Massachusetts Institute of Technology

Developer reference

`molSimplify` version 1.0

by

E. Ioannidis

KULiK GROUP

March, 2016

Contents

1 Directory tree **4**

2 Classes **5**

2.1 atom3D.py 5
2.1.1 __init__() 5
2.1.2 coords() 5
2.1.3 distance() 5
2.1.4 distancev() 5
2.1.5 ismetal() 5
2.1.6 symbol() 5
2.1.7 translate() 5
2.1.8 __repr__() 5
2.2 globalvars.py 6
2.2.1 mybash() 6
2.2.2 __init__() 6
2.2.3 amass() 6
2.2.4 metals() 6
2.2.5 elementsbynum() 6
2.3 mGUI.py 6
2.3.1 __init__() 6
2.3.2 qDBload() 7
2.3.3 enableDB() 7
2.3.4 qaddDB() 7
2.3.5 qdelDB() 7
2.3.6 dbchange() 7
2.3.7 postprocGUI() 7
2.3.8 qcDBload() 7
2.3.9 searchDBW() 7
2.3.10 qaddcDB() 8
2.3.11 cdbchange() 8
2.3.12 runGUI() 8
2.3.13 drawligs() 8
2.3.14 enableffinput() 8
2.3.15 dirload() 8
2.3.16 qdumpS() 8
2.3.17 getscreensize() 8
2.3.18 sliderChanged() 8

2.3.19 matchgeomcoord() . . . 8
2.3.20 enableemol() . . . 9
2.3.21 qcinput() . . . 9
2.3.22 jobdef() . . . 9
2.3.23 qctdef() . . . 9
2.3.24 qcgdef() . . . 9
2.3.25 qcqdef() . . . 9
2.3.26 enableqeinput() . . . 9
2.3.27 enablejinput() . . . 9
2.3.28 jobenable() . . . 9
2.3.29 setupp() . . . 9
2.3.30 pdload() . . . 10
2.4 mol3D.py . . . 10
2.4.1 __init__() . . . 10
2.4.2 addatom() . . . 10
2.4.3 alignmol() . . . 10
2.4.4 centermass() . . . 10
2.4.5 centersym() . . . 10
2.4.6 convert2mol3D() . . . 10
2.4.7 combine() . . . 10
2.4.8 coordsvec() . . . 11
2.4.9 copymol3D() . . . 11
2.4.10 deleteatom() . . . 11
2.4.11 deleteatoms() . . . 11
2.4.12 deleteHs() . . . 11
2.4.13 distance() . . . 11
2.4.14 findcloseMetal() . . . 11
2.4.15 findAtomsbySymbol() . . . 11
2.4.16 findsubMol() . . . 11
2.4.17 getAtom() . . . 11
2.4.18 getAtoms() . . . 12
2.4.19 getBondedAtoms() . . . 12
2.4.20 getBondedAtomsnotH() . . . 12
2.4.21 getHs() . . . 12
2.4.22 getHsbyAtom() . . . 12
2.4.23 getClosestAtom() . . . 12
2.4.24 getMask() . . . 12
2.4.25 getClosestAtomnoHs() . . . 12
2.4.26 getOBmol() . . . 12

2.4.27 initialize() . . . 12
2.4.28 maxdist() . . . 13
2.4.29 mindist() . . . 13
2.4.30 mindistmol() . . . 13
2.4.31 mindistnoH() . . . 13
2.4.32 molsize() . . . 13
2.4.33 overlapcheck() . . . 13
2.4.34 printxyz() . . . 13
2.4.35 readfromxyz() . . . 13
2.4.36 rmsd() . . . 13
2.4.37 sanitycheck() . . . 13
2.4.38 translate() . . . 14
2.4.39 writegxyz() . . . 14
2.4.40 writexyz() . . . 14
2.4.41 writemxyz() . . . 14
2.4.42 __repr__() . . . 14
2.5 `mWidgets.py` . . . 16

3 Scripts . . . 16
3.1 `addtodb.py` . . . 16
3.2 `dbinteract.py` . . . 16
3.3 `generator.py` . . . 16
3.4 `geometry.py` . . . 16
3.5 `grabguivars.p` . . . 16
3.6 `inparse.py` . . . 16
3.7 `io.py` . . . 16
3.8 `jobgen.py` . . . 16
3.9 `postmold.py` . . . 16
3.10 `postmwfn.py` . . . 16
3.11 `postparse.py` . . . 16
3.12 `postproc.py` . . . 16
3.13 `qcgen.py` . . . 16
3.14 `rungen.py` . . . 16
3.15 `structgen.py` . . . 16

4 Data files . . . 16
4.1 Core dictionary . . . 16
4.2 Ligand dictionary . . . 16
4.3 Extra molecule dictionary . . . 16

4.4 Metal-ligand bond lengths 16
4.5 Backbone files 16

1 Directory tree

The files that comprise this project are shown in the following directory tree:

```
molSimplify
 ├─main.py # start of the program
 ├─pybel.py # modified pybel
 ├─Bind/ # local extra molecules folder
 │  ├─⋆.mol # extra molecules mol files
 │  └─bind.dict # dictionary of extra molecules
 ├─Cores/ # local cores folder
 │  ├─⋆.mol # local cores mol files
 │  └─cores.dict # local cores dictionary
 ├─Classes/ # classes definitions
 │  ├─atom3D.py # atom3D class definition
 │  ├─globalvars.py # global variables
 │  ├─mGUI.py # main GUI class definition
 │  ├─mol3D.py # mol3D class definition
 │  └─mWidgets.py # custom GUI widgets class
 ├─Data/ # folder with useful data
 │  ├─⋆.dat # backbones for common geometries
 │  └─ML.dat # metal-ligand bond lengths database
 ├─icons/ # icons for GUI folder
 │  └─⋆.png # icons for GUI
 ├─Ligands/ # local ligands folder
 │  ├─⋆.mol # local ligands mol files
 │  ├─ligands.dict # local ligands dictionary
 │  └─simple_ligands.dict # simple ligands alternative names
 └─Scripts/ # folder with main scripts
    ├─addtodb.py # local database interaction
    ├─dbinteract.py # external database interaction
    ├─enerator.py # initial driver called by GUI
    ├─geometry.py # geometric operations
    ├─grabguivars.py # gathers data from GUI to input file
    ├─inparse.py # parses input files
    ├─io.py # loads cores/ligands/extra molecules
    ├─jobgen.py # generates jobscripts
    ├─postmold.py # parses molden files for MO info
    ├─postmwfn.py # uses Multiwfn for post processing
    ├─postparse.py # generates runs summary and nbo
    ├─postproc.py # driver for post processing
    ├─qcgen.py # generates quantum chemistry input
    ├─rungen.py # coordinates file generation
    ├─structgen.py # builds/modifies structures
```

The routines included in each of the aforementioned files are explained in detail next.

2 Classes

2.1 `atom3D.py`

This file defines a custom atom class named atom3D. Each object has the following attributes: mass, atomic number, covalent radius, 3D coordinates and symbol.

2.1.1 __init__()

This is the constructor of the class the takes as arguments the symbol and the 3D coordinates of the atom. It pulls the atomic mass from a database included in `globalvars.py` and if it doesn't find the corresponding atomic symbol in the database it assumes a carbon atom.

2.1.2 coords()

Returns a list with the 3D coordinates of the atom.

2.1.3 distance()

Calculates the distance with another atom in 3D space.

2.1.4 distancev()

Returns the difference vector with respect to another atom in 3D space.

2.1.5 ismetal()

Returns true or false if the atom is a metal or not. It compares the symbol with a database from `globalvars.py`.

2.1.6 symbol()

Returns the symbol of the atom.

2.1.7 translate()

Translates the atom in 3D space by δx, δy, δz specified as input.

2.1.8 __repr__()

Prints the various methods available in the class.

2.2 globalvars.py

This class offers data and various global functions useful throughout the program. In the files there is a big dictionary with properties of atoms, a list of metals and a list of element symbols.

2.2.1 mybash()

This function uses the subprocess module in python to run a shell command and catch the output by redirecting `stdout`. It returns the output of the shell command.

2.2.2 __init__()

The constructor of the globalvars class is mainly responsible for finding the appropriate paths that are required for the program to run. It looks up the system type (linux or OSX) and then tries to read the configuration file `/.molSimplify` in order to assign 3 paths. The attribute `installdir` of the Class contains the installation directory of the program needed to read the local molecule database, icons and data, the `cemdbdir` attribute contains the folder where the external chemical databases are located and the attribute `multiwfn` contains the path of the Multiwfn executable needed for post processing. It also has information about the path of the home directory of the user, jobs running directory and some global counters such as the number of structures generated.

2.2.3 amass()

Returns the dictionary with the atomic properties.

2.2.4 metals()

Returns a list of metals.

2.2.5 elementsbynum()

Returns a list of elements sorted by atomic number.

2.3 mGUI.py

This file defines the main GUI class.

2.3.1 __init__()

The constructor of the GUI is responsible for creating all objects that will be visible when the user starts the program and interacts with it. It uses many different widgets provided by the PyQt5 library and contains windows, push buttons, check buttons, dropdown boxes, sliders, editable text boxes and menu bars. The user can interact with many of these widgets through corresponding callback functions that are described next.

2.3.2 qDBload()

Function that loads a molecule from a file to be added in the local molecule database.

2.3.3 enableDB()

Callback for button that enables addition/removal from local database and brings up the corresponding window.

2.3.4 qaddDB()

Adds the specified molecule to the local database.

2.3.5 qdelDB()

Deletes the specified molecule from the local database.

2.3.6 dbchange()

Enables and disables input options according to whether a core, ligand or extra molecule are added or removed.

2.3.7 postprocGUI()

Initiates the post processing by calling for the input file generation and then passing control to the post processing routines.

2.3.8 qcDBload()

Loads a molecule from a file to be used for interaction with external databases.

2.3.9 searchDBW()

Enables interface for interaction with external databases and loads the corresponding window. It also checks for existing databases and prompts the user to specify a different folder if no database is found or if no database folder is specified.

2.3.10 qaddcDB()

Initiates the screening or similarity search with external databases by constructing the appropriate input file and passing control to the corresponding module.

2.3.11 cdbchange()

Enables and disables input in the database.

2.3.12 runGUI()

Initiates structure generation by creating the appropriate input file and passing control to the corresponding module.

2.3.13 drawligs()

This module loads the ligands specified in the GUI, creates a .svg representation using babel and then converts this representation to a .png file using imagemagick. The final png file is then loaded and visualized.

2.3.14 enableffinput()

Enables input for the force field optimization.

2.3.15 dirload()

The user browses for the running directory that is defined in this routine.

2.3.16 qdumpS()

Grabs all data from the GUI and creates corresponding input files.

2.3.17 getscreensize()

Returns the screen size in pixels.

2.3.18 sliderChanged()

Changes the text in the distortion slider.

2.3.19 matchgeomcoord()

Matches the corresponding geometries with the coordination number specified.

2.3.20 enableemol()

Enables input for the specification of extra molecule placement.

2.3.21 qcinput()

Brings up the corresponding window for specifying the input for quantum chemistry input file generation.

2.3.22 jobdef()

Creates a file with the default options for jobscript generation.

2.3.23 qctdef()

Creates a file with the default options for terachem input file generation.

2.3.24 qcgdef()

Creates a file with the default options for GAMESS input file generation.

2.3.25 qcqdef()

Creates a file with the default options for QChem input file generation.

2.3.26 enableqeinput()

Enables input for quantum chemistry input file generation.

2.3.27 enablejinput()

Enables input for jobscript generation.

2.3.28 jobenable()

Loads the window for jobscript generation.

2.3.29 setupp()

Loads the window for post processing. It also checks for a valid Multiwfn installation and prompts the user to specify a valid path if no functioning Multiwfn code is found.

2.3.30 pdload()

Loads directory with results to be used for post processing.

2.4 mol3D.py

This class defines a 3D molecule class. Each mol3D object has the following attributes: a list of atom3D objects (atoms), the number of atoms comprising the molecule, the total mass, the total size in Angstrom, charge, an openbabel molecule object (OBmol), a list of connection atoms, denticity, identifier used in naming folders (ident) and a globalvars object (globs).

2.4.1 __init__()

The constructor of the class initializes the various attributes of a mol3D object.

2.4.2 addatom()

Adds an atom3D object to the molecule.

2.4.3 alignmol()

Aligns the molecule so that an atom of the molecule matches another atom (not from the same molecule) specified in the input.

2.4.4 centermass()

Calculates the center of mass of the molecule.

2.4.5 centersym()

Calculates a center of symmetry treating all atoms, heavy and light as equivalent.

2.4.6 convert2mol3D()

Converts an openbabel molecule object to a mol3D atom by extracting the corresponding 3D coordinates.

2.4.7 combine()

Combines two molecules into one.

2.4.8 coordsvec()

Returns a vector with the xyz coordinates of all the atoms in the molecule.

2.4.9 copymol3D()

Copies all the attributes of one mol3D to another.

2.4.10 deleteatom()

Deletes an atom from the molecule specified with its index.

2.4.11 deleteatoms()

Deletes a list of atoms.

2.4.12 deleteHs()

Deletes all Hydrogens from the molecule.

2.4.13 distance()

Calculates the distance between two molecules.

2.4.14 findcloseMetal()

Returns the index of the metal closest to a specified atom.

2.4.15 findAtomsbySymbol()

Returns a list with the indices of all atoms with the specified symbol.

2.4.16 findsubMol()

This module runs a connectivity graph search and finds a submolecule within the current molecule. The user specifies the two atoms that would be disconnected if the submolecule was separated from the current molecule. It loops over all the atomic connections starting from the first reference atom and eliminating atoms that don't belong to the submolecule.

2.4.17 getAtom()

Returns the atom3 object that corresponds to the specified index.

2.4.18 getAtoms()

Returns the list of atom3D objects in the molecule.

2.4.19 getBondedAtoms()

Returns list of all the atoms bonded to a specified atom in the molecule. It uses the covalent radii of the atoms to detect bonding.

2.4.20 getBondedAtomsnotH()

Returns atoms that are bonded to the reference atom and that are not Hydrogens.

2.4.21 getHs()

Returns all the Hydrogens in the molecule.

2.4.22 getHsbyAtom()

Returns the list of Hydrogens bonded to a specific atom in the molecule.

2.4.23 getClosestAtom()

Returns the index of the closest atom with respect to a specified atom in the molecule.

2.4.24 getMask()

Returns the center of mass of the atoms specified by an atomic mask. This mask can be a list of atomic indices, an expression such as 1-5, an atomic symbol or any combination of those.

2.4.25 getClosestAtomnoHs()

Returns the closest non-Hydrogen atom with respect to a specified atom in the molecule.

2.4.26 getOBmol()

Loads an openbabel molecule from a file using pybel and saves it to the OBmol attribute of the corresponding mol3D object.

2.4.27 initialize()

Reinitializes the molecule.

2.4.28 maxdist()

Calculates the maximum distance between the atoms in 2 molecules.

2.4.29 mindist()

Calculates the minimum distance between the atoms in 2 molecules.

2.4.30 mindistmol()

Calculates the minimum distance between all atoms in the molecule.

2.4.31 mindistnoH()

Calculates the minimum distance between the non-Hydrogen atoms in 2 molecules.

2.4.32 molsize()

Calculates the molecular size based on the maximum distance of an atom from the center of mass of the molecule.

2.4.33 overlapcheck()

Checks for overlap between two molecules by looping over all atoms and comparing their distance to the sum of their covalent radii.

2.4.34 printxyz()

Prints the xyz coordinates of the molecule.

2.4.35 readfromxyz()

Creates a mol3D object from xyz file.

2.4.36 rmsd()

Calculates the RMSD of two molecules.

2.4.37 sanitycheck()

Checks if atoms within a molecule overlap.

2.4.38 translate()

Translates the molecule by δx,δy,δz.

2.4.39 writegxyz()

Writes a xyz-like file for GAMESS input.

2.4.40 writexyz()

Writes an xyz file with the coordinates of the complex.

2.4.41 writemxyz()

Writes an xyz file combinding 2 molecules.

2.4.42 __repr__()

Prints all the available methods within the class.

2.5 `mWidgets.py`

This class contains various widgets that are redefined for using them in the GUI and is basically an extension of the existing Widgets provided by PyQt5.

3 Scripts

3.1 `addtodb.py`

3.2 `dbinteract.py`

3.3 `generator.py`

3.4 `geometry.py`

3.5 `grabguivars.p`

3.6 `inparse.py`

3.7 `io.py`

3.8 `jobgen.py`

3.9 `postmold.py`

3.10 `postmwfn.py`

3.11 `postparse.py`

3.12 `postproc.py`

3.13 `qcgen.py`

3.14 `rungen.py`

3.15 `structgen.py`

4 Data files

4.1 Core dictionary

4.2 Ligand dictionary

4.3 Extra molecule dictionary

4.4 Metal-ligand bond lengths

4.5 Backbone files