

py-MEMdyn

Python Membrane Dynamics

Python - Membrane Dynamics

A Python library for Molecular Dynamics simulations in biological membranes

Manual V. 1.1

Xabier Bello, Mauricio Esguerra, David Rodríguez, Hugo Gutiérrez de Terán

hugo.gutierrez@icm.uu.se

py-MEMdyn is a python library developed to automate the process of setting up the simulation, via Molecular Dynamics (MD), of G-protein Coupled Receptors (GPCR's) embedded in a cell membrane. The protocol can be adapted to insert other transmembrane proteins, not only GPCR's. The library has been adapted from that described in Gutiérrez de Terán et al. (2011) [1], and is implemented in the web-based service for modeling and simulation of GPCR's available at <http://gpcr-modsim.org>.

- DESCRIPTION

The fully automated pipeline available by using **py-MEMdyn** 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.

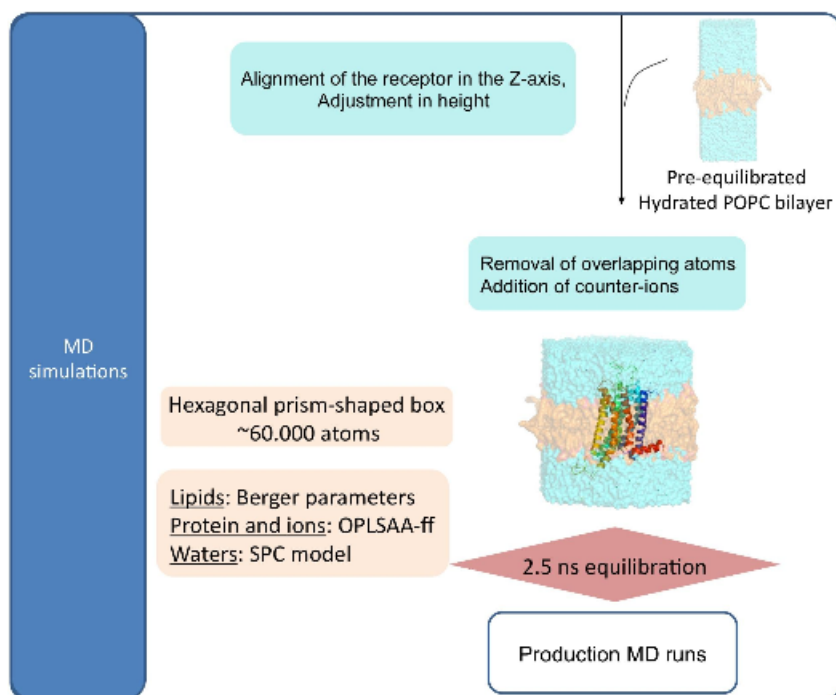


Figure 1: Pipeline of the process followed to embed a G-protein Coupled Receptor into a cell membrane made of a POPC (Palmitoyl-Oleoyl-Phosphatidyl-Choline) phospholipid bilayer.

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 different 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 GPCR's. **py-MEMdyn** can explicitly handle these elements allowing a broader audience in the field of GPCR's 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 **py-MEMdyn** library makes it a unique tool for researchers in the GPCR field interested in exploring dynamic processes of these receptors.

- INSTALLATION

py-MEMdyn is a python library that can be used in any unix platform provided that the following dependencies are installed:

- git (for downloading)
- Python 2.7
- Gromacs 4.0.5 for version 1.0
- Gromacs 4.6.5 for version 1.1
- 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.

py-MEMdyn is hosted in a bitbucket repository at:

<https://bitbucket.org/gpcrmodsim/pymemdyn.git>

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

You will need to create a free personal account at bitbucket and send an e-mail to: gpcruser@gmail.com and you will be given access to the free repository.

To install **py-MEMdyn** follow these steps:

1. Clone the current version of **py-MEMdyn**

```
git clone https://username@bitbucket.org/gpcrmodsim/pymemdyn.git
```

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

2. The previous command will create a *pymemdyn* directory. You now 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 or *.cshrc* 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 *run.py* in a terminal:

```
run.py --help
```

You should obtain the following help output:

```
usage: run.py [-h] [-b OWN_DIR] [-r REPO_DIR] -p PDB [-l LIGAND]
              [--alo ALOSTERIC] [--waters WATERS] [--ions IONS] [--cho CHO]
              [-q QUEUE] [--debug]
```

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

optional arguments:

-h, --help	show this help message 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.REPO_DIR.
-p PDB	Name of the pdb to insert into MD (mandatory)
-l LIGAND	Name of the ligand, without extension. Three files must be present along with the molecule pdb: the ligand, its itp and its force field.
--alo ALOSTERIC	Name of the allosteric interaction, without extension. Three files must be present along with the molecule pdb: the allosteric, its itp and its force field.
--waters WATERS	Crystalized water molecules. File name without extension.
--ions IONS	Crystalized ions file name without extension.
--cho CHO	Crystalized cholesterol molecules file name without extension.
-q QUEUE	Queueing system to use (slurm, pbs, pbs_ib and svgd supported)
--debug	

3. Updates are very easy thanks to the git versioning system. Once **py-MEMdyn** has been downloaded 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 **py-MEMdyn**. For example the stable version 1.0 which works well and has been tested extensively against gromacs version 4.0.5 can be cloned with:

```
git clone https://username@bitbucket.org/gpcrmodsim/pymemdyn \
--branch stable/1.0 --single-branch pymemdyn-1.0
```

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 **py-MEMdyn** 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/gromacs405/bin` Provided that in your case gromacs is installed in */opt*. The program will prepend this line to the binaries names, so calling “*/opt/gromacs405/bin/grompp*” should point to that binary.
6. Similarly, in that file you specify which queuing system you are using. We will assume that you will use “slurm”, but you can use any of the possible options by uncommenting the one that best suits your system.

- **COMPULSORY!** Option `-p`: In the simplest case, **py-MEMdyn** only needs a `pdb` file with the receptor. This should be readable by Gromacs (i.e., accomplish the PDB standards, see the GROMACS manual for details) and accessed from the working directory. And as the error says, this is the bare minimum to perform an MD simulation! we will assume that the file is called `gpcr.pdb` Thus `run.sh -p gpcr.pdb` should work! Accessory options: The common user should not take care of the `-b`, `-r` or `-debug` options: The working directory (`OWN_DIR`) is set by default to that where the program is invoked, and the repository (`REPO_DIR`) is specified in the `settings.py` file (by default, `templates/` subdirectory in the **py-MEMdyn** instalation).
- Considering non-protein elements: ligand(s), structural waters, structural lipids, cholesterol molecules, explicit ions.
 - Option `-l` (specifying an orthosteric ligand). Lets assume that we have docked a ligand in the orthosteric binding site. We can include this in the simulation as long as we have generated the requested library and parameter files in gromacs. Thus, 3 files are needed, that should share a root name (i.e., `lig`):
 - * `lig.pdb`: a standard `pdb` file where the atom names are explicitly considered in the `itp` and `ff` files (see bellow)
 - * `lig.itp`: we refer to as the library file, and collects the atom charges and the bonded parameters (i.e., bonds, angles, dihedrals and torsions) as derived with the OPLS forcefield in the Gromacs standard nomenclature.
 - * `lig.ff`: we will refer to as the “force-field file”, which collects the OPLS2005 atom types and non-bonded parameters in the Gromacs standard nomenclature. For the users used to Gromacs, this file is generally non-existing and the parameters listed here are merged onto the standard forcefield file in gromacs (i.e. `ffoplsaa.itp`). But in our protocol, this is needed as a separate file.
`run.sh -p gpcr.pdb -l lig`
 - Option `-alo` (specifying an allosteric ligand): This should be treated exactly the same as the ligand: if the allosteric modulator is called `allo`, we need to have `alo.pdb`, `alo.itp` and `alo.ff`. `run.sh -p gpcr.pdb -alo allo`
 - Option `-water` (specifying structural waters): If structural waters are present (i.e., coming from an x-ray structure) these should be included as a separate `pdb` file (i.e. `hoh.pdb`). The corresponding `itp` file (`hoh.itp`) is also needed, but in this case the `ff` file is avoided as the parameters are in the standard forcefield.
 - Option `-ions` (specifying structural ions) If we want to consider structural ions (i.e., the sodium ion in the A2A high resolution structure 4EIY), we should name the needed files i.e. `ion-local` and provide the same information as in the case of structural waters: `ion-local.pdb` and `ion-local.itp`.
 - Option `-cho` (specifying cholesterol molecules): As in the previous case, the `pdb` and `itp` files are needed (i.e., `cho.pdb` and `cho.itp`).
 - Option `-queue`: if other queue than the default one, indicated in the `settings.py`, is needed (in our example, `slurm` in `cuelebre`) this should be indicated in the option. Possibilities are listed in the `settings.py` file:
 - * `slurm` as implemented in `cuelebre`
 - * `pbs` as implemented in `garibaldi.scripps.edu`
 - * `pbs_ib` infiniband, as implemented in `garibaldi.scripps.edu`
 - * `svgd`, as implemented in `svgd.cesga.es`

To summarize, the following command should work in the most complex case, `run.sh -p gpcr.pdb -l lig -waters hoh -alo allo -cho cho`

RUNNING WITH QUEUES

≈ 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_queueename.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/2.7.3
module load gromacs/4.0.7
python ~/bin/pymoldyn/run.py -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.

- DEBUGGING

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 `--debug` option, like:

```
run.sh -p gpcr.pdb -l lig --waters hoh --debug
```

If everything 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 pymoldyn 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 run.py and change:

1. Line 211 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 140 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!

STAGE	RESTRAINTS	FORCE CON- STANTS	TIME (ns)
Minimization			(\approx 500 steps)
Equilibration	Protein Heavy Atoms	1000	0.5
		800	0.5
		600	0.5
		400	0.5
		200	0.5
		200	2.5

Table 1: Simulation steps performed by default using pyMEMdyn

- OUTPUT

The performed equilibration includes the following stages:

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

INPUTS:

```
- 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
- ffoplsaانب_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_example.mdp    # Example of a parameter file to configure a production phase (see TIPS)
```

STRUCTURES:

```
- hexagon.pdb        # Initial structure of the system, with the receptor centered in the box
- confout.gro         # Final structure of the system (see TIPS)
```

TRAJECTORY FILES

```
- traj_EQ.xtc        # Trajectory of the whole system in .xtc format: 1 snapshot/50 ps
- ener_EQ.edr         # Energy file of the trajectory
```

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
```

LOGS:

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
```

- TIPS

- If you want to configure a .tpr input file for production phase, you can use the template 'prod.mdp' file by introducing the number 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
```

- 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` NOTE: these tips work for GROMACS version 4.0.5.

REFERENCES

- [1] D. Rodríguez, A. Piñeiro, and H. Gutiérrez-de-Terán. 2011, Molecular dynamics simulations reveal insights into key structural elements of adenosine receptors. *Biochemistry*, **50**, 4194-4208.