ProDy

Protein Dynamics & Sequence Analysis

Interactions Analysis

Release

Mikulska-Ruminska, Karolina

May 17, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

This tutorial shows how to detect water molecules that might form hydrogen bonds with protein structures (called water bridges). The prediction method introduced here helps evaluate the significance of water molecules on the stability and dynamics of protein structure.

1.1 Required Programs

The latest version of **ProDy_** is required.

1.2 Recommended Programs

Besides **ProDy_**, the **Matplotlib_** library and **VMD_** program are required for some steps in the tutorial. **IPython_** is highly recommended for interactive usage.

Moreover, in the case of the lack of hydrogen atoms in protein structures, additional packages such as Openbabel[1] or PDBfixer[2] are required to predict hydrogen bonds.

They can be installed using a conda or pip.

1.3 Getting Started

To follow this tutorial, you will need the following files:

```
1.5M Feb 29 20:20 5kqm_all_sci.pdb
446K Feb 29 20:20 addH_5kqm.pdb
3.8M Feb 29 20:20 NAMD_D2_sample.dcd
 78M Feb 29 20:20 pebp1_50frames.pdb
```

We recommend that you will follow this tutorial by typing commands in an IPython session, e.g.:

```
$ ipython
```

or with pylab environment:

```
$ ipython --pylab
```

First, we will make necessary imports from ProDy and Matplotlib packages.

```
In [1]: from prody import *

In [2]: from pylab import *
```

[1] https://github.com/openbabel
[2] https://github.com/openmm/pdbfixer

```
In [3]: import matplotlib
```

We have included these imports in every part of the tutorial so that the code copied from the online pages is complete. You do not need to repeat imports in the same Python session.

CHAPTER

TWO

PROTEIN PREPARATION

Since PDB structures often lack hydrogen atoms in water molecules, we need to add them. We can use the `addMissingAtoms()` function. This function utilizes either Openbabel[3] or PDBFixer[4]. Those are external packages; therefore, they should be installed independently (see Recommended Programs).

If we prefer not to install additional tools, we can alternatively provide a PDB structure with hydrogens already added by other software.

Here, we will fetch a structure of LMW-PTP (PDB: **5KQM**) from Protein Data Bank (PDB) in uncompressed form using `compressed=False` and add missing atoms using `addMissingAtoms()`:

```
In [1]: pdb = '5kqm'

In [2]: filename = fetchPDB(pdb, compressed=False)
```

```
@> Connecting wwPDB FTP server RCSB PDB (USA).
@> 5kqm downloaded (5kqm.pdb)
@> PDB download via FTP completed (1 downloaded, 0 failed).
```

```
In [3]: filename2 = addMissingAtoms(filename)
```

```
@> Hydrogens were added to the structure.
Structure addH_5kqm.pdb is saved in the local directry.
```

A new file containing hydrogens is now generated, prefixed with `'addH_'`. For protein structures lacking hydrogens, results will also be computed without applying angle criteria.

[3] https://github.com/openbabel
[4] https://github.com/openmm/pdbfixer

CHAPTER

THREE

WATER BRIDGES DETECTION IN A SINGLE PDB STRUCTURE

3.1 Water bridges prediction

To analyze the structure we need to parse a structure `addH_5kqm.pdb` using `parsePDB()`:

```
In [1]: atoms = parsePDB(filename2)
```

```
@> 2815 atoms and 1 coordinate set(s) were parsed in 0.03s.
```

Before analysis, we can verify the number of water molecules present in our PDB structure and later compare how many of them contributed meaningfully to protein stability.

```
In [2]: water_molecules = atoms.select('water')

In [3]: len(water_molecules)
```

```
363
```

Subsequently, we can utilize `calcWaterBridges()` along with one of two methods for detecting water bridges: `'chain'` or `'cluster'`.

1. `Method 'chain' (default)` which will detect water molecules between pairs of hydrophilic residues:

```
In [4]: waterBridges_chain = calcWaterBridges(atoms)
```

```
@> 45 water bridges detected.
@> SER7 OG_21 A ARG40 NH1_285 A 4.900955519079926 1 ['A_1289']
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> GLY14 O_72 A GLU50 N_347 A 4.544135231262378 1 ['A_1274']
@> GLY14 O_72 A SER47 O_328 A 5.288116583434976 1 ['A_1274']
@> ARG18 NH2_105 A ASN95 ND2_714 A 4.570361692470302 1 ['A_1261']
@> ARG18 NH2_105 A ASP92 OD1_690 A 5.373355841557489 1 ['A_1261']
@> PRO20 O_115 A GLU23 OE1_139 A 4.571172934816621 1 ['A_1292']
@> GLU23 OE1_139 A GLU23 OE2_140 A 2.206470711339718 1 ['A_1244']
@> GLU23 OE1_139 A SER71 O_514 A 5.272447154784958 1 ['A_1244']
@> GLU23 OE1_139 A HIS72 ND1_523 A 3.2114691342125647 1 ['A_1244']
@> GLU23 OE2_140 A SER71 O_514 A 4.934310286149422 1 ['A_1244']
@> GLU23 OE2_140 A HIS72 ND1_523 A 4.127239392136104 1 ['A_1244']
@> ARG27 NE_171 A ARG27 NH2_174 A 2.298703982682415 1 ['A_1339']
@> ARG27 NE_171 A VAL41 N_287 A 5.672199573357763 1 ['A_1339']
@> ARG27 NH1_173 A SER71 N_511 A 6.128045528551498 1 ['A_1319']
@> ARG27 NH2_174 A VAL41 N_287 A 4.642634596864156 1 ['A_1339']
@> SER36 N_239 A SER36 OG_244 A 2.9687596736684503 1 ['A_1318']
@> SER36 N_239 A GLU37 N_245 A 2.743692767056837 1 ['A_1318']
@> SER36 OG_244 A GLU37 N_245 A 3.0545251676815504 1 ['A_1318']
@> ARG40 NH2_286 A ASP42 OD2_301 A 5.163938516287738 1 ['A_1246']
```

```
@> ARG40 NH2_286 A THR84 OG1_621 A 3.9229717052255175 1 ['A_1262']
@> ARG40 NH2_286 A ASP81 OD1_598 A 4.365525627000715 1 ['A_1262']
@> THR46 OG1_323 A ASP98 OD2_741 A 3.80450101853055 1 ['A_1313']
@> SER47 O_328 A GLU50 N_347 A 5.10489901956934 1 ['A_1274']
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 1 ['A_1267']
@> SER71 O_514 A HIS72 ND1_523 A 4.463734423103597 1 ['A_1244']
@> ARG75 NH1_548 A GLN76 O_553 A 3.303749082481901 1 ['A_1256']
@> LYS79 NZ_582 A GLN105 NE2_800 A 3.8752885053889865 1 ['A_1249']
@> LYS79 NZ_582 A LYS102 O_772 A 4.929221236666094 1 ['A_1249']
@> ASP81 OD1_598 A ASP81 OD2_599 A 2.192005930648912 1 ['A_1239']
@> ASP81 OD1_598 A THR84 OG1_621 A 4.415462942886057 1 ['A_1262']
@> ASP92 OD1_690 A ASN95 ND2_714 A 3.3343465626716116 1 ['A_1261']
@> GLU93 N_692 A SER94 N_701 A 2.720476612654481 1 ['A_1338']
@> GLU93 O_695 A GLU93 OE2_700 A 4.362719106245552 1 ['A_1251']
@> LYS102 O_772 A GLN105 NE2_800 A 4.2363221076778395 1 ['A_1249']
@> GLY117 N_886 A LEU125 O_953 A 3.8595291163560352 1 ['A_1264']
@> LEU125 N_950 A ILE126 N_958 A 2.9289776373335465 1 ['A_1325']
@> LEU125 N_950 A ILE126 O_961 A 4.854966117286505 2 ['A_1325', 'A_1275']
@> ILE126 N_958 A ILE126 O_961 A 2.8761442940158615 2 ['A_1325', 'A_1275']
@> TYR131 N_998 A TYR132 N_1010 A 2.849420467393326 1 ['A_1298']
@> ASP137 OD1_1054 A THR140 OG1_1081 A 5.251346017927213 1 ['A_1308']
@> GLN144 NE2_1119 A CYS148 SG_1149 A 6.149862843999044 1 ['A_1278']
@> ARG147 NE_1140 A ARG147 NH2_1143 A 2.278232867816633 1 ['A_1304']
@> ARG150 NH1_1165 A ARG150 NH2_1166 A 2.3112059622629917 1 ['A_1328']
```

These results may vary slightly depending on the position of added hydrogen atoms.

2. `Method 'cluster'` which will detect water molecules between multiple hydrophilic residues:

```
In [5]: waterBridges_cluster = calcWaterBridges(atoms, method='cluster')
```

```
@> 45 water bridges detected.
@> SER7 OG_21 A ARG40 NH1_285 A 4.900955519079926 1 ['A_1289']
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> GLY14 O_72 A SER47 O_328 A GLU50 N_347 A 5.288116583434976 4.544135231262378 5.10489901956934 1 [
@> ARG18 NH2_105 A ASN95 ND2_714 A ASP92 OD1_690 A 4.570361692470302 5.373355841557489 3.33434656267
@> PRO20 O_115 A GLU23 OE1_139 A 4.571172934816621 1 ['A_1292']
@> SER71 O_514 A HIS72 ND1_523 A GLU23 OE2_140 A GLU23 OE1_139 A 4.463734423103597 4.934310286149422
@> SER71 O_514 A HIS72 ND1_523 A GLU23 OE2_140 A GLU23 OE1_139 A 4.463734423103597 4.934310286149422
@> ARG27 NE_171 A ARG27 NH2_174 A VAL41 N_287 A 2.298703982682415 5.672199573357763 4.64263459686415
@> SER71 N_511 A ARG27 NH1_173 A 6.128045528551498 1 ['A_1319']
@> SER36 N_239 A SER36 OG_244 A GLU37 N_245 A 2.9687596736684503 2.743692767056837 3.054525167681550
@> SER36 N_239 A SER36 OG_244 A GLU37 N_245 A 2.9687596736684503 2.743692767056837 3.054525167681550
@> SER7 OG_21 A ARG40 NH1_285 A 4.900955519079926 1 ['A_1289']
@> ASP42 OD2_301 A ARG40 NH2_286 A 5.163938516287738 1 ['A_1246']
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.922971705225
@> ARG27 NE_171 A ARG27 NH2_174 A VAL41 N_287 A 2.298703982682415 5.672199573357763 4.64263459686415
@> THR46 OG1_323 A ASP98 OD2_741 A 3.80450101853055 1 ['A_1313']
@> GLY14 O_72 A SER47 O_328 A GLU50 N_347 A 5.288116583434976 4.544135231262378 5.10489901956934 1 [
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 2 ['A_1267', 'A_1252']
@> SER71 N_511 A ARG27 NH1_173 A 6.128045528551498 1 ['A_1319']
@> SER71 O_514 A HIS72 ND1_523 A GLU23 OE2_140 A GLU23 OE1_139 A 4.463734423103597 4.934310286149422
@> GLN76 O_553 A ARG75 NH1_548 A 3.303749082481901 1 ['A_1256']
@> GLN105 NE2_800 A LYS102 O_772 A LYS79 NZ_582 A 4.2363221076778395 3.8752885053889865 4.9292212366
@> ASP81 OD2_599 A ASP81 OD1_598 A 2.192005930648912 1 ['A_1239']
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.922971705225
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.922971705225
```

```
@> ARG18 NH2_105 A ASN95 ND2_714 A ASP92 OD1_690 A 4.570361692470302 5.373355841557489 3.33434656267
@> GLU93 N_692 A SER94 N_701 A 2.720476612654481 1 ['A_1338']
@> GLU93 OE2_700 A GLU93 O_695 A 4.362719106245552 1 ['A_1251']
@> GLU93 N_692 A SER94 N_701 A 2.720476612654481 1 ['A_1338']
@> ARG18 NH2_105 A ASN95 ND2_714 A ASP92 OD1_690 A 4.570361692470302 5.373355841557489 3.33434656267
@> THR46 OG1_323 A ASP98 OD2_741 A 3.80450101853055 1 ['A_1313']
@> GLN105 NE2_800 A LYS102 O_772 A LYS79 NZ_582 A 4.2363221076778395 3.8752885053889865 4.9292212366
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> LEU125 O_953 A GLY117 N_886 A 3.8595291163560352 1 ['A_1264']
@> ILE126 O_961 A LEU125 N_950 A ILE126 N_958 A 4.854966117286505 2.8761442940158615 2.9289776373335
@> ILE126 O_961 A LEU125 N_950 A ILE126 N_958 A 4.854966117286505 2.8761442940158615 2.9289776373335
@> TYR132 N_1010 A TYR131 N_998 A 2.849420467393326 1 ['A_1298']
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 2 ['A_1267', 'A_1252']
@> THR140 OG1_1081 A ASP137 OD1_1054 A 5.251346017927213 1 ['A_1308']
@> CYS148 SG_1149 A GLN144 NE2_1119 A 6.149862843999044 1 ['A_1278']
@> ARG147 NE_1140 A ARG147 NH2_1143 A 2.278232867816633 1 ['A_1304']
@> CYS148 SG_1149 A GLN144 NE2_1119 A 6.149862843999044 1 ['A_1278']
@> ARG150 NH1_1165 A ARG150 NH2_1166 A 2.3112059622629917 1 ['A_1328']
```

The `'chain' method` detected **42** water bridges, and the `'cluster' method` second **49**. The total number of water molecules in the crystal structure is **363**. As we can see, many of them are not significant for protein stability.

3.2 Save results in PDB file

We can use `savePDBWaterBridges()` to save the results in a PDB file. The file will contain water molecules that are forming associations with protein structure. Residues involved in water bridging can be displayed using the occupancy column in any graphical visualization tool.

```
In [6]: savePDBWaterBridges(waterBridges_cluster, atoms, filename2[:-4]+
   ...:                                                    '_wb_cluster.pdb')
   ...:

In [7]: savePDBWaterBridges(waterBridges_chain, atoms, filename2[:-4]+
   ...:                                                    '_wb_chain.pdb')
   ...:
```

The results can be displayed in **VMD_** program. Below we can see a comparison between results obtained by `'chain'` vs. `'cluster'` (additional molecules are shown in green) method.

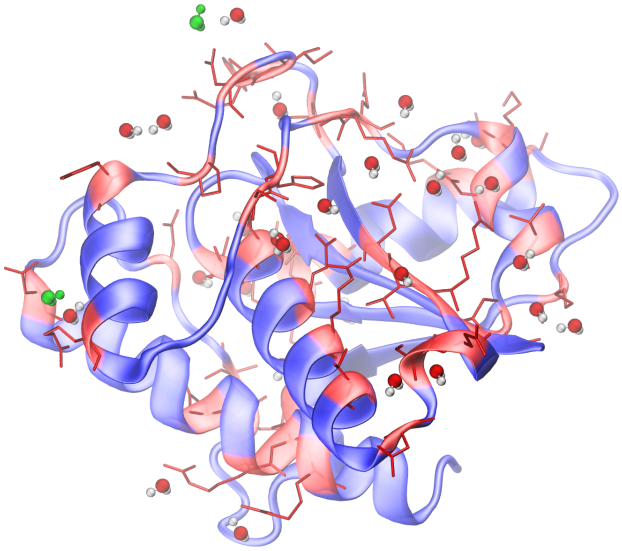
3.3 Access to the raw data

To have access to the raw data, we need to include an additional parameter `ouput='info'` in `calcWaterBridges()`.

The atomic output can also be transformed to this detailed information using `getWaterBridgesInfoOutput()`.

```
In [8]: waterBridges_cluster = calcWaterBridges(atoms, method='cluster', output='info')

In [9]: waterBridges_cluster
```

```
[['SER7',
  'OG_21',
  'A',
  'ARG40',
  'NH1_285',
  'A',
```



```
 4.900955519079926,
 1,
 ['A_1289']],
['SER7',
 'OG_21',
 'A',
 'LYS110',
 'NZ_838',
 'A',
 4.70722699686344,
 1,
 ['A_1316']],
['GLY14',
 'O_72',
 'A',
 'SER47',
 'O_328',
 'A',
 'GLU50',
 'N_347',
 'A',
 5.288116583434976,
 4.544135231262378,
 5.10489901956934,
 1,
 ['A_1274']],
['ARG18',
 'NH2_105',
 'A',
 'ASN95',
 'ND2_714',
 'A',
 'ASP92',
 'OD1_690',
 'A',
 4.570361692470302,
 5.373355841557489,
```

```
  3.3343465626716116,
  2,
  ['A_1261', 'A_1300']],
 ['PRO20',
  'O_115',
  'A',
  'GLU23',
  'OE1_139',
  'A',
  4.571172934816621,
  1,
  ['A_1292']],
 ['SER71',
  'O_514',
  'A',
  'HIS72',
  'ND1_523',
  'A',
  'GLU23',
  'OE2_140',
  'A',
  'GLU23',
  'OE1_139',
  'A',
  4.463734423103597,
  4.934310286149422,
  5.272447154784958,
  4.127239392136104,
  3.2114691342125647,
  2.206470711339718,
  1,
  ['A_1244']],
  ..
  ..
```

The distances are between combinations of protein atoms. 2 atoms gives 1 distance, 3 atoms gives 3 distances, 4 atoms gives 6 distances, etc.

```
In [10]: waterBridges_chain = calcWaterBridges(atoms, output='info')
```

```
@> 45 water bridges detected.
@> SER7 OG_21 A ARG40 NH1_285 A 4.900955519079926 1 ['A_1289']
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> GLY14 O_72 A GLU50 N_347 A 4.544135231262378 1 ['A_1274']
@> GLY14 O_72 A SER47 O_328 A 5.288116583434976 1 ['A_1274']
@> ARG18 NH2_105 A ASN95 ND2_714 A 4.570361692470302 1 ['A_1261']
@> ARG18 NH2_105 A ASP92 OD1_690 A 5.373355841557489 1 ['A_1261']
@> PRO20 O_115 A GLU23 OE1_139 A 4.571172934816621 1 ['A_1292']
@> GLU23 OE1_139 A GLU23 OE2_140 A 2.206470711339718 1 ['A_1244']
@> GLU23 OE1_139 A SER71 O_514 A 5.272447154784958 1 ['A_1244']
@> GLU23 OE1_139 A HIS72 ND1_523 A 3.2114691342125647 1 ['A_1244']
@> GLU23 OE2_140 A SER71 O_514 A 4.934310286149422 1 ['A_1244']
@> GLU23 OE2_140 A HIS72 ND1_523 A 4.127239392136104 1 ['A_1244']
@> ARG27 NE_171 A ARG27 NH2_174 A 2.298703982682415 1 ['A_1339']
@> ARG27 NE_171 A VAL41 N_287 A 5.672199573357763 1 ['A_1339']
@> ARG27 NH1_173 A SER71 N_511 A 6.128045528551498 1 ['A_1319']
@> ARG27 NH2_174 A VAL41 N_287 A 4.642634596864156 1 ['A_1339']
@> SER36 N_239 A SER36 OG_244 A 2.9687596736684503 1 ['A_1318']
```

```
@> SER36 N_239 A GLU37 N_245 A 2.743692767056837 1 ['A_1318']
@> SER36 OG_244 A GLU37 N_245 A 3.0545251676815504 1 ['A_1318']
@> ARG40 NH2_286 A ASP42 OD2_301 A 5.163938516287738 1 ['A_1246']
@> ARG40 NH2_286 A THR84 OG1_621 A 3.9229717052255175 1 ['A_1262']
@> ARG40 NH2_286 A ASP81 OD1_598 A 4.365525627000715 1 ['A_1262']
@> THR46 OG1_323 A ASP98 OD2_741 A 3.80450101853055 1 ['A_1313']
@> SER47 O_328 A GLU50 N_347 A 5.10489901956934 1 ['A_1274']
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 1 ['A_1267']
@> SER71 O_514 A HIS72 ND1_523 A 4.463734423103597 1 ['A_1244']
@> ARG75 NH1_548 A GLN76 O_553 A 3.303749082481901 1 ['A_1256']
@> LYS79 NZ_582 A GLN105 NE2_800 A 3.8752885053889865 1 ['A_1249']
@> LYS79 NZ_582 A LYS102 O_772 A 4.929221236666094 1 ['A_1249']
@> ASP81 OD1_598 A ASP81 OD2_599 A 2.192005930648912 1 ['A_1239']
@> ASP81 OD1_598 A THR84 OG1_621 A 4.415462942886057 1 ['A_1262']
@> ASP92 OD1_690 A ASN95 ND2_714 A 3.3343465626716116 1 ['A_1261']
@> GLU93 N_692 A SER94 N_701 A 2.720476612654481 1 ['A_1338']
@> GLU93 O_695 A GLU93 OE2_700 A 4.362719106245552 1 ['A_1251']
@> LYS102 O_772 A GLN105 NE2_800 A 4.2363221076778395 1 ['A_1249']
@> GLY117 N_886 A LEU125 O_953 A 3.8595291163560352 1 ['A_1264']
@> LEU125 N_950 A ILE126 N_958 A 2.9289776373335465 1 ['A_1325']
@> LEU125 N_950 A ILE126 O_961 A 4.854966117286505 2 ['A_1325', 'A_1275']
@> ILE126 N_958 A ILE126 O_961 A 2.8761442940158615 2 ['A_1325', 'A_1275']
@> TYR131 N_998 A TYR132 N_1010 A 2.849420467393326 1 ['A_1298']
@> ASP137 OD1_1054 A THR140 OG1_1081 A 5.251346017927213 1 ['A_1308']
@> GLN144 NE2_1119 A CYS148 SG_1149 A 6.149862843999044 1 ['A_1278']
@> ARG147 NE_1140 A ARG147 NH2_1143 A 2.278232867816633 1 ['A_1304']
@> ARG150 NH1_1165 A ARG150 NH2_1166 A 2.3112059622629917 1 ['A_1328']
```

We can check which residues are involved in water bridges using the code below. First, we need to extract residue names and display them without repetitions.

```
In [11]: allresidues = []

In [12]: for i in waterBridges_chain:
   ....:     allresidues.append(i[0])
   ....:     allresidues.append(i[3])
   ....:

In [13]: import numpy as np

In [14]: allresidues_once = np.unique(allresidues)

In [15]: allresidues_once
```

```
array(['ARG147', 'ARG150', 'ARG18', 'ARG27', 'ARG40', 'ARG65', 'ARG75',
       'ASN95', 'ASP135', 'ASP137', 'ASP42', 'ASP81', 'ASP92', 'ASP98',
       'CYS148', 'GLN105', 'GLN144', 'GLN76', 'GLU23', 'GLU37', 'GLU50',
       'GLU93', 'GLY117', 'GLY14', 'HIS72', 'ILE126', 'LEU125', 'LYS102',
       'LYS110', 'LYS79', 'PRO20', 'SER36', 'SER47', 'SER7', 'SER71',
       'SER94', 'THR140', 'THR46', 'THR84', 'TYR131', 'TYR132', 'VAL41'],
      dtype='<U6')
```

We can also count how many times each residue was involved in water bridges (with different waters) and display the number of counts as a histogram.

```
In [16]: from collections import Counter

In [17]: aa_counter = Counter(allresidues)
```

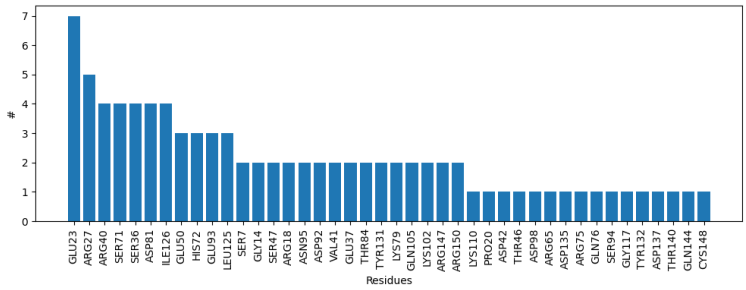
```
In [18]: sorted_aa_counter = dict(sorted(aa_counter.items(), key=lambda item: item[1], reverse=True))

In [19]: sorted_aa_counter
```

```
{'GLU23': 7,
 'ARG27': 5,
 'ARG40': 4,
 'SER71': 4,
 'SER36': 4,
 'ASP81': 4,
 'ILE126': 4,
 'GLU50': 3,
 'HIS72': 3,
 'GLU93': 3,
 'LEU125': 3,
 'SER7': 2,
 'GLY14': 2,
 'SER47': 2,
 'ARG18': 2,
 'ASN95': 2,
 'ASP92': 2,
 'VAL41': 2,
 'GLU37': 2,
 'THR84': 2,
 'TYR131': 2,
 'LYS79': 2,
 'GLN105': 2,
 'LYS102': 2,
 'ARG147': 2,
 'ARG150': 2,
 'LYS110': 1,
 ..
 'CYS148': 1}
```

```
In [20]: import matplotlib.pyplot as plt

In [21]: values = list(sorted_aa_counter.values())

In [22]: labels = list(sorted_aa_counter.keys())

In [23]: plt.figure(figsize=(10, 4))

In [24]: plt.bar(labels, values)

In [25]: plt.xticks(rotation=90)

In [26]: plt.xlabel('Residues')

In [27]: plt.ylabel('#')

In [28]: plt.tight_layout()

In [29]: plt.show()
```

Based on the results, we can see that there is one residue, GLU23, which often interacts with water molecules.

There are also options to save the output, which is especially important for trajectories. The information on how to do it you will find in that particular section.


#
0
1
2
3
4
5
6
7
GLU23
ARG27
ARG40
SER71
SER36
ASP81
ILE126
GLU50
HIS72
GLU93
LEU125
SER7
GLY14
SER47
ARG18
ASN95
ASP92
VAL41
GLU37
THR84
TYR131
LYS79
GLN105
LYS102
ARG147
ARG150
LYS110
PRO20
ASP42
THR46
ASP98
ARG65
ASP135
ARG75
GLN76
SER94
GLY117
TYR132
ASP137
THR140
GLN144
CYS148
Residues

CHAPTER

FOUR

WATER BRIDGES DETECTION IN A TRAJECTORY

Now, we will perform calculations for a trajectory file that was obtained using the **NAMD_** package. We will use `calcWaterBridgesTrajectory()`, for which we need to provide PDB and DCD files.

The system (protein in a water box) can be found in `5kqm_all_sci.pdb`. The trajectory, `NAMD_D2_sample.dcd`, has dcd format. If we want to analyze trajectories with different formats, we need to convert them to `dcd` file format or save the trajectory as a multi-model PDB (using **VMD_** or another tool).

4.1 Parse structure with trajectory

```
In [1]: PDBtraj_file = "5kqm_all_sci.pdb"

In [2]: coords_traj = parsePDB(PDBtraj_file)

In [3]: trajectory = parseDCD("NAMD_D2_sample.dcd")
```

```
@> 19321 atoms and 1 coordinate set(s) were parsed in 0.18s.
@> DCD file contains 17 coordinate sets for 19321 atoms.
@> DCD file was parsed in 0.01 seconds.
@> 3.76 MB parsed at input rate 748.06 MB/s.
@> 17 coordinate sets parsed at input rate 3382 frame/s.
```

The analysis od water bridges can be performed on selected frames by using `start_frame` or `stop_frame`.

```
In [4]: wb_traj = calcWaterBridgesTrajectory(coords_traj, trajectory, start_frame=5,
   ...:                                      stop_frame=15, output='info')
   ...:
```

```
@> Frame: 5
@> 101 water bridges detected.
@> Frame: 6
@> 107 water bridges detected.
@> Frame: 7
@> 90 water bridges detected.
@> Frame: 8
@> 97 water bridges detected.
@> Frame: 9
@> 122 water bridges detected.
@> Frame: 10
@> 101 water bridges detected.
@> Frame: 11
@> 130 water bridges detected.
@> Frame: 12
@> 132 water bridges detected.
@> Frame: 13
```

```
@> 126 water bridges detected.
@> Frame: 14
@> 88 water bridges detected.
@> Frame: 15
@> 105 water bridges detected.
```

Because of the amount of data, detailed results will not be displayed. We instead access the raw data by using `output='info'`.

```
In [5]: wb_traj
```

```
[[['THR5',
   'OG1_8',
   'P',
   'TRP39',
   'NE1_547',
   'P',
   3.261452627054999,
   1,
   ['3W_13313']],
  ['THR5',
   'O_15',
   'P',
   'ASP86',
   'OD1_1269',
   'P',
   5.986350034086454,
   2,
   ['3W_12974', '3W_18431']],
  ['THR5',
   'O_15',
   'P',
   'LYS110',
   'NZ_1667',
   'P',
   7.375256709599827,
   2,
   ['3W_12974', '3W_18431']],
  ['THR5',
   'O_15',
   'P',
   'LYS6',
   'NZ_32',
   'P',
   6.414308925017051,
   2,
   ['3W_12974', '3W_12152']],
  ['LYS6',
   'NZ_32',
   'P',
   'TYR87',
   'OH_1286',
   'P',
   4.891713264838611,
   1,
   ['3W_9209']]
   ...
   ...
```

```
    ]]
```

4.2 Save the results

The results can be saved using `saveWaterBridges()` in two formats. The `txt` file will contain all the results for analysis and can be visualized in a text editor, and the `wb` file will restore data for further analysis. It can be loaded using `parseWaterBridges()` as shown below.

First, we have to return the calculation without `output='info'`.

We can suppress the logged output using `confProDy()` to set the verbosity to `'none'`.

```
In [6]: confProDy(verbosity='none')

In [7]: wb_traj = calcWaterBridgesTrajectory(coords_traj, trajectory,
   ...:                                      stop_frame=15)
   ...:
```

```
In [8]: saveWaterBridges(wb_traj,'wb_saved.txt')

In [9]: saveWaterBridges(wb_traj,'wb_saved.wb')
```

To load the `wb` file, use `parseWaterBridges()` and protein coordinates as follows:

```
In [10]: waterBridges = parseWaterBridges('wb_saved.wb', coords_traj)
```

Loaded results from a `.wb` file are `Atomic` type and therefore can be used for analysis later.

4.3 Analysis of the results:

4.4 Information about residues contributing to water bridges

The data can be analyzed using `calcWaterBridgesStatistics()`. The following analysis provides details about the pairs of residues engaged in water bridges, their frequency of occurrence, and the average distance between them. The standard deviation offers insights into the variation in distance throughout the simulation. Moreover, the analysis can be saved using the `filename` option.

We can recover logged output using `confProDy()` again with a different verbosity.

```
In [11]: confProDy(verbosity='debug')

In [12]: analysisAtomic = calcWaterBridgesStatistics(waterBridges, trajectory,
   ....:                                             filename='data.txt')
   ....:
```

```
@> RES1          RES2          PERC      DIST_AVG  DIST_STD
@> ARG40P        SER7P         12.500    4.901     0.000
@> ASP92P        ARG18P        68.750    4.285     1.159
@> ASN95P        ARG18P        68.750    5.099     1.192
@> GLU23P        PRO20P        12.500    4.571     0.000
@> HSE72P        GLU23P        12.500    3.669     0.458
@> VAL41P        ARG27P        56.250    5.565     0.781
@> SER71P        ARG27P        75.000    6.116     0.445
@> ASN34P        ASP32P        25.000    4.218     0.652
@> GLU37P        SER36P        75.000    3.700     1.154
@> THR84P        ARG40P        50.000    4.235     0.671
@> ARG75P        ASP42P        68.750    3.159     0.652
```

```
@> ASN95P          THR46P          62.500     4.067      0.842
@> TYR49P          SER47P          50.000     4.320      0.757
..
..
```

The output is a dictionary, so we can use `dict.items()`[5] to inspect it.

```
In [13]: for item in list(analysisAtomic.items())[:5]:
   ....:     print(item)
   ....:
```

```
((40, 7), {'percentage': 12.5, 'distAvg': 4.9006157, 'distStd': 0.0})
((7, 40), {'percentage': 12.5, 'distAvg': 4.9006157, 'distStd': 0.0})
((92, 18), {'percentage': 68.75, 'distAvg': 4.2853837, 'distStd': 1.159262})
((18, 92), {'percentage': 68.75, 'distAvg': 4.2853837, 'distStd': 1.159262})
((95, 18), {'percentage': 68.75, 'distAvg': 5.0986476, 'distStd': 1.1916962})
```

To have easier access to the data, we can use `getWaterBridgeStatInfo()`.

```
In [14]: wb_stat_info = getWaterBridgeStatInfo(analysisAtomic, coords_traj)

In [15]: wb_stat_info
```

```
{('SER7P', 'ARG40P'): {'percentage': 12.5,
  'distAvg': 4.9006157,
  'distStd': 0.0},
 ('ARG18P', 'ASP92P'): {'percentage': 68.75,
  'distAvg': 4.2853837,
  'distStd': 1.159262},
 ('ARG18P', 'ASN95P'): {'percentage': 68.75,
  'distAvg': 5.0986476,
  'distStd': 1.1916962},
 ('PRO20P', 'GLU23P'): {'percentage': 12.5,
  'distAvg': 4.571081,
  'distStd': 0.0},
   ...
   ...
```

To obtain maps of interactions for the protein structure, we can use `showWaterBridgeMatrix()`, which is equipped with three paramaters: `'percentage'` (how often two residues were forming water bridges), `'distAvg'` (how close there were on average), and `'distStd'` (how stable that water bridge was).

```
In [16]: showWaterBridgeMatrix(analysisAtomic, 'percentage')
```

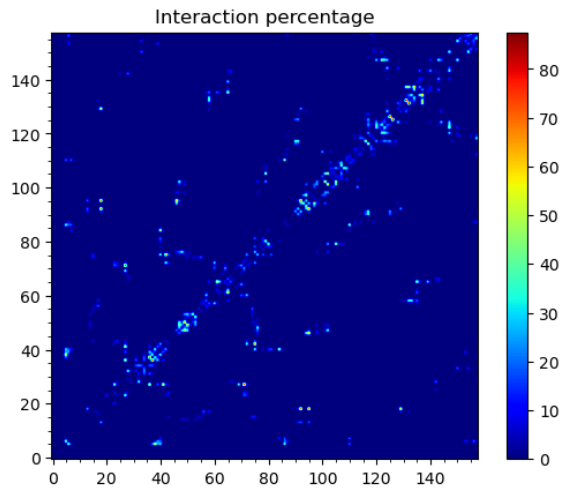
```
In [17]: showWaterBridgeMatrix(analysisAtomic, 'distAvg')
```

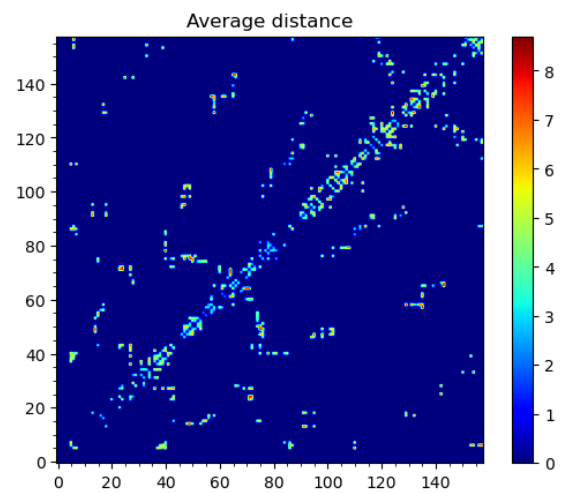
```
In [18]: showWaterBridgeMatrix(analysisAtomic, 'distStd')
```

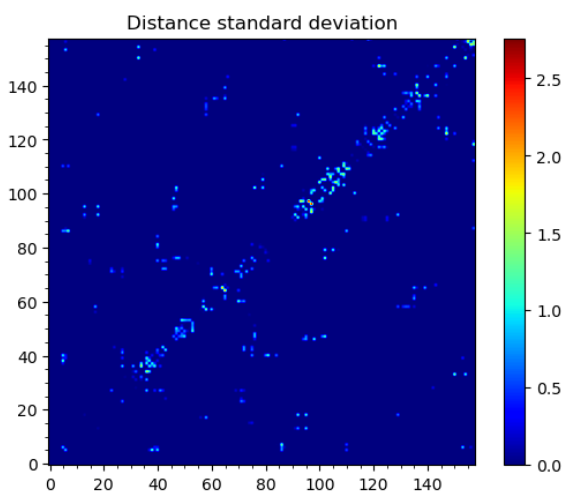
Raw data of the matrices can be obtained with `calcWaterBridgeMatrix()`. The type of the data in the matrix can be selected using the following strings for the second argument: `'percentage'`,`'distAvg'`,`'distStd'`.

```
In [19]: M1 = calcWaterBridgeMatrix(analysisAtomic, 'percentage')

In [20]: M2 = calcWaterBridgeMatrix(analysisAtomic, 'distAvg')

In [21]: M3 = calcWaterBridgeMatrix(analysisAtomic, 'distStd')
```

[5] http://docs.python.org/library/stdtypes.html#dict.items


Interaction percentage
0
20
40
60
80
100
120
140
0
10
20
30
40
50
60
70
80


Average distance
0
20
40
60
80
100
120
140
0
1
2
3
4
5
6
7
8


Distance standard deviation
0
20
40
60
80
100
120
140
0.0
0.5
1.0
1.5
2.0
2.5

```
In [22]: M1
```

```
array([[ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  , ...,  0.  ,  0.  ,  0.  ],
       ...,
       [ 0.  ,  0.  ,  0.  , ...,  0.  , 12.5 , 31.25],
       [ 0.  ,  0.  ,  0.  , ..., 12.5 ,  0.  , 12.5 ],
       [ 0.  ,  0.  ,  0.  , ..., 31.25, 12.5 ,  0.  ]])
```

```
In [23]: M2
```

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 4.58851337,
        5.82083416],
       [0.        , 0.        , 0.        , ..., 4.58851337, 0.        ,
        3.52366138],
       [0.        , 0.        , 0.        , ..., 5.82083416, 3.52366138,
        0.        ]])
```

```
In [24]: M3
```

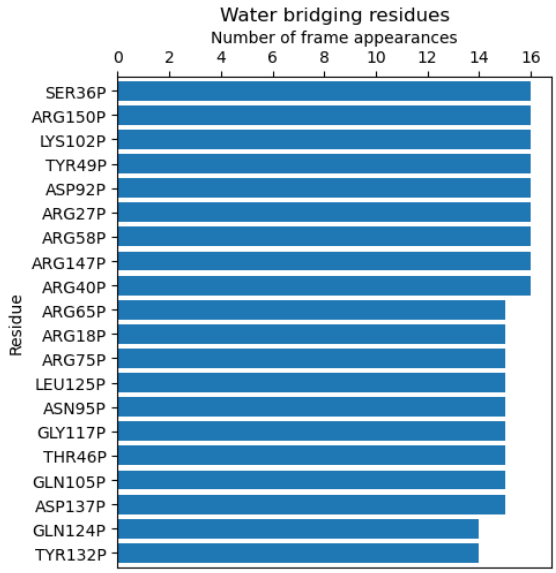
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 1.71697354,
        1.38650537],
       [0.        , 0.        , 0.        , ..., 1.71697354, 0.        ,
        1.27207112],
       [0.        , 0.        , 0.        , ..., 1.38650537, 1.27207112,
        0.        ]])
```

4.5 Statistical analysis for water bridges

To visualize the results in a more accessible way, we can use the `calcBridgingResiduesHistogram()` function, which will show how often each residue was contributing to the water bridges in the trajectory.

```
In [25]: wb_res_hist = calcBridgingResiduesHistogram(waterBridges)

In [26]: wb_res_hist
```

```
[('LEU96P', 1),
 ('MET63P', 1),
 ('PHE152P', 1),
 ('LEU29P', 1),
 ('PRO130P', 1),
 ('PHE85P', 1),
```




```
 ('PRO54P', 1),
 ('ILE16P', 1),
 ('CYS148P', 1),
 ('VAL25P', 1),
 ('ILE77P', 1),
 .
 .
 ('ARG75P', 15),
 ('ARG18P', 15),
 ('ARG65P', 15),
 ('ARG40P', 16),
 ('ARG147P', 16),
 ('ARG58P', 16),
 ('ARG27P', 16),
 ('ASP92P', 16),
 ('TYR49P', 16),
 ('LYS102P', 16),
 ('ARG150P', 16),
 ('SER36P', 16)]
```

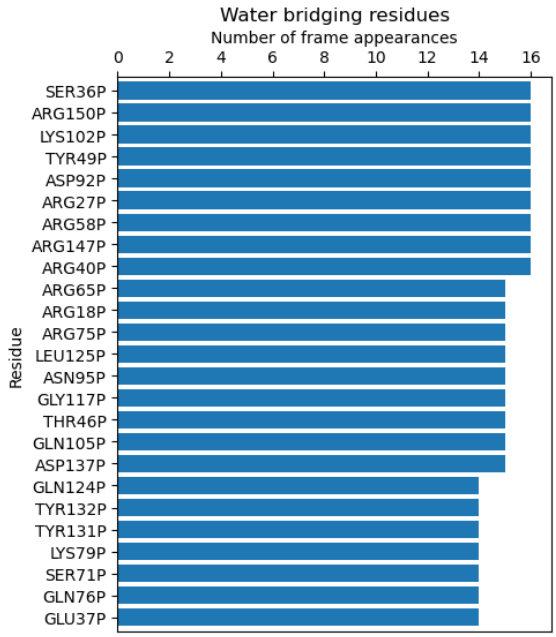
The `clip` option can be used to include different number of results on the histogram.

```
In [27]: calcBridgingResiduesHistogram(waterBridges, clip=25)
```

If we are interested in one particular residue, we can also use `calcWaterBridgesDistribution()` to find their partners in water bridges. Below we can see results for arginine 147 or aspartic acid 92 from `chain P` using the nomenclature for them corresponding to the keys of the dictionary.

```
In [28]: calcWaterBridgesDistribution(waterBridges, 'ARG147P')
```

```
[('GLN122P', 8),
 ('ARG150P', 7),
 ('GLN143P', 6),
 ('LYS123P', 6),
 ('GLN124P', 5),
 ('ASP120P', 5),
 ('GLN144P', 3),
```




```
 ('THR140P', 2)]
```

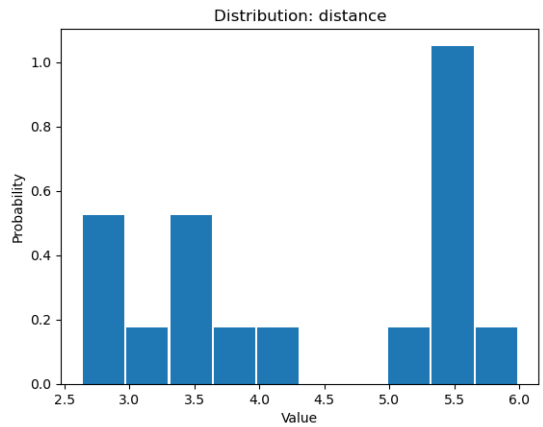
```
In [29]: calcWaterBridgesDistribution(waterBridges, 'ASP92P')
```

```
[('ARG18P', 11),
 ('ASN95P', 10),
 ('SER94P', 5),
 ('MET91P', 5),
 ('ASP129P', 4),
 ('LEU13P', 3),
 ('CYS90P', 1)]
```

Once we select a pair of residues which are supported by interactions with water molecules, we can use `calcWaterBridgesDistribution()` to obtain histograms with results such as distances between them (`metric='distance'`), the number of water molecules which were involved (`metric='waters'`), and information about residue part which was involved in water bridges, i.e. backbone or side chain (`metric='location'`).

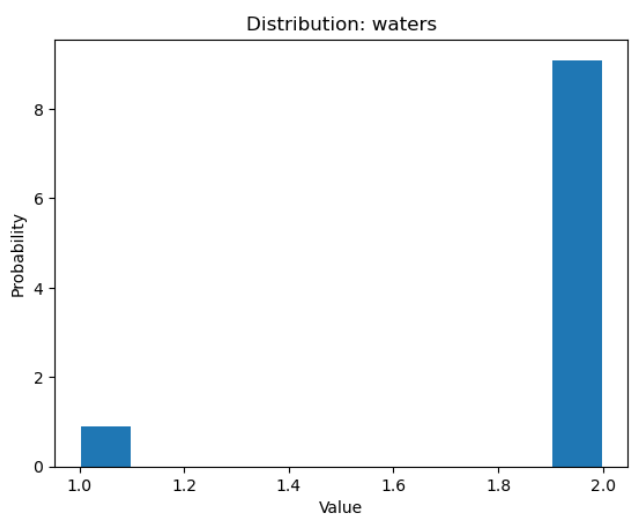
```
In [30]: calcWaterBridgesDistribution(waterBridges,  'ASP92P', 'ARG18P',
   ....:                              trajectory=trajectory, metric='distance')
   ....:
```

```
[5.3736005,
 5.3736005,
 5.167575,
 2.681302,
 5.371548,
 2.6318514,
 3.0394073,
 4.0884595,
 5.4406505,
 3.4112484,
 2.805657,
 5.4176636,
```




```
 3.5104342,
 5.991175,
 5.470093,
 3.4345005,
 3.6427624]
```

```
In [31]: calcWaterBridgesDistribution(waterBridges, 'ARG147P', 'GLN122P',
   ....:                                                 metric='waters')
   ....:
```




```
[2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2]
```

```
In [32]: calcWaterBridgesDistribution(waterBridges, 'ARG147P', 'GLN122P',
   ....:                              trajectory=trajectory, metric='location')
   ....:
```

```
{'ARG147P': {'backbone': 7, 'side': 86},
'GLN122P': {'backbone': 21, 'side': 25}}
```

4.6 Save results as PDB file

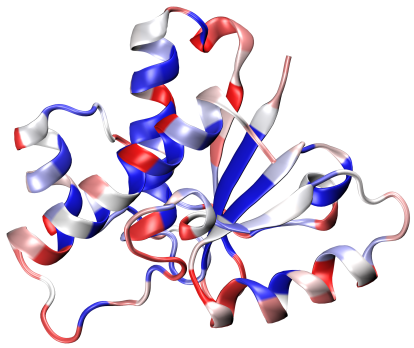
The results can be stored as a PDB file using `savePDBWaterBridges()` (single PDB file, single frame) or using `savePDBWaterBridgesTrajectory()` to save all the results (large number of frames saved each independently).

`5kqm_all_sci_multi_0.pdb` `5kqm_all_sci_multi_1.pdb` .. `5kqm_all_sci_multi_15.pdb`

Those results can be displayed in any program for visualization. The results for the protein structure will be storage in the `B-factor (*beta*) column` (average values of contributions of each residue in water bridging) and `Occupancy column` (results for particular frame). Water molecules will be included in each frame.

```
In [33]: savePDBWaterBridges(waterBridges[0], coords_traj, PDBtraj_file[:-4]+'_frame0.pdb')

In [34]: savePDBWaterBridgesTrajectory(waterBridges, coords_traj,
   ....:                               filename=PDBtraj_file[:-4]+'_multi.pdb',
   ....:                               trajectory=trajectory)
   ....:
```

Results saved in PDB file can be displayed as follows:



CHAPTER

FIVE

WATER BRIDGES DETECTION IN AN ENSEMBLE PDB

This time we will use an ensemble stored in a multi-model PDB, which contains 50 frames from MD simulations from PE-binding protein 1 (PDB: **1BEH**). Simulations were performed using **NAMD_** and saved as a multi-model PDB using **VMD_**.

We need to remember to align the protein structure before performing the analysis. Otherwise, when all structures are uploaded to the visualization program, they will be spread out in space. We could do this inside ProDy by converting the `Atomic` object to an `Ensemble` object and using its `Ensemble.iterpose()` method to do this, but here we demonstrate the process for parsing a multi-model PDB file directly.

5.1 Initial analysis

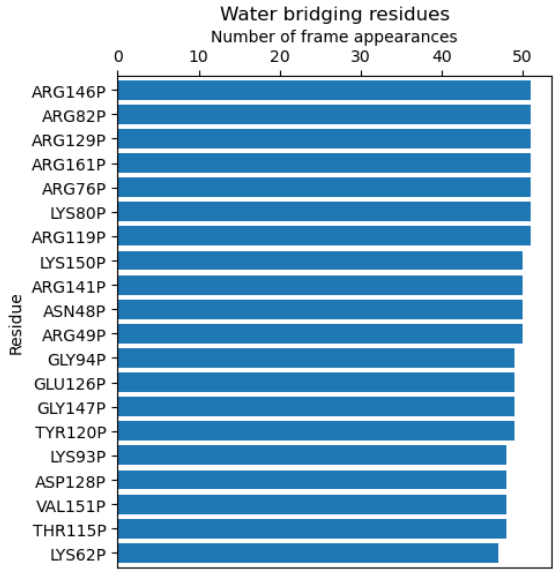
```
In [1]: ens = 'pebp1_50frames.pdb'

In [2]: coords_ens = parsePDB(ens)

In [3]: bridgeFrames_ens = calcWaterBridgesTrajectory(coords_ens, coords_ens)
```

```
@> 20195 atoms and 51 coordinate set(s) were parsed in 1.88s.
@> Frame: 0
@> 161 water bridges detected.
@> Frame: 1
@> 127 water bridges detected.
@> Frame: 2
@> 168 water bridges detected.
@> Frame: 3
@> 132 water bridges detected.
@> Frame: 4
@> 142 water bridges detected.
@> Frame: 5
@> 166 water bridges detected.
@> Frame: 6
@> 150 water bridges detected.
@> Frame: 7
@> 159 water bridges detected.
@> Frame: 8
@> 147 water bridges detected.
@> Frame: 9
@> 136 water bridges detected.
@> Frame: 10
@> 127 water bridges detected.
@> Frame: 11
@> 135 water bridges detected.
```

```
...
...
```

Analysis of the results is similar to that presented in trajectory analysis. Below are examples showing which residues are most frequently involved in water bridge formation (`calcBridgingResiduesHistogram()`), details of those interactions (`calcWaterBridgesStatistics()`), and results saved as a PDB structure for further visualization (`savePDBWaterBridgesTrajectory()`). Other functions can be explored in the trajectory analysis.

```
In [4]: calcBridgingResiduesHistogram(bridgeFrames_ens)
```




```
[('VAL34P', 1),
 ('VAL177P', 1),
 ('PRO43P', 1),
 ('LEU41P', 2),
 ('MET92P', 2),
 ('VAL164P', 3),
 ('LEU14P', 3),
 ('TYR169P', 3),
 ('PHE154P', 4),
 .
 .
 ('ARG49P', 50),
 ('ASN48P', 50),
 ('ARG141P', 50),
 ('LYS150P', 50),
 ('ARG119P', 51),
 ('LYS80P', 51),
 ('ARG76P', 51),
 ('ARG161P', 51),
 ('ARG129P', 51),
 ('ARG82P', 51),
 ('ARG146P', 51)]
```

```
In [5]: analysisAtomic_ens = calcWaterBridgesStatistics(bridgeFrames_ens,
   ...:                                                 coords_ens)
   ...:
```

```
In [6]: for item in analysisAtomic_ens.items():
   ...:     print(item)
   ...:
```

```
@> RES1            RES2            PERC      DIST_AVG  DIST_STD
@> VAL3P           HSE26P          19.608    5.581     0.696
@> ASP4P           SER6P           13.725    3.817     0.560
@> SER6P           LYS7P           43.137    4.394     1.114
@> LYS7P           GLU36P          1.961     6.088     0.000
@> LYS7P           LEU37P          7.843     6.353     0.433
@> GLY10P          SER13P          43.137    4.759     0.612
@> GLY10P          ARG76P          11.765    5.309     0.586
@> LEU12P          SER13P          45.098    2.767     0.080
@> SER13P          GLU16P          25.490    4.449     1.133
@> GLN15P          ASP18P          7.843     3.732     0.174
@> GLU16P          ARG82P          45.098    4.550     1.086
@> GLU16P          VAL17P          17.647    3.438     0.952
@> GLU16P          LYS150P         21.569    5.056     0.929
@> GLU16P          GLU83P          9.804     5.476     1.138
@> GLU16P          ALA152P         7.843     7.307     0.450
@> VAL17P          GLU83P          1.961     7.262     0.000
@> VAL17P          LYS150P         13.725    6.303     0.572
@> GLN22P          GLU126P         33.333    6.458     1.216
@> GLN22P          HSE23P          37.255    4.738     0.669
@> HSE23P          GLU126P         7.843     7.911     0.239
@> PRO24P          ASP56P          17.647    5.592     0.910
@> THR28P          SER52P          43.137    3.970     0.677
@> THR28P          ILE53P          5.882     5.849     0.027
@> TYR29P          THR51P          7.843     3.583     0.286
@> ALA30P          ARG49P          17.647    5.206     0.304
...
...
```

```
In [7]: savePDBWaterBridgesTrajectory(bridgeFrames_ens, coords_ens, ens[:-4]+'_ens.pdb')
```

```
@> All 51 coordinate sets are copied to pebp1_50frames Selection 'protein' + pebp1_50frames Selectio
@> All 51 coordinate sets are copied to pebp1_50frames Selection 'protein' + pebp1_50frames Selectio
@> All 51 coordinate sets are copied to pebp1_50frames Selection 'protein' + pebp1_50frames Selectio
..
..
```

5.2 Detecting water centers

The previous function generated multiple PDB files in which we can find protein and water molecules for each frame that form water bridges with the protein structure. Now we can use another function `findClusterCenters()` which will extract water centers (they refer to the oxygens from water molecules that are forming clusters). We need to provide a file pattern as show below. Now all the PDB files with prefix `'pebp1_50frames_ens_'` will be analyzed.

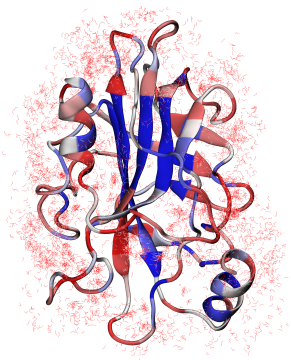
```
In [8]: findClusterCenters('pebp1_50frames_ens_*.pdb')
```

```
@> 3269 atoms and 1 coordinate set(s) were parsed in 0.11s.
@> 3161 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 3173 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3173 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3218 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3251 atoms and 1 coordinate set(s) were parsed in 0.04s.
```

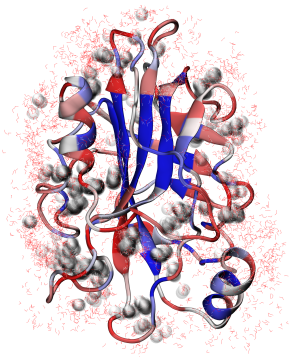
```
@> 3215 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3230 atoms and 1 coordinate set(s) were parsed in 0.03s.
@> 3230 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3224 atoms and 1 coordinate set(s) were parsed in 0.03s.
@> 3158 atoms and 1 coordinate set(s) were parsed in 0.03s.
..
..
@> Results are saved in clusters_pebp1_50frames_ens_.pdb.
```

This function generated one PDB file with water centers. We used default values, such as `distC` (distance to other molecule) and `numC` (min number of molecules in a cluster), but those values could be changed if the molecules are more widely distributed or we would like to have more numerous clusters. Moreover, this function can be applied to different types of molecules by using the `selection` parameter. We can provide the whole molecule, and by default, the center of mass will be used as a reference.

Saved PDB files using `savePDBWaterBridgesTrajectory()` in the previous step can be upload to **VMD_** or other program for visualization:



After uploading a new PDB file with water centers we can see the results as follows:



CHAPTER

SIX

CHANGES IN THE DEFAULT PARAMETERS

There are a lot of parameters that can be changed in the water bridge analysis, including the distances, angles, and the number of involved water molecules or residues.

`atoms`: Atomic object from which atoms are considered

`method`: `'cluster'` or `'chain'`, where 'chain' will find the shortest water bridging path between two protein atoms default is `'chain'`

`distDA`: maximal distance between water/protein donor and acceptor default is `3.5`

`distWR`: maximal distance between considered water and any residue default is `4`

`anglePDWA`: angle range where protein is donor and water is acceptor default is `(100, 200)`

`anglePAWD`: angle range where protein is acceptor and water is donor default is `(100, 140)`

`angleWW`: angle between water donor/acceptor default is `(140, 180)`

`maxDepth`: maximum number of waters in chain/depth of residues in cluster default is `2`

`maxNumRes`: maximum number of water+protein residues in cluster default is `None`

`donors`: which atoms to count as donors default is `['N', 'O', 'S', 'F']`

`acceptors`: which atoms to count as acceptors default is `['N', 'O', 'S', 'F']`

`output`: return information arrays, (protein atoms, water atoms), or just atom indices per bridge Options are `'info'`,`'atomic'` and `'indices'` Default is `'atomic'`

`isInfoLog`: whether to log information default is `True`

Here's an example of how to apply changes in parameters:

```
In [1]: waterBridges_2 = calcWaterBridges(atoms, method='cluster', distWR=3.0, distDA=3.2, maxDepth=
```

```
@> 17 water bridges detected.
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> GLY14 O_72 A SER47 O_328 A GLU50 N_347 A 5.288116583434976 4.544135231262378 5.10489901956934 1 [
@> PRO20 O_115 A GLU23 OE1_139 A 4.571172934816621 1 ['A_1292']
@> GLU23 OE2_140 A SER71 O_514 A HIS72 ND1_523 A 4.934310286149422 4.127239392136104 4.4637344231035
@> ASP42 OD2_301 A ARG40 NH2_286 A 5.163938516287738 1 ['A_1246']
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.922971705225
@> GLY14 O_72 A SER47 O_328 A GLU50 N_347 A 5.288116583434976 4.544135231262378 5.10489901956934 1 [
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 1 ['A_1267']
@> GLU23 OE2_140 A SER71 O_514 A HIS72 ND1_523 A 4.934310286149422 4.127239392136104 4.4637344231035
@> GLN105 NE2_800 A LYS102 O_772 A LYS79 NZ_582 A 4.2363221076778395 3.8752885053889865 4.9292212366
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.922971705225
@> GLN105 NE2_800 A LYS102 O_772 A LYS79 NZ_582 A 4.2363221076778395 3.8752885053889865 4.9292212366
```

```
@> SER7 OG_21 A LYS110 NZ_838 A 4.70722699686344 1 ['A_1316']
@> LEU125 O_953 A GLY117 N_886 A 3.8595291163560352 1 ['A_1264']
@> GLU50 OE2_355 A TYR131 OH_1009 A 5.157987010452818 1 ['A_1299']
@> ARG65 NH2_473 A ASP135 O_1037 A 4.820906553751068 1 ['A_1267']
```

Acknowledgments

Continued development of Protein Dynamics Software *ProDy* and associated programs is partially supported by the NIH[6]-funded Biomedical Technology and Research Center (BTRC) on *High Performance Computing for Multiscale Modeling of Biological Systems* (MMBios[7]) (P41 GM103712).

[6] http://www.nih.gov/
[7] http://mmbios.org/