Description of PLAMS functions used in script

Scalar numbers appear as lowercase letters, vectors as lowercase bold letters, points as uppercase letters, matrices as uppercase bold letters.

***distance\_to\_mol (self, other, result\_unit='angstrom', return\_atoms=False)***

*Calculate the distance between this molecule and some \*other\* molecule.*

*The distance is measured as the smallest distance between a pair of atoms, one belonging to each of the molecules. Returned distance is expressed in \*result\_unit\*.*

*If \*return\_atoms\* is ``False``, only a single number is returned. If \*return\_atoms\* is ``True``, this method returns a tuple ``(distance, atom1, atom2)`` where ``atom1`` and ``atom2`` are atoms fulfilling the minimal distance, with atom1 belonging to this molecule and atom2 to \*other\*.*

Distance between atoms is calculated as distance, , between two points in 3-dimentions and :

Distances between all atoms are calculated and compared to each other and smallest distance is returned as the shortest distance between two molecules.

***translate (self, vector, unit='angstrom')***

*Move this atom in space by \*vector\*, expressed in \*unit\*.*

*\*vector\* should be an iterable container of length 3 (usually tuple, list or numpy array). \*unit\* describes unit of values stored in \*vector\*.*

*This method requires all coordinates to be numerical values, :exc:`~exceptions.TypeError` is raised otherwise.*

Translation of molecule is done by translating all of its atoms, adding the given vector, , to the coordinates of each atom, .

***rotate (self, matrix)***

*Rotate this atom according to rotation \*matrix\*.*

*\*matrix\* should be a container with 9 numerical values. It can be a list (tuple, numpy array etc.) listing matrix elements row-wise, either flat (``[1,2,3,4,5,6,7,8,9]``) or in two-level fashion (``[[1,2,3],[4,5,6],[7,8,9]]``).*

*note:*

*This method does not check if supplied matrix is a proper rotation matrix.*

Rotation is done by multiplying original vector, , represented like column matrix, with the matrix given in argument of this function, M. This multiplication gives new vector, :

**vector\_to (self, point, unit='angstrom', result\_unit='angstrom')**

*Calculate a vector from this atom to \*point\*.*

*\*point\* should be an iterable container of length 3 (for example: tuple, |Atom|, list, numpy array). \*unit\* describes unit of values stored in \*point\*. Returned value is expressed in \*result\_unit\*.*

*This method requires all coordinates to be numerical values, :exc:`~exceptions.TypeError` is raised otherwise.*

This function makes vector, **,** between point and new point, given in argument, :

***rotate\_bond(self, bond, atom, angle, unit='radian')***

*Rotate given \*bond\* by an \*angle\* expressed in \*unit\*.*

*\*bond\* should be chosen in such a way, that it divides the molecule into two parts (using a bond being part of a ring results in an error). \*atom\* has to belong to \*bond\* and is used to pick which "half" of the molecule is rotated. Positive angle denotes counterclockwise rotation (looking along the bond, from the stationary part of the molecule).*

For rotation of vectors it is needed to define rotation axis and rotation angle in matrix notation. Cross product of unit matrices represents the axis of rotation. Bond and atom given in argument define a vector and its direction, . Vector is normalized and cross product expressed in matrix notation is made:

Angle given in argument define sine of the angle of rotation, Function is using Rodrigues rotation formula:

Or, using the half-angle formula:

Using *rotation()* vector is than rotated.

***rotation\_matrix (vec1, vec2)***

*Calculates the rotation matrix rotating from one vector to another, in this case from vec1 to vec2. Vectors can be any containers with 3 numerical values. They don't need to be normalized. Returns 3x3 numpy array.*

This function is using Rodrigues rotation formula. First vectors are normalized, unit vector for vec1 and vec2 are vectors a and b respectively.

Cross product expressed in matrix notation:

Rotation in matrix in this case is:

I represents identity matrix.

Third element on the right side is simplified version of:

From this equation we can see that its undefined for , which is true for vectors pointing opposite directions.

New functions in the script

Script is using functions from PLAMS library.

Input needed for the script are two list of Molecules (object in PLAMS):

- input\_cores – core molecules on which H atoms will be substituted with ligands

- input\_ligands – molecules that should be added to the core, also with substitution of H atom

Correct substitution is only done on H atoms because it’s bonded to only one other atom which means other atom and bond between other atom (lig\_other for ligand and core\_other for core molecule) and hydrogen is exactly defined. This is used for defining bond as a vector between these atoms.

Hydrogen that should be substituted has to be on the first place in coordinate files, both molecules: core and ligand.

***restr\_distance\_to\_mol* (self, other, excluded1 = None, excluded2 = None, result\_unit = 'angstrom', return\_atoms=False)**

*Arguments excluded1 and excluded2 are for atoms in this molecule which distances to \*other\* atoms are not taken into account.*

This same function as PLAMS function *distance\_to\_mol* with some restrictions. Two arguments are added to the function: excluded1 and excluded2. This is useful when there is an atom for which is known that it is the closest one but it is not of the interest for further procedure.

***rotation\_check (ligand,lig\_h,lig\_other,core)***

*Rotates ligand around the bond between ligand and core in order to find best possible orientation for ligand, in aim to avoid steric clash.*

*lig\_h is hydrogen on ligand which shoud be substituted; ligh\_other is atom connected to the lig\_h.*

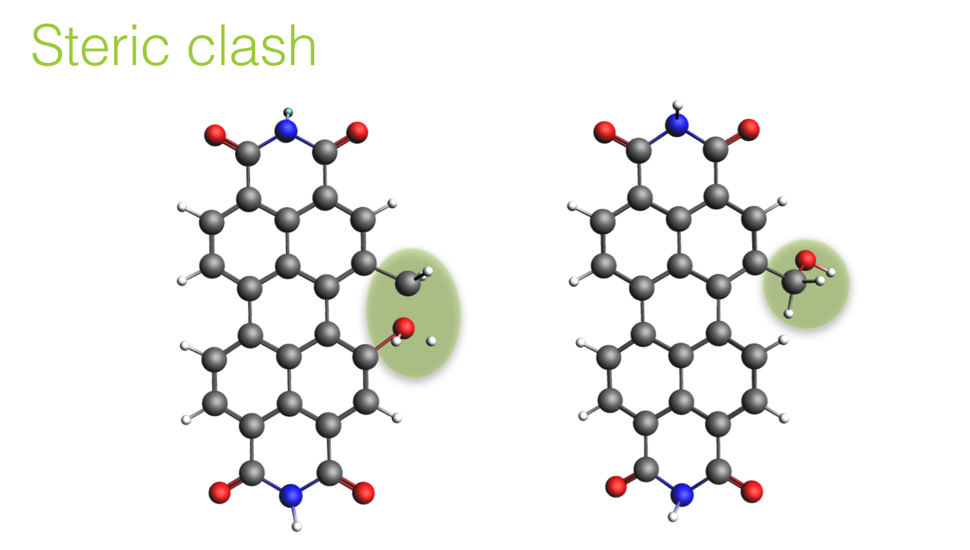


Image 1. Before(left) and after(right) rotation

First checks if ligand is already oriented so there is no steric clash. Shortest distance between two atoms will be used as an indicator that there is no steric clash. Distance larger or equal to is taken as boundary condition as most of organic bonds are shorter than this value. If this condition is not fulfilled, rotation of ligand is applied.

Bond is rotated in angle steps of 10 degrees (0.1745329252 radians) from 0 to 360 degrees. Therefore, ligand is back to the beginning position.

After every rotation shortest distance is calculated and placed in list. Indices in this list multiplied with angle step give values of angles for which there are no steric clashes. Indices of all distances larger that are placed in new list. First index from the new list is taken and ligand is rotated for appropriate value. If in all positions, distance is shorter that , program will check if any shortest distances are between two hydrogens. If that is the case, hydrogens boundary condition is lowered to , as the bond between hydrogens are approx. . Analogously to previous procedure, index of appropriate distance is used for calculating angle for which there is no steric slash.

\*Further improvement will involve giving more degrees of freedom for rotation of bond between hydrogen and lig\_other.

***connect\_two\_molecules (core, ligand)***

*Takes molecule coordinates as arguments. This function makes changes on ligand and core molecules to obtain best orientation is space for both of them and finally adds them together.*

*Connects two molecules in place of first atom on the coordinate list. First atom, for both ligand and core molecule, should be hydrogen.*

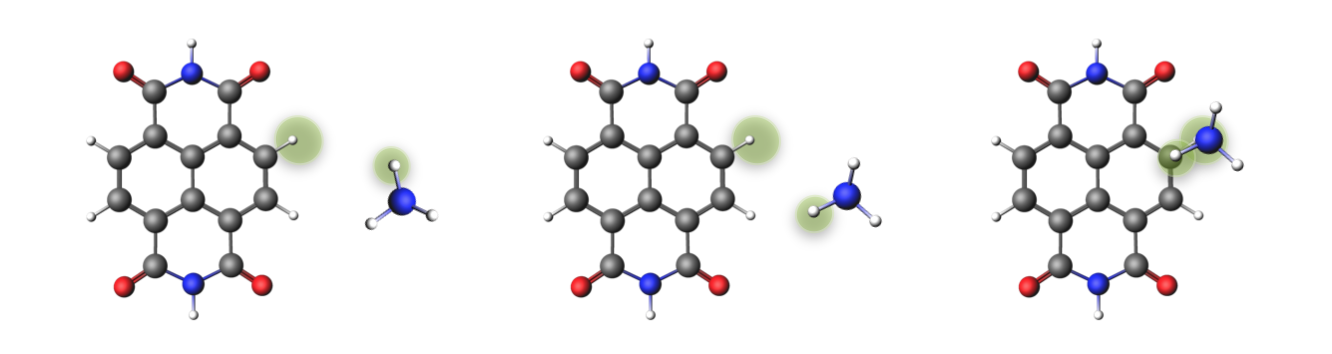


Image 2. Connecting core and ligand

Inside this function hydrogens from core and ligand are defined in variables core\_h and lig\_h. Atom connected to those hydrogen atoms are defined using function *other\_end* andnamed core\_other and lig\_other. Bond between those atoms is defined with function *vector\_to,* named core\_vector and lig\_vector.

Defined vectors are used for aligning core and ligand molecules. Rotation matrix is calculated using function *rotation\_matrix().* Ligand is than rotated using function *rotate().* At the end ligand is translated (*translate()*) to the position of core vector.

Next step is deleting hydrogen at the core molecule and resizing the distance between two “other” atoms (core\_other and lig\_other), depending on atom type. If it’s two carbon atom distance between them is set to 1.54 A, if nitrogen and carbon atom bond distance is set to 1.469 A. If it’s neither of these, distance is set to 1.5 A.

\*Further improvement will involve bond distances defined for other atoms.

After these transformation, hydrogen atom is at the same point in space as core\_other, but that hydrogen is still needed for ligand rotation. Function restr\_distance\_to\_mol is useful here. Excluded atoms are lig\_other and hydrogen atom on ligand for which distance is manually set earlier in script.

To avoid steric clash function rotation\_check is used.

When the good position for ligand is found, hydrogen on ligand is deleted and coordinates of new molecule is what is returned from this function.

Adding two molecules together

***Monosubstitution:***

Because the function that connects core and ligand changes ligand and core molecules, instead of using original variable, this script is using copies of ligand and core lists, more precisely deepcopy.

Number of copies of core molecules equals number of ligands in input\_ligands. To each copy one ligand is added to all core molecules.

Examlpe:

For core\_input = [-, +, \*] and ligand\_input = [1, 2] there will be two copies of core molecule and the first copy of cores firs ligand would be added: [-1, +1, \*1] and to the second copy - second ligand: [-2, +2, \*2]. In total there will be six different monosubstituted molecules: mono\_subs = [-1, +1, \*1, -2, +2, \*2].

***Disubstitution:***

Disubstitution is done in similar way like monosubstitution, just in case of disubtitution repetition of ligand-core-ligand combinations must be taken in account. Again, number of copies of monosubtituted molecules are equal to number of ligand. In each iteration one ligand is added to one if the copies of core molecule. To the later copies, adding ligands will start from a number equal to number of different cores multiplied with number of iteration.

Example:

For core\_input = [-, +, \*] and ligand\_input = [1, 2] leads to mono\_subs = [-1, +1, \*1, -2, +2, \*2]. There will be two copies of monosubstituted molecules. To the first copy, first ligand would be added starting from 1st item in list: [-11, +11, \*11, -21, +21, \*21]. Next iteration has value 1 multiplied with number of cores, 3, means that adding ligands will start from 3rd item in list: [-1, +1, \*1, -22, +22, \*22]. -12 will be the same molecule as -21, which is made in first iteration and it can be excluded here.