FreeSAS Documentation

Release 1.0

Guillaume Bonamis

July 17, 2015

CONTENTS

FreeSAS is a Python package with small angles scattering tools in a MIT type license.

CHAPTER

ONE

INTRODUCTION

FreeSAS has been write as a re-implementation of some ATSAS parts in Python for a better integration in the BM29 ESRF beam-line processing pipelines. It provides functions to read SAS data from pdb files and to handle them. Parts of the code are written in Cython and parallelized to speed-up the execution.

FreeSAS code is available on Github at https://github.com/kif/freesas .

CHAPTER

TWO

INSTALLATION

CHAPTER

THREE

USAGE

3.1 Freesas as a library

Here are presented some basics way to use FreeSAS as a library. Some abbreviations:

- DA = Dummy Atom
- DAM = Dummy Atoms Model
- NSD = Normalized Spatial Discrepancy

3.1.1 The SASModel class

This class allows to manipulate a DAM and to do some operations on it as it is presented here.

First, the method SASModel.read() can be used to read a pdb file containing data of a DAM :

```
from freesas.model import SASModel
model1 = SASModel()                 #create SASModel class object
model1.read("dammif-01.pdb")        #read the pdb file
#these 2 lines can be replaced by model1 = SASModel("dammif-01.pdb")
print model1.header                 #print pdb file content
print model1.atoms                  #print dummy atoms coordinates
print model1.rfactor                #print R-factor of the DAM
```

Some informations are extracted of the model atoms coordinates:

- fineness : average distance between a DA and its first neighbours
- radius of gyration
- Dmax : DAM diameter, maximal distance between 2 DA of the DAM
- center of mass
- inertia tensor
- canonical parameters : 3 parameters of translation and 3 euler angles, define the transformation to applied to the DAM to put it on its canonical position (center of mass at the origin, inertia axis aligned with coordinates axis)

```
print model1.fineness           #print the DAM fineness
print model1.Rg                 #print the DAM radius of gyration
print model1.Dmax               #print the DAM diameter
model1.centroid()               #calculate the DAM center of mass
print model1.com
model1.inertiatensor()          #calculate the DAM inertiatensor
print model1.inertensor
```

```
model1.canonical_parameters()  #calculate the DAM canonical_parameters
print model1.can_param
```

Other methods of the class for transformations and NSD calculation:

```
param1 = model1.can_param           #parameters for the transformation
symmetry = [1,1,1]                  #symmetry for the transformation
model1.transform(param1, symmetry)
#return DAM coordinates after the transformation

model2 = SASModel("dammif-02.pdb") #create a second SASModel
model2.canonical_parameters()
atoms1 = model1.atoms
atoms2 = model2.atoms
model1.dist(model2, atoms1, atoms2)#calculate the NSD between models

param2 = model2.can_param
symmetry = [1,1,1]
model1.dist_after_movement(param2, model2, symmetry)
#calculate the NSD, first model on its canonical position, second
#model after a transformation with param2 and symmetry
```

3.1.2 The AlignModels class

This other class contains lot of tools to align several DAMs, using the SASModel class presented before.

The first thing to do is to select the pdb files you are interested in and to create SASModels corresponding using the method of the class like following :

```
from freesas.align import AlignModels
inputfiles = ["dammif-01.pdb", "dammif-02.pdb", "dammif-03.pdb", ...]
align = AlignModels(inputfiles)        #create the class
align.assign_models()                  #create the SASModels
print align.models                     #SASModels ready to be aligned
```

Next, the different NSD between each computed models can be calculated and save as a 2d-array. But first it is necessary to give which models are valid and which ones are not and need to be discarded :

```
align.validmodels = numpy.ones((len(align.inputfiles)))
#here we keep all models as valid ones
align.makeNSDarray()                  #create the NSD table
align.plotNSDarray()                  #display the table as png file
align.find_reference()                #select the reference model
align.alignment_reference()           #align models with the reference
```

3.2 SuPyComb script

FreeSAS can also be used directly using command lines. Here is presented the way to use the program supycomb, the re-implementation of the supcomb of the Atsas package.

Supycomb has two different process, the first one is called when only two pdb files are put as arguments and a second one for more than two files.

With the first process, the program creates the two DAM provided by pdb files and align the second one on the first one (reference, do not move). The coordinates of the atoms of the aligned model are saved in a pdb file and the program return the final NSD between the two DAM. The name of the output can be modified.

The second one creates a model for each file put as argument. Models are first selected as valid or not using its R-factor value. The maximum value is the mean of R-factors plus twice the standard deviation. The figure of the R-factors is then displayed or saved automatically in png format.
Next, NSD between each valid DAM are computed to select best models using the mean of NSD with other models for each DAM. A maximal value for the NSD mean is create as the mean of the ND mean plus a standard deviation to discarded the to different models. The model with the lower NSD mean is the reference one. A second figure with the NSD table and the graphic with the NSD means is displayed or saved.
Finally, the valid models are aligned on the reference one and final positions are saved in pdb files called model-01.pdb, model-02.pdb, etc...

Several options are available for the supycomb program:

```
$ supcomb.py --help
usage: ./supcomb FILES [OPTIONS]

align several models and calculate NSD

positional arguments:
  FILE                  pdb files to align

optional arguments:
  -h, --help            show this help message and exit
  -m {SLOW,FAST}, --mode {SLOW,FAST}
                        Either SLOW or FAST, default: SLOW)
  -e {YES,NO}, --enantiomorphs {YES,NO}
                        Search enantiomorphs, YES or NO, default: YES)
  -q {ON,OFF}, --quiet {ON,OFF}
                        Hide log or not, default: ON
  -g {YES,NO}, --gui {YES,NO}
                        Save automatically figures or not, default: NO
  -o OUTPUT, --output OUTPUT
                        output filename, default: aligned.pdb
```

Slow mode / fast mode:
For the slow mode, the optimization of the NSD is done for each symmetry (ie. 8 times) whereas for the fast mode, the best symmetry is first choosen without optimization and only the NSD for this symmetry is optimized. The result is that the slow mode is nearly 8 times slower than the fast one. The NSD values thought are a few lower using the slow mode.

Enantiomorphs option:
This option can be used to authorize or not the program to look for enantiomorphs. If not, the program will not test 8 symmetries but only 4. The execution time will be nearly twice lower without enantiomorphs but only if you are using the slow mode, the gain is negligible for the fast mode. Moreover, it will not be able to recognize two enantiomorphs of the same protein.

GUI option:
You can choose to display the computed figures during the execution of the program to save it or not, or to save it automatically as png files with this option.

Output option:
This option allow to change the default filename of the output for the two models alignment process. It has to be a .pdb file !