# pymemdyn

***Release 1.6.1***

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

**Mar 29, 2023**

# CONTENTS:

CHAPTER

ONE

# PYMEMDYN VERSION 1.6.1

PyMemDyn is a standalone *python* package to setup membrane molecular dynamics calculations using the **GROMACS** set of programs. The package can be used either in a desktop environment, or in a cluster with popular queuing systems such as Torque/PBS or Slurm.

**PyMemDyn** is hosted in github at:

https://github.com/GPCR-ModSim/pymemdyn

You can download any version of **PyMemDyn** by cloning the repository to your local machine using git.

You will need to create a free personal account at github and send and e-mail to: gpcruser@gmail.com requesting access to the code. After request processing from us you will be given access to the free repository.

## 1.1 Dependencies

**GROMACS**
Pymemdyn is dependent on GROMACS. Download GROMACS here. Instructions for installation are here.

**LigParGen**
In order to automatically generate .itp files for ligands and allosterics, the program ligpargen is used. Install using their instructions: https://github.com/Isra3l/ligpargen. Do not forget to activate the conda environment in which you installed ligpargen (conda install py37 if you followed the instructions) before running pymemdyn. In case you are using a bash script, this should be done inside the script. See also "ligpargen_example" in the folder examples.

Testing was done using LigParGen v2.1 using BOSS5.0.

Pymemdyn can also be used without ligpargen installation, but then .itp files containing the parameters for the ligand and the allosteric should be provided in the same folder as their respective .pdb's.

**Queueing system**
A queuing system: although not strictly required, this is highly advisable since an MD simulation of 2.5 nanoseconds will be performed. However, if only membrane insertion and energy minimization is requested, this requirement can be avoided. Currently, the queuing systems supported include Slurm and PBS.

## 1.2 Installation

To install **PyMemDyn** follow these steps:

1. Clone **PyMemDyn** for python 3.7:

```
git clone https://username@github.com/GPCR-ModSim/pymemdyn.git
```

   Make sure to change *username* to the one you have created at github.

2. The previous command will create a *pymemdyn* directory. Now you have to tell your operating system how to find that folder. You achieve this by declaring the location of the directory in a .bashrc file .cshrc or .zshrc file in your home folder. An example of what you will have to include in your .bashrc file follows:

```
export PYMEMDYN=/home/username/software/pymemdyn
export PATH=$PYMEMDYN:$PATH
```

   or if your shell is csh then in your .cshrc file you can add:

```
setenv PYMEMDYN /home/username/software/pymemdyn
set path = ($path $PYMEMDYN)
```

   Notice that I have cloned *pymemdyn* in the software folder in my home folder, you will have to adapt this to wherever it is that you downloaded your *pymemdyn* to.

   After including the route to your *pymemdyn* directory in your .bashrc file make sure to issue the command:

```
source .bashrc
```

   or open a new terminal.

   To check if you have defined the route to the *pymemdyn* directory correctly try to run the main program called pymemdyn in a terminal:

```
pymemdyn --help
```

   You should obtain the following help output:

```
usage: pymemdyn [-h] [-v] [-b OWN_DIR] [-r REPO_DIR] -p PDB [-l LIGAND]
                [-a ALOSTERIC] [-w WATERS] [-i IONS] [-c CHO]
                [--res RESTRAINT] [-q QUEUE] [-d]

== Setup Molecular Dynamics for Membrane Proteins given a PDB. ==

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -b OWN_DIR            Working dir if different from actual dir
  -r REPO_DIR           Path to templates of fixed files. If not provided,
                        take the value from settings.TEMPLATES_DIR.
  -p PDB                Name of the pdb to insert into membrane for MD
                        (mandatory). Use the pdb extension. (e.g. -p
                        myprot.pdb)
  -l LIGAND, --lig LIGAND
                        Name of the ligand, without extension. See
                        input_guide.txt for details on how to generate the
```

(continues on next page)

(continued from previous page)

```
                    required pdb and forcefield files.
  -a ALLOSTERIC, --alo ALLOSTERIC
                    Name of the allosteric, without extension. See
                    input_guide.txt for details on how to generate the
                    required pdb and forcefield files.
  -w WATERS, --waters WATERS
                    Crystalized water molecules. File name without
                    extension.
  -i IONS, --ions IONS  Crystalized ions file name without extension.
  -c CHO, --cho CHO     Crystalized cholesterol molecules file name without
                    extension.
  --res RESTRAINT       Position restraints during MD production run. Options:
                    bw (Ballesteros-Weinstein Restrained Relaxation -
                    default), ca (C-Alpha Restrained Relaxation)
  --llc LIGPARGEN_LIGAND_CHARGE
                    Charge of ligand for ligpargen (when itp file should
                    be generated)
  --llo LIGPARGEN_LIGAND_NROFOPTIMIZATIONS
                    Number of optimizations that ligpargen should use to
                    generate itp file for ligand (only needed when itp is
                    not provided)
  --lac LIGPARGEN_ALLOSTERIC_CHARGE
                    Charge of allosteric for ligpargen (when itp file
                    should be generated)
  --lao LIGPARGEN_ALLOSTERIC_NROFOPTIMIZATIONS
                    Number of optimizations that ligpargen should use to
                    generate itp file for allosteric (only needed when itp
                    is not provided)
  -q QUEUE, --queue QUEUE
                    Queueing system to use (slurm, pbs, pbs_ib and svgd
                    supported)
  -d, --debug
```

3. Updates are very easy thanks to the git versioning system. Once **PyMemDyn** has been downloaded (cloned) into its own *pymemdyn* folder you just have to move to it and pull the newest changes:

```
cd /home/username/software/pymemdyn
git pull
```

4. You can also clone older stable versions of **PyMemDyn**. For example the stable version 1.4 which works well and has been tested extensively again GROMACS version 4.6.7 can be cloned with:

```
git clone https://username@github.com/GPCR-ModSim/pymemdyn.git \
--branch stable/1.4 --single-branch pymemdyn-1.4
```

   Now you will have to change your .bashrc or .cshrc files in your home folder accordingly.

5. To make sure that your GROMACS installation is understood by **PyMemDyn** you will need to specify the path to where GROMACS is installed in your system. To do this you will need to edit the settings.py file with any text editor (vi and emacs are common options in the unix environment). Make sure that only one line is uncommented, looking like: GROMACS_PATH = /opt/gromacs-2021/bin Provided that in your case gromacs is installed in /opt. The program will prepend this line to the binaries names, so calling /opt/gromacs-2021/bin/gmx should point to that binary.

## 1.3 Modules

### 1.3.1 Modeling Modules

The following modules define the objects to be modeled.

- **protein.py**. This module defines the ProteinComplex, Protein, Monomer, Dimer, Compound, Ligand, CrystalWaters, Ions, Cholesterol, Lipids, and Alosteric objects. These objects are started with the required files, and can then be passed to other objects.
- **membrane.py**. Defines the cellular membrane.
- **complex.py**. Defines the full complex, protein + membrane. It can include any of the previous objects.

### 1.3.2 Auxiliary Modules

- **queue.py**. Queue manager. That is, it receives objects to be executed.
- **recipes.py**. Applies step by step instructions for carrying a modeling step.
- **bw4posres.py**. Creates a set of distance restraints based on Ballesteros-Weinstein identities which are gathered by alignment to a multiple-sequence alignment using clustalw.
- **utils.py**. Puts the functions done by the previous objects on demand. For example, manipulate files, copy folders, call functions or classes from standalone modules like bw4posres.py, etc.
- **settings.py** This modules sets up the main environment variables needed to run the calculation, for example, the path to the gromacs binaries.

### 1.3.3 Execution Modules

- **gromacs.py**. Defines the Gromacs and Wrapper objects. * Gromacs will load the objects to be modeled, the modeling recipe, and run it. * Wrapper is a proxy for gromacs commands. When a recipe entry is sent to it this returns the command to be run.

### 1.3.4 Executable

- **pymemdyn** The main program to call which sends the run to a cluster.

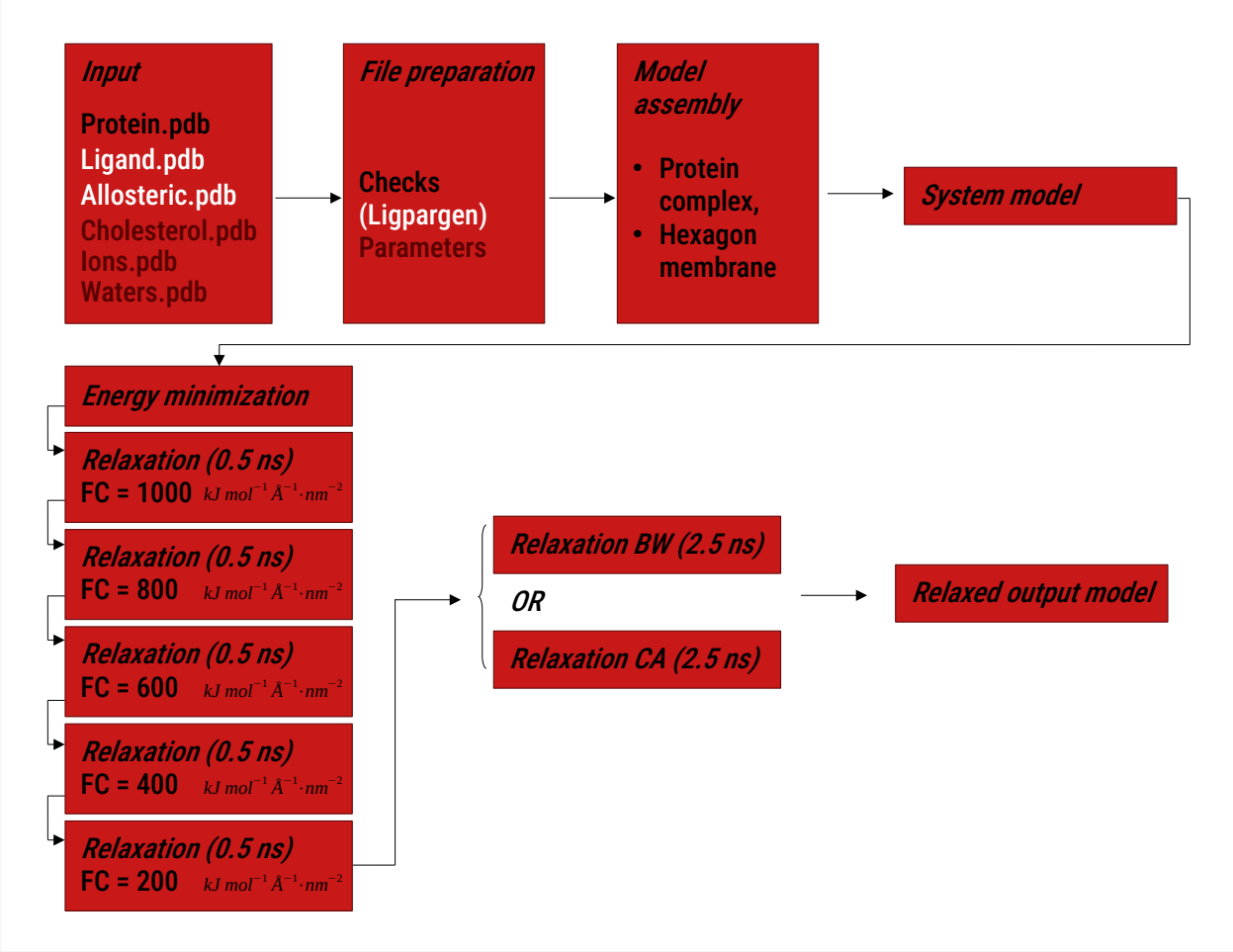
More information about all modules can be found in the Modules chapter.

CHAPTER

# TWO

# MANUAL

The fully automated pipeline available by using **PyMemDyn** allows any researcher, without prior experience in computational chemistry, to perform an otherwise tedious and complex process of membrane insertion and thorough MD equilibration, as outlined in Figure 1.




In the simplest scenario, only the receptor structure is considered. In such case the GPCR is automatically surrounded by a pre-equilibrated POPC (Palmitoyl-Oleoyl-Phosphatidyl-Choline) membrane model in a way that the TM (Trans-Membrane) bundle is parallel to the vertical axis of the membrane. The system is then soaked with bulk water and inserted into an hexagonal prism-shaped box, which is energy-minimized and carefully equilibrated in the framework of periodic boundary conditions (PBC). A thorough MD equilibration protocol lasting 2.5 ns follows.

But the simulation of an isolated receptor can only account for one part of the problem, and the influence of dif-

ferent non-protein elements in receptor dynamics such as the orthosteric (primary) ligand, allosteric modulator, or even specific cholesterol, lipid, water or ion molecules are key for a more comprehensive characterization of GPCRs. **PyMemDyn** can explicitly handle these elements allowing a broader audience in the field of GPCRs to use molecular dynamics simulations. These molecules should be uploaded in the same way they're present in the original PDB file of the receptor, so they are properly integrated into the membrane insertion protocol described above, together with force-field associated files (which can be either generated with external software like Macromodel, or by manual parameterization). In addition, it is also possible to perform MD simulations of receptor dimers, provided that a proper dimerization model exists (i.e., coming from X-ray crystallography or from a protein-protein docking protocol). The ease of use, flexibility and public availability of the **PyMemDyn** library makes it a unique tool for researchers in the GPCR field interested in exploring dynamic processes of these receptors.

## 2.1 Running with queues

$approx$ 90% of the time you will want to use some queueing system. We deal with queue systems tweaks as we stumble into them and it's out of our scope to cover them all. If you take a look at the source code dir, you'll found some files called "run_pbs.sh", "run_svgd.sh" and so on. Also there are specific queue objects in the source file queue.py we have to tweak for every and each queue. In you want to run your simulation in a supported queue, copy the "run_queuename.sh" file to your working directory, and edit it. E.g. the workdir to run an A2a.pdb simulation in svgd.cesga.es looks like: . .. A2a.pdb run_svgd.sh And run_svgd.sh looks like:

```
\$!/bin/bash
module load python/3.7
module load gromacs/2021
python ~/bin/pymoldyn/pymemdyn -p a2a.pdb
```

Now we just launch this script with:

```
qsub -l arch=amd,num_proc=1,s_rt=50:00:00,s_vmem=1G,h_fsize=1G -pe mpi 8 run_svgd.sh
```

and wait for the results. Note that we launch 1 process, but flag the run as mpi with reservation of 8 cores in SVGD queue.

## 2.2 Debugging

To also log 'debug'-messages to the log file (log.log), activate the debug mode with –debug:

```
pymemdyn -p protein.pdb --debug
```

If you are to set up a new system, it is a good idea to just run a few steps of each stage in the equilibration protocol just to test that the pdb file is read correctly and the membrane-insertion protocol works fine and the system can be minimized and does not "explode" during the equilibration protocol (i.e., detect if atom clashes and so on exist on your system).

To do this, use the –debugFast option, like:

```
pymemdyn -p gpcr.pdb -l lig --waters hoh --debugFast
```

If everythings works fine, you will see the list of output directories and files just as in a regular equilibration protocol, but with much smaller files (since we only use here 1000 steps of MD in each stage). NOTE that sometimes, due to the need of a smooth equilibration procedure (i.e. when a new ligand is introduced in the binding site without further refinement of the complex, or with slight clashes of existing water molecules) this kind of debugging equilibration procedure might crash during the first stages due to hot atoms or LINCS failure. This is normal, and you have two

options: i) trust that the full equilibration procedure will fix the steric clashes in your starting system, and then directly run the pymemdyn without the debugging option, or ii) identify the hot atoms (check the mdrun.log file in the last subdirectory that was written in your output (generally eq/mdrun.log and look for "LINCS WARNING"). What if you want to check partial functions of pymoldyn? In order to do this you must edit the file pymemdyn and change:

1. Line 260 comment with "#" this line [that states: run.clean()], which is the one that deletes all the output files present in the working directory.
2. In the last two lines of this file, comment (add a "#") the line: run.moldyn()
3. And uncomment (remove the "#") the line: run.light_moldyn()
4. In the line 153 and within that block (ligh_moldyn) change the lines stating steps = ["xxxx"] and include only those steps that you want to test, which should be within a list of strings.

For the sake of clarity, these have been subdivided in two lines:

```
line 1-  steps = ["Init", "Minimization", "Equilibration", "Relax", "CARelax"]
```

Here you remove those strings that you do not want to be executed, i.e. if only membrane insertion and minimization is wished, remove "Equilibration", "Relax", "CARelax" so the line states:

```
steps = ["Init", "Minimization"]

line 2-  steps = ["CollectResults"]
```

This only accounts for the preparation of the output files for analysis, so if you only are interested on this stage, comment the previous line. The last assignment is the one that runs. NOTE that you must know what you do, otherwise you might have crashes in the code if needed files to run intermediate stages are missing!

## 2.3 Output

The performed equilibration includes the following stages:

Table 1: Output

| STAGE | RESTRAINED ATOMS | FORCE CONSTANT | TIME |
|---|---|---|---|
| | | kJ/(mol A ·nm^2) | ns |
| Minimization | | | (Max. 500 steps) |
| Equil. 1 | Protein Heavy Atoms | 1000 | 0.5 |
| Equil. 2 | Protein Heavy Atoms | 800 | 0.5 |
| Equil. 3 | Protein Heavy Atoms | 600 | 0.5 |
| Equil. 4 | Protein Heavy Atoms | 400 | 0.5 |
| Equil. 5 | Protein Heavy Atoms | 200 | 0.5 |
| Equil. 6 | Venkatakrishnan Pairs / | 200 / | 2.5 / |
| Equil. 6 | C-alpha Atoms | 200 | 2.5 |

In this folder you will find several files related to this simulation:

### 2.3.1 INPUT:

```
- popc.itp              # Topology of the lipids
- ffoplsaa_mod.itp      # Modified OPLSAA-FF, to account for lipid modifications
- ffoplsaabon_mod.itp   # Modified OPLSAA-FF(bonded), to account for lipid modifications
- ffoplsaanb_mod.itp    # Modified OPLSAA-FF(non-bonded), to account for lipid modifications
- topol.tpr             # Input for the first equilibration stage
- topol.top             # Topology of the system
- protein.itp           # Topology of the protein
- index.ndx             # Index file with appropriate groups for GROMACS
- prod.mdp              # Example of a parameter file to configure a production run (see TIPS)
```

### 2.3.2 STRUCTURES:

```
- hexagon.pdb          # Initial structure of the system, with the receptor centered in the box
- confout.gro          # Final structure of the system (see TIPS)
- load_gpcr.pml        # Loads the initial structure and the trajectory in pymol
```

### 2.3.3 TRAJECTORY FILES:

```
- traj_pymol.xtc       # Trajectory of the whole system for visualization in pymol. 1 snapshot/100 ps
- traj_EQ.xtc          # Trajectory of the whole system in .xtc format: 1 snapshot/50 ps
- ener_EQ.edr          # Energy file of the trajectory
- load_gpcr.pml        # Script to load the equilibration trajectory in pymol.
```

### 2.3.4 REPORTS:

In the "reports" subfolder, you will find the following files:

- tot_ener.xvg, tot_ener.log # System total energy plot and log - temp.xvg, temp.log # System temperature plot and log - pressure.xvg, pressure.log # System pressure plot and log - volume.xvg, volume.log # System volume plot and log - rmsd-all-atom-vs-start # All atoms RMSD plot - rmsd-backbone-vs-start.xvg # Backbone RMSD plot - rmsd-calpha-vs-start.xvg # C-Alpha RMSD plot - rmsf-per-residue.xvg # Residue RMSF plot

## 2.4 LOGS:

The logger in pymemdyn will write log messages to the file log.log, which is regenerated every run.

In the "logs" subfolder, you will find the log files of mdrun:

- eq_{force_constant}.log # log of stages with restrained heavy atoms of the receptor - eqCA.log # log of the stage with restrained C-alfa atoms of the receptor

**NOTE ON GROMACS METHODS** To integrate the equations of motion we have selected the leap-frog integrator with a 2 femtosecond timestep. Longe-range electrostatic interactions in periodic boundary conditions are treated with the particle mesh Ewald method. We use a Nose-Hoover thermostat with a tau_t of 0.5 picoseconds and a Parinello-Rahman barostat with a tau_p of 2.0. The pressure coupling is semiisotropic, meaning that it's isotropic in the x and y

directions but different in the z direction. Since we are using pressure coupling we are working with an NPT ensemble. This is done both in the all-atom restrained steps and in the alpha-carbon atom restrained part. All of these details are more explicitly stated in the Rodriguez et al. [1] publication.

**TIPS**

NOTE: these tips work for GROMACS version >= 4.5 and < 5.0. For later versions, adjustments are required, but the principle remains the same.

- If you want to configure a .tpr input file for a **production** run, you can use the template 'prod.mdp' file by introducing the number of steps (nsteps), and thus the simulation time, you want to run.

After that, you just have to type:

```
grompp -f prod.mdp -c confout.gro -p topol.top -n index.ndx -o topol_prod.tpr
mdrun -s topol_prod.tpr -o traj.trr -e ener.edr -c confout.gro -g production.log -x traj_prod.xtc
```

- If you want to create a PDB file of your system after the equilibration, with the receptor centered in the box, type:

  echo 1 0 | trjconv -pbc mol -center -ur compact -f confout.gro -o confout.pdb

- If you want to create an xmgrace graph of the root mean square deviation for c-alpha atoms in the 5.0 ns of simulation you can use:

  echo 3 3 | g_rms -f traj_EQ.xtc -s topol.tpr -o rmsd-calpha-vs-start.xvg

- You may want to get a pdb file of your last frame. You can first check the total time of your trajectory and then use this time to request the last frame with:

  gmxcheck -f traj_pymol.xtc echo 1 | trjconv -b 5000 -e 5000 -f traj_pymol.xtc -o last51.pdb

## 2.5 References

[1] Rodríguez D., Piñeiro A. and Gutiérrez-de-Terán H.
Molecular Dynamics Simulations Reveal Insights into Key Structural Elements of Adenosine Receptors
Biochemistry (2011), 50, 4194-208.

CHAPTER

THREE

# MODULES

## 3.1 Pymemdyn

This is the main script for the pymemdyn commandline tool. In this script the following things are accomplished:

1. Command line arguments are parsed.
2. (If necessary) a working directory is created.
3. (If necessary) previous Run files are removed.
4. A run is done.

### 3.1.1 Usage

```
usage: pymemdyn [-h] [-v] [-b OWN_DIR] [-r REPO_DIR] -p PDB [-l LIGAND]
            [-a ALLOSTERIC] [-w WATERS] [-i IONS] [-c CHO]
            [--res RESTRAINT] [--llc LIGPARGEN_LIGAND_CHARGE]
            [--llo LIGPARGEN_LIGAND_NROFOPTIMIZATIONS]
            [--lac LIGPARGEN_ALLOSTERIC_CHARGE]
            [--lao LIGPARGEN_ALLOSTERIC_NROFOPTIMIZATIONS] [-q QUEUE] [-d]

== setup molecular dynamics for membrane proteins given a pdb. ==

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -b OWN_DIR            Working dir if different from actual dir
  -r REPO_DIR           Path to templates of fixed files. If not provided,
                        take the value from settings.TEMPLATES_DIR.
  -p PDB                Name of the pdb to insert into membrane for MD
                        (mandatory). Use the pdb extension. (e.g. -p
                        myprot.pdb)
  -l LIGAND, --lig LIGAND
                        Name of the ligand, without extension. See
                        input_guide.txt for details on how to generate the
                        required pdb and forcefield files.
  -a ALLOSTERIC, --alo ALLOSTERIC
                        Name of the allosteric, without extension. See
                        input_guide.txt for details on how to generate the
                        required pdb and forcefield files.
```

(continues on next page)

(continued from previous page)

```
-w WATERS, --waters WATERS
                      Crystalized water molecules. File name without
                      extension.
-i IONS, --ions IONS  Crystalized ions file name without extension.
-c CHO, --cho CHO     Crystalized cholesterol molecules file name without
                      extension.
--res RESTRAINT       Position restraints during MD production run. Options:
                      bw (Ballesteros-Weinstein Restrained Relaxation -
                      default), ca (C-Alpha Restrained Relaxation)
--llc LIGPARGEN_LIGAND_CHARGE
                      Charge of ligand for ligpargen (when itp file should
                      be generated)
--llo LIGPARGEN_LIGAND_NROFOPTIMIZATIONS
                      Number of optimizations that ligpargen should use to
                      generate itp file for ligand (only needed when itp is
                      not provided)
--lac LIGPARGEN_ALLOSTERIC_CHARGE
                      Charge of allosteric for ligpargen (when itp file
                      should be generated)
--lao LIGPARGEN_ALLOSTERIC_NROFOPTIMIZATIONS
                      Number of optimizations that ligpargen should use to
                      generate itp file for allosteric (only needed when itp
                      is not provided)
-q QUEUE, --queue QUEUE
                      Queueing system to use (slurm, pbs, pbs_ib and svgd
                      supported)
-d, --debug
```

## 3.2 Run module

class run.Run(*pdb*, **args*, ***kwargs*)

Bases: object

clean()

Removes all previously generated files

moldyn()

Run all steps in a molecular dynamics simulation of a membrane protein

light_moldyn()

This is a function to debug a run in steps

## 3.3 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

## 3.4 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.

## 3.5 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 assign 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.

## 3.6 Complex module

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

Bases: object

setMembrane(*membrane*)

Set the membrane pdb file

getMembrane()

setComplex(*complex*)

Set the complex object

getComplex()

## 3.7 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.

## 3.8 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

## 3.9 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(*debugFast=False*)

run_recipe: Run recipe for the complex

select_recipe(*stage=''*, *debugFast=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)

## 3.10 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

## 3.11 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

## 3.12 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/rkupper/apps/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

## 3.13 Settings module

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

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX

## T



## U



## W