Interactions Analysis

Release

Mikulska-Ruminska, Karolina

December 04, 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  2024 5kqm_all_sci.pdb
446K Feb 29  2024 addH_5kqm.pdb
2.9M Aug  5 10:24 case_study1.pdb
1.5M Aug  5 10:24 case_study4a.pdb
1.5M Aug  5 10:24 case_study4b.pdb
3.8M Feb 29  2024 NAMD_D2_sample.dcd
 78M Feb 29  2024 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.

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

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

In [2]: from pylab import *

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.

1.4 How to Cite

If you benefited from WatFinder Calculations in your research, please cite the following paper:

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 3.3343465626716
@> 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 4.642634596864156
@> 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 3.0545251676815504
@> SER36 N_239 A SER36 OG_244 A GLU37 N_245 A 2.9687596736684503 2.743692767056837 3.0545251676815504
@> 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 3.9229717052225
@> ARG27 NE_171 A ARG27 NH2_174 A VAL41 N_287 A 2.298703982682415 5.672199573357763 4.642634596864156
@> 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 4.929221236666
@> 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 3.9229717052225
@> ASP81 OD1_598 A THR84 OG1_621 A ARG40 NH2_286 A 4.415462942886057 4.365525627000715 3.9229717052225
@> 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 4.92922123666
@> 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 2.92897763733354
@> ILE126 O_961 A LEU125 N_950 A ILE126 N_958 A 4.854966117286505 2.8761442940158615 2.92897763733354
@> 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.

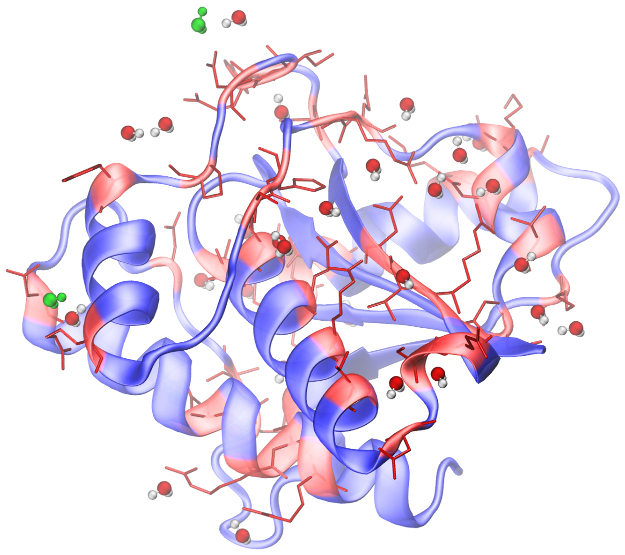
3.3 Access to the raw data

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.

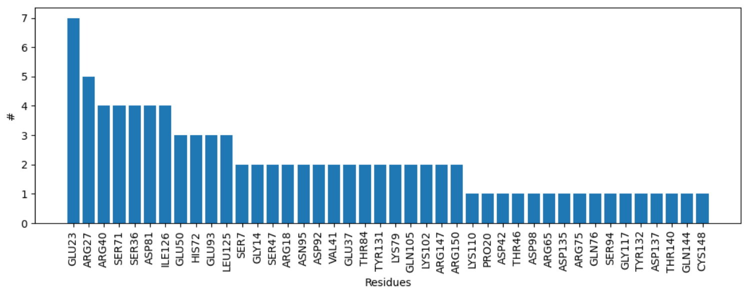
```
In [16]: from collections import Counter

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.


7
6
5
4
3
2
1
0
#
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')
```

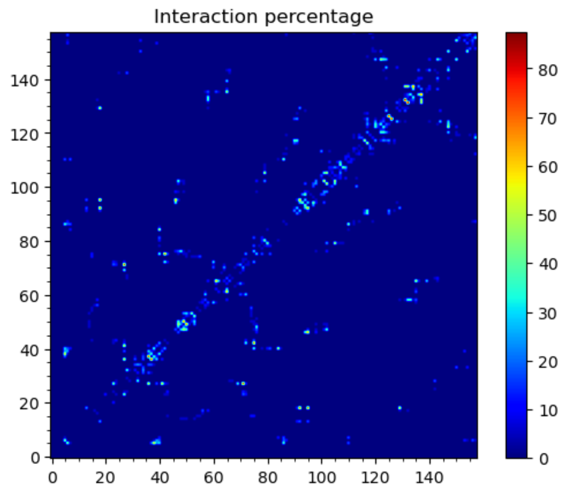
```
In [17]: showWaterBridgeMatrix(analysisAtomic, 'distAvg')
```

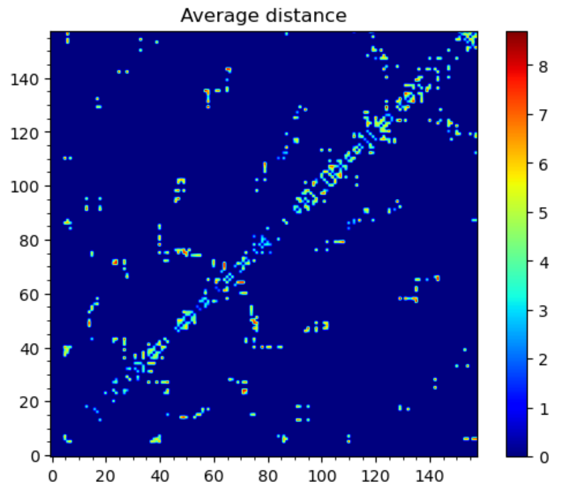
```
In [18]: showWaterBridgeMatrix(analysisAtomic, 'distStd')
```

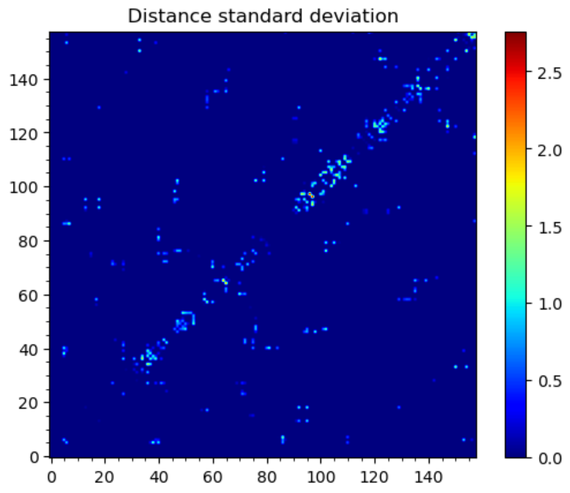
Raw data of the matrices can be obtained with `calcWaterBridgeMatrix()`. The type of the data in the matrix can be selected using the following strings for the second argument: `'percentage'`, `'distAvg'`, `'distStd'`.

```
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
```

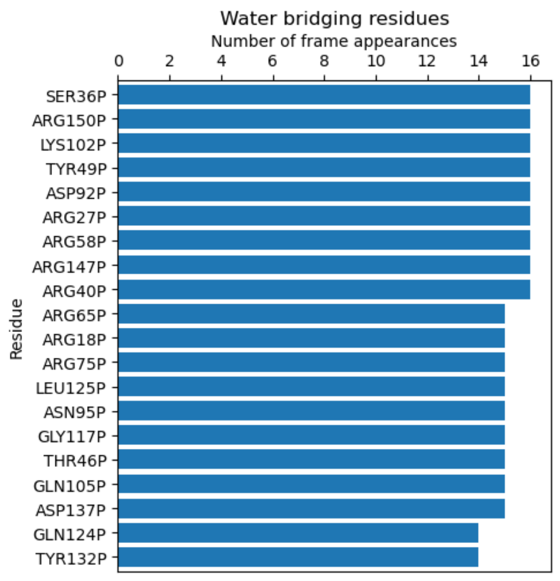
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
  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

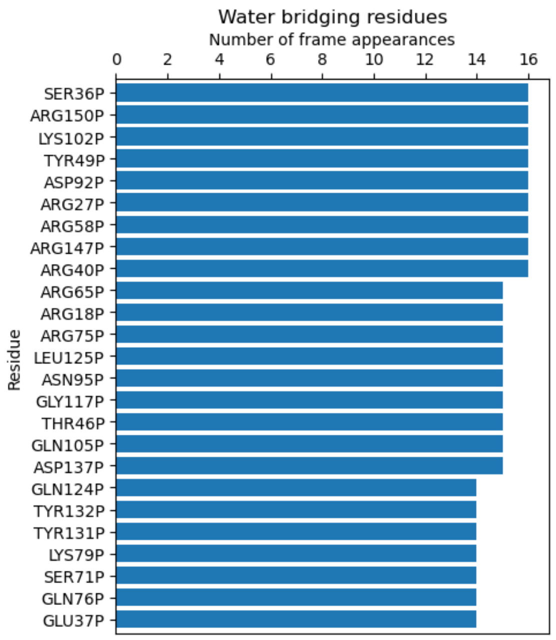
The `clip` option can be used to include different number of results on the histogram.

```
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)]
```

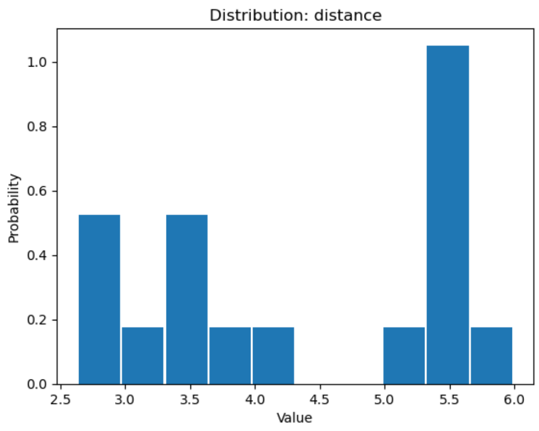
```
In [29]: calcWaterBridgesDistribution(waterBridges, 'ASP92P')
```

```
[('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')`.

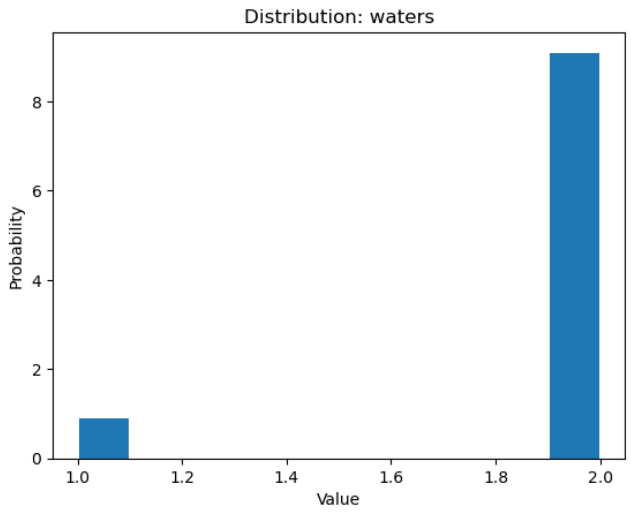
```
In [30]: calcWaterBridgesDistribution(waterBridges,  'ASP92P', 'ARG18P',
   ....:                                  trajectory=trajectory, metric='distance')
   ....:
```

```
[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

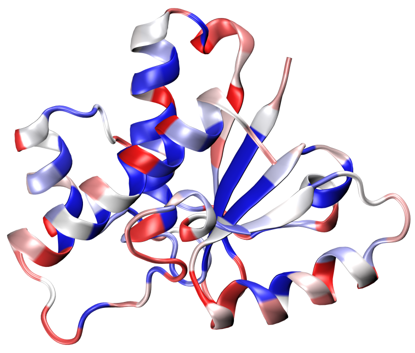
The results can be stored as a PDB file using `savePDBWaterBridges()` (single PDB file, single frame) or using `savePDBWaterBridgesTrajectory()` to save all the results (large number of frames saved each independently).

`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

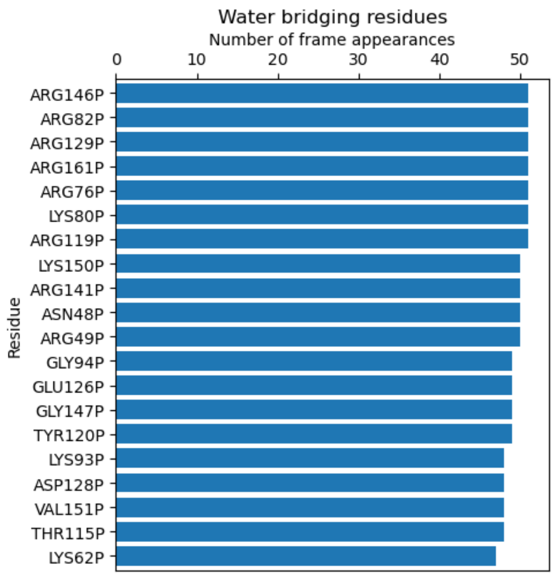
```
In [1]: ens = 'pebp1_50frames.pdb'

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

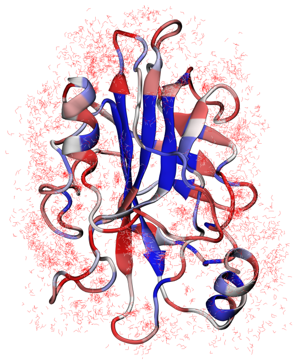
The previous function generated multiple PDB files in which we can find protein and water molecules for each frame that form water bridges with the protein structure. Now we can use another function `findClusterCenters()` which will extract water centers (they refer to the oxygens from water molecules 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')
```

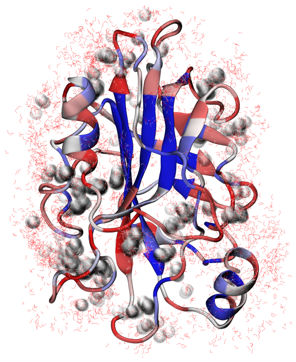
```
@> 3269 atoms and 1 coordinate set(s) were parsed in 0.11s.
@> 3161 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 3173 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3173 atoms and 1 coordinate set(s) were parsed in 0.04s.
@> 3218 atoms and 1 coordinate set(s) were parsed in 0.04s.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

FINDING SIGNIFICANT CRYSTALLOGRAPHIC WATERS ACROSS HETEROGENEOUS X-RAY STRUCTURES

Now, we will illustrate how to find significant protein-water interactions that are observed across different PDB structures. In this example, we will use 76 heterogeneous PDB structures of p38 MAP kinase (MAPK; analyzed in Ensemble Analysis), and analyze them with WatFinder to obtain water bridges present in numerous crystal structures.

6.1 Parse multiple PDB structures

First, we need to download all PDB structures that will be used for analysis. We can do it by providing directly a list with PDB IDs. In this case we are providing 76 structures storage in a list called `pdbids`.

```
In [1]: pdbids = ['5UOJ', '1A9U', '1BL6', '1BL7', '1BMK', '1DI9', '1IAN', '1KV1',
   ...:           '1KV2', '1LEW', '1LEZ', '1M7Q', '1OUK', '1OUY', '1OVE', '1OZ1',
   ...:           '5UOJ', '1R39', '1R3C', '1W7H', '1W82', '1W83', '1W84', '1WBN',
   ...:           '1WBO', '1WBS', '1WBT', '1WBV', '1WBW', '1WFC', '1YQJ', '1YW2',
   ...:           '1YWR', '1ZYJ', '1ZZ2', '1ZZL', '2BAJ', '2BAK', '2BAL', '2BAQ',
   ...:           '2EWA', '2FSL', '2FSM', '2FSO', '2FST', '2GFS', '2GHL', '2GHM',
   ...:           '2GTM', '2GTN', '2I0H', '2NPQ', '2OKR', '2OZA', '3HVC', '3MH0',
   ...:           '3MH3', '3MH2', '2PUU', '3MGY', '3MH1', '2QD9', '2RG5', '2RG6',
   ...:           '2ZAZ', '2ZB0', '2ZB1', '3BV2', '3BV3', '3BX5', '3C5U', '3L8X',
   ...:           '3CTQ', '3D7Z', '3D83', '2ONL']
   ...:
```

```
In [2]: pdbids
```

```
['5UOJ', '1A9U', '1BL6', '1BL7', '1BMK', '1DI9',  '1IAN', '1KV1',
'1KV2', '1LEW', '1LEZ', '1M7Q', '1OUK', '1OUY', '1OVE', '1OZ1',
'5UOJ', '1R39', '1R3C', '1W7H', '1W82', '1W83', '1W84', '1WBN',
'1WBO', '1WBS', '1WBT', '1WBV', '1WBW', '1WFC', '1YQJ', '1YW2',
'1YWR', '1ZYJ', '1ZZ2', '1ZZL', '2BAJ', '2BAK', '2BAL', '2BAQ',
'2EWA', '2FSL', '2FSM', '2FSO', '2FST', '2GFS', '2GHL', '2GHM',
'2GTM', '2GTN', '2I0H', '2NPQ', '2OKR', '2OZA', '3HVC', '3MH0',
'3MH3', '3MH2', '2PUU', '3MGY', '3MH1', '2QD9', '2RG5', '2RG6',
'2ZAZ', '2ZB0', '2ZB1', '3BV2', '3BV3', '3BX5', '3C5U', '3L8X',
'3CTQ', '3D7Z', '3D83', '2ONL']
```

6.2 Protein preparation

We need to prepare each PDB structure for the analysis with WatFinder. First, we download each PDB file directly from the Protein Data Bank.

```
In [3]: infiles = fetchPDB(pdbids, compressed=False)
```

```
@> Connecting wwPDB FTP server RCSB PDB (USA).
@> 5uoj downloaded (5uoj.pdb)
@> 1a9u downloaded (1a9u.pdb)
@> 1bl6 downloaded (1bl6.pdb)
@> 1bl7 downloaded (1bl7.pdb)
@> 1bmk downloaded (1bmk.pdb)
@> 1di9 downloaded (1di9.pdb)
@> 1ian downloaded (1ian.pdb)
@> 1kv1 downloaded (1kv1.pdb)
@> 1kv2 downloaded (1kv2.pdb)
@> 1lew downloaded (1lew.pdb)
@> 1lez downloaded (1lez.pdb)
..
..
@> 3ctq downloaded (3ctq.pdb)
@> 3d7z downloaded (3d7z.pdb)
@> 3d83 downloaded (3d83.pdb)
@> 2onl downloaded (2onl.pdb)
@> PDB download via FTP completed (76 downloaded, 0 failed).
```

Next, we parse all PDB files and exclude those structures that don't contain water molecules using the `filterStructuresWithoutWater()` function. PDB files need to contain at least `1` water molecule. Structures that will be analyzed are provided using `filenames`.

```
In [4]: structures = parsePDB(infiles)
```

```
@> 76 PDBs were parsed in 2.65s.
```

```
In [5]: new_structures = filterStructuresWithoutWater(structures, 1, filenames=infiles)
```

```
@> WARNING 2onl doesn't contain water molecules
@> WARNING 1kv2 doesn't contain water molecules
@> WARNING 1ian doesn't contain water molecules
```

We can now check how many structures are left that contain water molecules, and those will be further taken into analysis with WatFinder.

```
In [6]: len(new_structures)
```

```
73
```

```
In [7]: infiles
```

```
['5uoj.pdb',
'1a9u.pdb',
'1bl6.pdb',
'1bl7.pdb',
'1bmk.pdb',
'1di9.pdb',
'1kv1.pdb',
'1lew.pdb',
'1lez.pdb',
'1m7q.pdb',
'1ouk.pdb',
..
..
'3ctq.pdb',
'3d7z.pdb',
'3d83.pdb']
```

Finally, we will add hydrogen atoms, which are typically missing in the PDB files, using `fixStructuresMissingAtoms()` function and *'pdbfixer'* method. The new files will contain the `'addH_'` prefix and will be stored in `new_pdbids`.

```
In [8]: new_pdbids = fixStructuresMissingAtoms(infiles, method='pdbfixer')
```

```
@> Hydrogens were added to the structure. New structure is saved as addH_5uoj.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1a9u.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1bl6.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1bl7.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1bmk.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1di9.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1kv1.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_1lew.pdb.
..
..
@> Hydrogens were added to the structure. New structure is saved as addH_3l8x.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_3ctq.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_3d7z.pdb.
@> Hydrogens were added to the structure. New structure is saved as addH_3d83.pdb.
```

6.3 Aligning all PDB heterogeneous structures onto first PDB

In the next step of protein preparation, we need to perform structural alignment of all PDB structures we will analyze. We will align all PDBs onto the first PDB structure of our list. Each aligned file is independently saved in the local directory with `'align__'` prefix.

```
In [9]: structures = parsePDB(new_pdbids)

In [10]: target = structures[0]

In [11]: rmsds = []

In [12]: for mobile in structures[1:]:
   ....:     try:
   ....:         i = mobile.getTitle()
   ....:         print (i)
   ....:         matches = matchChains(mobile.protein, target.protein, subset='bb')
   ....:         m = matches[0]
   ....:         m0_alg, T = superpose(m[0], m[1], weights=m[0].getFlags("mapped"))
   ....:         rmsds.append(calcRMSD(m[0], m[1], weights=m[0].getFlags("mapped")))
   ....:         writePDB('align__'+i+'.pdb', mobile)
   ....:     except: pass
   ....:
```

```
@> 73 PDBs were parsed in 4.79s.
@> Checking AtomGroup addH_1a9u: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_1a9u (len=351) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_1a9u (len=351) and Chain A from addH_5uoj (len=343):
```

```
@>  Match: 343 residues match with 99% sequence identity and 98% overlap.

addH_1a9u

@> Checking AtomGroup addH_1bl6: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_1bl6 (len=351) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_1bl6 (len=351) and Chain A from addH_5uoj (len=343):
@>  Match: 343 residues match with 99% sequence identity and 98% overlap.

addH_1bl6

@> Checking AtomGroup addH_1bl7: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_1bl7 (len=351) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_1bl7 (len=351) and Chain A from addH_5uoj (len=343):
@>  Match: 343 residues match with 99% sequence identity and 98% overlap.

addH_1bl7

@> Checking AtomGroup addH_1bmk: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_1bmk (len=351) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_1bmk (len=351) and Chain A from addH_5uoj (len=343):
@>  Match: 343 residues match with 100% sequence identity and 98% overlap.

..
..

addH_3c5u

@> Checking AtomGroup addH_3l8x: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_3l8x (len=326) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=95%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_3l8x (len=326) and Chain A from addH_5uoj (len=343):
@>  Match: 325 residues match with 99% sequence identity and 95% overlap.

addH_3l8x

@> Checking AtomGroup addH_3ctq: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_3ctq (len=336) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=53%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_3ctq (len=336) and Chain A from addH_5uoj (len=343):
```

```
@>  Match: 336 residues match with 99% sequence identity and 98% overlap.

addH_3ctq

@> Checking AtomGroup addH_3d7z: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_3d7z (len=349) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_3d7z (len=349) and Chain A from addH_5uoj (len=343):
@>  Match: 342 residues match with 99% sequence identity and 98% overlap.

addH_3d7z

@> Checking AtomGroup addH_3d83: 1 chains are identified
@> Checking AtomGroup addH_5uoj: 1 chains are identified
@> Trying to match chains based on residue numbers and names:
@>   Comparing Chain A from addH_3d83 (len=349) and Chain A from addH_5uoj (len=343):
@>  Failed to match chains (seqid=5%, overlap=98%).
@> Trying to match chains based on local sequence alignment:
@>  Comparing Chain A from addH_3d83 (len=349) and Chain A from addH_5uoj (len=343):
@>  Match: 342 residues match with 99% sequence identity and 98% overlap.

addH_3d83
```

To see how different the protein structures are we will also compute `RMSD` (Root Mean Square Deviation) values:

```
In [13]: rmsds
```

```
[3.5502773224903406,
3.5518560313809213,
3.531791159385768,
3.536308595458991,
3.5883843009524425,
1.3508050136935905,
4.024771814068961,
3.6180331783111113,
3.6451321851562795,
3.6225864041371074,
3.7010843221966856,
0.7266335022815086,
1.2192209064105432e-14,
3.6814519776524275,
..
..
1.3153012774723138,
1.7110265263755273,
1.3953681417287447,
4.01269231318287,
4.071641705884,
3.5869450244694794,
3.5599082354788183,
3.7815283489614484,
2.190815934106486,
1.9023911636752533,
2.184819914027742,
3.944364439138517,
4.082500149787881,
1.9054336876325983,
3.7413357286577353,
3.680180759491109]
```

6.4 Analyzing PDB structures with certain pattern

When protein structures are prepared, i.e., hydrogens are added, protein structures that lack water molecules are eliminated, and they are all aligned, we can finally start the analysis with WatFinder. To do it, we will use the prefix name `namePrefix` to select the PDB structure for the analysis. We are using the current directory to find those files.

The code below will analyze all found PDB structures with `'align__'` prefix in the current directory and analyze them using `calcWaterBridges()` function. Structures that are not protein structures or water molecules will be ignored. The analyzed structure will be saved using `savePDBWaterBridges()` function with `'wb_'` prefix in the same directory.

```
In [14]: import os

In [15]: namePrefix = 'align__'

In [16]: directory = os.getcwd()

In [17]: align_files = [file for file in os.listdir(directory) if file.startswith(namePrefix)]

In [18]: for file in align_files:
   ....:     print (file)
   ....:     try:
   ....:         atoms = parsePDB(file)
   ....:         waterBridges = calcWaterBridges(atoms)
   ....:         savePDBWaterBridges(waterBridges, atoms, 'wb_'+file)
   ....:     except:
   ....:         print("This protein doesn't contain water bridges")
   ....:
```

```
@> 7359 atoms and 1 coordinate set(s) were parsed in 0.07s.

align__addH_1wbn.pdb

@> 59 water bridges detected using method chain.
@> PHE5 N_69 A ARG91 NH2_1506 A 6.573956038794295 1 ['B_5724']
@> GLU19 N_323 A ARG20 N_338 A 2.748689869737945 1 ['B_5829']
@> LYS51 NZ_809 A ASP98 OD1_1613 A 4.4370056344341045 1 ['B_6345']
@> ARG54 N_843 A ASP98 OD1_1613 A 5.677863770820855 1 ['B_6048']
@> ILE59 N_929 A ILE60 N_948 A 2.840243123396305 1 ['B_6069']
@> LYS63 NZ_1012 A TRP334 NE1_5377 A 5.914715715907233 2 ['B_7260', 'B_7287']
@> LYS63 NZ_1012 A LEU329 N_5280 A 6.702891465628844 2 ['B_7260', 'B_7287']
@> MET75 O_1239 A LEU83 N_1361 A 4.862355190645785 1 ['B_6135']
@> MET75 O_1239 A GLY82 N_1354 A 4.490536938941713 1 ['B_6162']
@> MET75 O_1239 A ILE81 O_1340 A 3.76368888193485 1 ['B_6162']
@> MET75 O_1239 A HIS77 N_1273 A 4.194540380065494 1 ['B_6162']
@> HIS77 N_1273 A GLY82 N_1354 A 6.3842001065129494 1 ['B_6162']
@> HIS77 N_1273 A ILE81 O_1340 A 5.919256794564666 1 ['B_6162']
@> ILE81 O_1340 A GLY82 N_1354 A 2.259809283988364 1 ['B_6162']
@> GLY82 N_1354 A LYS162 NZ_2660 A 5.02103296145325 1 ['B_6624']
@> GLY82 N_1354 A HIS104 ND1_1711 A 5.9096712260497215 1 ['B_6624']
@> GLU94 O_1544 A LYS335 NZ_5406 A 5.4192233760936634 1 ['B_6336']
@> GLU95 O_1559 A ASN97 N_1589 A 3.2168667986101007 1 ['B_6348']
@> HIS104 ND1_1711 A LYS162 NZ_2660 A 5.082646357951735 1 ['B_6624']
@> GLY107 N_1754 A ALA108 N_1761 A 3.190471281801484 1 ['B_6402']
@> ALA108 N_1761 A VAL155 O_2545 A 4.347411183681612 1 ['B_6420']
@> ASP109 N_1771 A ASP109 OD2_1782 A 3.906022145354529 1 ['B_6444']
@> LYS118 NZ_1933 A THR215 O_3508 A 5.582298899199147 1 ['B_6492']
@> LYS118 NZ_1933 A LEU213 O_3470 A 5.187238186164194 1 ['B_6504']
@> TYR129 OH_2121 A ASP313 O_5049 A 4.453922989904518 1 ['B_6537']
@> ARG133 O_2183 A PRO315 N_5071 A 6.180698423317547 1 ['B_7170']
@> ARG133 O_2183 A GLU314 O_5061 A 5.992423883538281 1 ['B_7170']
@> LYS136 NZ_2246 A PRO311 O_5022 A 5.667678978206158 1 ['B_7149']
@> SER140 O_2312 A ALA317 O_5106 A 4.792094531621847 1 ['B_7179']
@> ARG146 O_2400 A ARG186 NE_3047 A 4.181967240426447 1 ['B_6573']
@> ARG183 NH1_2981 A HIS225 N_3648 A 4.030232127309791 1 ['B_6666']
@> TRP184 O_2992 A LEU219 N_3562 A 6.271808750272923 1 ['B_6801']
@> TRP184 O_2992 A PHE220 N_3581 A 4.859018933900133 1 ['B_6801']
@> TYR185 N_3011 A ARG186 N_3032 A 2.704904619390487 1 ['B_6687']
@> ARG186 O_3037 A TRP204 NE1_3348 A 3.5636415364062675 1 ['B_6690']
@> GLU189 OE2_3094 A SER290 OG_4704 A 4.031646065814807 1 ['B_6732']
@> GLU189 OE2_3094 A ASN198 N_3243 A 5.818566919783599 1 ['B_6732']
@> ASN198 N_3243 A SER290 OG_4704 A 5.702313653246375 1 ['B_6732']
@> VAL201 O_3293 A SER205 OG_3368 A 3.4239025979136706 1 ['B_6699']
@> LEU219 N_3562 A PHE220 N_3581 A 2.904216589719166 1 ['B_6801']
@> THR223 O_3627 A ASP224 OD2_3647 A 3.765190433430952 2 ['B_6810', 'B_6804']
@> ARG234 NE_3826 A MET265 SD_4317 A 4.2630924221743065 1 ['B_6840']
@> THR238 O_3882 A MET262 O_4257 A 4.170953008606065 1 ['B_6966']
@> GLY240 N_3905 A LEU243 N_3937 A 4.966423864311221 1 ['B_6879']
@> VAL270 O_4385 A ILE272 N_4416 A 3.2146626572628105 1 ['B_6993']
@> LEU288 N_4664 A ASP289 N_4683 A 2.8610966428976172 1 ['B_7059']
@> SER290 N_4695 A SER290 OG_4704 A 2.9945961330369735 1 ['B_7068']
@> ASP291 O_4711 A ARG293 O_4745 A 4.600850356184167 1 ['B_7086']
@> ALA306 N_4941 A GLN307 N_4951 A 2.858702852693857 1 ['B_7131']
@> ALA306 N_4941 A GLN307 NE2_4965 A 5.641271310617846 1 ['B_7131']
@> ALA306 O_4946 A GLN307 O_4956 A 3.778994178349576 1 ['B_7146']
@> GLN307 N_4951 A GLN307 NE2_4965 A 3.8062708784320645 1 ['B_7131']
@> GLU314 N_5056 A GLU314 O_5061 A 3.3455619856759498 1 ['B_7152']
@> GLU314 O_5061 A PRO315 N_5071 A 2.248233306398604 1 ['B_7170']
@> SER323 OG_5196 A ARG327 NH2_5265 A 4.567763238172486 1 ['B_7248']
@> SER323 OG_5196 A ARG327 NH1_5262 A 4.2918016030566895 1 ['B_7248']
@> ARG327 NH1_5262 A ARG327 NH2_5265 A 2.2885257263137766 1 ['B_7248']
@> LEU329 N_5280 A TRP334 NE1_5377 A 3.839466890077318 1 ['B_7287']
@> ASP340 OD1_5485 A ASP340 OD2_5486 A 2.1913256261906864 1 ['B_7293']
@> 5896 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 23 water bridges detected using method chain.
@> GLU19 OE2_337 A ARG20 N_338 A 5.878384216772496 2 ['B_5680', 'B_5683']
@> GLN22 OE1_396 A ASN23 N_400 A 4.279627670720901 1 ['B_5689']
@> ARG64 NE_1031 A GLU68 OE1_1112 A 5.168174339164652 1 ['B_5698']
@> LEU72 O_1181 A LEU83 N_1361 A 3.8260938305274212 1 ['B_5707']
@> HIS77 N_1273 A VAL80 O_1324 A 4.745099893574423 1 ['B_5713']
@> ALA90 N_1475 A ASP98 N_1603 A 6.407529633173774 1 ['B_5722']
@> ALA90 N_1475 A ASN97 N_1589 A 6.004686836130592 1 ['B_5722']
@> ASN97 N_1589 A ASP98 N_1603 A 2.782156717368737 1 ['B_5722']
@> HIS139 ND1_2300 A ALA296 N_4797 A 5.051173527013304 1 ['B_5746']
@> LEU148 N_2431 A SER205 OG_3368 A 4.436225084460889 1 ['B_5755']
@> LEU148 N_2431 A ASP202 OD1_3314 A 4.948488860248146 1 ['B_5755']
@> TRP184 O_2992 A PHE220 N_3581 A 4.863268242653289 1 ['B_5773']
@> ALA187 N_3056 A SER205 OG_3368 A 5.024188292649868 1 ['B_5866']
@> ALA187 N_3056 A VAL201 O_3293 A 4.655099032244106 1 ['B_5866']
@> ASN198 OD1_3253 A SER290 OG_4704 A 3.8403593581851165 1 ['B_5782']
@> VAL201 O_3293 A SER205 OG_3368 A 3.1895346055498424 1 ['B_5866']
@> ASP202 OD1_3314 A SER205 OG_3368 A 4.915096031615251 1 ['B_5755']
@> TRP204 NE1_3348 A GLN228 NE2_3710 A 6.121850128841768 1 ['B_5776']
@> CYS208 O_3398 A LEU219 N_3562 A 4.934281203985034 1 ['B_5788']
@> VAL236 O_3859 A LEU288 N_4664 A 5.696742929077984 1 ['B_5809']
@> ALA301 O_4868 A HIS302 O_4878 A 3.24225816368777 1 ['B_5833']
@> GLN322 OE1_5183 A GLU325 OE2_5232 A 4.976302744005838 1 ['B_5836']
@> ASP328 OD1_5278 A LEU329 N_5280 A 3.8415040023407494 1 ['B_5839']

align__addH_1bl6.pdb

@> 6357 atoms and 1 coordinate set(s) were parsed in 0.06s.

align__addH_2zb1.pdb

@> 64 water bridges detected using method chain.
@> ARG2 NE_32 A ALA88 O_1462 A 4.772681217093805 1 ['B_5868']
@> ARG2 NE_32 A THR86 O_1434 A 5.221993776327199 1 ['B_5868']
@> ARG2 NH1_35 A PHE5 O_74 A 6.437162262985144 1 ['B_5835']
@> ARG2 NH1_35 A PHE5 N_69 A 4.8099134087839905 1 ['B_5835']
@> ARG2 NH1_35 A PRO3 O_45 A 3.668070882630269 1 ['B_6087']
@> ARG2 NH2_38 A GLU16 OE2_292 A 5.190005105970515 1 ['B_5970']
@> THR4 N_55 A GLU19 OE1_336 A 4.543989656678368 1 ['B_5769']
@> PHE5 N_69 A PHE5 O_74 A 2.7251291712504195 1 ['B_5835']
@> TYR6 OH_108 A ASN23 N_400 A 4.61897456152337 1 ['B_5976']
@> TYR6 OH_108 A LEU24 N_414 A 5.237830562360719 1 ['B_5976']
@> GLU19 N_323 A ARG20 N_338 A 2.8113743969809497 1 ['B_5805']
@> TYR21 OH_381 A ASP38 OD2_608 A 4.634256251007273 1 ['B_5793']
@> ASN23 N_400 A LEU24 N_414 A 2.911750504421695 1 ['B_5976']
@> LEU50 N_795 A VAL97 N_1597 A 4.7754868861719215 1 ['B_5694']
@> LYS61 NZ_994 A ASP324 OD1_5226 A 5.063379701345736 1 ['B_6114']
@> ARG68 NH1_1133 A ASP317 O_5111 A 3.6191801557811405 1 ['B_5685']
@> ARG68 NH1_1133 A PHE320 N_5146 A 5.8201812686547845 1 ['B_5685']
@> LYS71 NZ_1195 A SER340 OG_5494 A 3.351553222015128 1 ['B_5691']
@> LYS74 O_1238 A TYR135 OH_2251 A 3.3942828403066243 1 ['B_5856']
@> GLY80 N_1336 A HIS102 N_1683 A 5.906673598566283 1 ['B_5946']
@> VAL84 O_1398 A THR86 N_1429 A 4.836624959618022 2 ['B_5877', 'B_5898']
@> THR86 O_1434 A ALA88 O_1462 A 3.76180289223133 1 ['B_5868']
@> HIS102 ND1_1693 A LEU103 N_1700 A 4.864742028103854 1 ['B_5919']
@> HIS102 ND1_1693 A LEU103 O_1705 A 5.994732854765089 1 ['B_5919']
@> LEU103 N_1700 A LEU103 O_1705 A 2.783313313301253 1 ['B_5919']
@> ASP107 N_1753 A ASN110 OD1_1808 A 6.078192165438666 1 ['B_6234']
@> ASP107 OD1_1763 A ASN109 ND2_1795 A 4.464301625114504 1 ['B_6063']
@> ASN109 OD1_1794 A SER149 OG_2477 A 5.7476750952015365 1 ['B_5886']
@> ASN109 OD1_1794 A SER149 N_2468 A 4.792361943760091 1 ['B_5886']
@> LYS134 NZ_2228 A GLU310 OE1_5017 A 4.764214940575204 1 ['B_5706']
@> HIS137 ND1_2282 A ALA292 N_4745 A 5.275386241783629 1 ['B_5667']
@> ALA139 O_2305 A TYR316 N_5085 A 4.429644342382354 1 ['B_5820']
@> LEU146 N_2413 A SER201 OG_3316 A 4.344889641866642 1 ['B_5625']
@> LEU146 N_2413 A ASP198 OD1_3262 A 4.96568384011709 1 ['B_5625']
@> SER149 N_2468 A SER149 OG_2477 A 2.849899296466454 1 ['B_5886']
@> GLY174 N_2843 A ARG179 NH1_2929 A 3.6460610252709738 1 ['B_5640']
@> GLY174 N_2843 A HIS221 N_3596 A 5.748087855974372 1 ['B_5640']
@> ARG179 NH1_2929 A HIS221 N_3596 A 4.06007031958807 1 ['B_5640']
@> ALA183 N_3004 A SER201 OG_3316 A 4.966023157416805 1 ['B_5700']
@> ALA183 N_3004 A VAL197 O_3241 A 4.810734975032401 1 ['B_5700']
@> GLU185 OE2_3042 A ASN194 N_3191 A 5.811400347592652 1 ['B_5766']
@> GLU185 OE2_3042 A SER286 OG_4652 A 4.244290989081684 1 ['B_5766']
@> LEU188 O_3084 A SER245 OG_4006 A 5.444725337425206 1 ['B_5964']
@> ASN194 N_3191 A SER286 OG_4652 A 5.824057005215518 1 ['B_5766']
@> VAL197 O_3241 A SER201 OG_3316 A 3.369221423415205 1 ['B_5700']
@> ASP198 OD1_3262 A SER201 OG_3316 A 5.045291864699207 1 ['B_5625']
@> TRP200 NE1_3296 A GLN224 NE2_3658 A 6.015626151947945 1 ['B_5619']
@> ASP220 OD1_3594 A ASP220 OD2_3595 A 2.2032276323612154 1 ['B_5715']
@> ARG230 NH1_3777 A ARG230 NH2_3780 A 2.300537545879223 1 ['B_6021']
@> VAL232 O_3807 A GLY233 O_3824 A 3.3189909611205644 1 ['B_6048']
@> VAL232 O_3807 A GLY236 N_3853 A 7.877346507549355 2 ['B_6048', 'B_5649']
@> GLY233 O_3824 A GLY236 N_3853 A 6.694337084431885 2 ['B_6048', 'B_5649']
@> THR234 O_3830 A GLN257 OE1_4196 A 6.429353388949777 1 ['B_5907']
@> GLY236 N_3853 A LEU239 N_3885 A 4.738406799758755 1 ['B_5772']
@> LYS260 N_4231 A LYS260 O_4236 A 2.880827138167091 1 ['B_6081']
@> ASN262 N_4270 A ASN265 ND2_4325 A 5.6168396808169625 2 ['B_6201', 'B_6204']
@> ASP285 OD1_4641 A SER286 OG_4652 A 4.554287759902752 1 ['B_5742']
@> PRO307 O_4970 A GLU310 N_5004 A 4.266950198912568 1 ['B_5718']
@> TYR316 OH_5104 A GLN318 OE1_5131 A 4.835579075974251 1 ['B_5748']
@> ASP317 O_5111 A PHE320 N_5146 A 5.125772234502817 1 ['B_5685']
@> LEU325 N_5228 A TRP330 NE1_5325 A 3.5740752650161127 1 ['B_5679']
@> SER332 O_5363 A SER332 OG_5367 A 3.17139165036424 1 ['B_6015']
@> SER332 O_5363 A ASP336 OD1_5433 A 3.7261532442990046 1 ['B_6015']
@> SER332 OG_5367 A ASP336 OD1_5433 A 4.741911534392011 1 ['B_6015']
@> 5486 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 8 water bridges detected using method chain.
@> GLU87 N_1455 A GLU87 OE2_1469 A 4.688252766223255 1 ['B_5450']
@> GLU88 O_1475 A ASN90 N_1505 A 3.170165453095468 1 ['B_5423']
@> LEU141 N_2340 A SER186 OG_3095 A 4.557214170960148 1 ['B_5399']
@> TRP165 O_2719 A PHE201 N_3308 A 4.851787093432685 1 ['B_5417']
@> ALA249 N_4079 A GLU264 OE2_4312 A 5.04425475169524 1 ['B_5411']
@> ALA249 N_4079 A GLU264 OE1_4311 A 5.336986509257824 1 ['B_5411']
@> GLU264 OE1_4311 A GLU264 OE2_4312 A 2.1973031197356447 1 ['B_5411']
@> TYR301 O_4865 A GLN303 N_4893 A 3.291610092340829 1 ['B_5426']

..
..

align__addH_1ouk.pdb

@> 5854 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 32 water bridges detected using method chain.
@> TYR6 OH_108 A ASN23 N_400 A 4.615102490736257 1 ['B_5692']
@> GLU19 N_323 A ARG20 N_338 A 2.7776072436541495 1 ['B_5689']
@> LEU52 O_818 A VAL99 N_1615 A 5.222420415860832 1 ['B_5701']
@> LEU52 O_818 A ASP98 OD1_1613 A 4.411748292910645 1 ['B_5701']
@> GLY82 N_1354 A HIS104 ND1_1711 A 5.5944181109387925 1 ['B_5839']
@> GLY82 N_1354 A THR103 OG1_1695 A 3.721591326301155 1 ['B_5839']
@> GLY82 N_1354 A GLY82 O_1360 A 2.760033514289275 1 ['B_5839']
@> GLY82 O_1360 A HIS104 ND1_1711 A 5.042785044000984 1 ['B_5839']
@> GLY82 O_1360 A THR103 OG1_1695 A 2.4203406784996195 1 ['B_5839']
@> GLU95 O_1559 A ASN97 N_1589 A 3.19543080037731 1 ['B_5722']
@> ASP98 OD1_1613 A VAL99 N_1615 A 4.4735462443122245 1 ['B_5701']
@> THR103 OG1_1695 A HIS104 ND1_1711 A 4.038137441940282 1 ['B_5839']
@> TYR137 OH_2269 A ASP318 N_5111 A 5.034224965175872 1 ['B_5743']
@> HIS139 ND1_2300 A ALA296 N_4797 A 5.180029922693496 1 ['B_5746']
@> ARG146 O_2400 A SER205 OG_3368 A 5.5475649613141105 1 ['B_5749']
@> ASP147 N_2419 A LEU148 N_2431 A 2.918157466621704 1 ['B_5752']
@> ASP147 N_2419 A SER205 OG_3368 A 5.836999657358222 1 ['B_5752']
@> ASP147 N_2419 A ASP202 OD1_3314 A 4.6087231420427015 1 ['B_5752']
@> ASP147 OD2_2430 A LYS149 NZ_2468 A 4.241426764663042 1 ['B_5758']
@> LEU148 N_2431 A SER205 OG_3368 A 4.471605304585819 1 ['B_5752']
@> LEU148 N_2431 A ASP202 OD1_3314 A 4.979517446500211 1 ['B_5752']
@> ASP165 N_2702 A ASP165 OD2_2713 A 3.3099394254276047 1 ['B_5764']
@> TRP184 O_2992 A GLN228 NE2_3710 A 3.6392044460293795 1 ['B_5773']
@> TRP184 O_2992 A TRP204 NE1_3348 A 5.492021940961269 1 ['B_5773']
@> TRP184 O_2992 A ARG186 O_3037 A 5.064636907025022 1 ['B_5773']
@> ARG186 O_3037 A GLN228 NE2_3710 A 4.172959381542071 1 ['B_5773']
@> ARG186 O_3037 A TRP204 NE1_3348 A 3.614997372059904 1 ['B_5773']
@> ASP202 OD1_3314 A SER205 OG_3368 A 4.973505604701778 1 ['B_5752']
@> TRP204 NE1_3348 A GLN228 NE2_3710 A 6.1119838023345565 1 ['B_5773']
@> CYS208 O_3398 A LEU219 N_3562 A 4.95488466061522 1 ['B_5782']
@> LEU219 N_3562 A PHE220 N_3581 A 2.9722883440204777 1 ['B_5776']
@> ALA301 O_4868 A HIS309 ND1_4999 A 4.853743503729881 1 ['B_5824']

align__addH_1bl7.pdb

@> 6020 atoms and 1 coordinate set(s) were parsed in 0.05s.

align__addH_1lez.pdb

@> 16 water bridges detected using method chain.
@> ARG2 NH2_38 A PHE5 N_69 A 4.571194592226415 1 ['C_5912']
@> TYR6 OH_108 A ASN23 N_400 A 4.50858436762583 1 ['C_5969']
@> GLU19 N_323 A ARG20 N_338 A 2.772673980113782 1 ['C_5762']
@> SER58 OG_925 A ILE59 N_927 A 3.3482071321828335 1 ['C_5972']
@> ARG70 NH1_1149 A PHE317 N_5100 A 5.769974090063145 1 ['C_5693']
@> LYS73 NZ_1211 A SER337 OG_5448 A 3.0176429543602397 1 ['C_5978']
@> THR88 OG1_1453 A ASP98 N_1601 A 4.317795386536977 1 ['C_5732']
@> THR88 OG1_1453 A ASN97 N_1587 A 4.64242630528477 1 ['C_5732']
@> ASN97 N_1587 A ASP98 N_1601 A 2.8289607632485843 1 ['C_5732']
@> ASP98 OD2_1612 A TYR100 OH_1648 A 5.2055084285783275 1 ['C_5729']
@> ASP147 N_2416 A LEU148 N_2428 A 2.8996084218390594 1 ['C_5705']
@> ASP147 N_2416 A ASP195 OD1_3220 A 4.452267849085452 1 ['C_5705']
@> LEU148 N_2428 A ASP195 OD1_3220 A 4.954386541237979 1 ['C_5705']
@> TRP177 O_2898 A PHE213 N_3487 A 4.693564210703845 1 ['C_5765']
@> VAL194 O_3199 A SER198 OG_3274 A 3.5576895592504996 1 ['C_5717']
@> TRP197 NE1_3254 A GLN221 NE2_3616 A 6.029092883676613 1 ['C_5702']
```

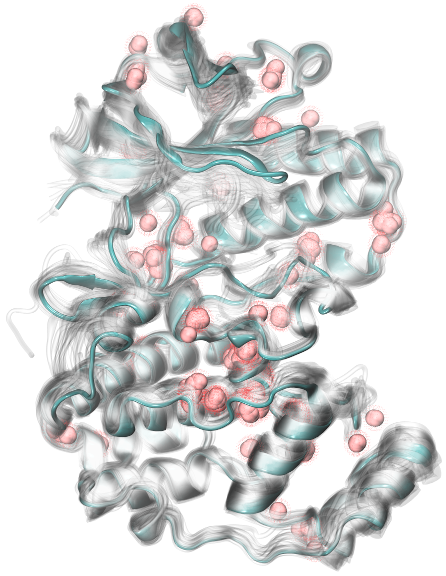
6.5 Finding clusters of water within homologous structures

Once the PDB files with selected water bridges are saved, we can start checking water clustering using `findClusterCenters()` function. With this kind of analysis, we should check the names of oxygens that are forming water molecules. If the name is different from the default one, we should set a new `selection` parameter. In this case we will use the following selection: `'resname HOH and name O'`. We will use default criteria of `distC` and `numC`, which are set to 0.3 and 3, respectively.

```
In [19]: findClusterCenters('wb_*.pdb', selection = 'resname HOH and name O')
```

```
@> 5730 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5712 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5703 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5724 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5661 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5381 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5726 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5655 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5700 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5688 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5762 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5645 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5622 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5735 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5780 atoms and 1 coordinate set(s) were parsed in 0.05s.
..
..
@> 5470 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5773 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5781 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5399 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5311 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5408 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5419 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5475 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5398 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5462 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5657 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> Results are saved in clusters_wb_.pdb.
```

The function will create a file called `clusters_wb_.pdb` which will contain water clusters. We can upload this file to any graphical visualization program (in this tutorial we used **VMD_**) and display water clusters. Additionally, we should upload the protein structure we analyzed to see where water clusters are localized with respect to the protein structure.

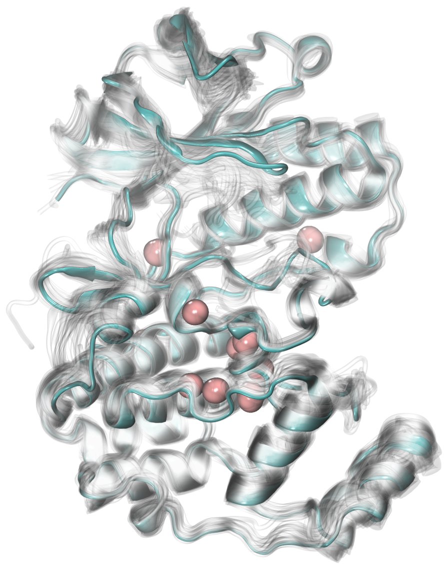


If we would like to use more restricted criteria to see more conserved water molecules across different protein structures, we can change the default parameters for `distC` and `numC`. In the example below, we will use `distC=0.2` and `numC=5`. It means that we are looking for at least 5 water molecules among our set of data that are localized with 0.2 Angstrom from each other.

```
In [20]: findClusterCenters('wb_*.pdb', selection = 'resname HOH and name O',
   ....:                                                  distC=0.2, numC=5)
   ....:
```

```
@> 5730 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5712 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5703 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5724 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5661 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5381 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5726 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5655 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5700 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5688 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5762 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5645 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5622 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5735 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5780 atoms and 1 coordinate set(s) were parsed in 0.05s.
..
..
@> 5470 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5773 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5781 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5399 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5311 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5408 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5419 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5475 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5398 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5462 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5657 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> Results are saved in clusters_wb_.pdb.
```

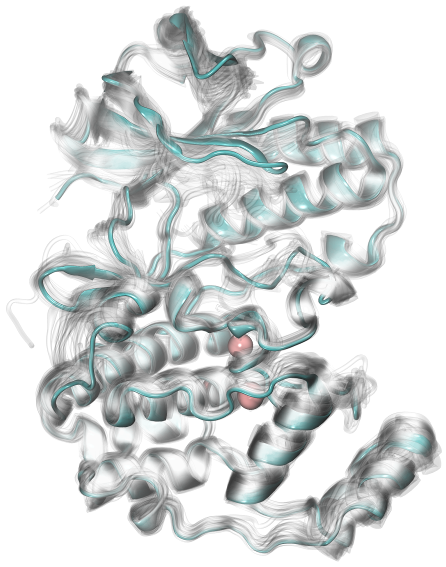
After displaying new results in the visualization program, we can see a smaller number of water clusters and only those that were more preoccupied if we compare it with the previous figure.



We can increase the number of molecules `numC` to 10 to see which places are occupied more often by water bridges. Now, we will see only two: the most significantly preoccupied water positions across the heterogeneous structures of p38 MAP kinase.

```
In [21]: findClusterCenters('wb_*.pdb', selection = 'resname HOH and name O',
   ....:                                                  distC=0.2, numC=10)
   ....:
```

```
@> 5730 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5712 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5703 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5724 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5661 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5381 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5726 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5655 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5700 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5688 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5762 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5645 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5622 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5735 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5780 atoms and 1 coordinate set(s) were parsed in 0.05s.
..
..
@> 5470 atoms and 1 coordinate set(s) were parsed in 0.06s.
@> 5773 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5781 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5399 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5311 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5408 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5419 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5475 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5398 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5462 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> 5657 atoms and 1 coordinate set(s) were parsed in 0.05s.
@> Results are saved in clusters_wb_.pdb.
```



CHAPTER

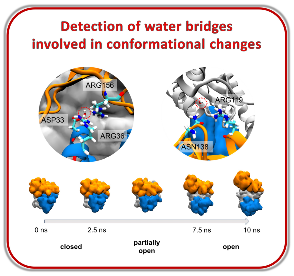
SEVEN

ADDITIONAL EXAMPLES OF WATFINDER UTILITY

Here, we present several more case studies on how to use WatFinder.

7.1 Case study 1

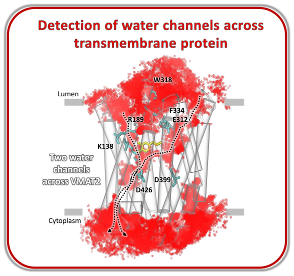
Detection and quantification of the formation of inter-residue water bridges as adenylate kinase transitions from the closed to open state



IPython notebook (ipynb):

7.2 Case study 2

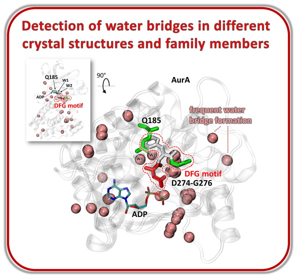
Identification of water influx and clusters into the vesicular monoamine transporter VMAT2


Detection of water channels across
transmembrane protein
W318
Lumen
F334
R189
E312
K138
Two water channels across VMAT2
D399
D426
Cytoplasm

IPython notebook (ipynb):

7.3 Case study 3

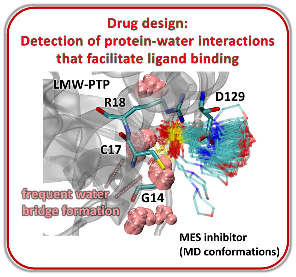
Identification of key protein-water interactions observed across various Aurora kinase A crystal structures identified by BLAST


Detection of water bridges in different crystal structures and family members
AurA
Q185
frequent water bridge formation
DFG motif
D274-G276
ADP

IPython notebook (ipynb):

7.4 Case study 4

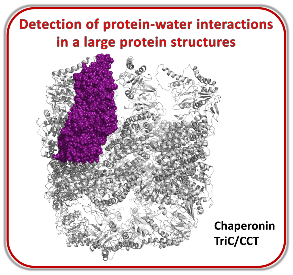
Derection of protein-water interactions that facilitate ligand binding


Drug design:
Detection of protein-water interactions that facilitate ligand binding
LMW-PTP
R18
D129
C17
frequent water bridge formation
G14
MES inhibitor (MD conformations)

IPython notebooks (ipynb): (protein only) and (protein+ligand)

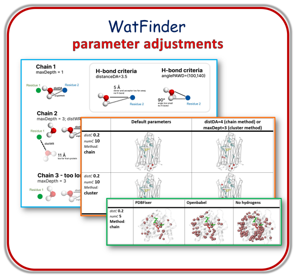
7.5 Case study 5

Water bridges detection in a large chaperonin TRiC/CCT structure including timing for selected regions


Detection of protein-water interactions in a large protein structures
Chaperonin TriC/CCT

IPython notebook (ipynb):

7.6 Example 6: WatFinder Parameter Adjustments



IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

IPython notebook (ipynb):

The description of all cases are available in the supplementary file of WatFinder paper *[JK24]*. Trajectories are in tutorial's files.

CHAPTER

EIGHT

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`

The above criteria are used to find hydrogen bonds in a PDB structure which are used to predict possible water bridges.

The angles are calculated so that the hydrogen atom that makes the hydrogen bond is the vertex of the angle, where the donor and acceptor atoms form the two ends. The angle is measured between the line connecting the donor to the hydrogen and the line connecting the hydrogen to the acceptor.

Most PDB structures do not have hydrogen atoms, therefore we can ignore this criterium. Hydrogen bonds will be calculated just based on distance and atom types.

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=3
```

```
@> 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 4.46373442310359
@> 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 4.46373442310359
@> GLN105 NE2_800 A LYS102 O_772 A LYS79 NZ_582 A 4.2363221076778395 3.8752885053889865 4.92922123666
@> 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 4.92922123666
@> 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/

[JK24] Krieger JM, Doljanin F, Bogetti AT, Zhang F, Manivarma T, Bahar I, Mikulska-Ruminska K. WatFinder: A ProDy tool for protein-water interactions. *Bioinformatics* **2024** btae516.