Pymemdyn

A program to perform GPCR MD simulations in the membrane

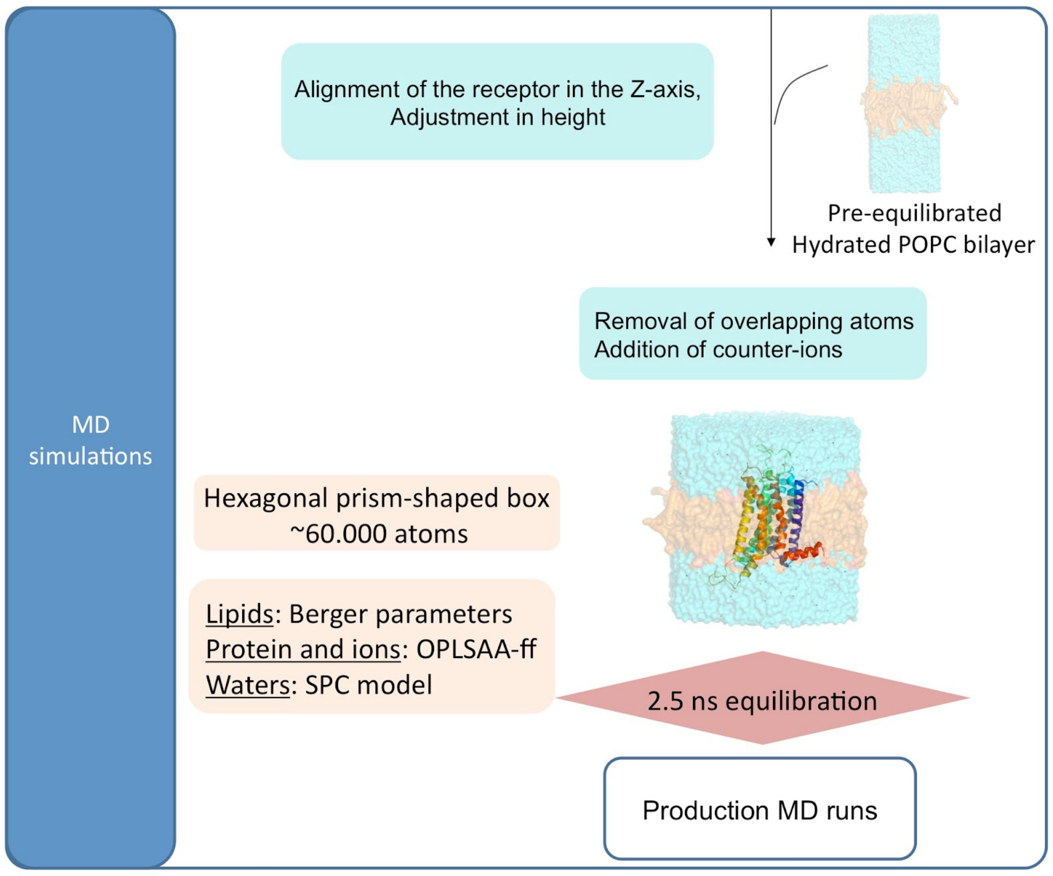
Manual v1.0

Xabier Bello, David Rodríguez and Hugo Gutiérrez-de-Terán

Description: Pymemdyn is a python program written to automate the process of membrane insertion and molecular dynamics (MD) equilibration of GPCR structures. The protocol is adapted from that described in the following reference, and is implemented in the web-based service for the modeling and simulation of GPCRs GPCR-ModSim.

This fully automated pipeline allows any researcher, with no prior experience in computational chemistry, to perform the otherwise tedious and complex process of membrane insertion and thorough MD equilibration, as outlined in Fig. 1. In the simplest scenario, only the receptor structure is considered. This will be automatically embedded in a pre-equilibrated POPC membrane model in a way that the TM bundle is parallel to the membrane vertical axis. The system is then soaked with bulk water and inserted into a hexagonal-prism shaped box, which is energy-minimized and carefully equilibrated in the framework of periodic boundary conditions (PBC). It follows a thorough MD equilibration protocol that lasts 2.5 ns. 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 ligand, an allosteric modulator, or even specific cholesterol, lipid, water or ion molecules is key for a more comprehensive characterization of GPCRs. To allow a broader use of MD simulations to researchers in the field, PyMemDyn can explicitly handle these elements. Each molecule should be uploaded as docked to the original PDB file of the receptor, so they are properly integrated into the membrane insertion protocol described above, together with the force-field associated files (which can be either generated with external software like Macromodel, either 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 module makes it a unique tool for researchers in the GPCR field interested in exploring dynamic processes of these receptors.

Fig1: General scheme of


MD simulations
Alignment of the receptor in the Z-axis, Adjustment in height
Pre-equilibrated Hydrated POPC bilayer
Removal of overlapping atoms
Addition of counter-ions
Hexagonal prism-shaped box
~60.000 atoms
Lipids: Berger parameters
Protein and ions: OPLSAA-ff
Waters: SPC model
2.5 ns equilibration
Production MD runs

Pymemdyn

Installation

Pymemdyn is a command line program that can be used in any unix platform, provided that the dependencies are installed. These include:

- A unix-line text terminal.
- ssh for secure-shell connection.
- git (repository access).
- Python (required >= 2.7).
- Gromacs (currently only version 4.0.5 is supported).
- Queuing system: although not strictly required, this is highly advisable since a 5 ns MD simulation will be performed. However, if only membrane insertion and energy

minimization is requested, this requirement can be avoided.[1] Currently, the queuing systems supported include Slurm, PBS.

In order to install the last version, the user must have granted access to the repository, located at the cluster "Cuelebre" at USC. The following steps are needed though. It is assumed a unix-like terminal and that the ssh protocol for communications is enabled in the user computer.

1. Have a user created in Trasgu (trasgu.usc.es, only accessible by ssh) and in Cuelebre (i.e., our cluster) (i.e., username).

2. Create a tunnel through trasgu:
   ```
   ssh -l username -L 7777:cuelebre.inv.usc.es:22 trasgu.usc.es
   ```
   Adding the `-f -N` flags to that line will force the tunnel to go background:
   ```
   ssh -f -N -l username -L 7777:cuelebre.inv.usc.es:22 trasgu.usc.es
   ```

3. (This step only needed the first time that we download the pymemdyn program) Create a local copy of the repository: Situate the cursor in the local directory where you want the program to be installed (we will assume /home/localuser) and type:
   ```
   git clone ssh://username@localhost:7777/home/slurm/pymoldyn/share 
   pymoldyn/
   ```
   If your local user is different from the trasgu user, adding the "username@" is mandatory, as Git is working through the tunnel using the local user name

4. For every update, locate the cursor IN the local pymoldyn directory and type:
   ```
   git pull
   ```
   Or of you are on a remote computer you might need to put the full options:
   ```
   git pull ssh://localhost:7777/home/slurm/pymoldyn/share
   ```

5. Make sure that your gromacs installation is understood by the program. To specify the path of Gromacs, edit the file (with a text editor, i.e.. the Unix program "vi") settings.py, and 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" at the cuelebre cluster.

Usage

The program is executed through the simple shell script run.sh, located in the pymoldyn directory. This program simply calls the python program, so in the case of a local installation

[1] This feature is not yet implemented. However if you want to easily do it, open the source file "run.py" and look for a line "`def moldyn(self):`". You'll see inside that function a list of steps. Change that line for "`steps = ["Init", "Minimization"]`", and the program will stop after doing only that two steps.

(not needed if you are running in Cuelebre!) it should point to the right path where the executable is.[2] You can call the script using the full path, or create an alias:

```
/home/localuser/pymoldyn/run.sh
```

An output like this one shows you all the options:

```
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]

run.py: error: argument -p is required
```

If you call that wrapper with the -h/--help flag, you will get a more verbose help:

```
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 alosteric interaction, without extension. Three
                   files must be present along with the molecule pdb: the
                   alosteric, its itp and its force field.
  --waters WATERS  Crystalized water molecules file name without extensions.
  --ions IONS      Crystalized ions file name without extensions.
  --cho CHO        Crystalized cholesterol molecules file name without
                   extensions.
  -q QUEUE         Queue system to use (slurm, pbs, pbs_ib and svgd supported)
  --debug
```

- **COMPULSORY! Option -p**: In the simplest case, Pymoldyn 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!
- **Accesory 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 pymoldyn instalation.
- **Considering non-protein elements**: ligand(s), structural waters, structural lipids, cholesterol molecules, explicit ions.

[2] Currently, there are no enviromental variables set so the user must edit “run.sh” and make sure that the default line: “/home/slurm/pymoldyn/run.py” is changed and acounts for the local instalation path (i.e., /home/localuser/pymoldyn/run.py). Otherwise an error appears like:
python: can't open file '/home/slurm/pymoldyn/run.py': [Errno 2] No such file or directory

- 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 alo
  ```
- 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[3] 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 implemebted in garibaldi.scripps.edu
  - pbs_ib infiniband, as implemented in garibaldi.scripps.edu
  - svgd, as implemented in svgd.cesga.es

[3] The simple straightforward name “ion” should be avoided since it is used in the gromacs standard files. Although it sounds redundant, the program needs to treat the ions provided by the user in a separate manner as the bulk ions used to neutralize the system.

To summarize, the following command should work in the most complex case[4],[5]

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

Running with queues

99% of the time you will want to use some queue 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/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 just fine and the system can be minimized and does not explote 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
```

[4] The needed files should be provided as a tar file in a test case (To do)

[5] Note that --alo and --ion are incompatible if the ion to be considered is exploring the same binding site as the allosteric modulator (i.e., Na+ and amiloride in A2a) but this does not need to be necessarily the case.

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 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()
And uncomment (remove the "#") the line:
run.light_moldyn().
3. 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 shake 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 hte output files for analysis, so if you only are interested on this stage, comment the previous line

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!

Output

The performed equilibration includes the following stages:

STAGE	CONSTRAINED ATOMS	FORCE CONSTANT	TIME
		kJ·mol^(-1)·nm^(-2)	ns

Minimization	-	-	(Max. 500 steps)
Eq1	Protein Heavy Atoms	1000	0.5
Eq2	Protein Heavy Atoms	800	0.5
Eq3	Protein Heavy Atoms	600	0.5
Eq4	Protein Heavy Atoms	400	0.5
Eq5	Protein Heavy Atoms	200	0.5
Eq6	Protein C-alfa Atoms	200	2.5

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
- 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_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.