PLUMED's collective variables in the High Throughput Molecular Dynamics analysis environment

An object model for CVs

GitHub
@tonigi
PLUMEDws2017

Toni Giorgino
National Research Council of Italy
toni.giorgino@cnr.it

May 27, 2017

Motivation

Idea – Combine PLUMED with a high-level Python-based molecular environment

Overview

1. Markov w/ large-scale MD, HTMD*
2. PLUMED-based projections: *MetricPlumed2*
3. (Not-) writing PLUMED CVs
4. Plumed-based Markov Models
5. Availability

* acellera®

Markov models from large-scale biomolecular simulations

Noe et al., Cur. Op. Str. Biol. 2014

De Fabritiis et al., PNAS 2012

Pérez-Hernández et al., JCP 2013

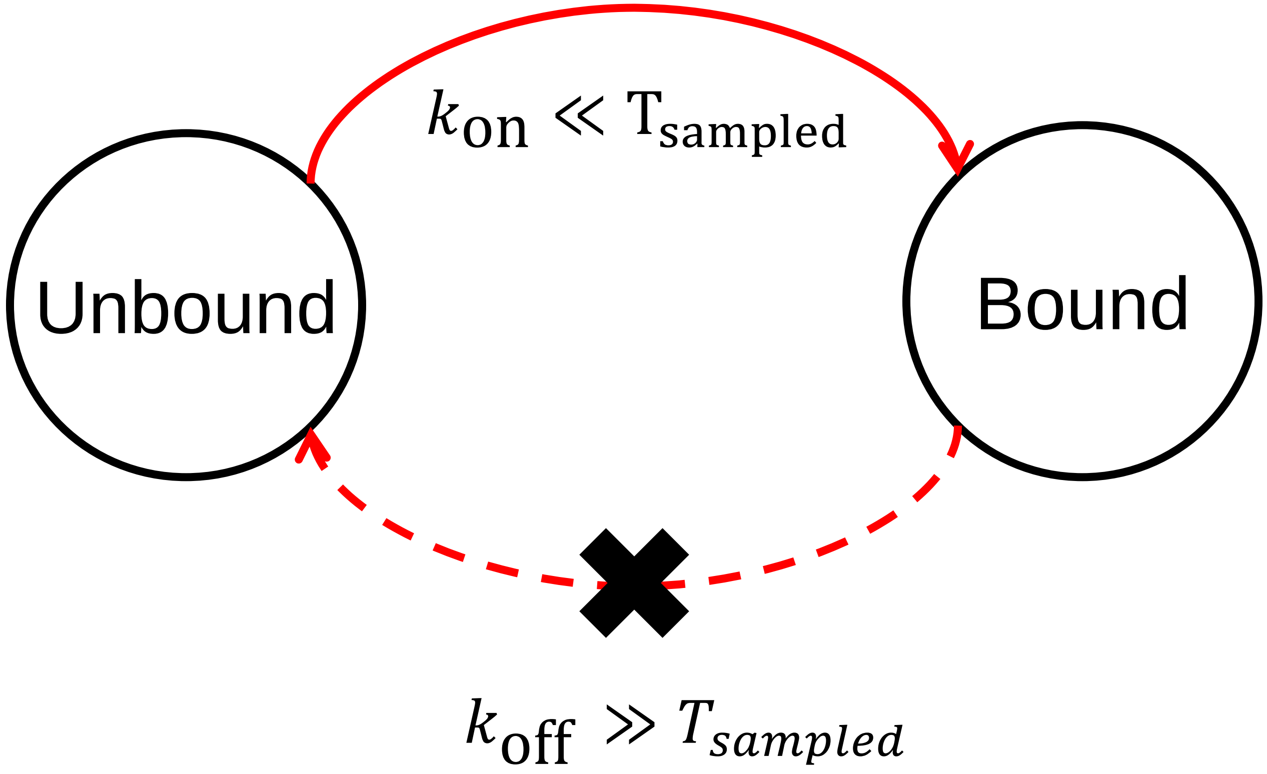
Markov Models

- Estimate $P(x_{t+\tau}|x_t)$ from observations
  - Assume discrete states, time-independent *P*
  - Need no force bias*
  - Suitable for large scale MD (…requires it)
  - Efficient use of parallel trajectories
  - Bias can be "statistic" (adaptive schemes)
- Success cases:
  - Ligand-binding kinetics (on & off)
  - Large conformational transitions
  - Protein folding...

* Not unlike forward-flux sampling

Move beyond this model…

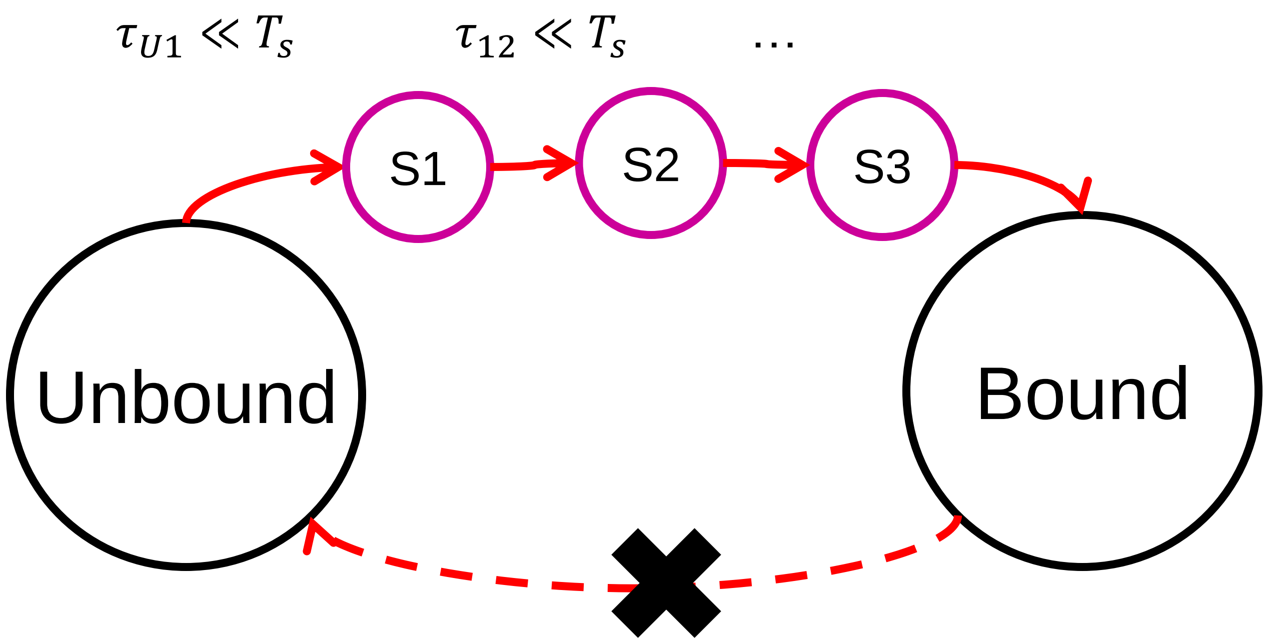
On the timescales sampled by unbiased MD…


$k_{\mathrm{on}} \ll \mathrm{T}_{\mathrm{sampled}}$
Unbound
Bound
$k_{\mathrm{off}} \gg T_{sampled}$

k_{on} - estimated counting events vs sampled time

Idea

- We introduce several *intermediate steps*

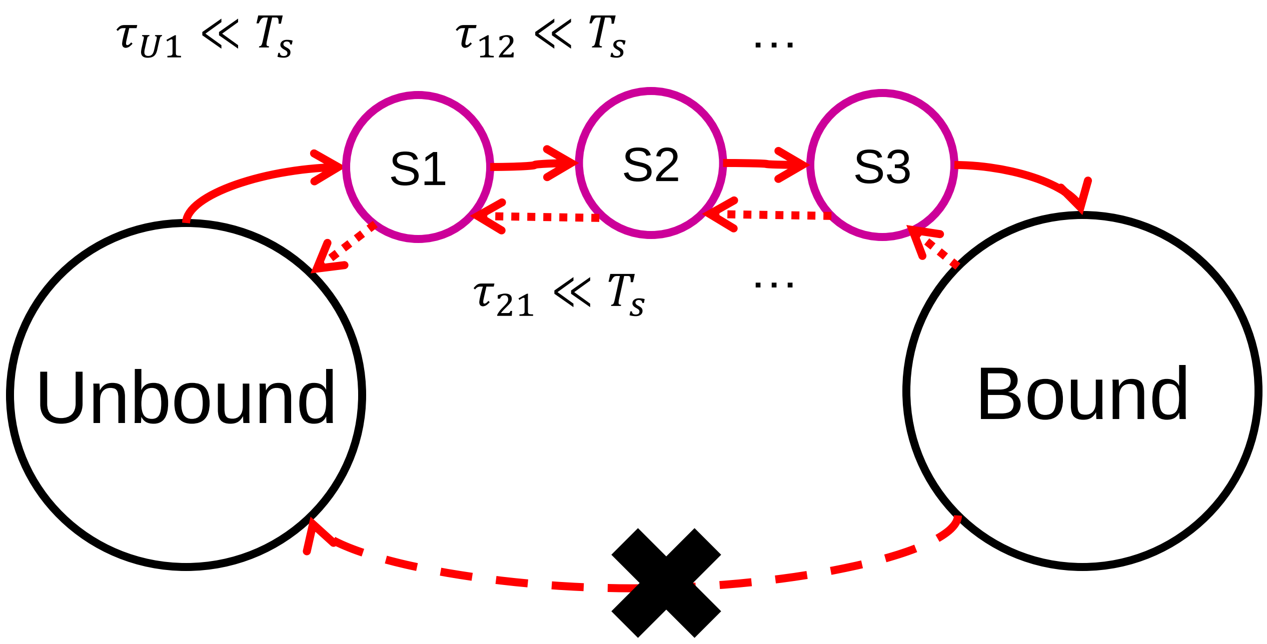


k_{on} - reconstructed combining rates of intermediate steps

Buch et al., PNAS 2012

Idea

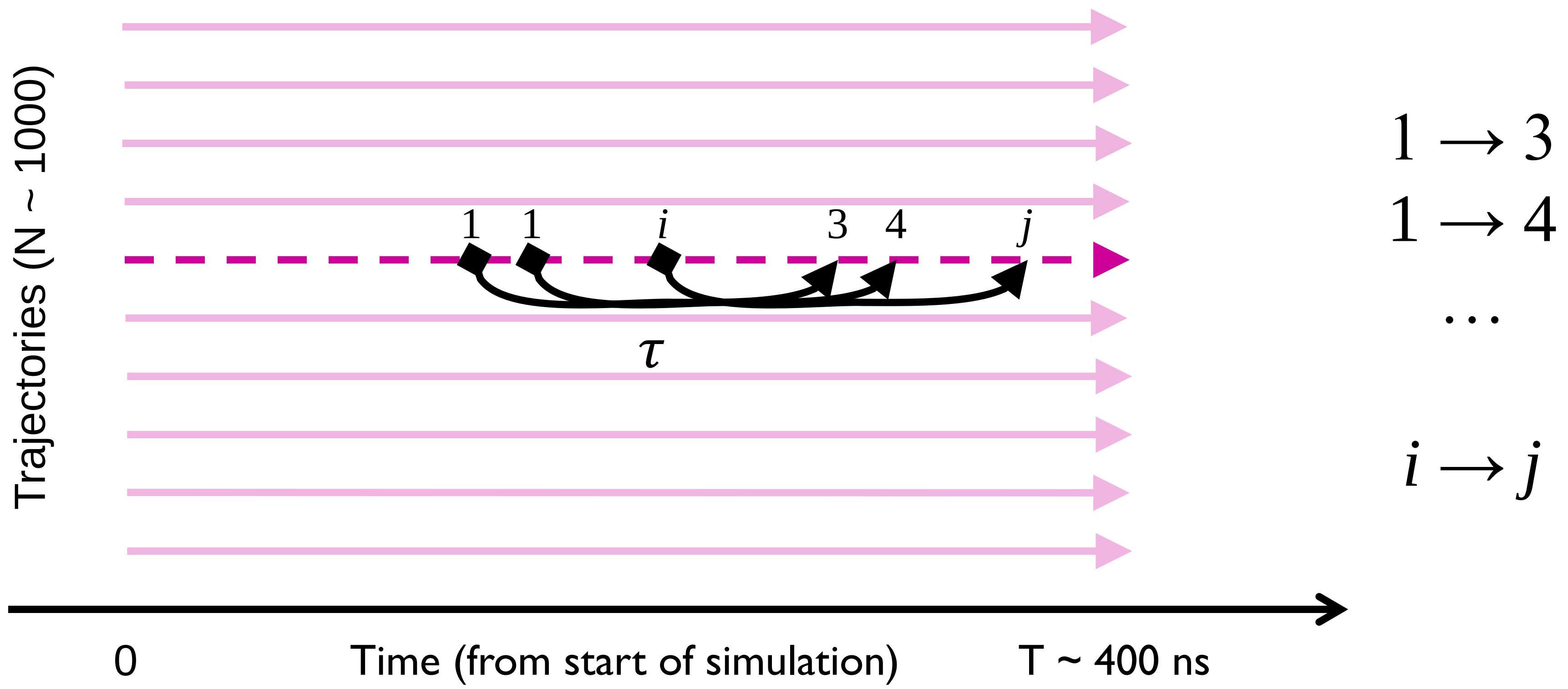
- We introduce several *intermediate steps*



k_{on} - reconstructed combining rates of intermediates steps

k_{off} - may be also reconstructed, if states are fine-grained!

Trajectories → States → Markov state model



$$N\left(X_{t-\tau} \xrightarrow{\text{lag } \tau} X_t\right) \Rightarrow$$

$$P_{ij} = P(X_t = j \mid X_{t-\tau} = i)$$

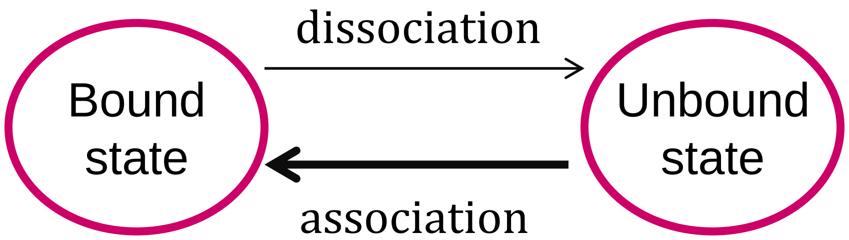
P_{ij} = Time-independent probability to change state from i to j at each point in time (constant)

Why do we care?

1. Thanks to memoryless-ness and homogeneity assumptions, one can accumulate counts from **different** trajectories in the **same** matrix.

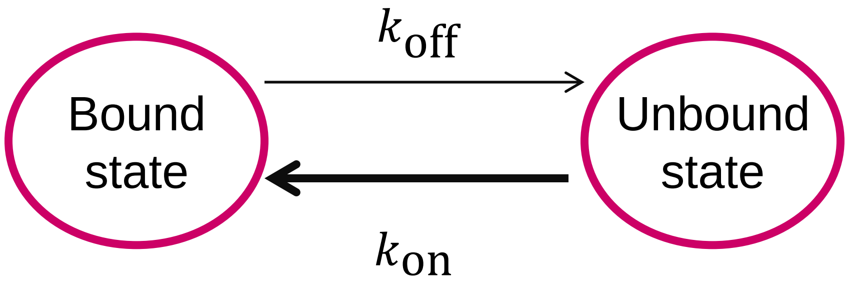
2. Because equilibrium probabilities *are* the free energy of binding (ΔG), an important determinant of drug potency.

$$K_D = \frac{p_U}{p_B} = \exp\frac{-\Delta G}{RT} \quad \left(\propto \frac{[P][L]}{[PL]}\right)$$



3. Because **mean-first-passage times** correspond to inter-state kinetics...

$$K_D = \frac{k_{\text{off}}}{k_{\text{on}}} \propto \frac{\tau_{\text{on}}}{\tau_{\text{off}}} \sim \frac{t_U^{(B)}}{t_B^{(U)}}$$



Code by yourself, or…

- Building matrices builds on non-trivial algorithms (MLE of reversible process, bootstrapping, convergence, …)
- Extraction of observables from *large* amounts of trajectories
- → Strongly motivated analysis frameworks, e.g. MSMBuilder, PyEMMA, **HTMD***

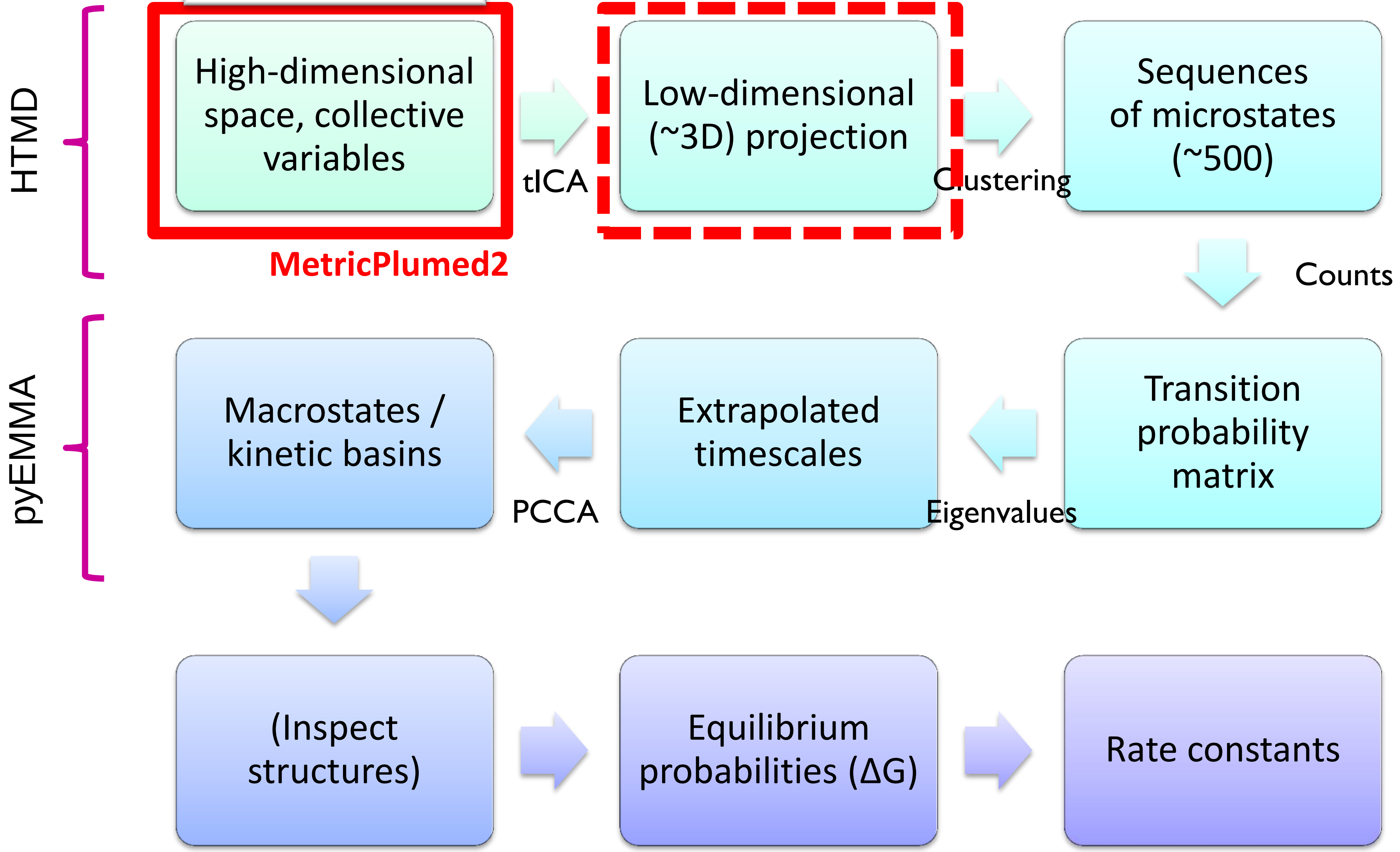
* Deals with *building, run and analysis*; we'll only see the latter

Harrigan, Biophys J 2017; Scherer, JCTC 2015; Doerr, JCTC 2016

Markov Workflow Overview

- Start in an arbitrary space, high dimensional
  - **This is what Plumed addresses well…**
  - …and the motivation of this talk
- Reduce dimension, try to preserve "slow" DOF (e.g. use tICA)
  - MM have this problem in common with metadynamics
- Compute transition matrix at several lag times
  - Cluster, eigenvectors, convergence, etc.
- If bad projection, **do not resimulate**: pick new projections, **re-calculate CVs** and model

Typ M analysis


HTMD
High-dimensional space, collective variables
MetricPlumed2
tICA
Low-dimensional (~3D) projection
Clustering
Sequences of microstates (~500)
Counts
pyEMMA
Transition probability matrix
Eigenvalues
Extrapolated timescales
PCCA
Macrostates / kinetic basins
(Inspect structures)
Equilibrium probabilities (ΔG)
Rate constants

The MetricPlumed2 object model

Basic Usage (1 trajectory)

- Install HTMD via *conda* (see website)
- Start Python and import packages

```
from htmd import *                         # Experimental, so...
from htmd.projections.metricplumed2 import *
                                           # Check ...
htmd.projections.metricplumed2._getPlumedRoot()
```

- Instantiate *MetricPlumed2* objects
- Then `metric.project(molecule)`

MetricPlumed2 – string form

- Pass METAINP as strings:

```
metric = MetricPlumed2(['d1: DISTANCE ATOMS=1,200',
                        'd2: DISTANCE ATOMS=5,6'])
mol=Molecule(“1KDX”)                                # 17 frames
metric.project(mol)
    array([[ 22.44125175,   2.80880904],
           [ 21.7255497 ,   2.9548099 ],
           [ 22.68965721,   2.99287391], …
```

- **Note** CVs are strings; atoms are inconveniently selected as serials. Solution in next slides.

Object-oriented CV creation (1)

- Instead of strings, we can build and pass *PlumedCV* objects. Construct with the following arguments, in order…
  1. CV name (DISTANCE, GYRATION,…)
  2. unique label (default autogenerated)
  3. CV arguments, as Python named arguments

```
gyr1 = PlumedCV("GYRATION", "gyr1", ATOMS="10-20", TYPE="RADIUS")
str(gyr1)
    'rgyr: GYRATION ATOMS=10-20 TYPE=RADIUS'
```

Object-oriented CV creation (2)

- *Atom groups* and *centers of masses* are also objects. They can be passed *in lieu* of atom indices or lists.
- *PlumedGroup* and *PlumedCOM* constructors:
  1. the molecule corresponding to the system
  2. the group label
  3. an atom selection (VMD syntax)

```
protCA = PlumedCOM(m, "protCA", "chain A and name CA")
lig    = PlumedCOM(m, "lig",    "resname BEN and noh")
dCAlig = PlumedCV("DISTANCE", "dCAlig", ATOMS=[protCA,lig])
```

CV dependencies

- Note dependencies between CVs; e.g.
  - `dCAlig: DISTANCE ATOMS=protCA,lig`
  - `COMBINE ARG=dCAlig POWERS=2.0`
- *Topological sort** ensures correct evaluation

```
dCAlig2=PlumedCV("COMBINE",     ARG=[dCAlig],
                 POWERS="2.0", label="dCAlig2")
myDistMetric=MetricPlumed2(dCAlig2)
print(myDistMetric)
    protCA: COM ATOMS=2,10,17,21,25,37,44,50...
    lig: COM ATOMS=1632,1633,1634,1635,1636,1637,...
    dist: DISTANCE ATOMS=protCA,lig
    dCAlig2: COMBINE ARG=dist POWERS=2.0
```

* and deduplication

Multi-trajectory usage

- Instantiate *MetricXXX* objects
  - `myPlumedMetric = MetricPlumed2(…)`
- Create a *simlist* object
  - `slist = simlist( […], ‘name.pdb’)`
- Put it in a *Metric* object
  - `metr = Metric(slist)`
  - `metr.set(myPlumedMetric)`
- .project() – yields a *MetricData* object
  - `data = metr.project()`

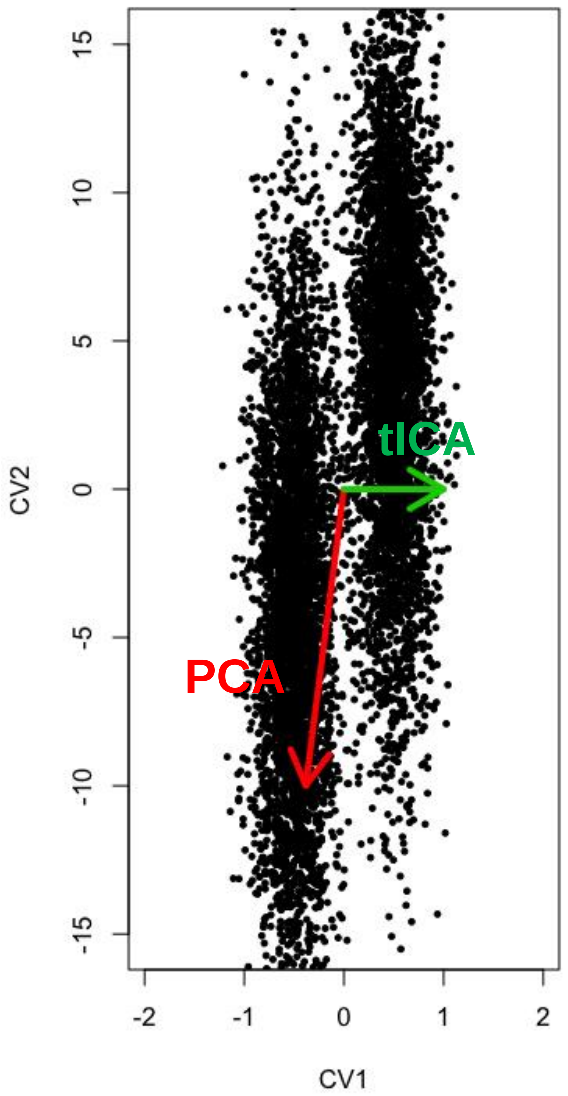
Time-lagged independent component analysis

- A low-dimensional projection on the "slow" DOF (based on lagged autocorrelation)
- Instantiate a *TICA* object

```
tica = TICA(data, 2,
                units='ns')
```

- Reduce to 3 dimensions

```
dataTica = tica.project(3)
```



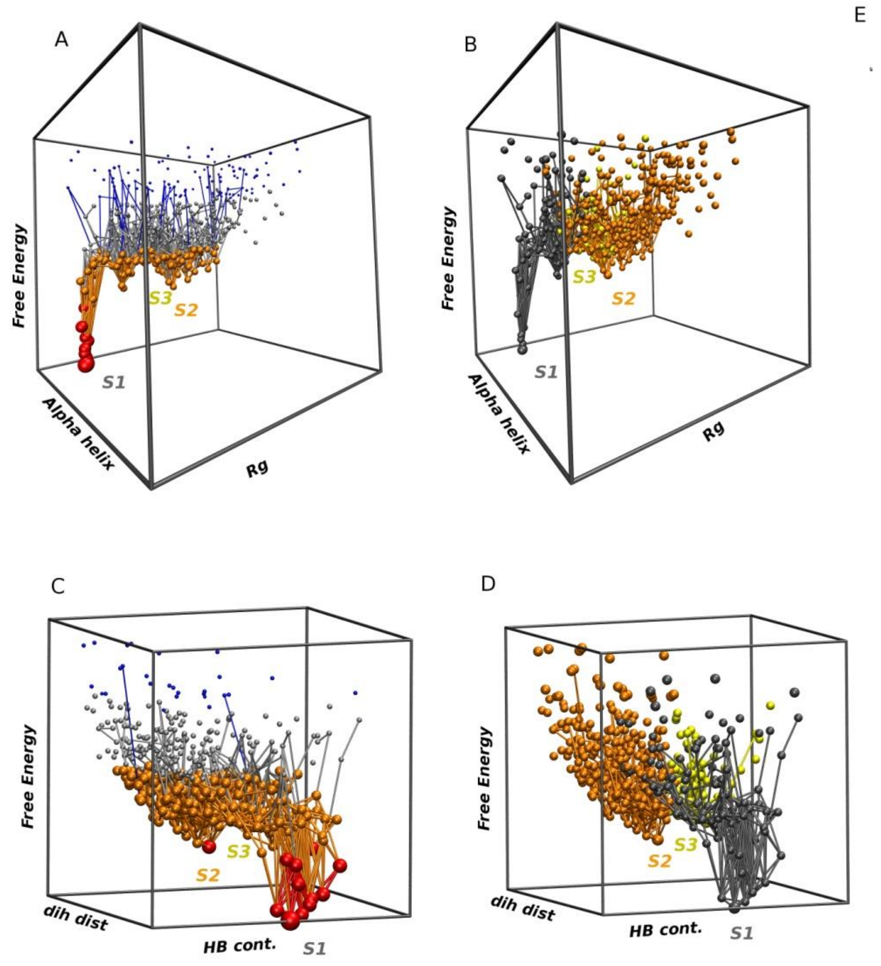
J. Chem. Phys. 139, 015102 (2013).

Discretization

- Clustering gets you discrete states
  - `data.cluster(MiniBatchKMeans(n_clusters=100))`
- Model built here
  - `model=Model(data)`
- Eigenvalues = relaxation timescales
  - `model.plotTimescales()`
- Macrostates
  - `model.markovModel(2,4,units="ns")`
  - `model.eqDistribution()`

Examples

Villin headpiece example (1)

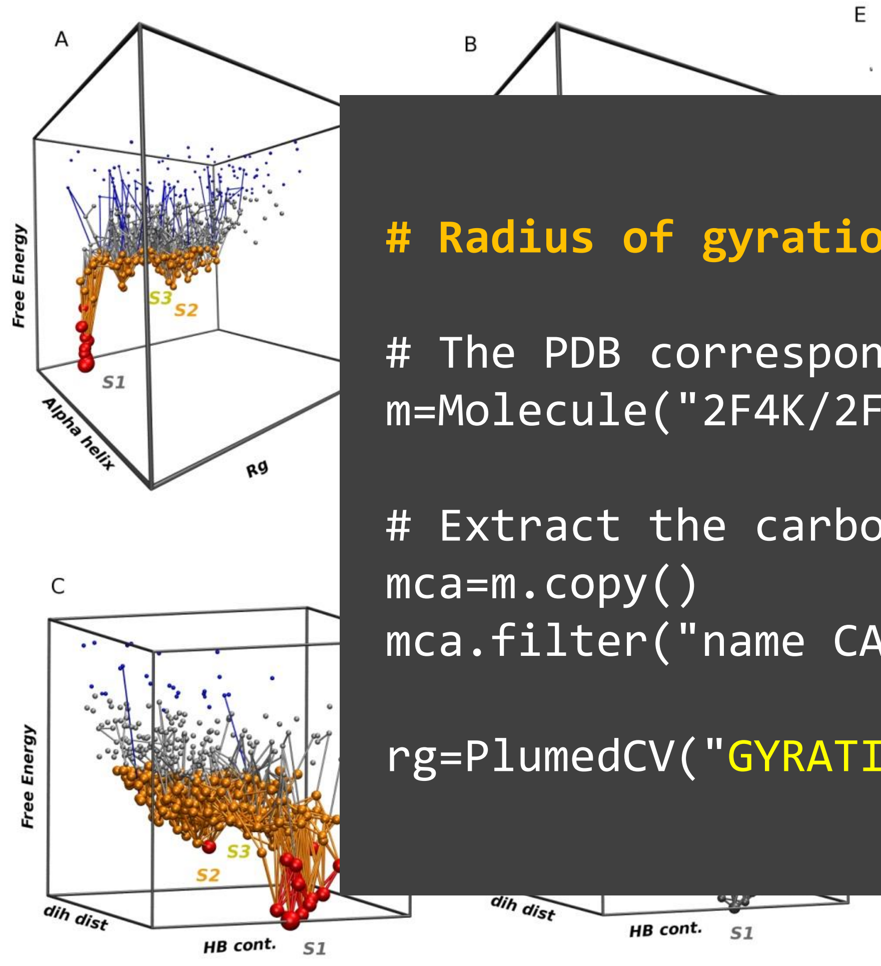


"The collective variables employed were the content of α-helix, the radius of gyration, […] and the number of hydrophobic contacts.

The number of hydrophobic contacts was computed […] between all the possible hydrophobic residues side-chain pairs" *

* Giorgino, Laio, Rodriguez CPC 2017

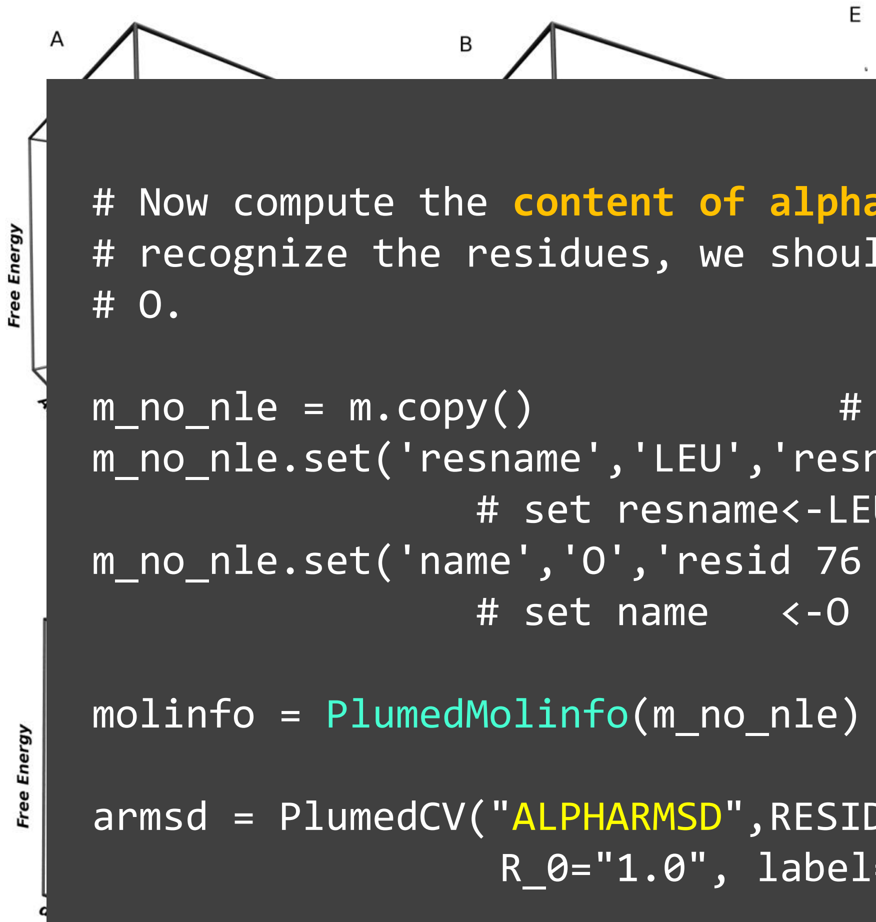
Villin headpiece example (I)



```
# Radius of gyration of Cα

# The PDB corresponding to the structure
m=Molecule("2F4K/2F4k-0.pdb")

# Extract the carbon-alpha atoms only
mca=m.copy()
mca.filter("name CA")

rg=PlumedCV("GYRATION",ATOMS=mca,label="rg")
```

Giorgino, Laio, Rodriguez CPC 2017

Villin headpiece example (2)



```
# Now compute the content of alpha-helix. For PLUMED's MOLINFO to
# recognize the residues, we should rename NLE as LEU, and OT1 as
# O.

m_no_nle = m.copy()                  # Work on a copy
m_no_nle.set('resname','LEU','resname NLE')
                # set resname<-LEU where resname==NLE
m_no_nle.set('name','O','resid 76 and name OT1')
                # set name   <-O   where resid==76 and name==OT1

molinfo = PlumedMolinfo(m_no_nle)

armsd = PlumedCV("ALPHARMSD",RESIDUES="42-76",
                 R_0="1.0", label="armsd")
```

Giorgino, Laio, Rodriguez CPC 2017

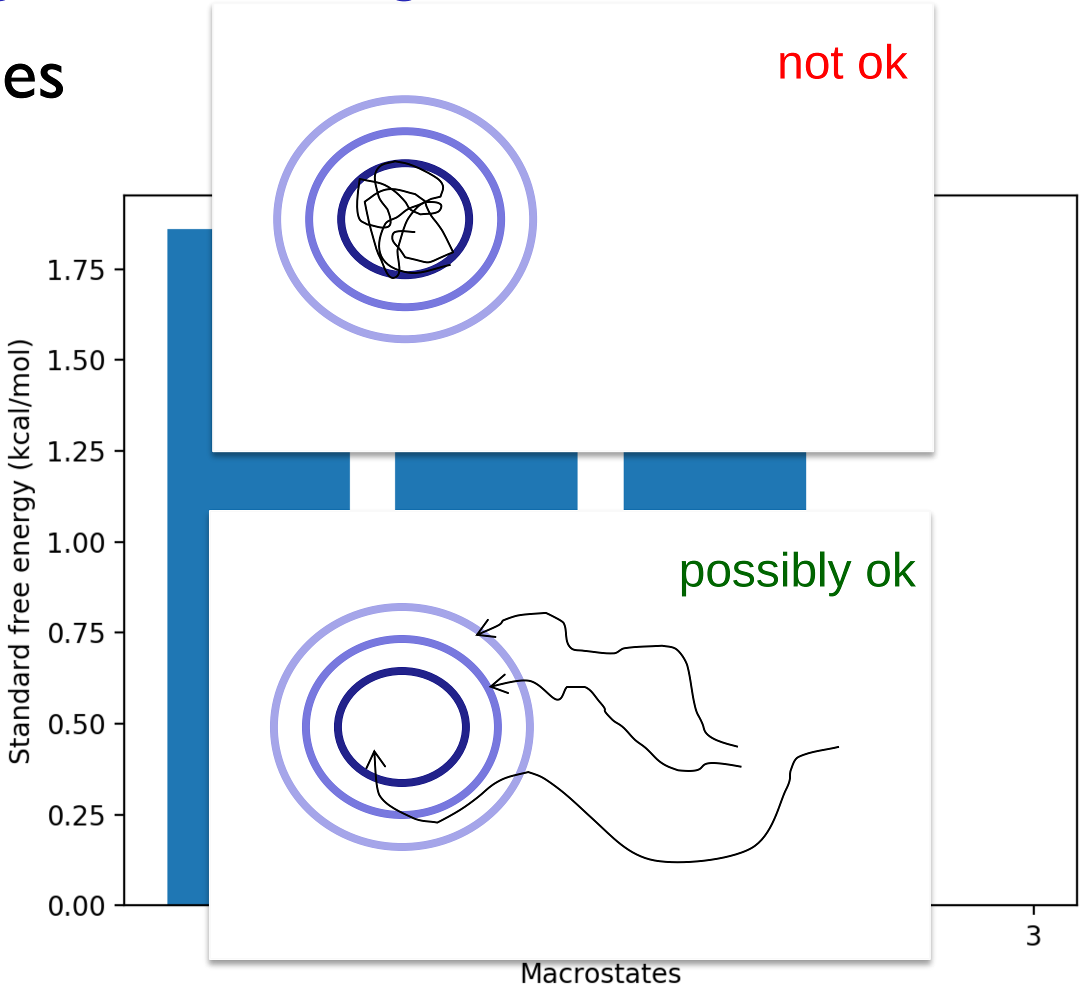
```
# Number of hydrophobic contacts - i.e. coordination number (3.5 A)
# of heavy atoms in the sidechains of hydrophobic residues.

# First, make a group for each hydrophobic sidechain […]
for resid in hyd_resid:
    pg.append(PlumedGroup(m,
                          "hg_{}".format(i),
                          "resid {} and not name N CA C O and noh".
                                 format(resid)))

# Second, form COORDINATION CVs between all the pairs. (17*16/2=136 pairs).
for g1 in range(ngroups):
    for g2 in range(g1+1,ngroups):
        group_pairs.append(PlumedCV("COORDINATION",
                                    GROUPA=pg[g1],
                                    GROUPB=pg[g2],
                                    R_0="3.5",
                                    label="c_{}".format(i)))

# Third, sum all contacts counted above
hydrophobic_contacts_sum = PlumedCV("COMBINE", ARG=group_pairs,
                                         PERIODIC="NO", label="hbc")
```

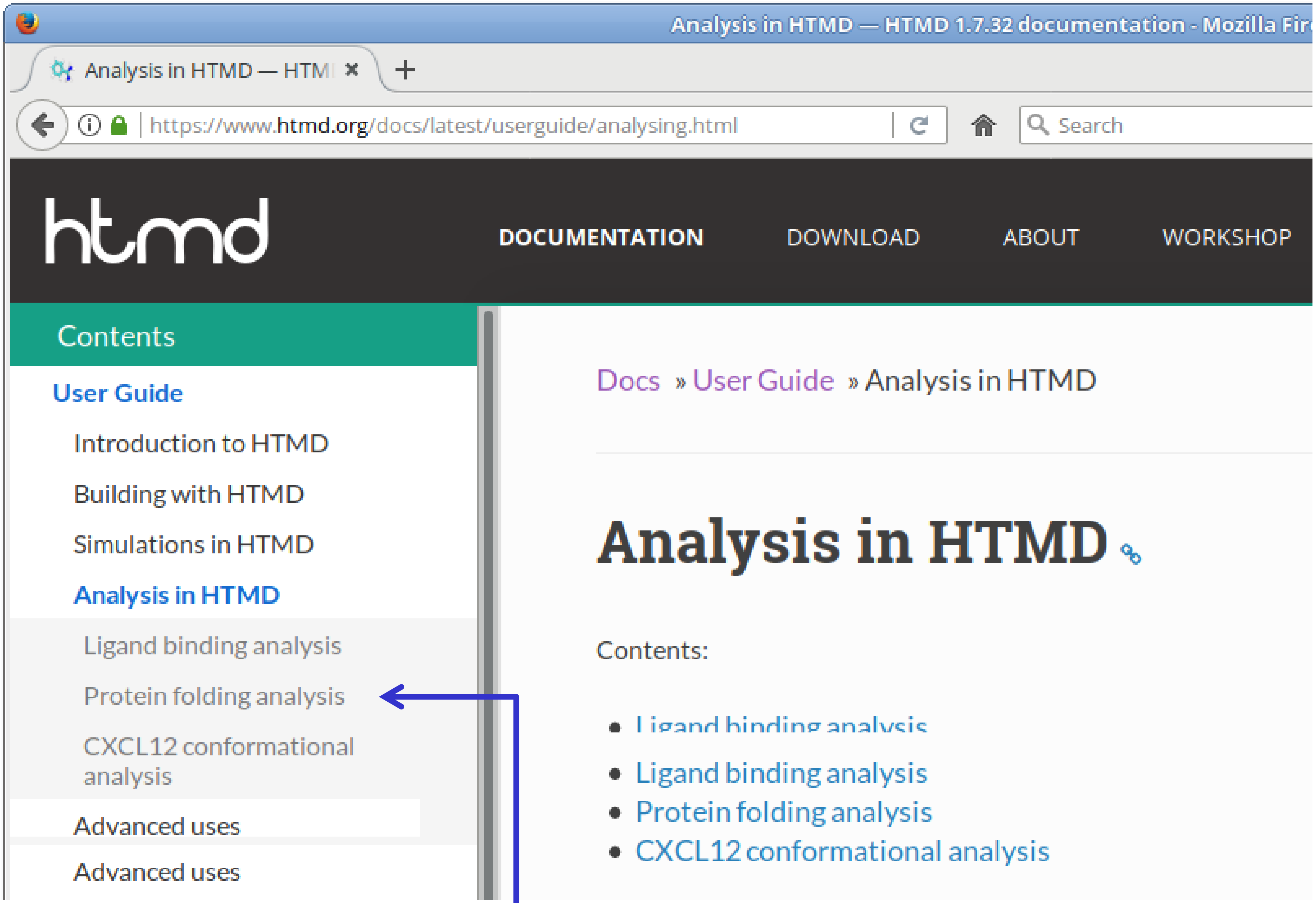
Outputs…

```
MOLINFO STRUCTURE=/var/folders/qz/7p0f8wdj4zdd8nwxm89xzhy80000gn/T/tmp8zgqt19m.pdb
armsd: ALPHARMSD RESIDUES=42-76 R_0=1.0
lab_1: GROUP ATOMS=5,24,35,47,62,74,94,116,126,142,162,169,186,200,224,235,245,265,275,289,310,322,341,365,384,401,418,436,455,474,496,511,533,540,562
rg: GYRATION ATOMS=lab_1
hg_15: GROUP ATOMS=542,545,547,551
hg_16: GROUP ATOMS=558,559,564,567,568,570,572,574,576
hg_14: GROUP ATOMS=457,460,463,466
hg_13: GROUP ATOMS=438,441,443,447
hg_12: GROUP ATOMS=367,370,373,376
hg_11: GROUP ATOMS=343,346,347,349,351,352,353,355,357,359
hg_10: GROUP ATOMS=324,327,329,333
hg_9: GROUP ATOMS=291,294,296,300
hg_8: GROUP ATOMS=267
hg_7: GROUP ATOMS=247,250,251,253,255,257,259
hg_6: GROUP ATOMS=237
hg_5: GROUP ATOMS=171,174,177,178
hg_4: GROUP ATOMS=144,147,148,150,152,154,156
hg_3: GROUP ATOMS=128,130,134
hg_2: GROUP ATOMS=118
hg_1: GROUP ATOMS=76,79,80,82,84,86,88
hg_0: GROUP ATOMS=7,10,12,16
c_1: COORDINATION GROUPA=hg_0 GROUPB=hg_1 R_0=3.5
c_2: COORDINATION GROUPA=hg_0 GROUPB=hg_2 R_0=3.5
c_3: COORDINATION GROUPA=hg_0 GROUPB=hg_3 R_0=3.5
c_4: COORDINATION GROUPA=hg_0 GROUPB=hg_4 R_0=3.5
c_5: COORDINATION GROUPA=hg_0 GROUPB=hg_5 R_0=3.5

[…]

c_130: COORDINATION GROUPA=hg_12 GROUPB=hg_16 R_0=3.5
c_131: COORDINATION GROUPA=hg_13 GROUPB=hg_14 R_0=3.5
c_132: COORDINATION GROUPA=hg_13 GROUPB=hg_15 R_0=3.5
c_133: COORDINATION GROUPA=hg_13 GROUPB=hg_16 R_0=3.5
c_134: COORDINATION GROUPA=hg_14 GROUPB=hg_15 R_0=3.5
c_135: COORDINATION GROUPA=hg_14 GROUPB=hg_16 R_0=3.5
c_136: COORDINATION GROUPA=hg_15 GROUPB=hg_16 R_0=3.5
hbc: COMBINE
ARG=c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8,c_9,c_10,c_11,c_12,c_13,c_14,c_15,c_16,c_17,c_18,c_19,c_20,c_21,c_22,c_23,c_24,c_25,c_26,c_27,c_28,c_29,c_30,c_31,c_32,c_33,
c_34,c_35,c_36,c_37,c_38,c_39,c_40,c_41,c_42,c_43,c_44,c_45,c_46,c_47,c_48,c_49,c_50,c_51,c_52,c_53,c_54,c_55,c_56,c_57,c_58,c_59,c_60,c_61,c_62,c_63,c_64,c_65,
c_66,c_67,c_68,c_69,c_70,c_71,c_72,c_73,c_74,c_75,c_76,c_77,c_78,c_79,c_80,c_81,c_82,c_83,c_84,c_85,c_86,c_87,c_88,c_89,c_90,c_91,c_92,c_93,c_94,c_95,c_96,c_97,
c_98,c_99,c_100,c_101,c_102,c_103,c_104,c_105,c_106,c_107,c_108,c_109,c_110,c_111,c_112,c_113,c_114,c_115,c_116,c_117,c_118,c_119,c_120,c_121,c_122,c_123,c_124,
c_125,c_126,c_127,c_128,c_129,c_130,c_131,c_132,c_133,c_134,c_135,c_136 PERIODIC=NO
# Rendered PlumedStatement
```

No free lunch demo: 1μs Ace-Ala3-Nme

- Part of examples at: github.com/tonigi/PLUMEDws2017
- 6 Ramachandran angles
- The system is trapped
- How to «shoot» trajectories:
  - «bathtub» not ok
  - «shower» maybe
  - adaptive spawning
  - or your favourite string-like method





Analysis in HTMD — HTMD 1.7.32 documentation - Mozilla Fir

Analysis in HTMD — HTMI

https://www.htmd.org/docs/latest/userguide/analysing.html

Search

htmd

DOCUMENTATION DOWNLOAD ABOUT WORKSHOP

Contents

User Guide

Introduction to HTMD

Building with HTMD

Simulations in HTMD

Analysis in HTMD

Ligand binding analysis

Protein folding analysis

CXCL12 conformational analysis

Advanced uses

Advanced uses

Docs » User Guide » Analysis in HTMD

Analysis in HTMD

Contents:

- Ligand binding analysis
- Ligand binding analysis
- Protein folding analysis
- CXCL12 conformational analysis

Each come with 2GB (200 μs) of trajectories

- Still preliminary. To try out:
  1. Get HTMD
  2. If needed, use branch toni-devel-plumed from GitHub and set `PYTHONPATH`
- Examples at github.com/toni/PLUMEDws2017
- Implementation
  - `plumed` invoked externally (I/O via XTC)
  - Poor diagnostics; in case of error, inputs are kept in a directory
  - Parallelized, if multiple trajectories

In summary

- *MetricPlumed2* gets:
  - Sophisticated CVs, courtesy of Plumed, and
  - A Markov "workflow", courtesy of HTMD
  - METAINP files written instantiating objects
- Use cases
  - CV-based Markov-type calculations
  - Scripted analysis (meta-generation of CVs)
  - Bonus: Python-based notebooks generally readable (and reproducible)

Acknowledgements

Prof. G. De Fabritiis & Computational Biophysics Group – Universitat Pompeu Fabra, Barcelona (MD)

Acellera Ltd. (Funding)

Prof. A. Laio, A. Rodriguez – SISSA (Villin, METAGUI 3, clustering)

Prof. F. Passamonti – Univ. Insubria (Biostatistics)

Prof. J. Selent – UPF (GPCR)

Volunteers of GPUGRID.net (MD compute time)