Protein Function Discovery
and Department of Biomedical and Molecular Sciences
Molecular Modelling and Crystallographic Computing Facility

Crystallography and Modelling:

- Home
- Software List
- Crystallography Links
- Calculations

Other:

- Software
- Links
- Rob's Home

Introduction to structural alignment with PyMOL

Using the "align" command

With two structures (hereafter referred to as **structure1** and **structure2**) loaded into ***PyMOL*** it is a simple matter to type the command:

```
align structure2, structure1
```

and ***PyMOL*** will first do a sequence alignment and then try to align the structures to minimize the **RMSD** (Root Mean Square Deviation: see footnote 1) between the aligned residues. This often works very well for homologous structures, but if you have to get the **RMSD** for the backbone atoms of a particular set of non-homologous residues, this can be difficult. You may need to specify the particular residues to match. For example, you may know that only part of **structure1** should match to part of **structure2**. In this case, you may wish to use a command like:

```
align structure2 and resi 1-100, structure1 and resi 300-400
```

or in short form:

```
align structure2 & i. 1-100, structure 1 & i. 300-400
```

Furthermore, you may wish to restrict the alignment to just the backbone atoms, so you can say:

```
align structure2 and resi 1-100 and name n+ca+c+o, structure1 and resi 300-400 and name n+ca+c+o
```

or in short form:

```
align structure2 & i. 1-100 & n. n+ca+c+o, structure1 & i. 300-400 & n. n+ca+c+o
```

When the align command runs, it will print out some information like:

```
PyMOL>align structure1 & n. ca, structure2 & n. ca
 Match: read scoring matrix.
 Match: assigning 349 x 66 pairwise scores.
 MatchAlign: aligning residues (349 vs 66)...
 ExecutiveAlign: 47 atoms aligned.
 Executive: RMS =   12.490 (47 to 47 atoms)
```

or if you specify a set of atoms to use (you can specify an upper limit beyond the end of the chain so that you do not have to figure out the last residue number):

```
PyMOL>align structure1 & i. 188-500 & n. ca, structure2 & n. ca
 Match: read scoring matrix.
 Match: assigning 63 x 66 pairwise scores.
 MatchAlign: aligning residues (63 vs 66)...
 ExecutiveAlign: 7 atoms aligned.
 Executive: RMS =    0.353 (7 to 7 atoms)
```

Note that one can specify a cutoff for atoms to be excluded from the alignment calculation:

```
PyMOL>align structure1 & n. ca & i. 288-500, structure2 & n. ca, cutoff=1
 Match: read scoring matrix.
 Match: assigning 63 x 66 pairwise scores.
 MatchAlign: aligning residues (63 vs 66)...
 ExecutiveAlign: 7 atoms aligned.
 ExecutiveRMS: 3 atoms rejected during cycle 1 (RMS=0.35).
 ExecutiveRMS: 2 atoms rejected during cycle 2 (RMS=0.18).
 Executive: RMS =    0.038 (2 to 2 atoms)
```

*Of course an **RMSD** calculation with only 2 atoms is pretty meaningless.*

One can also add the "object" parameter to have a set of lines drawn that connect the atoms that have been matched. Try something like:

```
align structure2 & i. 1-100 & n. n+ca+c+o, structure1 & i. 300-400 & n. n+ca+c+o, object=matches
```

and examine the new object called "matches".

When "align" doesn't do the job

In the case in which you have relatively little homology, the first thing you should do is try the **cealign** command that is now built into PyMOL. For more information see the command examples at http://pymolwiki.org/index.php/Cealign.

In the event that you wish to calculate an **RMSD** for the backbone atoms of even the non-homologous residues the **align** and **cealign**commands may allow you to get a structural alignment, but the **RMSD** value that it calculates only applies to the homologous residues. The **fit** command will also not work, because it expects to have a set of residues that have the same residue name, residue number, chain name and atom name to match. There are a few alternatives to get around this problem.

First the hard way

One alternative is to create a **.pml** script file that uses the **pair_fit** command:

```
pair_fit \
structure2 & i. 196 & n. ca,  structure1 & i. 8 & n. ca, \
structure2 & i. 197 & n. ca,  structure1 & i. 9 & n. ca, \
structure2 & i. 198 & n. ca,  structure1 & i. 10 & n. ca, \
structure2 & i. 199 & n. ca,  structure1 & i. 11 & n. ca, \
structure2 & i. 200 & n. ca,  structure1 & i. 12 & n. ca, \
structure2 & i. 201 & n. ca,  structure1 & i. 13 & n. ca, \
structure2 & i. 202 & n. ca,  structure1 & i. 14 & n. ca, \
structure2 & i. 203 & n. ca,  structure1 & i. 15 & n. ca, \
structure2 & i. 204 & n. ca,  structure1 & i. 16 & n. ca, \
structure2 & i. 205 & n. ca,  structure1 & i. 17 & n. ca, \
structure2 & i. 206 & n. ca,  structure1 & i. 18 & n. ca, \
structure2 & i. 207 & n. ca,  structure1 & i. 19 & n. ca, \
structure2 & i. 208 & n. ca,  structure1 & i. 20 & n. ca, \
```

If these commands are stored in a macro file called **fitting.pml** (you can of course choose any name you like) then they are called with the "@" (at) sign like this:

```
@fitting.pml
```

To include all backbone atoms (n, ca, c & o) you would need to expand the list of atom pairs like this:

```
pair_fit \
structure2 & i. 196 & n. n,  structure1 & i. 8 & n. n, \
structure2 & i. 196 & n. ca,  structure1 & i. 8 & n. ca, \
structure2 & i. 196 & n. c,  structure1 & i. 8 & n. c, \
structure2 & i. 196 & n. o,  structure1 & i. 8 & n. o, \
structure2 & i. 197 & n. n,  structure1 & i. 9 & n. n, \
structure2 & i. 197 & n. ca,  structure1 & i. 9 & n. ca, \
structure2 & i. 197 & n. c,  structure1 & i. 9 & n. c, \
structure2 & i. 197 & n. o,  structure1 & i. 9 & n. o, \
structure2 & i. 198 & n. n,  structure1 & i. 10 & n. n, \
structure2 & i. 198 & n. ca,  structure1 & i. 10 & n. ca, \
structure2 & i. 198 & n. c,  structure1 & i. 10 & n. c, \
structure2 & i. 198 & n. o,  structure1 & i. 10 & n. o, \
etc.
```

An example of this is here. This can be rather time consuming to type out and therefore prone to error.

The not quite so hard way

The second alternative is to change the residue names of both structures to "GLY", change the chain names of both structures to match and alter the residue numbers to match. Before doing this, you may wish to create a copy of your molecules to work on. You can do this with the **create** command like this:

```
create newstruct, structure1
```

where **newstruct** will be the new object name.

The modification of residue names, chain names and residue numbers can be done with the **alter** command. In the example above, the residue number **196** in **structure2** corresponds with the residue number **8** for **structure1**, so we need to subtract 188 from the residue numbers in **structure2** to match **structure1**. For example:

```
alter structure1, resn='GLY'
alter structure1, chain=''
alter structure1, resi=str(int(resi)-188)
```

This will change all residue names to glycine, change the chain identifier to a blank and the third line will renumber the residue numbers. The operation **str(int(resi)-188)**, means to take the residue number (which is stored as a character string), convert it to an integer, substract 188 and reconvert it to a string. Don't worry, it works! You'll just need to change the object name (here **structure1**) and the number to subtract (**188**) to whatever is appropriate for your situation.

Similarly, you may need to alter the other structure:

```
alter structure2, resn='GLY'
alter structure2, chain=''
```

In this case the original residue numbering can be used. Once this is done, you can use the **fit** command to fit **structure2** onto

structure1 and calculate the **RMSD**:

```
fit structure2 & resi 8-35, structure1 & resi 8-35
```

to calculate the **RMSD** for the backbone atoms for residues 8 to 35 of the two structures.

A situation in which this does not work would be one in which the residue numbering gets out of sync between the two structures due to an insertion. In this case, one would need to resort to the **pair_fit** command with a list of matched atoms, or be creative with the residue re-numbering.

Now the easy way!

Go to http://pldserver1.biochem.queensu.ca/~rlc/work/pymol/ and download the python script **fitting.py**. Install it by putting it in your ***PyMOL*** directory (if using Windows) or somewhere else (if using **Linux**, which you should be!) and issue the command:

```
run fitting.py
```

Follow the instructions (type "help fitting" to see them). For example:

```
fitting 1xuu, c. a & (i. 296-309 or i. 335-340), 1ame, i. 8-21 or i. 47-52
```

and you should get the **RMSD** value that you really want!

Footnotes

- **RMSD**: Root Mean Square Deviation is the square root of the mean of the square of the distances between the matched atoms.

  $$RMSD = SQRT[\{SUM(d_{ii})^2\}/N]$$

  where d_{ii} is the distance between the i^{th} atom of structure 1 and the i^{th} atom of structure 2 and N is the number of atoms matched in each structure.

Back to text