pymemdyn

Release 1.6.1

H. Gutierrez de Teran **X. Bello** **M. Esguerra**
R. L. van den Broek **R.V. Küpper**

Mar 20, 2023

CONTENTS:

CHAPTER

ONE

MODULES

1.1 Pymemdyn module

1.2 Protein module

This module handles the protein and all submitted molecules around it.

class protein.**ProteinComplex**(**args*, ***kwargs*)

Bases: object

setMonomer(*value*)

Sets the monomer object.

getMonomer()

setLigand(*value*)

Sets the ligand object

getLigand()

setWaters(*value*)

Sets the crystal waters object

getWaters()

setIons(*value*)

Sets the ions object

getIons()

setCho(*value*)

Sets the cholesterol object

getCho()

setAlosteric(*value*)

Sets the allosteric object

getAlosteric()

set_nanom()

Convert dimension measurements to nanometers for GROMACS

class protein.**Protein**(**args*, ***kwargs*)

Bases: object

check_number_of_chains()

Determine if a PDB is a Monomer or a Dimer

class protein.**Monomer**(**args*, ***kwargs*)

Bases: object

delete_chain()

PDBs which have a chain column mess up with pdb2gmx, creating an unsuitable protein.itp file by naming the protein ie "Protein_A". Here we remove the chain value

According to http://www.wwpdb.org/documentation/format33/sect9.html, the chain value is in column 22

class protein.**Oligomer**(**args*, ***kwargs*)

Bases: *Monomer*

delete_chain()

Overload the delete_chain method from Monomer

class protein.**Sugar_prep**(**args*, ***kwargs*)

Bases: object

create_itp(*pdbfile: str*, *charge: int*, *numberOfOptimizations: int*) → None

Call ligpargen to create gromacs itp file and corresponding openmm pdb file. Note that original pdb file will be replaced by opnemm pdb file.

Parameters

- **pdbfile** – string containing local path to pdb of molecule. In commandline -i.
- **charge** – interger charge of molecule. In commandline -c.
- **numberOfOptimizations** – number of optimizations done by ligpargen. In cmdline -o.

Returns
None

Writes itp file and new pdf file to current dir. old pdb is saved in dir ligpargenInput. unneccessary ligpargen output is saved in dir ligpargenOutput.

lpg2pmd(*sugar*, **args*, ***kwargs*)

Converts LigParGen structure files to PyMemDyn input files.

Original files are stored as something_backup.pdb or something_backup.itp.

class protein.**Compound**(**args*, ***kwargs*)

Bases: object

This is a super-class to provide common functions to added compounds

check_files(**files*)

Check if files passed as *args exist

class protein.**Ligand**(**args*, ***kwargs*)

Bases: *Compound*

check_forces()

A force field must give a set of forces which match every atom in the pdb file. This showed particularly important to the ligands, as they may vary along a very broad range of atoms

class protein.**CrystalWaters**(**args*, ***kwargs*)

Bases: *Compound*

setWaters(*value*)

Set crystal waters

getWaters()

Get the crystal waters

property number

Get the crystal waters

count_waters()

Count and set the number of crystal waters in the pdb

class protein.**Ions**(**args*, ***kwargs*)

Bases: *Compound*

setIons(*value*)

Sets the crystal ions

getIons()

Get the crystal ions

property number

Get the crystal ions

count_ions()

Count and set the number of ions in the pdb

class protein.**Cholesterol**(**args*, ***kwargs*)

Bases: *Compound*

setCho(*value*)

Sets the crystal cholesterol

getCho()

Get the crystal cholesterols

property number

Get the crystal cholesterols

check_pdb()

Check the cholesterol file meets some standards

count_cho()

Count and set the number of cho in the pdb

class protein.**Alosteric**(**args*, ***kwargs*)

Bases: *Compound*

This is a compound that goes as a ligand but it's placed in an allosteric site rather than an orthosteric one.

check_pdb()

Check the allosteric file meets some standards

check_itp()

Check the force field is correct

1.3 Membrane module

class membrane.**Membrane**(**args*, ***kwargs*)

Bases: object

Set the characteristics of the membrane in the complex.

set_nanom()

Convert some measurements to nanometers to comply with GROMACS units.

1.4 Bw4posres module

Date: June 23, 2015 Email: mauricio.esguerra@gmail.com

Description: With this code we wish to do various task in one module:

1. Translate pdb to fasta without resorting to import Bio.
2. Align the translated fasta sequence to a Multiple Sequence Alignment (MSA) and place Marks coming from a network of identified conserved pair-distances of Venkatakrishnan et al. clustalo –profile1=GPCR_inactive_BWtags.aln –profile2=mod1.fasta -o withbwtags.aln –outfmt=clustal –wrap=1000 –force -v -v -v
3. Translate Marks into properly identified residues in sequence. Notice that this depends on a dictionary which uses the Ballesteros-Weinstein numbering.
4. From sequence ID. pull the atom-numbers of corresponding c-alphas in the matched residues.

---

class bw4posres.**Run**(*pdb*, ***kwargs*)

Bases: object

A pdb file is given as input to convert into one letter sequence and then align to curated multiple sequence alignment and then assign Ballesteros-Weinstein numbering to special positions.

pdb2fas()

From pdb file convert to fasta sequence format without the use of dependencies such as BioPython. This pdb to fasta translator checks for the existance of c-alpha residues and it is based on their 3-letter sequence id.

clustalalign()

Align the produced fasta sequence with clustalw to assing Ballesteros-Weinstein marks.

getcalphas()

Pulls out the atom numbers of c-alpha atoms. Restraints are placed on c-alpha atoms.

makedisre()

Creates a disre.itp file with atom-pair id's to be restrained using and NMR-style Heaviside function based on Ballesteros-Weinstein tagging.

1.5 Complex module

class complex.**MembraneComplex**(**args*, ***kwargs*)

Bases: object

setMembrane(*membrane*)

Set the membrane pdb file

getMembrane()

setComplex(*complex*)

Set the complex object

getComplex()

1.6 Queue module

A multi-producer, multi-consumer queue.

exception queue.**Empty**

Bases: Exception

Exception raised by Queue.get(block=0)/get_nowait().

exception queue.**Full**

Bases: Exception

Exception raised by Queue.put(block=0)/put_nowait().

class queue.**Queue**(*maxsize=0*)

Bases: object

Create a queue object with a given maximum size.

If maxsize is <= 0, the queue size is infinite.

task_done()

Indicate that a formerly enqueued task is complete.

Used by Queue consumer threads. For each get() used to fetch a task, a subsequent call to task_done() tells the queue that the processing on the task is complete.

If a join() is currently blocking, it will resume when all items have been processed (meaning that a task_done() call was received for every item that had been put() into the queue).

Raises a ValueError if called more times than there were items placed in the queue.

join()

Blocks until all items in the Queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls task_done() to indicate the item was retrieved and all work on it is complete.

When the count of unfinished tasks drops to zero, join() unblocks.

qsize()

Return the approximate size of the queue (not reliable!).

empty()

Return True if the queue is empty, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() == 0 as a direct substitute, but be aware that either approach risks a race condition where a queue can grow before the result of empty() or qsize() can be used.

To create code that needs to wait for all queued tasks to be completed, the preferred technique is to use the join() method.

full()

Return True if the queue is full, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() >= n as a direct substitute, but be aware that either approach risks a race condition where a queue can shrink before the result of full() or qsize() can be used.

put(*item*, *block=True*, *timeout=None*)

Put an item into the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until a free slot is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Full exception if no free slot was available within that time. Otherwise ('block' is false), put an item on the queue if a free slot is immediately available, else raise the Full exception ('timeout' is ignored in that case).

get(*block=True*, *timeout=None*)

Remove and return an item from the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until an item is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Empty exception if no item was available within that time. Otherwise ('block' is false), return an item if one is immediately available, else raise the Empty exception ('timeout' is ignored in that case).

put_nowait(*item*)

Put an item into the queue without blocking.

Only enqueue the item if a free slot is immediately available. Otherwise raise the Full exception.

get_nowait()

Remove and return an item from the queue without blocking.

Only get an item if one is immediately available. Otherwise raise the Empty exception.

class queue.**PriorityQueue**(*maxsize=0*)

Bases: *Queue*

Variant of Queue that retrieves open entries in priority order (lowest first).

Entries are typically tuples of the form: (priority number, data).

class queue.**LifoQueue**(*maxsize=0*)

Bases: *Queue*

Variant of Queue that retrieves most recently added entries first.

class queue.**SimpleQueue**

Bases: `object`

Simple, unbounded, reentrant FIFO queue.

empty()

Return True if the queue is empty, False otherwise (not reliable!).

get(*block=True*, *timeout=None*)

Remove and return an item from the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until an item is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Empty exception if no item was available within that time. Otherwise ('block' is false), return an item if one is immediately available, else raise the Empty exception ('timeout' is ignored in that case).

get_nowait()

Remove and return an item from the queue without blocking.

Only get an item if one is immediately available. Otherwise raise the Empty exception.

put(*item*, *block=True*, *timeout=None*)

Put the item on the queue.

The optional 'block' and 'timeout' arguments are ignored, as this method never blocks. They are provided for compatibility with the Queue class.

put_nowait(*item*)

Put an item into the queue without blocking.

This is exactly equivalent to *put(item)* and is only provided for compatibility with the Queue class.

qsize()

Return the approximate size of the queue (not reliable!).

1.7 Recipes module

This module describes the commandline or python commands for all the phases of pymemdyn. It consists of:

- Init
- Minimization
- Equilibration
- Relaxation
- Collecting results

class recipes.**BasicInit**(***kwargs*)

Bases: object

class recipes.**LigandInit**(***kwargs*)

Bases: *BasicInit*

class recipes.**LigandAlostericInit**(***kwargs*)

Bases: *LigandInit*

class recipes.**BasicMinimization**(***kwargs*)

Bases: object

class recipes.**LigandMinimization**(***kwargs*)

Bases: *BasicMinimization*

class recipes.**LigandAlostericMinimization**(***kwargs*)

Bases: *BasicMinimization*

class recipes.**BasicEquilibration**(***kwargs*)

Bases: object

class recipes.**LigandEquilibration**(***kwargs*)

Bases: *BasicEquilibration*

class recipes.**LigandAlostericEquilibration**(***kwargs*)

Bases: *LigandEquilibration*

class recipes.**BasicRelax**(***kwargs*)

Bases: object

class recipes.**LigandRelax**(***kwargs*)

Bases: *BasicRelax*

class recipes.**LigandAlostericRelax**(***kwargs*)

Bases: *LigandRelax*

class recipes.**BasicCARelax**(***kwargs*)

Bases: object

class recipes.**BasicBWRelax**(***kwargs*)

Bases: object

class recipes.**BasicCollectResults**(***kwargs*)

Bases: object

class recipes.**BasicCACollectResults**(***kwargs*)

Bases: *BasicCollectResults*

class recipes.**BasicBWCollectResults**(***kwargs*)

Bases: *BasicCollectResults*

1.8 Gromacs module

class gromacs.**Gromacs**(**args*, ***kwargs*)

Bases: object

set_membrane_complex(*value*)

set_membrane_complex: Sets the monomer object

get_membrane_complex()

property membrane_complex

count_lipids(***kwargs*)

count_lipids: Counts the lipids in source and writes a target with N4 tags

get_charge(***kwargs*)

get_charge: Gets the total charge of a system using gromacs grompp command

get_ndx_groups(***kwargs*)

get_ndx_groups: Run make_ndx and set the total number of groups found

get_ndx_sol(***kwargs*)

get_ndx_sol: Run make_ndx and set the last number id for SOL found

make_ndx(***kwargs*)

make_ndx: Wraps the make_ndx command tweaking the input to reflect the characteristics of the complex

make_topol_lipids(***kwargs*)

make_topol_lipids: Add lipid positions to topol.top

manual_log(*command*, *output*)

manual_log: Redirect the output to file in command["options"]["log"] Some commands can't be logged via flag, so one has to catch and redirect stdout and stderr

relax(***kwargs*)

relax: Relax a protein

run_recipe(*debug=False*)

run_recipe: Run recipe for the complex

select_recipe(*stage=''*, *debug=False*)

select_recipe: Select the appropriate recipe for the complex

set_box_sizes()

set_box_sizes: Set length values for different boxes

set_chains(***kwargs*)

set_chains: Set the REAL points of a dimer after protonation

set_grompp(***kwargs*)

set_grompp: Copy template files to working dir

set_itp(***kwargs*)

set_itp: Cut a top file to be usable later as itp

set_options(*options*, *breaks*)

set_options: Set break options from recipe

set_popc(*tgt=''*)

set_popc: Create a pdb file only with the lipid bilayer (POP), no waters. Set some measures on the fly (height of the bilayer)

set_protein_height(***kwargs*)

set_protein_height: Get the z-axis center from a pdb file for membrane or solvent alignment

set_protein_size(***kwargs*)

set_protein_size: Get the protein maximum base width from a pdb file

set_stage_init(***kwargs*)

set_stage_init: Copy a set of files from source to target dir

set_steep(***kwargs*)

set_steep: Copy the template steep.mdp to target dir

set_water(***kwargs*)

set_water: Create a water layer for a box

class gromacs.**Wrapper**(**args*, ***kwargs*)

Bases: object

generate_command(*kwargs*)

generate_command: Receive some variables in kwargs, generate the appropriate command to be run. Return a set in the form of a string "command -with flags"

run_command(*kwargs*)

run_command: Run a command that comes in kwargs in a subprocess, and return the output as (output, errors)

1.9 Groerrors module

exception groerrors.**GromacsError**

Bases: BaseException

exception groerrors.**IOGromacsError**(*command*, *explain*)

Bases: *GromacsError*

Exception raised with "File input/output error" message

class groerrors.**GromacsMessages**(*gro_err=''*, *command=''*, **args*, ***kwargs*)

Bases: object

Load an error message and split it along as many properties as possible

```
e = {'Can not open file':  <class 'groerrors.IOGromacsError'>, 'File input/output
error':  <class 'groerrors.IOGromacsError'>, 'srun:  error:  Unable to create job
step':  <class 'groerrors.IOGromacsError'>}
```

check()

Check if the GROMACS error message has any of the known error messages. Set the self.error to the value of the error

1.10 Broker module

This is a lame broker (or message dispatcher). When Gromacs enters a run, it should choose a broker from here and dispatch messages through it.

Depending on the broker, the messages may be just printed or something else

class broker.**Printing**

Bases: object

dispatch(*msg*)

Simply print the msg passed

1.11 Utils module

utils.**clean_pdb**(*src=[]*, *tgt=[]*)

Remove incorrectly allocated atom identifiers in pdb file

utils.**clean_topol**(*src=[]*, *tgt=[]*)

Clean the src topol of path specifics, and paste results in target

utils.**concat**(***kwargs*)

Make a whole pdb file with all the pdb provided

utils.**getbw**(***kwargs*)

Call the Ballesteros-Weistein based pair-distance restraint module.

utils.**make_cat**(*dir1*, *dir2*, *name*)

Very tight function to make a list of files to inject in some GROMACS suite programs

utils.**make_ffoplsaanb**(*complex=None*)

Join all OPLS force fields needed to run the simulation

utils.**make_topol**(*template_dir='/home/rebecca/Documents/8STAGE/pymemdyn/templates'*, *target_dir=''*, *working_dir=''*, *complex=None*)

Make the topol starting from our topol.top template

utils.**tar_out**(*src_dir=[]*, *tgt=[]*)

Tar everything in a src_dir to the tar_file

1.12 Settings module

This module handles the local settings for pymemdyn on your machine. The settings are mostly paths and run settings.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

I

J

L

M

N

O

P

Q

R

S

T

U

W