CAMM rotamer library

Version 1.3

Wiktor 'Nanotekton' Beker

October 2, 2017

1 Download

The package `rotabase` can be downloaded from http://156.17.246.1/CAMM/rotabase-1.3.tar.gz .

2 Package contents

- BioPython objects: `PDBParser`, `Selection`
- PyMolecule `multipoles` module
- Interface with database: `base`, `load_base`,`restrict_base`, `print_ref`, `read_camm`, `read_camm`
- Functions calculating $\{\theta_i\}$ defining rotamer: `chis`, `get_rot`
- Input/Output functions: `drop`, `load`, `dump`

3 Loading data from the base

As with any other Python package, you can either import the whole `rotabase` or its contents. In either way, it already provides classes and object required to handle PDB files (`PDBParser`, `Selection`) and multipole moments (`multipoles` from `pymolecule`). It provides also the default dictionary `base` containing CAMMs for amino acid rotamers (HF/6-31G* by default).

3.1 Loading the library

`load_base(method='HF',basis_set='6-31Gd')` returns a dictionary containing CAMMs obtained with given method and basis set. Currently available options:

- Hartree-Fock (`'HF'`) : 6-31G(d) (`'6-31Gd'`)
- MP2 (`'MP2'`) : 6-31G(d) (`'6-31Gd'`)

3.2 Resctricting the number of multipole components kept in base

By default, CAMM multipole moments up to rank 9 are loaded. However, this amount of components leads to substancially increased time of loading CAMM to the given residue or rotation of `PyMolecule` object. Moreover, interaction energy calculated with interpolated multipole moments truncated higher than R^{-5} becomes more inaccurate. Therefore, one may decide to truncate CAMM arrays to components lower of equal of a certain rank. This can be done with `restrict_base` method, which modifies base dictionary provided by cutting out components higher than a given rank L– for now only $L = 4$ is available.

```
mybase=load_base()  # load base with default parameters
restrict_base(mybase,L=4)  #now mybase is modified;L=4 is a default value of this parameter
```

3.3 Loading PDB

For a given PDB file, a list of `Residue` objects may be obtained with tools from Bio.PDB, loaded together with `rotabase`.

```
p=PDBParser(Quiet=True)
protein=p.get_structure('prot','some_file.pdb')
residue=Selection.unfold_entities(protein,'R')
```

3.4 Loading CAMMs corresponding to `Residue` object

`read_camm(residue, base, mode='nearest', expo=2.0, corr=False)` returns CAMM (`PyMultipoles` object) corresponding to the `residue` (a `Residue` object, as described above; for further information look into BioPython manual). There are two mandatory arguments: `residue` and `base` (a dictionary containing CAMM library). The rest descibe a way in which CAMMs are adjusted for particular conformation (this is a result of fact that dihedrals defining each rotamer are spread acroos a certain *range*, therefore a record from libary has to be adjusted to actual geometry). There are few modes of CAMM interpolation currently available, chosed with following options to `mode` keyword.

`'nearest'` nearest rotamer, with dihedrals adjusted

`'interpolation'` multidimensional linear interpolation between neighbouring rotamers

`'shepard2''` Shepard interpolation; additional keyword `expo` specifies the exponent

`'interOH'` takes the nearest rotamer and interpolate OH group only

The remaining flag `corr` toggles a final simple correction to the geometry (correcting the angles and bond distances).

WARNING. There may be problems connected to different atom naming schemes in some PDB files. In such a case, you can print the reference PDB coordinates for a given resname with `print_ref(resname)` and chek the atom names. Unfortunately, automatic correction of such issues is not implemented yet.

3.5 Loading CAMM of specified residue/rotamer, fitted only to backbone

One my be interested in loading CAMMs of specified residue and rotamer into position defined by a given resudue. This can be done with `load_camm` function.
`load_camm(residue, target_resname,target_rotid, base, coor_only=False)`
returns CAMM (`PyMultipoles` object) of rotamer `target_rotid` of aminoacid `target_resname` with backbone fitted to the given `Residue` object. Flag `coor_only` toggles rotation of CAMMs; if true, only atomic coordinates will be transformed, otherwise CAMM array will be also rotated. This may be useful when one's interested in atomic coordinates, which is severeal orders of magnitude faster than rotation of both CAMM components and atomic positions. In order to check IDs of possible rotamers, check the keys of `base[resname]` dictionary.

3.6 Including backbone conformation

Previous version of rotamer database didn't allow to change the conformation of backbone of aminoacids provided by it. Here, it included partially by additional set of 36 conformers of glycine. The idea if to approximate interaction energy of given aminoacid by the sum of interactions with its main chain (glicine) and sidechain; let AA_{target} and GLY_{target} be the given aminoacid and glycine with target conformation of main chain (ψ and ϕ angles as in PDB structure), while AA_{base} and GLY_{base} would depict those residues with a conformation present in rotamer database.

$$E_{int}(X, AA_{target}) \approx E_{int}(X, AA_{base}) + E_{int}(X, GLY_{target}) - E_{int}(GLY_{base}) \quad (1)$$

The above equation may be implemented by a following piece of code.

```
#A- Residue object, base- loaded CAMM rotamer database

#read CAMMs in a standard way, without adjusting backbone
a  =read_camm(A,base)

#read glycine with target conformation
```

```
ab =read_camm(A,base,backbone=True)

#read 'phantom glycine' with conformation corresponding to the one present in rotabase
abp=phantom_gly(a,A,base)

#calculate energy
mtp=m.energy(inh_camm[i],a,4) + m.energy(inh_camm[i],ab,4)-m.energy(inh_camm[i],abp,4)

#transform from au to kcal/mol
mtp*=627.5
```

4 Additional functions

4.1 Measuring dihedrals

`chis(<pymolecule object>)`
: returns and array of dihedral angles (degrees) defining rotamer

`get_rotamer(<Residue object>)`
: calculates dihedrals for given residue; returns a tuple: rotamer label and an array of dihedrals

4.2 Input/Output functions

`drop(<pymolecule object>, filename)`
: writes XYZ file (in Å) basing on coordinates of `pymolecule` object

`load(file.pkl.gz)`
: returns Python object serialiazed in gzipped Pickle file

`dump(object,file.pkl.gz)`
: serializes Python object into gzipped Pickle file