WESTPA Workshop Practicalities of WESTPA

ADAM PRATT

ALI SINAN SAGLAM

Overview

Part 1. WESTPA file format – HDF5

Part 2. Calculating free energy profiles

Part 3. Calculating rate constants

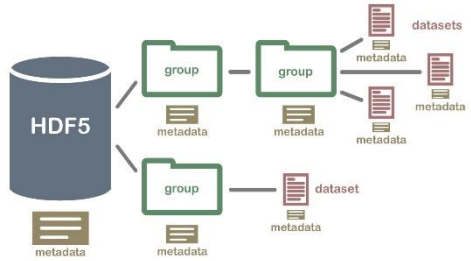
Part 4. Visualizing continuous trajectories

Required libraries/programs

- Python: https://www.python.org/
  - Anaconda python distribution: https://store.continuum.io/cshop/anaconda/
- Numpy: http://www.numpy.org/
- Matplotlib: http://matplotlib.org/
- WESTPA: https://github.com/westpa/westpa

Overview

Part 1. WESTPA file format – HDF5



Part 2. Calculating free energy profiles

Part 3. Calculating rate constants

Part 4. Visualizing continuous trajectories

WESTPA File format

- HDF (Hierarchial Data Format) is a portable, open source file format that allows for
  - Optimized large and complex data storage
  - Fast access for analysis
- Easy to use with Python thanks to h5py library

WESTPA File format

```
west.h5
  bin_topologies
  ibstates
  iterations
  summary
  tstates
```

WESTPA File format

west.h5
- bin_topologies – Contains binning information
- ibstates
- iterations
- summary
- tstates

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations
- summary
- tstates

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations – Iteration data is stored here (more on this later)
- summary
- tstates

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations – Iteration data is stored here (more on this later)
- summary – A summary of all iterations
- tstates

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations – Iteration data is stored here (more on this later)
- summary – A summary of all iterations

```
      (5, 1.0, 1.0, 1.0, 0.2, 0.2, 0.0, 0.012396097183227539, ''),
      (5, 1.0, 1.0, 1.0, 0.2, 0.2, 0.0, 0.012511014938354492, ''),
      (5, 1.0, 1.0, 1.0, 0.2, 0.2, 0.0, 0.012378931045532227, ''),
      (5, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, '')],
     dtype=[('n_particles', '<i8'), ('norm', '<f8'), ('min_bin_prob', '<f8'), ('max_bin_prob', '<f8'), ('min_seg_prob', '<f8'
), ('max_seg_prob', '<f8'), ('cputime', '<f8'), ('walltime', '<f8'), ('binhash', 'S64')])
```

- tstates

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations – Iteration data is stored here (more on this later)
- summary – A summary of all iterations
- tstates – Target state information (if it’s a steady state)

WESTPA File format

west.h5

- bin_topologies – Contains binning information
- ibstates – Contains information about the starting structures
- iterations – Iteration data is stored here (more on this later)
- summary – A summary of all iterations
- tstates – Target state information (if it's a steady state)

Attributes/Meta data:

- west_current_iteration – Keeps track of the current iteration (note that it doesn't determine the current iteration!)

WESTPA File format

```
west.h5
  iterations
    iter_00000001
      auxdata
      ibstates
      pcoord
      seg_index
      wtgraph
```

WESTPA File format

```
west.h5
  iterations
    iter_00000001
      auxdata – Contains the auxiliary data collected
      ibstates
      pcoord
      seg_index
      wtgraph
```

WESTPA File format

```
west.h5
  iterations
    iter_00000001
      auxdata – Contains the auxiliary data collected
      ibstates – Contains information about starting states
      pcoord
      seg_index
      wtgraph
```

WESTPA File format

```
west.h5
  iterations
    iter_00000001
      auxdata – Contains the auxiliary data collected
      ibstates – Contains information about starting states
      pcoord – Contains progress coordinate data
      seg_index
      wtgraph
```

WESTPA File format

west.h5
- iterations
  - iter_00000001
    - auxdata – Contains the auxiliary data collected
    - ibstates – Contains information about starting states
    - pcoord – Contains progress coordinate data
    - seg_index
    - wtgraph

```
 [ 8.5120821 ]],

[[ 7.45688248],
 [ 7.50038242],
 [ 7.47868252],
 [ 7.53348255],
 [ 7.53018236],
 [ 7.59098244],
 [ 7.60468245],
 [ 7.6382823 ],
 [ 7.68318224],
 [ 7.68018246],
 [ 7.66458225],
 [ 7.62328243],
 [ 7.59508228],
 [ 7.61598253],
 [ 7.61728239],
 [ 7.60998249],
 [ 7.62338257],
 [ 7.65088224],
 [ 7.69408226],
 [ 7.7090826 ],
 [ 7.71978235]]], dtype=float32)
```

WESTPA File format

west.h5
- iterations
  - iter_00000001
    - auxdata – Contains the auxiliary data collected
    - ibstates – Contains information about starting states
    - pcoord – Contains progress coordinate data
    - seg_index – Contains parent information, probabilites and more
    - wtgraph

WESTPA File format

west.h5
- iterations
  - iter_00000001
    - auxdata – Contains the auxiliary data collected
    - ibstates – Contains information about starting states
    - pcoord – Contains progress coordinate data
    - seg_index – Contains parent information, probabilites and more

```
       (0.2, 2, 1L, 2L, 0.0, 0.0, 1, 2), (0.2, 3, 1L, 3L, 0.0, 0.0, 1, 2),
       (0.2, 4, 1L, 4L, 0.0, 0.0, 1, 2)],
      dtype=[('weight', '<f8'), ('parent_id', '<i8'), ('wtg_n_parents', '<u8'), ('wtg_offset', '<u8'), ('cputime', '<f8'), ('w
alltime', '<f8'), ('endpoint_type', 'u1'), ('status', 'u1')])
```

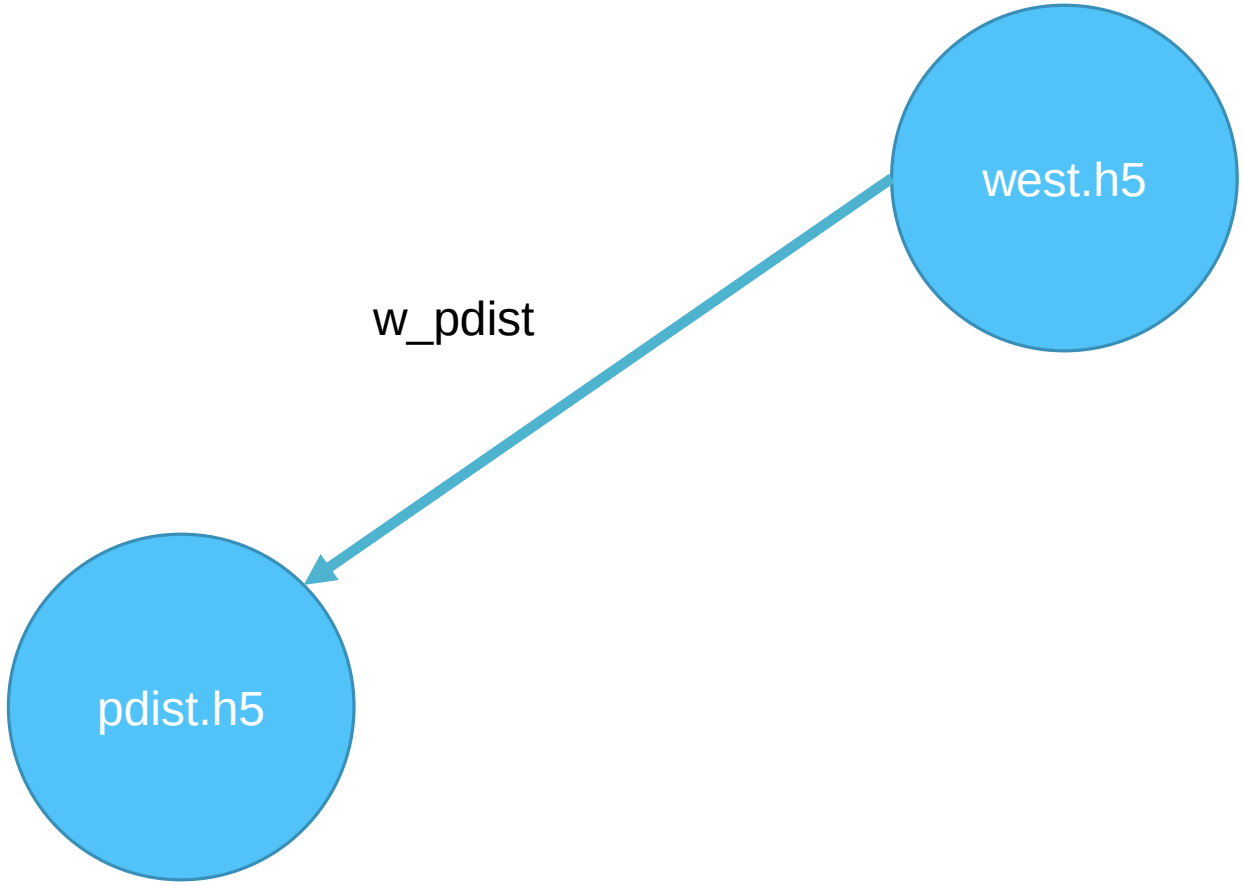
    - wtgraph

WESTPA File format

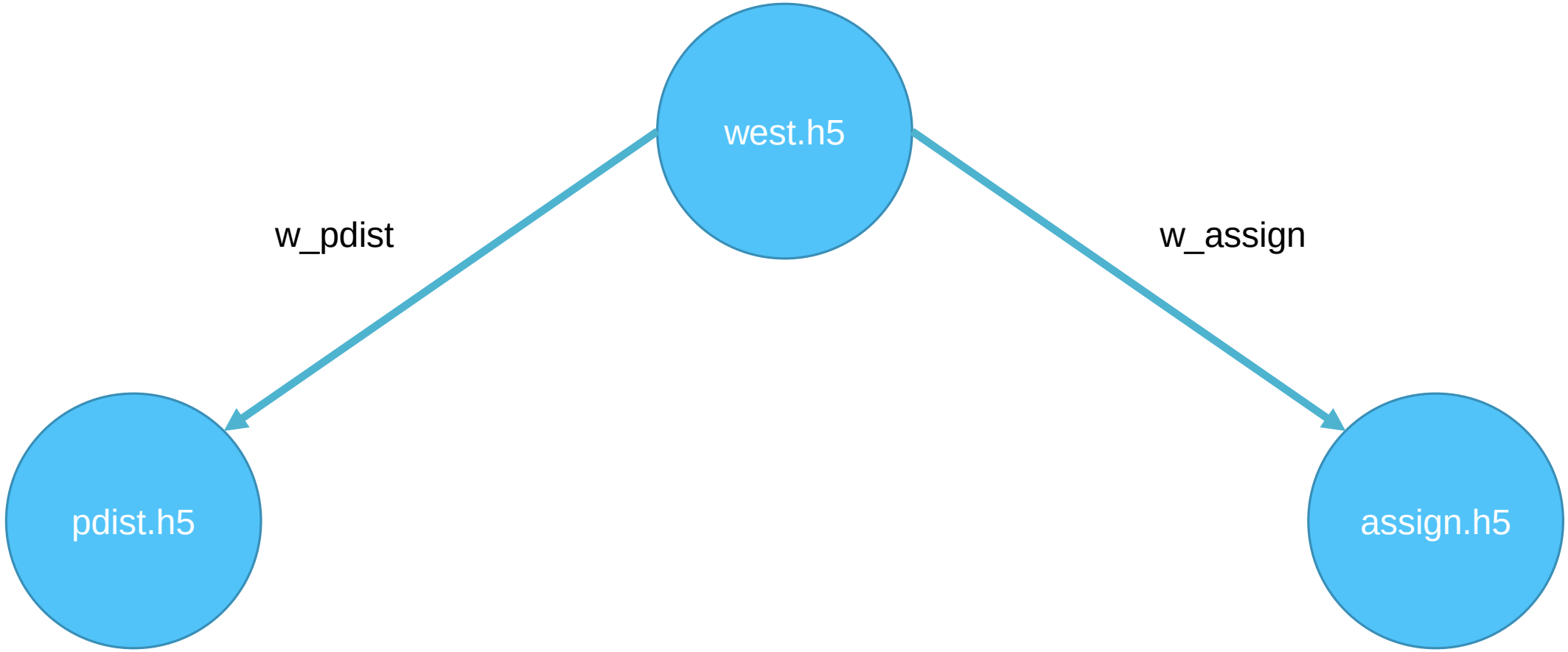
```
west.h5
  iterations
    iter_00000001
      auxdata – Contains the auxiliary data collected
      ibstates – Contains information about starting states
      pcoord – Contains progress coordinate data
      seg_index – Contains parent information, probabilites and more
      wtgraph – Contains connectivity information
```

Basic WESTPA tools

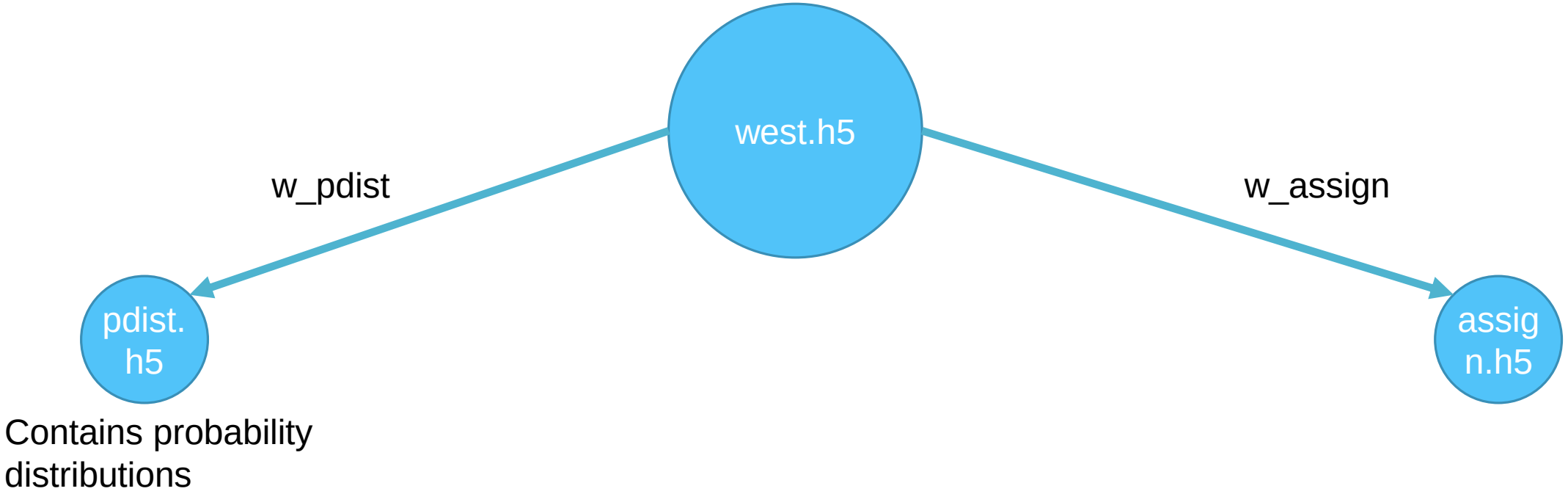
Basic WESTPA tools



Basic WESTPA tools

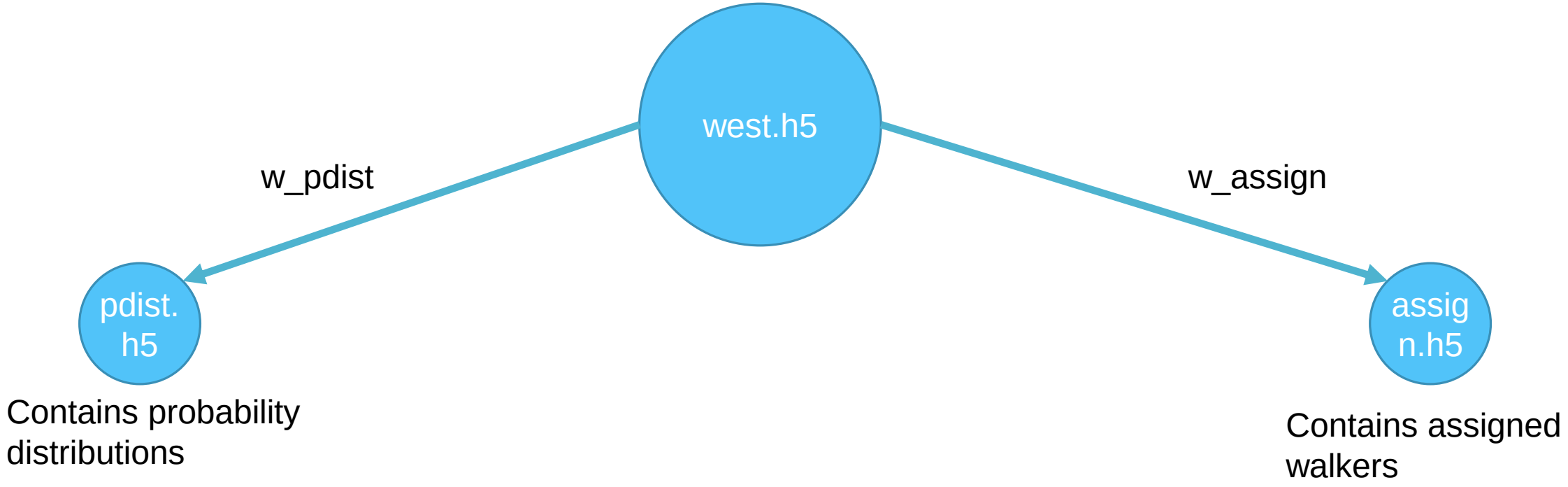


Basic WESTPA tools

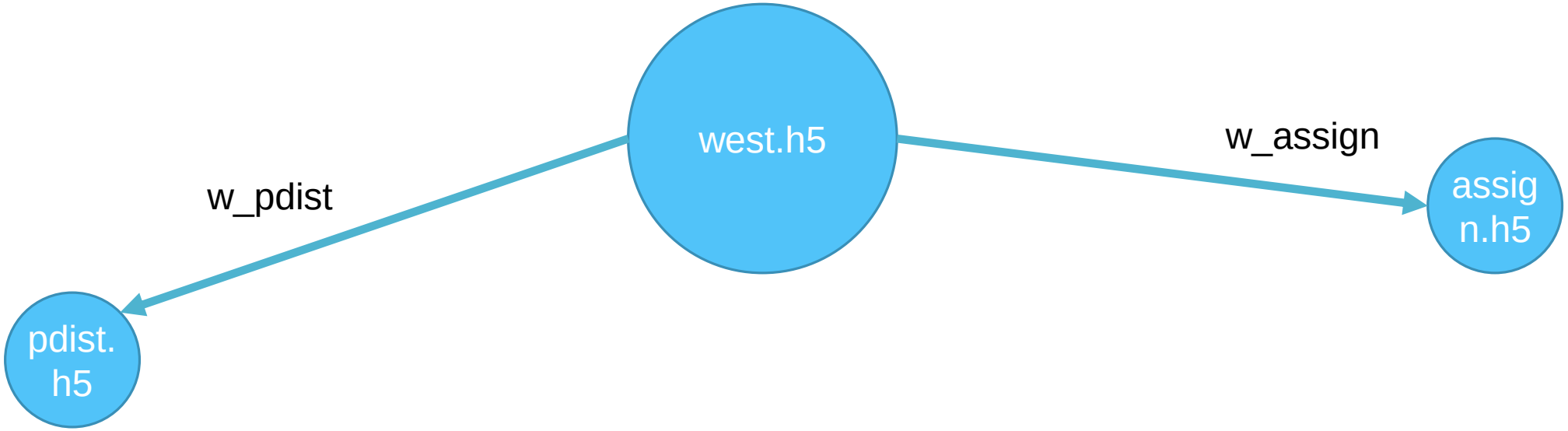


Contains probability distributions

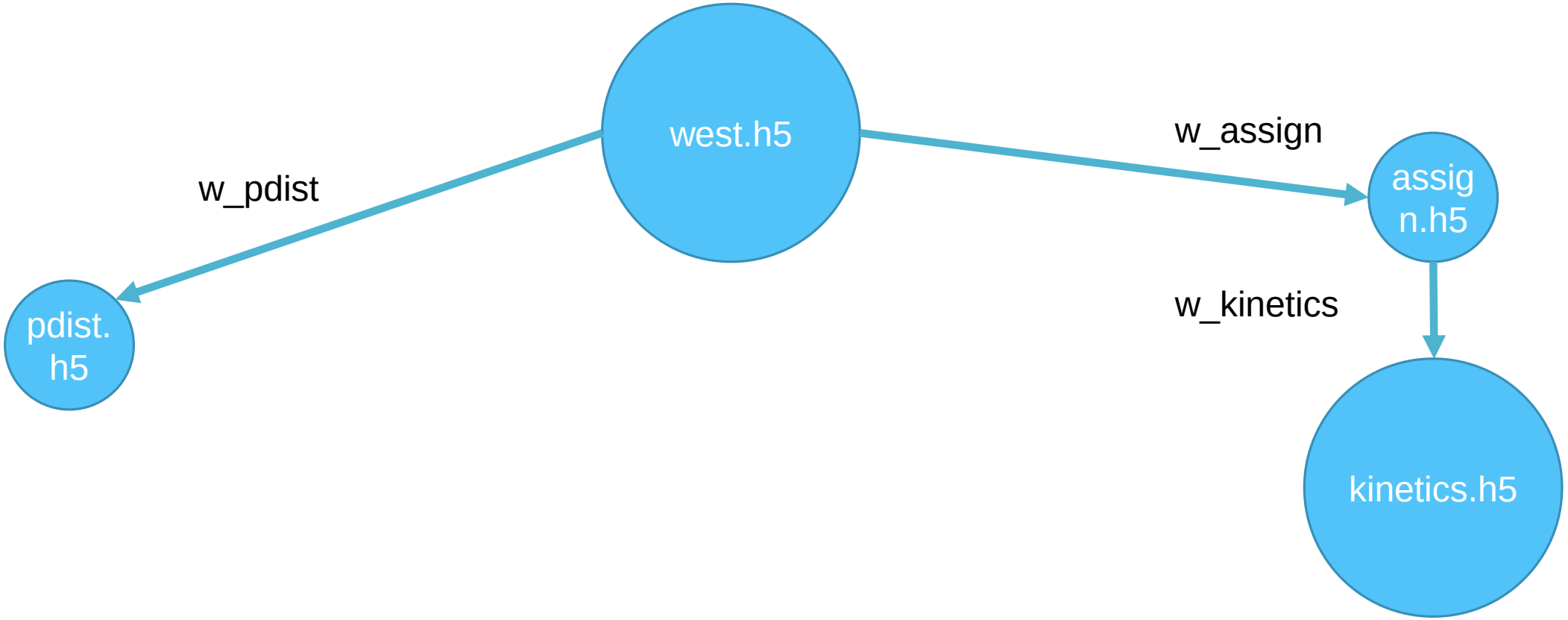
Basic WESTPA tools



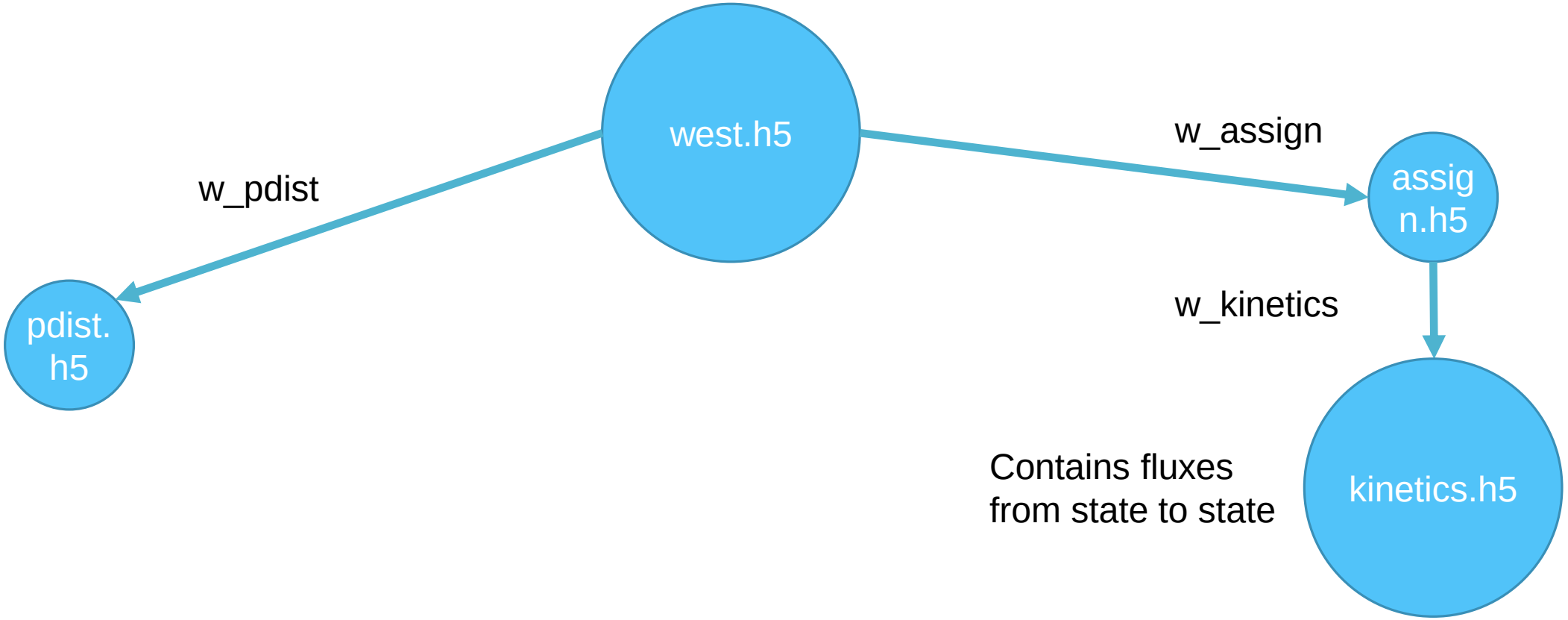
Basic WESTPA tools



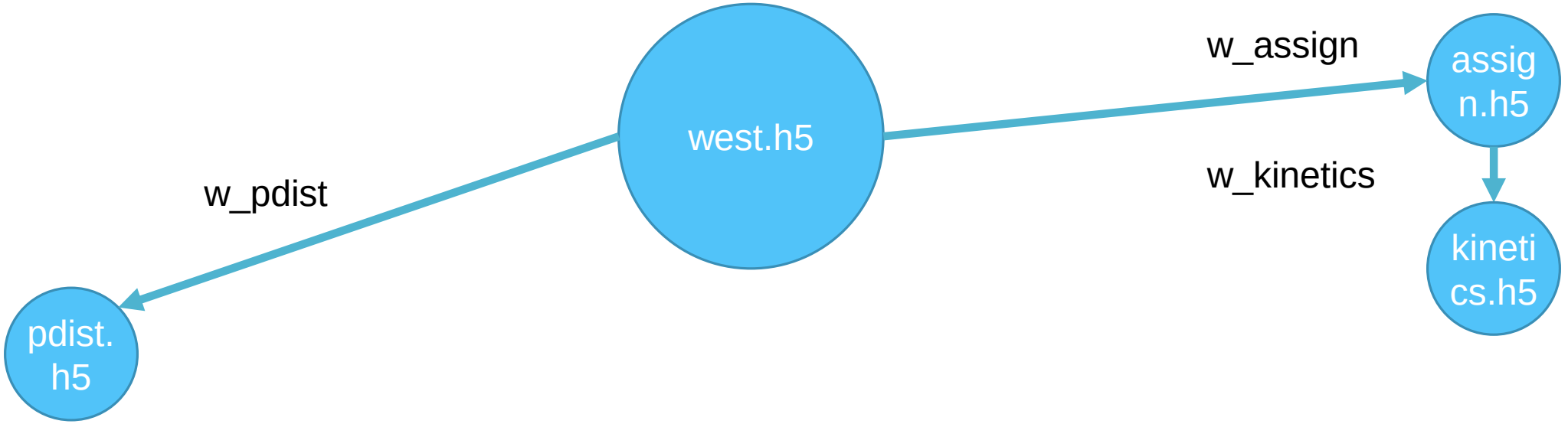
Basic WESTPA tools



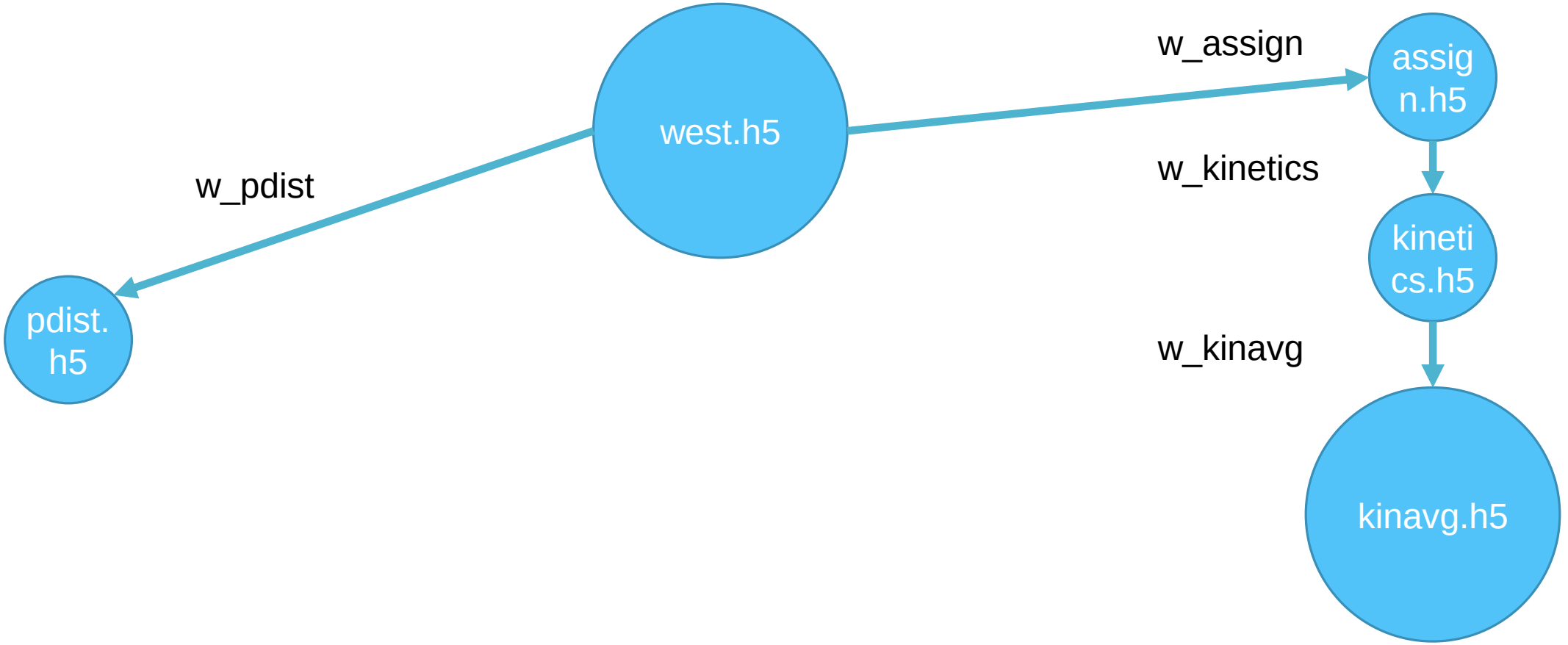
Basic WESTPA tools



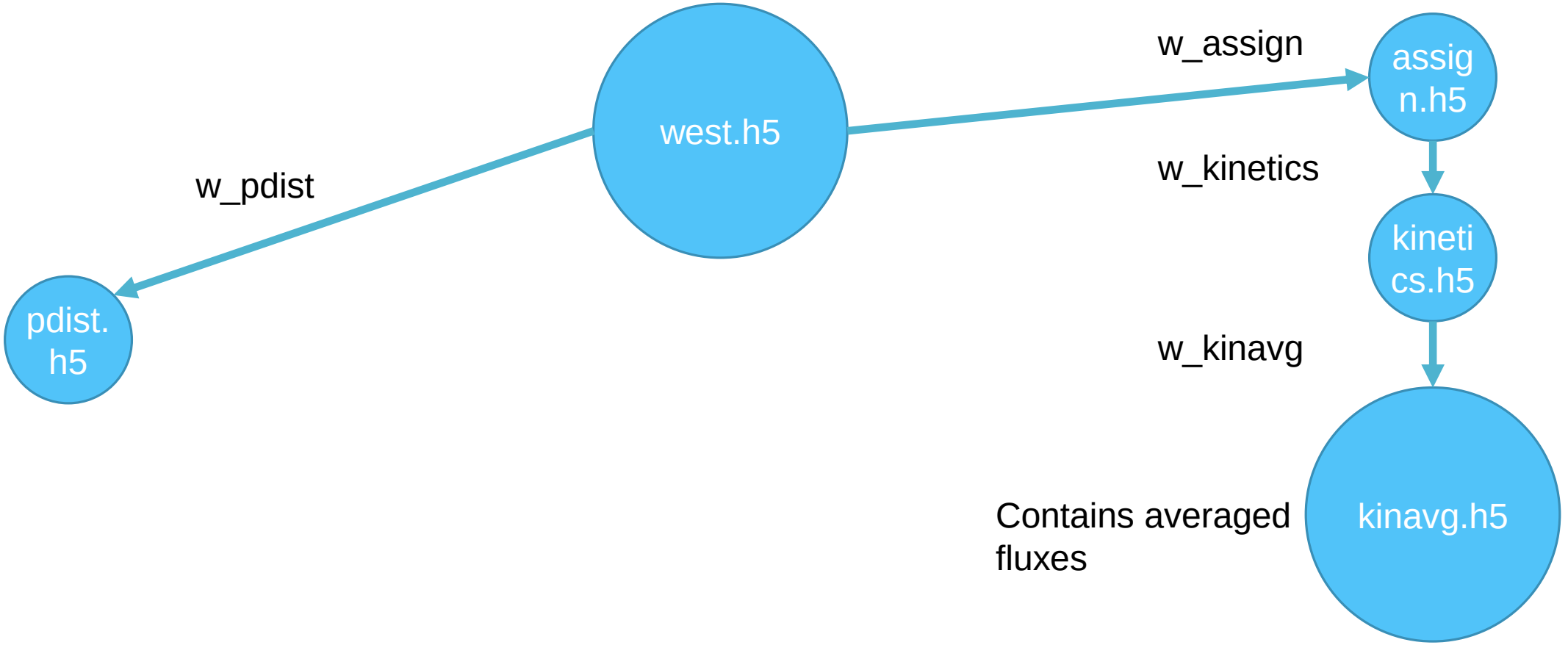
Basic WESTPA tools



Basic WESTPA tools



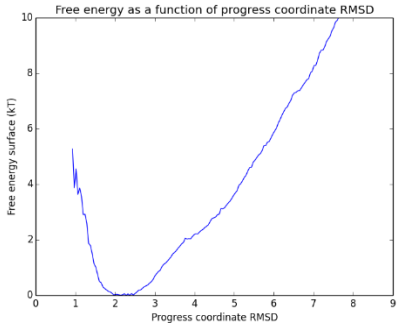
Basic WESTPA tools

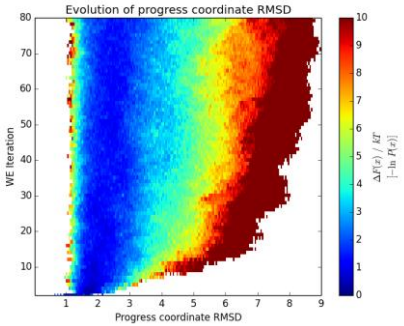


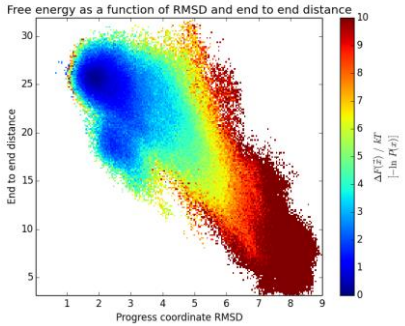
Overview

Part 1. WESTPA file format – HDF5

Part 2. Calculating free energy profiles







Part 3. Calculating rate constants

Part 4. Visualizing continuous trajectories

Generating histograms

`w_pdist`

- -W WEST_H5FILE
- --first-iter, --last-iter
- -b [[dimension 1 bins], [dimension 2 bins],…]
- -o OUTPUT
- --construct-dataset CONSTRUCT_DATASET

- For a full list and documentation: w_pdist –h
- Useful for:
  - Further analysis of probabilities using the optional h5 output
  - Plotting free energy profiles with either plothist or your own tool

```
w_pdist -W west.h5 --construct-dataset assignment.pull_data
                -o pdist.h5 -b 200
```

Pulling arbitrary data sets

```
def pull_data(n_iter, iter_group):

    auxdata = iter_group['auxdata']['end_to_end_dist'][...]
    pcoord  = iter_group['pcoord'][:,:,0]

    data = numpy.dstack( (pcoord, auxdata) )

    return data
```

- The function determines what is returned to be histogrammed for each iteration
- The shape should be (walker index, time index, data index)
- If the data is in another file the file has to be read (e.g a hdf5 file), if it's saved under the iteration group then you can access the dataset from the iter_group object
- The data has to be stacked so that each point have the dimensionality you want e.g. if you want a 3D histogram each point has to be 3 dimensional

Plotting 1D free energy profiles

```
plothist [average|evolution|instantaneous]
```

- --range
- --hdf5-output
- --first-iter, --last-iter
- -o PLOT_OUTPUT
- --post-process-function

- Useful for:
  - Following the extent of sampling during a simulation
  - Qualitatively helps refining binning schemes
- Matplotlib hook allows for a lot of customization for quick plots, reports etc.
- Matplotlib also allows for outputting into vector formats which can then be edited with a vector image editing software, suitable for publication quality plots

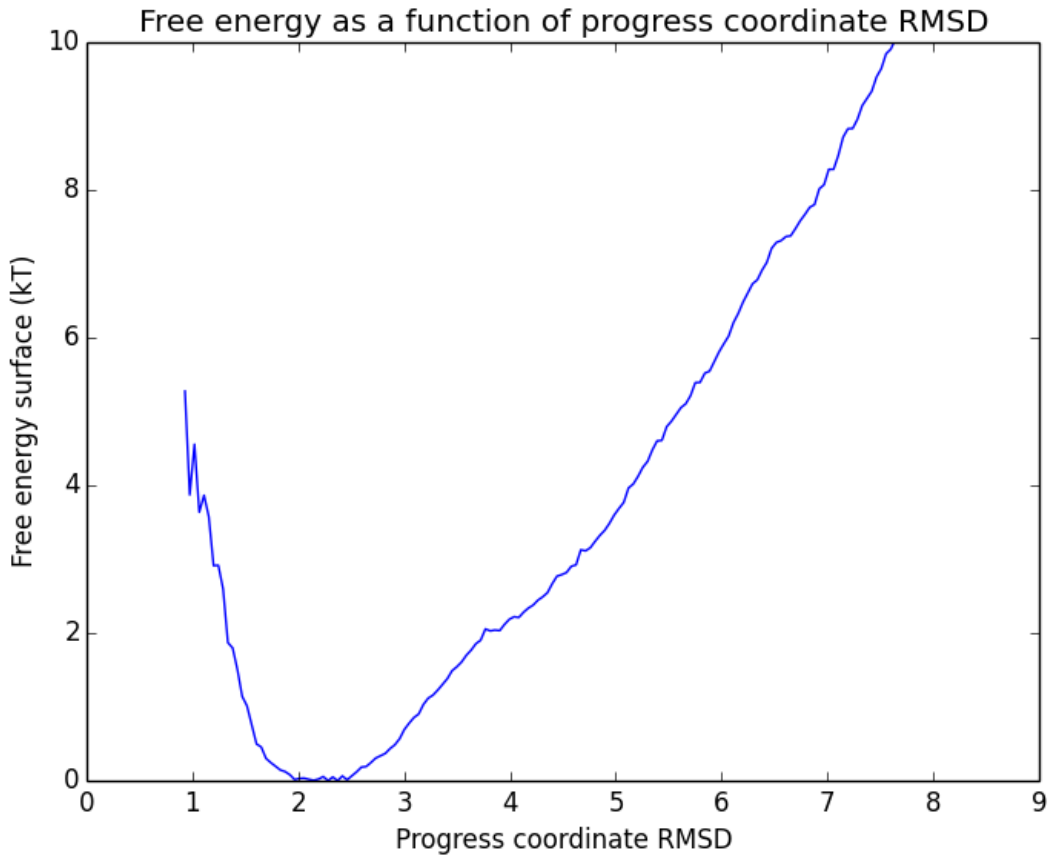
```
plothist average -o 1d_average.pdf --range 0,10
                 --postprocess-function plotting.avg_1d pdist.h5 0
```

Changing plotting options

```
def avg_1d(hist, midpoints, binbounds):
  import matplotlib.pyplot as plt
  plt.title(‘Free energy as a function of RMSD’)
  plt.xlabel(‘Progress coordinate RMSD’)
  plt.ylabel('Free energy surface (kT)')
```

- The function is called right after the plotting is completed by plothist. The pyplot interface can then be used to further modify the plotted figure
- Note that the histograms, midpoints of the bins and the bin boundaries are passed to the function as arguments which can be used in the function e.g. plotting bin boundaries

Plotting free energy profiles – 1D

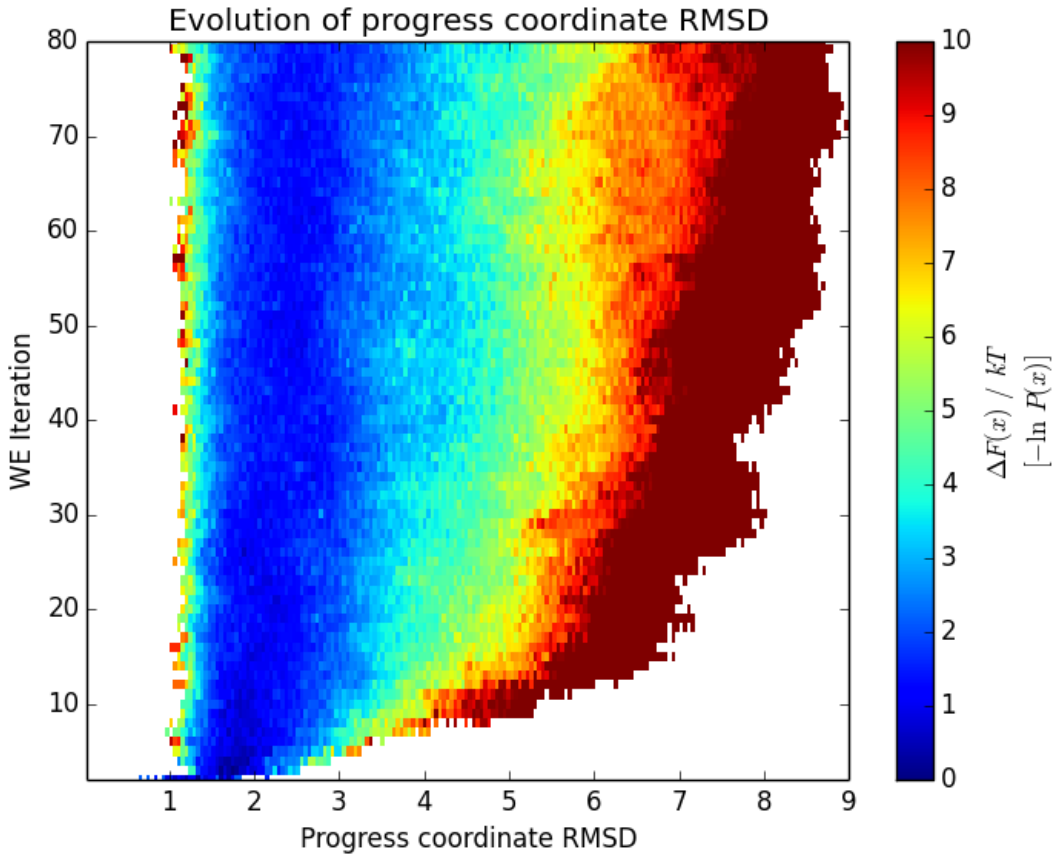


Free energy profile over time – 1D

```
plothist evolution -o 1d_evolution.pdf --range 0,10
      --postprocess-function plotting.evo_1d pdist.h5 0
```

- Useful for:
  - Following how the sampling progresses over time, giving an idea as to how good the WE parameters are
  - Gives a qualitative idea if the probability distribution is converged for a dimension

Free energy profile over time – 1D

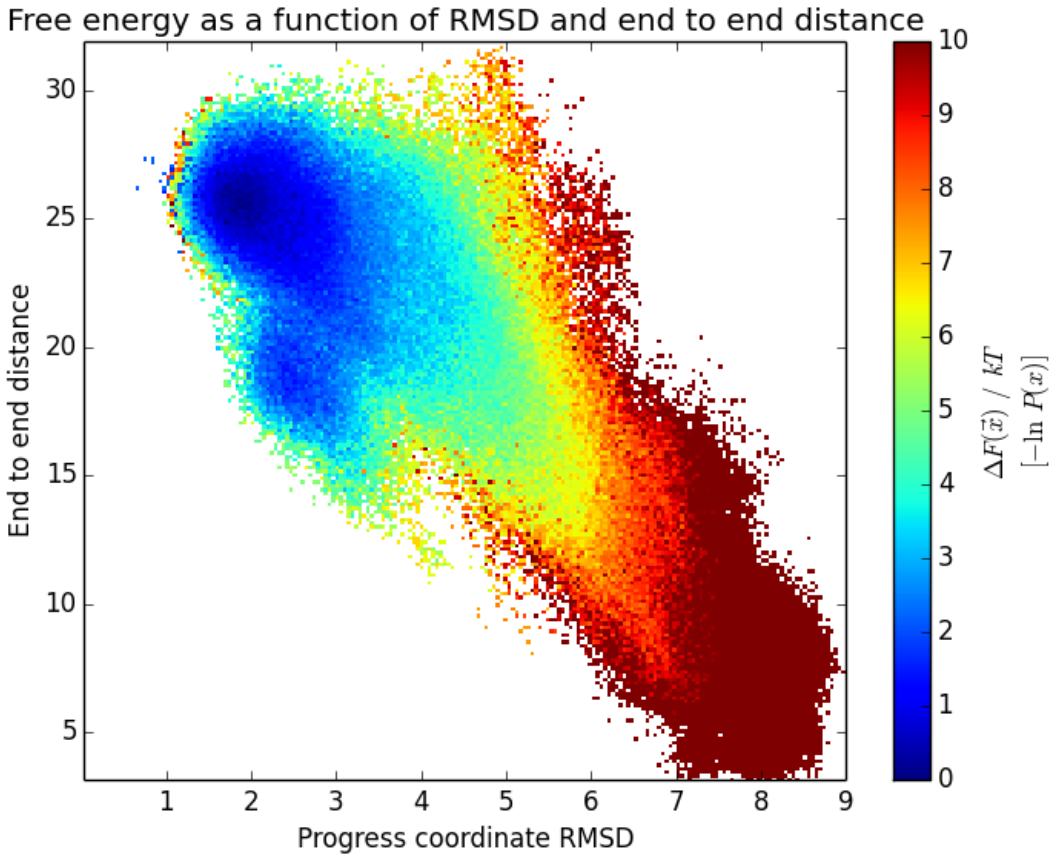


Plotting 2D free energy profiles

```
plothist average -o 2d_average.pdf --range 0,10
      --postprocess-function plotting.avg_2d pdist.h5 0 1
```

- Useful for:
  - Qualitatively looking at the correlation of data sets which in turn allows you to refine binning to include possibly orthogonal coordinates
  - Further refining state definitions using arbitrary data sets

Plotting 2D free energy profiles

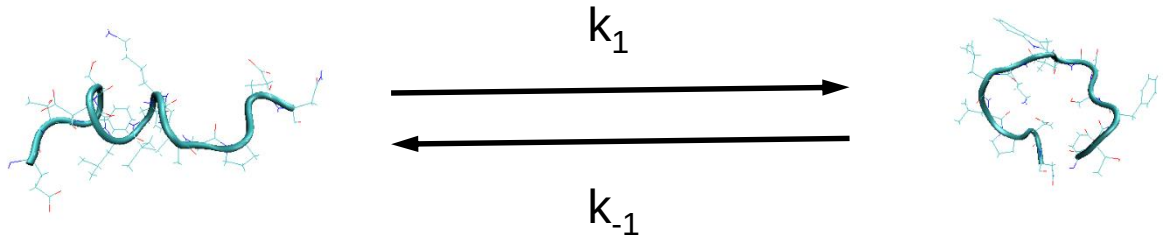


Overview

Part 1. WESTPA file format – HDF5

Part 2. Calculating free energy surfaces

Part 3. Calculating rate constants



Part 4. Extracting continuous trajectories

Rate constant calculations

- Calculation done in three steps
  - Assignment of every data point into a bin for a given set of bins and states (`w_assign`)
  - Calculating probability fluxes from states (`w_kinetics`)
  - Averaging the fluxes and calculating per tau rate constants (`w_kinavg`)
- The datasets, states and the binning determined at the assignment stage
- Bins and states are constructed in YAML file format and passed to `w_assign`
- Useful for measuring the kinetics as well as qualitatively looking at convergence

Assignment of simulation data into bins

`w_assign`

- `-W` WEST_H5FILE
- `--bins-from-file`, `--bins-from-system`, `--bins-from-expr`
- `--states-from-file`, `--states-from-function`, `--states`
- `-o` OUTPUT
- `--construct-dataset` CONSTRUCT_DATASET

- Useful for:
  - Implementing your own analyses that require assignment of simulation data into states

```
w_assign -W west.h5 --bins-from-file BINS --states-from-file STATES
              -o assign.h5 --construct-dataset assignment.pull_data
```

Bin boundaries

```
bins:
    type: RectilinearBinMapper
    boundaries: [[0.0, 2.0, 15.0, inf]]
```

- RectilinearBinMapper is the most common mapper but there are other ways to bin the data sets (see documentation for more information)
  - This mapper basically forms a grid in both dimensions (or in other words a Cartesian product of the two sets of bins) e.g. [[0.0, 1.0, inf], [0.0, 1.0, inf]] has 2 bins in each axis (0 and 1 let's say) and 4 bins total ((0,0), (0,1), (1,0), (1,1))
- Note that the boundaries has to be list of lists and this particular binning is just a 1D binning with three bins

State definitions

```
states:
  - label: bound
    coords:
      - [1.0]

  - label: unbound
    coords:
      - [16.0]
```

- States are defined in a similar manner but they also require a label
- State boundaries are actually defined by bin boundaries and the coordinate for the state boundary has to be a value that falls into the bin that is going to be defined as a “state”
- This particular state definition says the bin (0.0, 1.0) is the bound state and the bin (15.0, inf) is the unbound state

States defined over arbitrary data sets

```
bins:
    type: RectilinearBinMapper
    boundaries: [[0.0, 0.2, 15.0, inf],[0.0, inf]]

states:
  - label: bound_aux
    coords:
      - [1.0, 10.0]

  - label: unbound_aux
    coords:
      - [16.0, 15.0]
```

Calculating rate constants

```
w_kinetics trace
  • -W west.h5
  • -a assign_aux.h5
  • -o kinetics_aux.h5
```

- Calculates probability fluxes from bin to bin as well as conditional fluxes from state to state

```
w_kinavg trace
  • -W west.h5
  • –a assign_aux.h5
  • -k kinetics_aux.h5
  • -o kinavg_aux.h5
```

- From the result of w_kinetics this tool calculates average fluxes, overall rate constants and conditional rate constants from state to state

Calculating rate constants

```
w_kinetics trace -W west.h5 -a assign_aux.h5 -o kinetics_aux.h5

w_kinavg trace -W west.h5 –a assign_aux.h5
                -k kinetics_aux.h5 -o kinavg_aux.h5
```

Sample output:

fluxes into macrostates:
bound : mean=0.000e+00 CI=(0.000e+00, 0.000e+00) * tau^-1
unbound: mean=1.568e-04 CI=(8.290e-05, 2.570e-04) * tau^-1
fluxes from state to state:
bound   -> unbound: mean=1.568e-04 CI=(8.720e-05, 2.482e-04) * tau^-1
unbound -> bound : mean=0.000e+00 CI=(0.000e+00, 0.000e+00) * tau^-1
rates from state to state:
bound   -> unbound: mean=1.576e-04 CI=(8.164e-05, 2.563e-04) * tau^-1
unbound -> bound : mean=0.000e+00 CI=(0.000e+00, 0.000e+00) * tau^-1

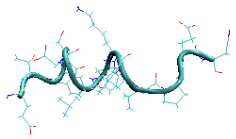
Overview

Part 1. WESTPA file format – HDF5

Part 2. Calculating free energy surfaces

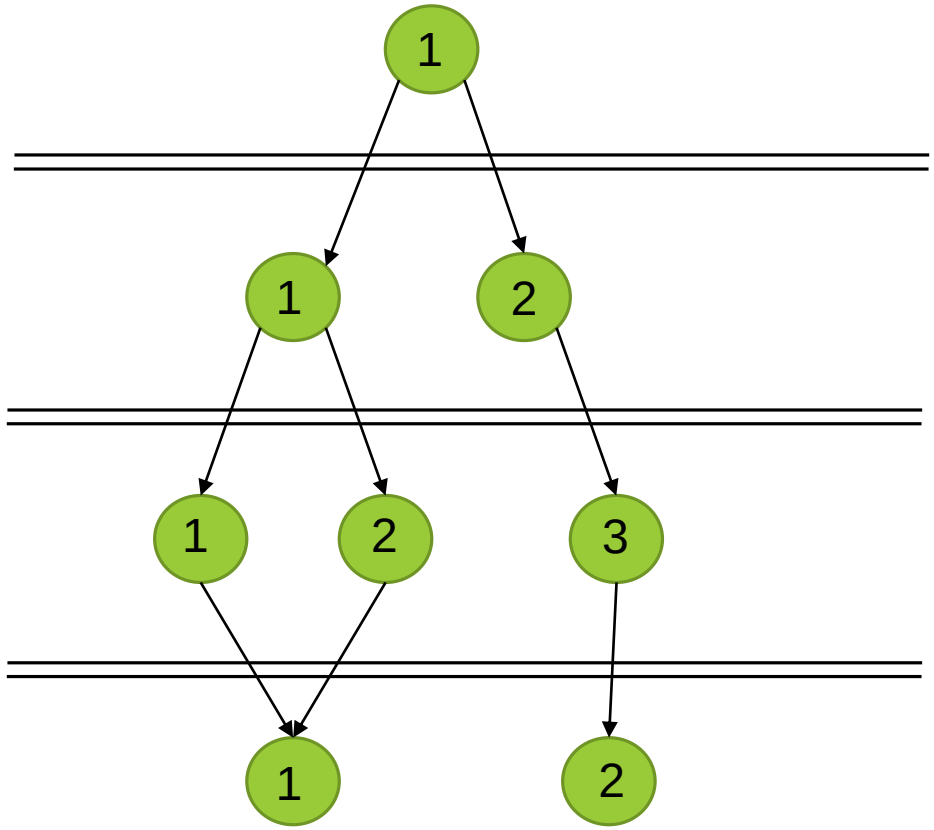
Part 3. Calculating rate constants

Part 4. Visualizing continuous trajectories



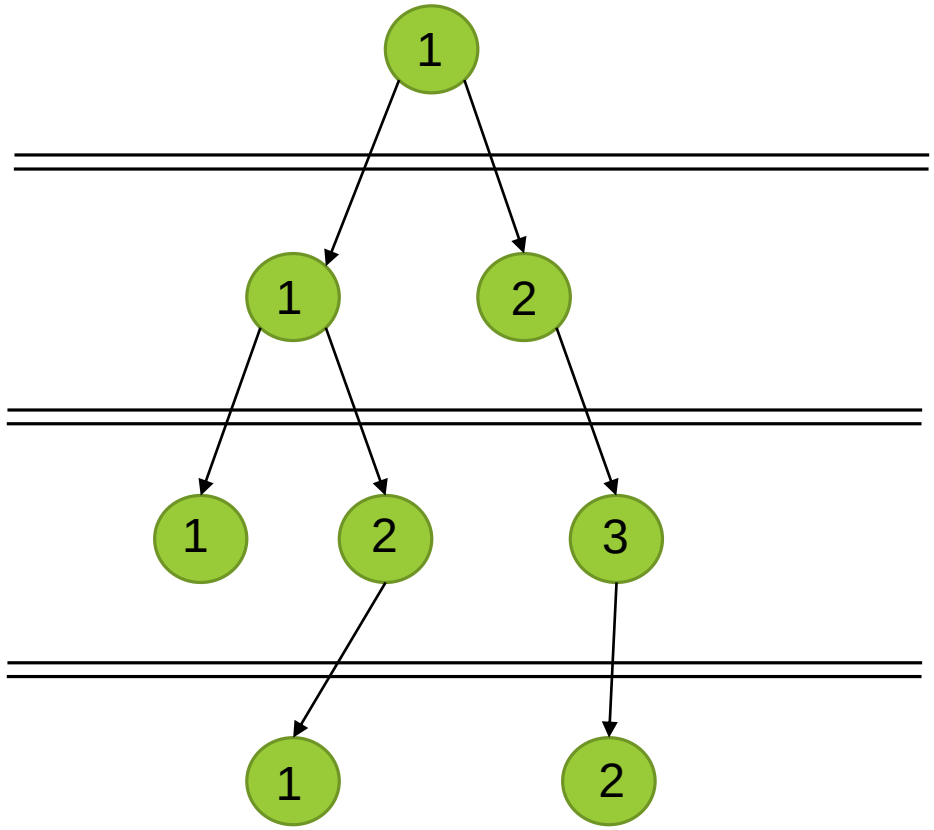
Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



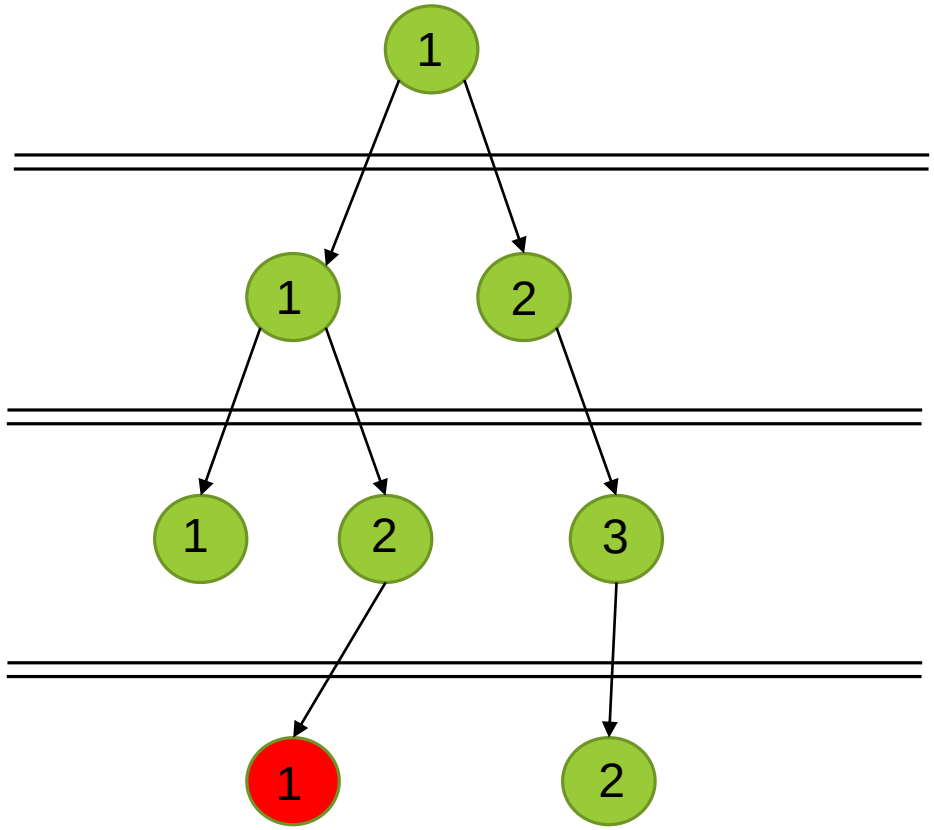
Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



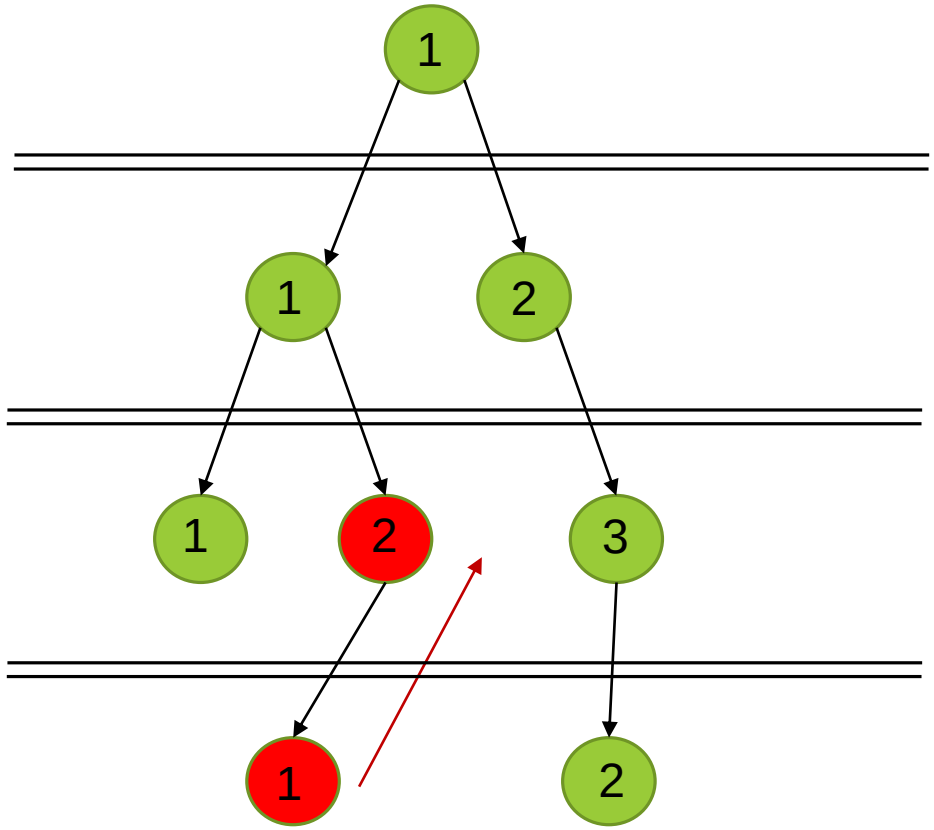
Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



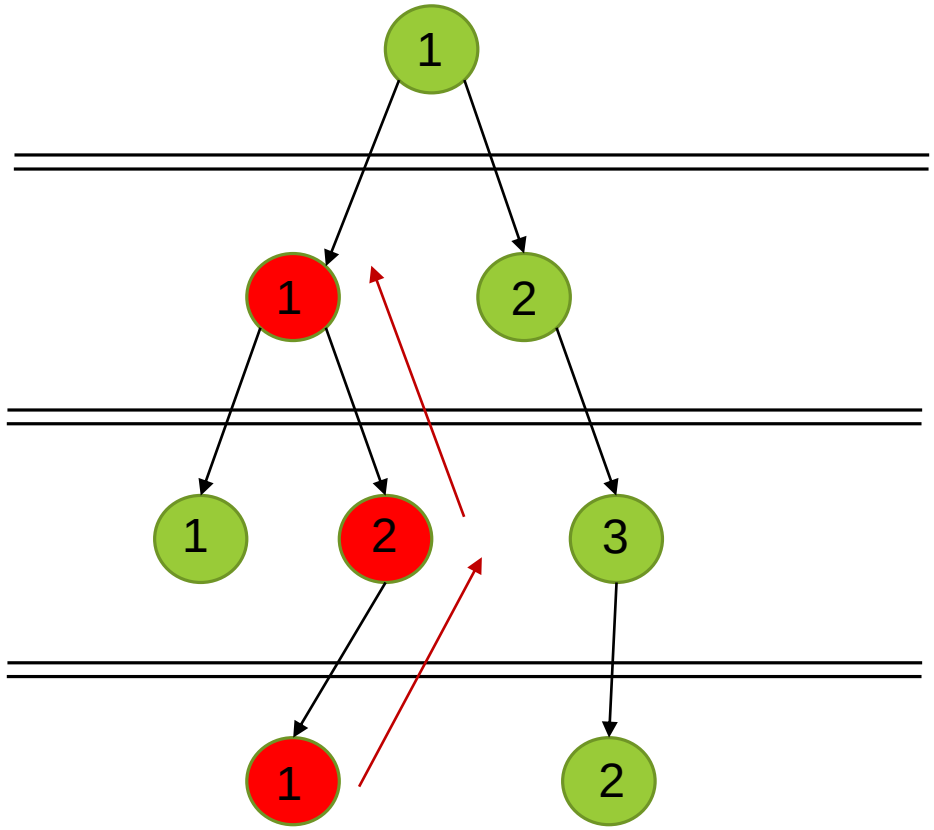
Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



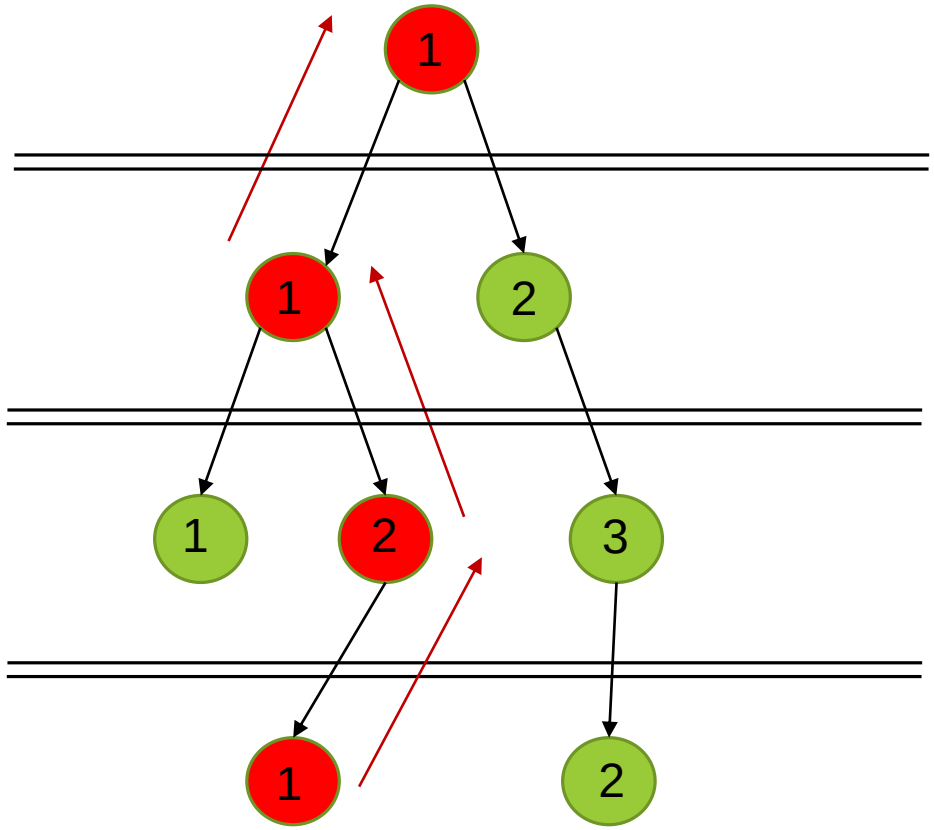
Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



Extracting trajectory history

1. Iteration
2. Iteration
3. Iteration
4. Iteration



Extracting trajectory history

Tracing tool: w_trace

```
usage: w_trace [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
               [-W WEST_H5FILE] [-d DSNAME] [--output-pattern OUTPUT_PATTERN]
               [-o OUTPUT]
               N_ITER:SEG_ID [N_ITER:SEG_ID ...]
```

```
w_trace -W west.h5 120:10
```

Extracting trajectory history

```
# column  0: iteration (0 => initial state)
# column  1: seg_id (or initial state ID)
# column  2: weight
# column  3: wallclock time (s)
# column  4: CPU time (s)
# columns 5 -- 6: final progress coordinate value
    0       0    0.0000000000000000e+00          0          0          1e-06          0
    1       2    2.5000000000000000e-01    311.479    309.314       0.361733          0
    2       0    2.5000000000000000e-01    308.769    307.023       0.442503          0
    3       4    2.5000000000000000e-01     333.15    331.626       0.558815          0
    4       5    2.5000000000000000e-01    380.693    379.153       0.556424          0
```

Extracting trajectory files according to the history

- A sample bash script to pull files from the output of `w_trace`

```
./trj_trace.sh traj_120_10_trace.txt
```

- Note that this script also copies over the initial state coordinate file as well

Extracting trajectory files according to the history

```
for i in `tail -n +9 $1|awk '{print $1"-"$2}'`;do
  # We need the iteration/walker indices as strings
  # so we can name the file properly
  iter=`echo $i|sed 's/-[0-9]*//'`
  p_iter=`printf "%06d" $iter`
  walk=`echo $i|sed 's/^[0-9]*//'|sed 's/-//'`
  p_walk=`printf "%06d" $walk`

  # Make the full path to the seg.xtc
  TRJ_PATH=`printf "${TRAJ_SEGS}/%06d/%06d" $iter $walk`

  # Copy into the folder!
  cp $TRJ_PATH/seg.xtc ${FULL_TRJ}/${p_iter}_${p_walk}.xtc
done
```

Creating a movie the trajectory

- An example of stitching together .xtc files using the GROMACS `trjcat` tool into a continuous trajectory:

```
trjcat -f *xtc -o full_traj.xtc –cat
```

- Note that the .xtc files will be concatenated in order given to `trjcat` so they have to be named in order

The movie!