Department of Biochemistry and Biophysics, University of Rochester Medical School

LOOS: Lightweight Object Oriented Structure Library

A tool for rapid development of MD analysis software

Dr. Alan Grossfield
alan_grossfield@urmc.rochester.edu
Twitter: @agrossfield

Outline

- **Introduction and Design goals**
- **Using LOOS Tools**
- **Developing with LOOS**
- **Example Problem: Domain motion**

Most analyses have the same basic structure

- **Curate the data (do this in advance)**
- **Specify the system**
- **Pick out the atoms of interest**
- **Loop over snapshots in trajectory**
  - Compute some geometric quantity
- **Output either time series or some kind of average**
- **Nearly all projects require some custom code**

What do we want?

- **Rapid development**
  - Analysis generally cheap compared to running simulation
  - Key is to quickly try new ideas
  - All standard tasks should be 1 line
- **Reasonable performance**
  - Data sets keep getting bigger
  - Scripting languages frequently too slow
- **Package independence**
  - Able to analyze all common trajectory formats
    - Reduce duplication of effort
    - Increase leverage to community
    - Some projects use multiple packages
- **Open source and cross-platform**

What do we use?

- **Existing tools**
  - Packages
    - CHARMM has lots of analysis built in
      - Adding code is hard, scripting is relatively easy but slow
    - Amber has ptraj and cpptraj
      - Basic analysis
      - Monolithic → hard to add functionality
    - Gromacs has a bunch of standalone tools
    - VMD can be scripted but primarily a GUI
  - Most packages only analyze their own file formats

Why use LOOS? Alternatives?

- **MDTraj**
  - Heavily pythonic
  - Memory inefficient because of pandas
  - Semi-abandonware
- **MDAnalysis**
  - Core python, built around Numpy arrays
  - More about big applications than rapid application development
  - EXTREMELY slow
- **LOOS**
  - C++ core for performance (often > 10^2 x faster than MDAnalysis)
  - Object-oriented so you think about physical objects not Numpy arrays

Design choices

- **C++**
  - High performance
  - Object-oriented design
    - Application code very simple
    - Hide the complexity of C++ from tool developers
    - Single binary supports all data formats
  - Python interface for rapid development
- **Very simple object hierarchy**
  - `Atoms` and `AtomicGroups`
- **Symbolic atom selection language**

Outline

- Introduction and Design goals
- **Using LOOS Tools**
- Developing with LOOS
- Example Problem: Domain motion

Defining terms

- **System file**
  - File that specifies the contents of the system
    - Atom names, residue names, etc
  - PDB, PSF, prmtop, gro, etc
  - Not all files have the same info
    - Connectivity, charge, mass not in all
    - Some applications depend on these properties
    - `gmxdump2pdb.pl`
      - Make a “fake” PSF from gromacs input
- **Trajectory**
  - File that specifies the coordinates for a trajectory
  - DCD, XTC, NETCDF, oldAmber
  - Also support a list of PDB files

Defining terms

- **Selection**
  - Subset of atoms
  - Chosen based on metadata
    - Atom or residue name or number, segid
  - NOT based on geometry
    - Not all model files have coordinates
    - Selections are often done before coordinates are known
    - Geometric selections are done programmatically

LOOS Tools (examples)

- **Trajectory manipulation**
  - aligner: align all frames of trajectory
  - subsetter: create new trajectory using a subset of atoms
  - merge-traj: incrementally merge multiple trajectory files
- **Structure analysis**
  - rdf / atomic-rdf: radial distribution function
  - ramachandran: backbone dihedrals
  - svd: principal component analysis
  - averager: average structure via iterative alignment
  - rmsds: all-to-all structural comparison

LOOS Tools (examples)

- **Membrane-specific tools**
  - order_params: compute ^{2}H quadrupolar splittings
  - mops: molecular order parameter, whole-chain analog of order parameters
  - density-dist: atom distributions along membrane normal
  - xy-rdf: RDF in plane of membrane (lateral ordering)
  - membrane_map: membrane properties around protein
- **Lots of others (≈100)**

Command line arguments

- **Use common sets of command-line arguments**
  - Consistency makes learning tools easier
- **Implemented via layer on top of BOOST `program_options`**
  - Kind of a pain, but less painful than every other solution we've found
  - Not required
    - Usually part of polishing for release
- **Provisional equivalent scheme in PyLOOS**
  - Build on top of `argparse`
  - Still under development, not used universally

LOOS Packages: collections of related tools

- **DensityTools**
  - 3D histograms of atom density
    - water distribution inside protein
    - lipid occupancy on protein surface
- **HydrogenBonds**
  - Quantify and count hydrogen bonds
- **Convergence**
  - Assess statistical errors and correlation times in biomolecular simulations
  - Implementations of most Zuckerman Lab algorithms
- **ElasticNetworks**
  - Anisotropic network model / Vibrational Subsystem Analysis
  - Implementations for a variety of spring functions

LOOS Packages: collections of related tools

- **Voronoi**
  - Perform Voronoi analysis on membrane systems
  - Area profiles for protein along membrane normal
  - Area/molecule for different system components
- **OptimalMembraneGenerator (`OMG.py`)**
  - Construct arbitrary membrane/membrane protein systems
  - Highly flexible and configurable
  - Can build non-membrane systems too (`solvate.py`)
  - NAMD only (for now)
  - Easily adapted for one-off special systems

LOOS Packages: User

- **Location in the tree for developing new C++ tools**
  - Trivial setup of paths, etc
  - Examples of LOOS idioms
    - Standard tasks with empty inner loop

LOOS Packages: PyLOOS

- **Core LOOS is C++**
  - Application development is fast
  - Some people don't like C++
  - Some tasks naturally scripted
- **PyLOOS: python interface to LOOS core**
  - Implemented via SWIG
  - Directory contains several tools
  - Future tools development should default to python
    - Performance is pretty good
    - Rapid development
    - Easy incorporation of libraries (e.g. `scipy`)
    - OMG and Voronoi both implemented this way

Symbolic selection of atoms

- **Most programs operate on subsets of atoms**
- **Need a clean way for end user to specify on the command line**
- **LOOS selection**
  - C-like syntax
  - Perl-style regular expressions
  - Access to atomic metadata
  - Similar in capability to selection in CHARMM or VMD
  - Available on command line and inside code
    - Can programmatically create selection strings

Selection examples

- **Select protein alpha carbons**
  - `segname == “PROT” && name == “CA”`
- **Select aromatic residues**
  - `resname == “TRP” || resname == “PHE” || resname == “TYR”`
- **Pattern match: atoms beginning with C but not CA**
  - `name =~ “^C” && !(name == “CA”)`
- **Lipids 7-12**
  - `segname -> “LP(\d+)” => 7 && segname -> “LP(\d+)” <=12`
  - Magic “->” operator interprets match as a number

Tool documentation

- **All tools have documentation**
- **"`toolname`" lists the command line options**
- **"`toolname --fullhelp`" does more**
  - Meaning of command line arguments
  - Algorithmic subtleties
  - Common use cases
  - Example command lines
  - Suggested workflows
  - Gotchas and alternative tools

Outline

- Introduction and Design goals
- Using LOOS Tools
- **Developing with LOOS**
- Example Problem: Domain motion

LOOS has a simple class hierarchy

- **LOOS is a (relatively) large package**
  - Most simulators aren't "real" programmers
  - Need to make it easy to learn
  - Most infrastructure hidden from tool developers
- **Most tasks only need 4 classes**
  - `Coord`
  - `Atom`
  - `AtomicGroup`
  - `Trajectory`

Class Design: Coord

- **Represents vectors and coordinates**
- **Operator overloading for vector operations**
  - Addition, subtraction, scalar multiplication
  - Dot products & cross products
- **Length and distance**
- **Imaging operations in periodic systems**
  - Rectangular boxes only
- **Templated to allow different types**
  - typedef GCoord for common use

Class Design: `Atom`

- **Fundamental data type**
- **Coordinates**
- **Critical metadata**
  - Atom and residue name and number, segment name
  - Connectivity
  - Charge, mass
- **Not all characteristics need to be available**
  - Some file formats contain different subsets of information
  - Mostly used for selection purposes

Class Design: Atom

- **Selection in LOOS is a copy**
  - Lightweight copying requires using pointers
  - Problem: memory management is hard
- **Solution: Shared pointers**
  - `pAtom` is typedef to shared pointer class
  - Use reference-counted shared pointers
  - All of the advantages of pointers without costs
  - Never see bare `Atom`s in LOOS

Class Design: AtomicGroup

- **AtomicGroups store pAtoms**
  - Inexpensive copying and subsetting
- **Structure data formats are subclasses**
  - Factory function calls correct code to read file
    - PDB, prmtop, psf, gro, tinker xyz
  - Returns AtomicGroup by up-casting
  - Application doesn’t know what format was used
    - Agnostic, except if needed info is missing
- **Selection, copying create new AtomicGroups**
  - Copy pointers → all point to the same data

Class Design: AtomicGroup

- **AtomicGroup is the workhorse class**
  - Try to make all common operations 1-liners
- **Key functionality**
  - Merging & splitting
  - Aligning two groups
  - Rotations & translations
  - Principal axes
  - Center of mass or centroid
  - Dipole moment
  - Contacts between groups
  - Lots of other stuff
  - This is where you look before you write something new

Class Design: Trajectory

- **Essence of most MD analysis is iterating over trajectory frames**
- **Trajectory class is the key**
  - Implements common access mechanism
  - Specific file formats are subclasses
    - CHARMM/NAMD: DCD
    - Amber: MDTRAJ, NETCDF
    - GROMACS: TRR, XTC
    - Tinker: ARC
    - PDB files
    - Differences in file formats hidden behind common interface
  - Factory function auto-detects trajectory format
    - Applications just use pointer to Trajectory

pyloos.Trajectory: a more pythonish Trajectory interface

- **Wrapper around Trajectory class**
- **More pythonish behavior**
  - Specify skip, stride, etc at creation
  - Treat the trajectory like an iterator

```
for frame in traj:
    # do something
```

  - VirtualTrajectory
    - Handle multiple pyloos.Trajectory objects as a single iterator
  - AlignedVirtualTrajectory
    - Iteratively align multiple trajectories behind the scenes

Output formats

- **Structures: PDB**
- **Trajectories**
  - DCD (default)
  - XTC
- **Matrices**
  - MATLAB format
- **Electron density**
  - XPLOR format
- **Other datasets**
  - Whitespace-delimited ASCII text
  - Formatted for easy plotting with gnuplot

Documentation

- **HTML in `Docs/`**
  - Generated from source via Doxygen
- **Short description of tools**
  - "`--fullhelp`" has more detail
- **Class documentation**
  - List of all methods
  - Inheritance diagrams
  - Explanation of algorithms
  - References where appropriate
- **GitHub wiki has short articles**

Outline

- Introduction and Design goals
- Using LOOS Tools
- Developing with LOOS
- **Example Problem: Domain motion**

Example problem

- **Track the motion of 2 chunks of a protein relative to each other**
  - Distance
  - Angle
  - Torsion
- **Pieces of the calculation**
  - Read structure
  - Select the 2 domains
  - Loop over trajectory
    - Compute centroid and principal axes for each domain
    - Compute distance
    - Compute angle between first axes

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)
```

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select "domains"**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

```
# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)
```

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```python
for frame in traj:
```

```python
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:
    # compute distance
    centroid1 = sel1.centroid()
    centroid2 = sel2.centroid()

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

    # compute distance
    centroid1 = sel1.centroid()
    centroid2 = sel2.centroid()
    diff = centroid2 - centroid1
    distance = diff.length()

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

```
# 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
```

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select "domains"**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

    # 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 between principal axes
    p1 = centroid1 + axis1
    p2 = centroid2 + axis2
    tors = loos.torsion(p1, centroid1, centroid2, p2)

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

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

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

PyLOOS Solution

- **Read command line**
- **Create system**
- **Select “domains”**
- **Loop over trajectory**
  - Compute distance
  - Compute angle
  - Compute torsion

```
#!/usr/bin/env python3

import sys
import loos
import loos.pyloos
import math

header = " ".join(sys.argv)
print("# ", header)

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

# create the system and trajectory
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system)

# apply selections to get atoms
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)

for frame in traj:

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

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

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

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

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

Installing LOOS

- **Current (version 3.3 and earlier)**
  - Must build locally
  - Easiest with conda (`conda_build.sh`)
  - OS libraries tested as well
- **Version 4.0 (end of summer)**
  - Conda only
  - Will be a package on conda-forge

Summary

- **LOOS is powerful analysis platform**
  - High-quality tools
  - Rapidly implement new analysis methods
- **Download**
  - https://github.com/GrossfieldLab/loos
- **References**
  - Romo, T. D.; Grossfield, A. *Conf Proc IEEE Eng Med Biol Soc* **2009**, 2332–2335
  - Romo et al, *J Comput Chem*, 2014, 35, 2305-2318
- **Dr. Tod D. Romo**
  - Most of low-level design and implementation

@agrossfield