Least-Squares Superpositions and Root-Mean-Squares Deviation

A least-squares superposition of two or more molecules is a process by which you determine the translation vector and rotation matrix that best aligns the atoms of one molecule to the corresponding atoms of the second molecule in such a way that the sum of the squares of the distances between corresponding atoms is minimized. The root-mean-squares deviation is a quantitative measure of that fit.

The whole process can be split into three steps:

1 - Correctly pairing up the atoms in the two molecules

2 - Determining the translation vector and rotation matrix and calculating rmsd

3 - (Iteratively improving the pairing up of atoms, removing highly variable positions)

Step 1 is trivial if you are comparing different conformers of the same molecule, e.g. snapshots of a molecular dynamics trajectory, different docking solutions for the same pair of molecules or the different conformers that satisfy the distance constraints of an NMR structures, as the different conformers contain exactly the same number of atoms, in the same order, carrying the same chain - , residue - and atom labels.

The process becomes more difficult when you compare different homologs of a protein: as the different molecules differ in sequence, you have different numbers of atoms in the side chains, and may even have insertions and deletions in the chains. You will probably restrict your least-squares-fit to the backbone atoms, or even just to the C-alpha atoms.

The different methods available in PyMOL differ in how they determine this pairing-up of atoms: **Fit** requires corresponding atoms to carry the same label, **Pair-fit** requires the user to tell the program exactly which atoms correspond, **Align** first aligns the sequences to determine which residue in one molecule correspond to which in the second, **Super** and **CEalign** perform a sequence-independent structural alignment.

Of course, this strongly affects the rmsd values you get from your least-squares superposition, therefore rmsd values can only be compared if the were obtained in the same way, and if you cite rmsd values, you have to indicate **exactly** which atoms you included in the comparison: all atoms, backbone only or C-alpha atoms, or only the Calpha atoms of a specific subset of residues? Without this information, an rmsd value is meaningless.

If you want to assess the structural flexibility of a molecule in MD or NMR, or compare docking solutions, you want to suppress step 3, as you want to highlight the differences between the to molecules. On the other hand, if you want to highlight the structural similarity between two distantly related molecules in a graphical superpositions, you may restrict the superposition to the conserved core structure and eliminate flexible loops from the lest-squares superposition, either explicitly by only entering a selection of subset of all atoms into the least-squares fit, or by allowing for several cycles of refinement in which all atom pairs for which the distance exceeds a preset threshold are eliminated from the superposition

Fit

https://pymolwiki.org/index.php/Fit

Fit superimposes the model in the first selection on to the model in the second selection. Only matching atoms in both selections will be used for the fit.

```
fit mobile, target [, mobile_state [, target_state [, quiet [, matchmaker [,
cutoff [, cycles [, object ]]]]]]]
```[1]

mobile = string: atom selection
target = string: atom selection
mobile_state = integer: object state {default=0, all states)
target_state = integer: object state {default=0, all states)
matchmaker = integer: how to match atom pairs {default: 0}
-1: assume that atoms are stored in the identical order
0/1: match based on all atom identifiers (segi,chain,resn,resi,name,alt)
2: match based on ID
3: match based on rank
4: match based on index (same as -1 ?)
cutoff = float: outlier rejection cutoff (only if cycles>0) {default: 2.0}
cycles = integer: number of cycles in outlier rejection refinement {default: 0}
object = string: name of alignment object to create {default: None}

Pair_fit

https://pymolwiki.org/index.php/Pair_fit

Pair_Fit fits a set of atom pairs between two models. Each atom in each pair must be specified individually, which can be tedious to enter manually. So long as the atoms are stored in PyMOL with the same order internally, you can provide just two selections. Otherwise, you may need to specify each pair of atoms separately, two by two, as additional arguments to pair_fit. Script files are recommended when using this command.

The Pair-fit wizard allows to select graphically which atom to fit to which

```
pair_fit (selection), (selection), [ (selection), (selection) [ ...] ]
```

Align

https://pymolwiki.org/index.php/Align

Align performs a **sequence alignment followed by a structural superposition**, and then carries out zero or more cycles of refinement in order to reject structural outliers found during the fit. align does a good job on proteins with decent sequence similarity (identity >30%). For comparing proteins with lower sequence identity, the super and cealign commands perform better.

```
align mobile, target [, cutoff [, cycles [, gap [, extend [, max_gap [, object
[, matrix [, mobile_state [, target_state [, quiet [, max_skip [, transform [,
reset ]]]]]]]]]]]]
```

---

[1] square brackets denote parameters that are not strictly required, but need only be used if you want them to deviate from the default value

mobile = string: atom selection of mobile object
target = string: atom selection of target object
cutoff = float: outlier rejection cutoff in RMS {default: 2.0}
cycles = int: maximum number of outlier rejection cycles {default: 5}
gap, extend, max_gap: sequence alignment parameters
object = string: name of alignment object to create {default: (no alignment object)}
matrix = string: file name of substitution matrix for sequence alignment {default: BLOSUM62}
mobile_state = int: object state of mobile selection {default: 0 = all states}
target_state = int: object state of target selection {default: 0 = all states}
quiet = 0/1: suppress output {default: 0 in command mode, 1 in API}
max_skip = ?
transform = 0/1: do superposition {default: 1}
reset = ?

The RMSD of the aligned atoms (after outlier rejection!) is reported in the text output. The all-atom RMSD can be obtained by setting cycles=0 and thus not doing any outlier rejection. The RMSD can also be captured with a python script, see the API paragraph below. Note that the output prints "RMS" but it is in fact "RMSD" and the units are Angstroms.

Super

https://pymolwiki.org/index.php/Super

Super aligns two selections. It does a **sequence-independent** (unlike align) **structure-based dynamic programming alignment** followed by a series of refinement cycles intended to improve the fit by eliminating pairing with high relative variability (just like align). super is more robust than align for proteins with low sequence similarity.

"super" performs a residue-based pairwise alignment followed by a structural superposition, and then carries out zero or more cycles of refinement in order to reject outliers.

```
super mobile, target [, object=name ]
```

Alternative Conformations: If super ever tells you no matched atoms, then instead of

super p1, p2

try

super p1 & alt A+'', p2 & alt B+''

Cealign

https://pymolwiki.org/index.php/Cealign

Cealign aligns two proteins using the CE algorithm. It is very robust for proteins with little to no sequence similarity (twilight zone). For proteins with decent structural similarity, the super command is preferred and with decent sequence similarity, the align command is preferred, because these commands are much faster than cealign.

```
cealign target, mobile [, target_state [, mobile_state [, quiet [, guide [, d0
[, d1 [, window [, gap_max [, transform [, object ]]]]]]]]]]
```

Note: The mobile and target arguments are swapped with respect to the align and super commands.

target = string: atom selection of target object (CE uses only CA atoms)
mobile = string: atom selection of mobile object (CE uses only CA atoms)
target_state = int: object state of target selection {default: 1}
mobile_state = int: object state of mobile selection {default: 1}
quiet = 0/1: suppress output {default: 0 in command mode, 1 in API}
guide = 0/1: only use "guide" atoms (CA, C4') {default: 1}
d0, d1, window, gap_max: CE algorithm parameters
transform = 0/1: do superposition {default: 1}
object = string: name of alignment object to create {default: (no alignment object)}

Rms

https://pymolwiki.org/index.php/Rms

Rms computes a RMS fit between two atom selections, but does not transform the models after performing the fit. To determine the RMS without any fitting, see Rms_Cur.
Fit, Rms, Rms_Cur are finicky and only work when all atom identifiers match: segi, chain, resn, name, alt. If they don't then you'll need to use the alter command to change the identifiers to that they do -- typically that means clearing out the SEGI field, renaming chains, and sometimes renumbering.

```
rms (selection), (target-selection)
```

Rms_cur

https://pymolwiki.org/index.php/Rms_cur

Rms_cur computes the RMS difference between two atom selections without performing any fitting.
By default, only matching atoms in both selections will be used for the fit (same chain, residue number, atoms names etc.). Alternate location mess up the match!

```
rms_cur mobile, target [, mobile_state [, target_state [, quiet [, matchmaker [,
cutoff [, cycles [, object ]]]]]]]
```

Intra_fit

https://pymolwiki.org/index.php/Intra_fit

Intra_fit fits all states of an object to an atom selection in the specified state. It returns the rms values to python as an array.
intra_fit (selection),state

Intra_rms

https://pymolwiki.org/index.php/Intra_rms

Intra_rms calculates rms fit values for all states of an object over an atom selection relative to the indicated state. Coordinates are left unchanged. The rms values are returned as a python array.

Intra_rms_cur

https://pymolwiki.org/index.php/Intra_rms_cur

Intra_rms_cur calculates rms values for all states of an object over an atom selection relative to the indicated state without performing any fitting. The rms values are returned as a python array.

Extra_fit

https://pymolwiki.org/index.php/Extra_fit

extra_fit aligns multiple objects to a reference object.

```
extra_fit [ selection [, reference [, method ]]]
```

It can use any of PyMOL's pairwise alignment methods (align, super, cealign, fit...). More precisely it can use any function which takes arguments **mobile** and **target**, so it will for example also work with tmalign. Additional keyword arguments are passed to the used method, so you can for example adjust outlier cutoff or create an alignment object.
There are several similar commands/scripts for this job, like "A > align > all to this" from the PyMOL panel, the "alignto" command, align_all.py and super_all.py scripts from Robert Campbell.

align_all.py and super_all.py

Color_by_conservation https://pymolwiki.org/index.php/Color_by_conservation

Get_raw_alignment https://pymolwiki.org/index.php/Get_raw_alignment

set_raw_alignment
https://pymolwiki.org/index.php/Set_raw_alignment

set_raw_alignment is an API-only function to create an alignment object from lists of atom indices.

The following require additional installation in PyMOL:

Mcsalign (psico)

https://pymolwiki.org/index.php/Mcsalign

mcsalign aligns two small-molecule selections based on Maximum-Common-Substructure.

TMalign

https://pymolwiki.org/index.php/TMalign
https://zhanglab.ccmb.med.umich.edu/TM-align/

tmalign is a python module (or script) that provides wrappers to TMalign, TMscore and MMalign. The executables can be downloaded from http://zhanglab.ccmb.med.umich.edu/TM-align/ and should be saved to any directory in PATH.
The module also provides the command **alignwithanymethod** which is useful to quickly test different alignment methods (with their respective default values).

Theseus

https://pymolwiki.org/index.php/Theseus
https://theobald.brandeis.edu/theseus/

theseus is a wrapper for the theseus program for maximum likelihood superpositioning of macromolecular structures. It produces results very similar to xfit.

LigAlign

https://pymolwiki.org/index.php/LigAlign
http://compbio.cs.toronto.edu/ligalign/

LigAlign is a tool to compare protein active-sites and investigate ligand binding. The active-site alignment is guided by the orientation of bound ligands in the protein active sites. LigAlign supports analysis of flexible ligand via automatic fragment-based alignment: first computing a natural fragmentation of the query ligand, aligning each fragment of the query independently against the baseline, and then permitting easy visualization of each active site subcavity.

We use protein-ligand complexes to compare the active sites of several proteins which interact with a chosen ligand. Beginning with a user-specified protein-ligand structure, LigAlign gathers experimental structures of other proteins bound to the ligand from the Protein Data Bank. The tool then aligns the ligands bound in the structures to minimize the ligand-to-ligand RMSD. This transformation also aligns the active sites. Finally, the user can examine the aligned active sites to identify structural patterns, such as conserved steric hindrance or hydrophobicity.

However, a flexible ligand can bend itself into different active sites, where the active site subcavities have different relative positions or orientations. Therefore, when comparing two active sites, a rigid RMSD-minimizing transform on docked flexible ligands may fail to align the correct portions of the active site. However, each subcavity should still exhibit chemical or geometric complementarity to the piece of the ligand which it binds. Please see the website for more information.

LigAlign simplifies a number of protein analysis tasks. For example, LigAlign will align similar but distinct ligands which, in the context of structure-based drug discovery, permits the comparison of the docking of different ligands. Alternatively, if the user only specifies one protein-ligand complex, LigAlign will find chemically similar ligands automatically via the Protein-Small Molecule Database. Finally, LigAlign improves workflow by automatically fetching necessary data from the Protein Data Bank.

LigAlign is contributed by Abraham Heifets and Ryan Lilien at the University of Toronto. The

Example: least-squares superposition of two scFv fragments with near-identical sequence, but a marked difference in the CDR-H3 conformation:

```
fetch 2a9n
fetch 2a9m

remove /2a9n//M+I
remove /2a9m//M+I
remove solvent

hide all
show cartoon

color lightpink, /2a9n//L
color magenta, /2a9m//L
color lightblue, /2a9n//H
color slate, /2a9m//H

util.cbaw /2a9n/E/H/ORE
show spheres, /2a9n/E/H/ORE
```

The “align” menu item in the graphical user interface does a Calpha-atoms only superposition:

```
Match: read scoring matrix.
Match: assigning 236 x 236 pairwise scores.
MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 1226.000
ExecutiveAlign: 236 atoms aligned.
ExecutiveRMS: 9 atoms rejected during cycle 1 (RMSD=1.58).
ExecutiveRMS: 6 atoms rejected during cycle 2 (RMSD=0.84).
ExecutiveRMS: 4 atoms rejected during cycle 3 (RMSD=0.79).
Executive: RMSD =    0.770 (214 to 214 atoms)
Executive: object "aln_2a9m_to_2a9n" created.
```

```
fit 2a9n, 2a9m
        Executive: RMSD =    1.880 (1730 to 1730 atoms)

fit (/2a9n///N+CA+C+O), (/2a9m///N+CA+C+O)
        ExecutiveRMS-Error: No atoms selected.

fit (2a9n and backbone), (2a9m and backbone)
        Executive: RMSD =    1.560 (928 to 928 atoms)

fit (2a9n and name CA), (2a9m and name CA)
        Executive: RMSD =    1.573 (232 to 232 atoms)

fit (2a9n and name CA), (2a9m and name CA), object=“fit_CA”


align 2a9n, 2a9m
        Match: read scoring matrix.
        Match: assigning 237 x 395 pairwise scores.
```

MatchAlign: aligning residues (237 vs 395)...
MatchAlign: score 1226.000
ExecutiveAlign: 1738 atoms aligned.
ExecutiveRMS: 74 atoms rejected during cycle 1 (RMSD=1.89).
ExecutiveRMS: 75 atoms rejected during cycle 2 (RMSD=1.07).
ExecutiveRMS: 42 atoms rejected during cycle 3 (RMSD=0.89).
ExecutiveRMS: 27 atoms rejected during cycle 4 (RMSD=0.84).
ExecutiveRMS: 19 atoms rejected during cycle 5 (RMSD=0.82).
Executive: **RMSD = 0.801 (1498 to 1498 atoms)**

align 2a9n, 2a9m, cycles=0
align 2a9n, 2a9m, cycles=0
Match: read scoring matrix.
Match: assigning 237 x 395 pairwise scores.
MatchAlign: aligning residues (237 vs 395)...
MatchAlign: score 1226.000
ExecutiveAlign: 1738 atoms aligned.
Executive: **RMSD = 1.889 (1735 to 1735 atoms)**

align (2a9n and backbone), (2a9m and backbone)
align (2a9n and backbone), (2a9m and backbone)
Match: read scoring matrix.
Match: assigning 236 x 236 pairwise scores.
MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 1226.000
ExecutiveAlign: 935 atoms aligned.
ExecutiveRMS: 36 atoms rejected during cycle 1 (RMSD=1.58).
ExecutiveRMS: 26 atoms rejected during cycle 2 (RMSD=0.88).
ExecutiveRMS: 14 atoms rejected during cycle 3 (RMSD=0.81).
ExecutiveRMS: 6 atoms rejected during cycle 4 (RMSD=0.78).
ExecutiveRMS: 4 atoms rejected during cycle 5 (RMSD=0.77).
Executive: **RMSD = 0.768** (846 to 846 atoms)

align (2a9n and name CA), (2a9m and name CA)
Match: read scoring matrix.
Match: assigning 236 x 236 pairwise scores.
MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 1226.000
ExecutiveAlign: 236 atoms aligned.
ExecutiveRMS: 9 atoms rejected during cycle 1 (RMSD=1.58).
ExecutiveRMS: 6 atoms rejected during cycle 2 (RMSD=0.84).
ExecutiveRMS: 4 atoms rejected during cycle 3 (RMSD=0.79).
Executive: **RMSD = 0.770 (214 to 214 atoms)**

super 2a9n, 2a9m
MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 915.875
ExecutiveAlign: 1658 atoms aligned.
ExecutiveRMS: 68 atoms rejected during cycle 1 (RMSD=1.19).
ExecutiveRMS: 52 atoms rejected during cycle 2 (RMSD=0.91).
ExecutiveRMS: 30 atoms rejected during cycle 3 (RMSD=0.85).
ExecutiveRMS: 21 atoms rejected during cycle 4 (RMSD=0.82).
ExecutiveRMS: 8 atoms rejected during cycle 5 (RMSD=0.80).
Executive: **RMSD = 0.793 (1477 to 1477 atoms)**

super (2a9n and backbone), (2a9m and backbone)

MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 916.007
ExecutiveAlign: 890 atoms aligned.
ExecutiveRMS: 22 atoms rejected during cycle 1 (RMSD=0.91).
ExecutiveRMS: 20 atoms rejected during cycle 2 (RMSD=0.81).
ExecutiveRMS: 7 atoms rejected during cycle 3 (RMSD=0.78).
ExecutiveRMS: 4 atoms rejected during cycle 4 (RMSD=0.77).
ExecutiveRMS: 2 atoms rejected during cycle 5 (RMSD=0.76).
Executive: **RMSD = 0.759 (833 to 833 atoms)**

super (2a9n and name CA), (2a9m and name CA)
MatchAlign: aligning residues (236 vs 236)...
MatchAlign: score 916.341
ExecutiveAlign: 224 atoms aligned.
ExecutiveRMS: 5 atoms rejected during cycle 1 (RMSD=0.89).
ExecutiveRMS: 6 atoms rejected during cycle 2 (RMSD=0.81).
ExecutiveRMS: 1 atoms rejected during cycle 3 (RMSD=0.76).
ExecutiveRMS: 1 atoms rejected during cycle 4 (RMSD=0.76).
Executive: **RMSD = 0.753 (209 to 209 atoms)**

cealign 2a9n, 2a9m
RMSD 1.576778 over 224 residues