Code

Relative motion of domains

Read command line
```
#!/usr/bin/env python3
import sys
import loos
import loos.pyloos
import math

system_file = sys.argv[1]
traj_file = sys.argv[2]
sel_string1 = sys.argv[3]
sel_string2 = sys.argv[4]
```

Create system
```
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)
```

Select "domains"
```
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)
```

Loop over trajectory
```
for frame in traj:
```

Compute distance
```
    # compute distance
    centroid1 = sel1.centroid()
    centroid2 = sel2.centroid()

    diff = centroid2 - centroid1
    distance = diff.length()
```

Compute axis angle
```
    # Compute angle between principal axes
    vectors1 = sel1.principalAxes()
    axis1 = vectors1[0]

    vectors2 = sel2.principalAxes()
    axis2 = vectors2[0]
    angle = math.acos(axis1 * axis2) * 180/math.pi
```

Compute torsion
```
    # Compute torsion between principal axes
    p1 = centroid1 + axis1
    p2 = centroid2 + axis2

    tors = loos.torsion(p1, centroid1, centroid2, p2)

    # write output
    print(traj.index(), distance, angle, tors)
```

Future directions

- Support QM file formats
- Integrated analysis workbench
  - Jupyter?
- New packages
  - NMR
  - X-ray scattering
  - Nucleic acids

New in LOOS 3.0

- New and improved tools
  - Track native hydrogen bonds
  - All-to-all structure comparisons across many trajectories
  - Improved clustering
- Better documentation
  - HowTo stories on GitHub wiki
- Conda install
  - Support non-standard environments
- Interoperate with WESTPA
- Port PyLOOS to Python 3.x
- Reweighting utility
  - Compute weighted average during trajectory analysis
  - Supported in C++ and python
- Velocity manipulation
  - Do analysis on velocities

Getting LOOS

LOOS Paper
J. Comput. Chem., 2014, 35, 2305-2318

Download poster

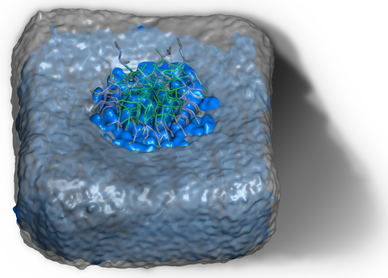
LOOS on GitHub

System Visualization

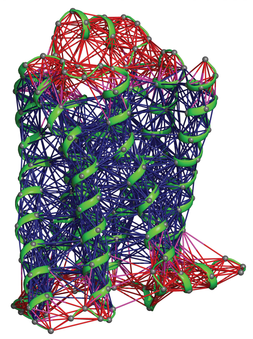
Made with LOOS

Water Density

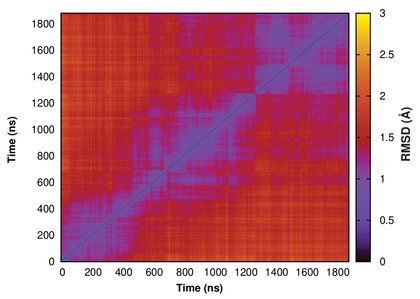
Lipid Density



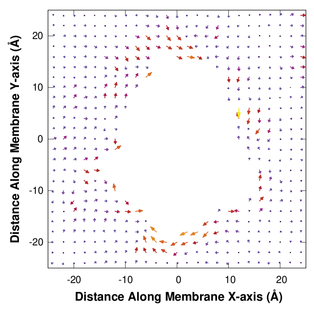
Elastic Network Models



All-to-All RMSD

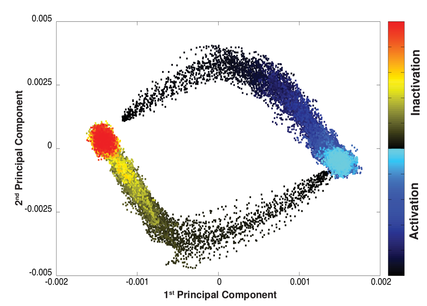


Membrane Properties



Principal Components

Structural Transitions



Tools:	custom software	water-hist	water-hist	ANM/VSA (rebond)	rmsds	membrane_map	svd & porcupine	transition_contacts & svd