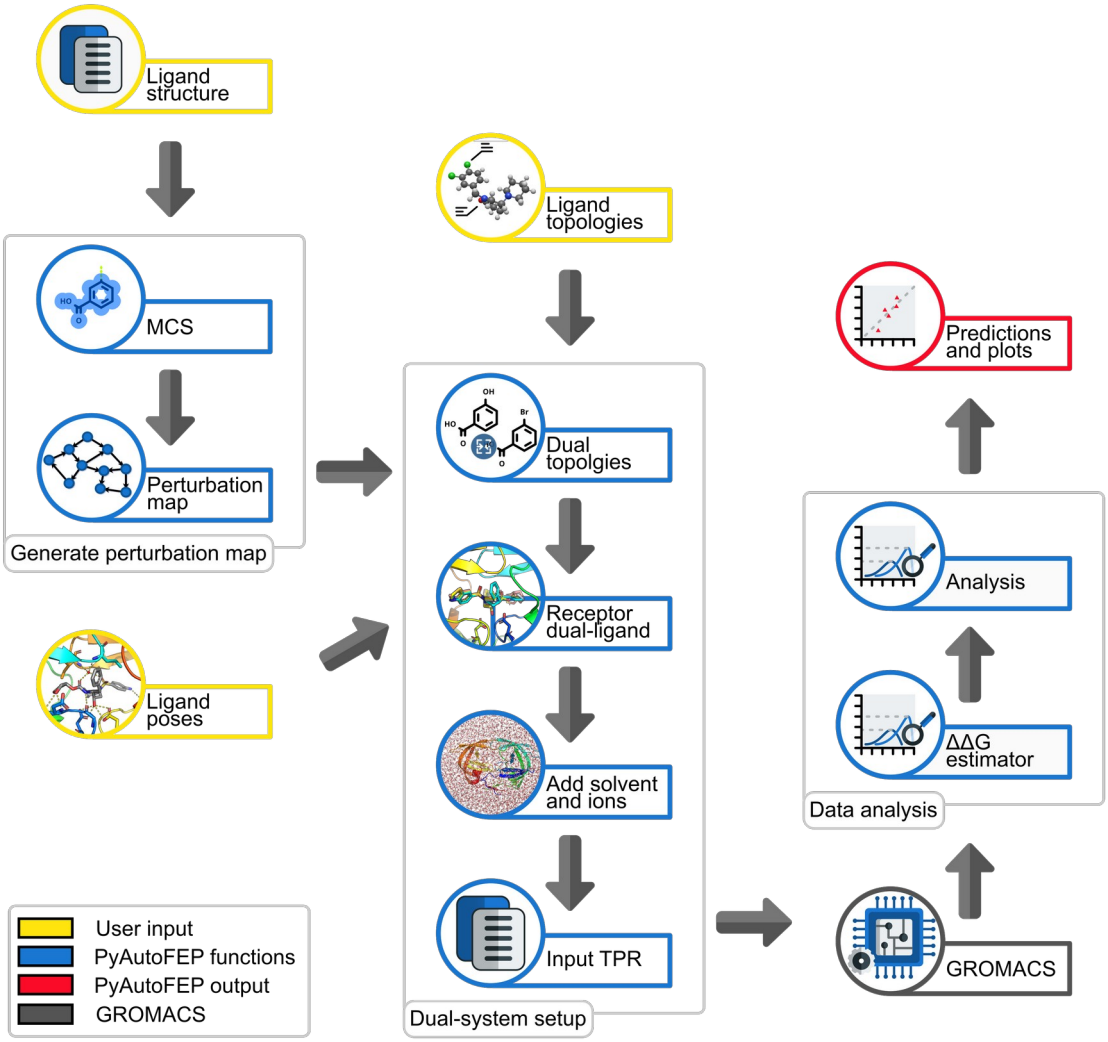
Page 1 – Introduction

NOTE: the following tutorial assumes the reader is familiarized with Molecular Dynamics (MD) and Free Energy Perturbation (FEP) theory. Also, knowledge of GROMACS tools, topologies and input files is useful. Linux command line is used throughout the tutorial. Finally the user is advised to read and reefer to the PyAutoFEP manual, which describes the program detail.

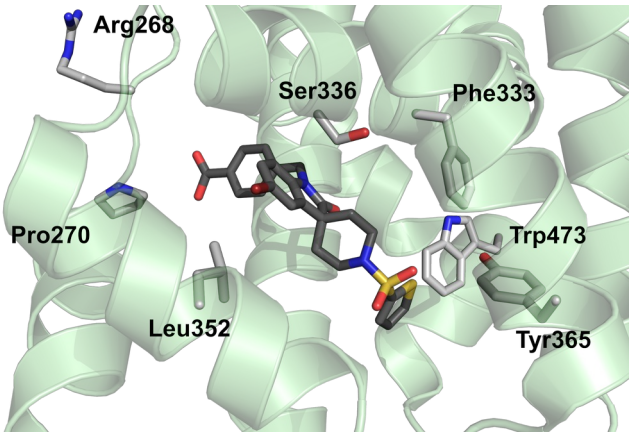
PyAutoFEP automates the setup and analysis of relative free energy of binding of small molecules binding to a macromolecular target using GROMACS [1]. PyAutoFEP automates the generation of perturbation maps, the setup of calculations, and the analysis. It uses a flexible λ scheme and integrates REST2 [2,3], allowing for partial tempering of perturbed regions which can speed up convergency. PyAutoFEP has three modules: a pertubation map generator, a module to setup the simulation systems and prepare inputs, and a analysis module. In this tutorial, these three modules will be used to prepare and analyze a set of perturbations.



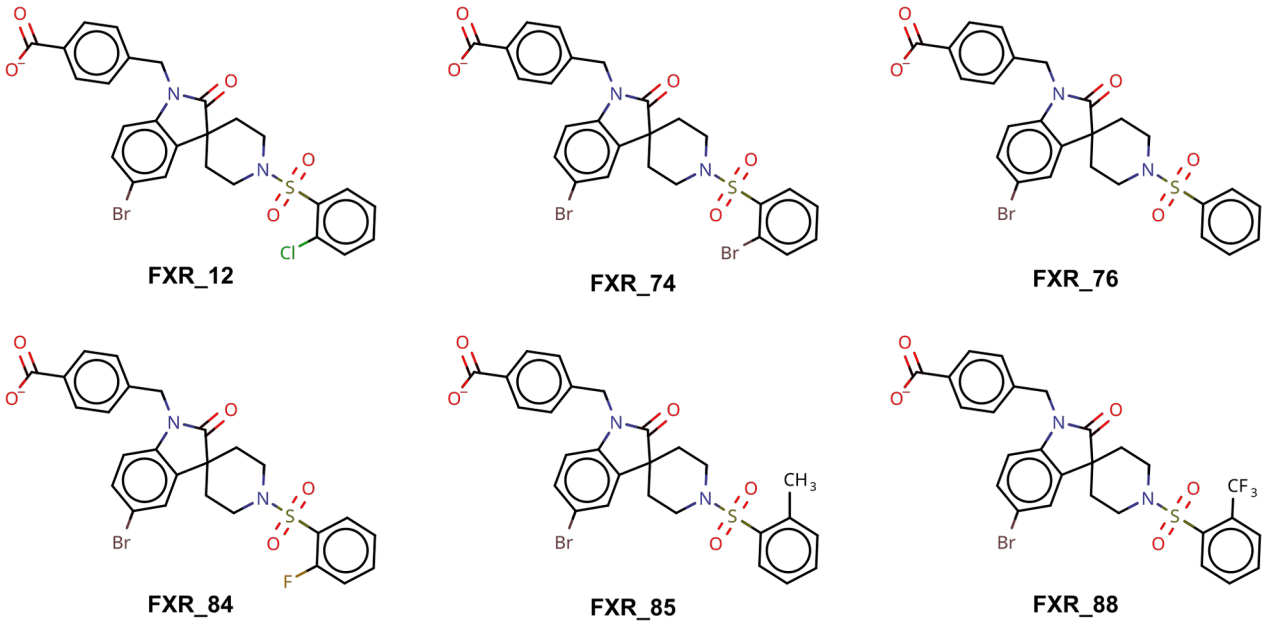
Overall workflow of the PyAutuFEP pipeline

The model system used in this tutorial is the Farnesoid X Receptor, a relevant nuclear receptor for bile acids nuclear bile acid receptor relevant to the regulation of bile secretion, cholesterol homeostasis and control of glycemic levels [4]. Because of the central role of bile secretion in the absorption of lipids,

FXR is a drug target for metabolic disorders associated with obesity, as type 2 diabetes [5]. Several crystallographic structures co-crystallized with diverse ligands are available for FXR, as well as Structure-Activity Relationship studies. Specifically, this tutorial uses the part of the Free Energy dataset 2 from the community challenge Grand Challenge 2, organized by D3R [6] (https://drugdesigndata.org/about/grand-challenge-2/fxr).



Farnesoid receptor binding site complexed to **FXR_10** (PDBID 5Q17 [6]). (*Figure created using PyMOL)*.



Selected molecules of the spiro series used in this tutorial. *(Figure created using MarvinSketch 20.3)*

Tutorial requirements

- PyAutoFEP (*see requirements in the manual or in Github:* https://github.com/lmmpf/PyAutoFEP)
  - *This tutorial assumes PyAutoFEP install dir to be in path, reader may have to fix PyAutoFEP execution commands otherwise*
- GROMACS

- openbabel (*version 2.4.x*)
- Tutorial data (https://github.com/luancarvalhomartins/PyAutoFEP/tree/master/docs/tutorial01)

The tutorial data contains the inputs files. All following commands assume the tutorial data was extracted to the working directory. GROMACS binary is used by PyAutoFEP to prepare the MD systems and to run MD.

Page 2 – macromolecular target and ligand input data

In order to setup the relative free energy of binding calculations, PyAutoFEP requires input files describing the ligands and its target. Note that PyAutoFEP does not parameterize small molecules and do not check for completeness, validity or consistency of parameters supplied. User is advised to make sure (a) all small molecules in each perturbation map are parameterized using a consistent method; and (b) small molecule force field and parameters are compatible with macromolecular force field of choice.

PyAutoFEP requires ligand structures (eg, mol2 or mol files) and GROMACS compatible topologies for these ligands (eg, top files). Ligand structures will be read and used to calculate MCS, alignments and perturbed atoms, while ligand topologies will be processed, but are finally used by GROMACS grompp. Make sure the atom order and atom naming in the ligand structure matches the ones in the topology. Currently, small molecules parametrization tools consistent with the major force fields has been reported [7–10]. As long as the topologies are GROMACS-compatible, PyAutoFEP will parse topologies generated by external tools alike.

Ligand structure and topologies

In this tutorial, LigParGen server (http://zarbi.chem.yale.edu/ligpargen; [8]) will be used to obtain small molecule parameters. In order to parameterize the ligands, first the mol files must be generated. This can be done using openbabel:

```
# Convert smiles in mol2 files
> obabel fxr_fe2_selected_ligands.smi -Oligand.mol --gen3d -m

# Create a dir
> mkdir lig_data

# Batch rename and move the mol files
> for eachfile in ligand*.mol; do mv $eachfile "lig_data/$(head -n1 $eachfile).mol"; done
```

NOTE: the likely most correct approach is to optimize the ligands using a higher level of theory. PyAutoFEP will use ligand geometry read from mol or mol2 input file to prepare the solvent leg – but not the complex leg, which is read from the poses data. The user is advised to validate ligand structure.

Next, LigParGen webserver is used to obtain ligand parameters using the .mol files in lig_data. The ligands are monoanions, so the charge model should be "1.14*CM1A" and the molecule charge "-1". For each molecule, the GROMACS compatible topology must be downloaded (the TOP button under GROMACS label) and saved to the `lig_data` directory as *`FXR_XX.itp`*, where *FXR XX* is the ligand name. The ligand name in the topology [moleculetype] directive will be wrong, but PyAutoFEP will fix it. The `lig_data` directory must contain the following files:

```
# lig_data must contain:
> ls lig_data
FXR_12.itp  FXR_74.itp  FXR_76.itp  FXR_84.itp  FXR_85.itp  FXR_88.itp
FXR_12.mol  FXR_74.mol  FXR_76.mol  FXR_84.mol  FXR_85.mol  FXR_88.mol
```

Finally, the crystallographic ligand **FXR_10** must be copied to a ligand file. This will be needed to use the core-constrained alignment later on.

```
# receptor_data must contain:
> grep "HETATM.*9MV" receptor_data/5q17.pdb > receptor_data/9mv.pdb
```

Protein structure

PyAutoFEP requires a prepared PDB describing the macromolecule. Macromolecular structures (eg, pdb files) will be used during the ligand alignment and processed to add the ligands. The PDB will be processed by GROMACS, so it must be contain hydrogens and the residue and atom name must match the GROMACS force field specification. In this tutorial, GROMACS compatible OPLS-AA/M force field will be used (force field data can be obtained here (link: http://zarbi.chem.yale.edu/oplsaam.html) and the `oplsaam.ff` directory should be extracted to the working directory.

A widely used tool to prepare PDB files is the PDB2PQR server (http://server.poissonboltzmann.org/pdb2pqr; [11]), which conveniently integrates the PropKa program [12] to predict titration states of residues. In this tutorial, the PDB2PQR webserver will be used to fix the 5Q17 (ref) pdb file (model missing atom, remove alternate locations and cocrystalization medium species) and add hydrogens to the assay pH (7.4). The AMBER naming scheme must be used in the output. Output pdb must be saved as `5q17_processed.pdb` in the `receptor_data` directory.

NOTE: if the command-line pdb2pqr is used, the following command can be used to prepare the PDB file: "`pdb2pqr30 --ffout=AMBER --titration-state-method=propka --with-ph=7.4 --protonate-all --drop-water 5Q17 5q17_processed.pdb`".

There will be, however, some residues with problematic namings. For the sake of the tutorial, a bash script to rename the residues and atoms is available along with the data.

```
# Replace residue names
> ./rename_res.sh receptor_data/5q17_processed.pdb
```

`rename_res.sh` is a trivial script using **`sed`** to edit `receptor_data/5q17_processed.pdb` in place and create a backup in `receptor_data/5q17_processed.pdb.bak`.

NOTE: rename_res.sh was included in the tutorial to simplify its pipeline and is not part of PyAutoFEP. It will not work with arbitrary PDB files and force fields. Users are responsible for making sure input PDB is consistent with the force field used.

Page 3 – Perturbation map generation

The first step in obtaining a set of relative energies of binding is generating a perturbation map. A perturbation map is a connected graph representing the perturbations in a set of molecules so that each vertex represents a molecule and each edge represents a perturbation.

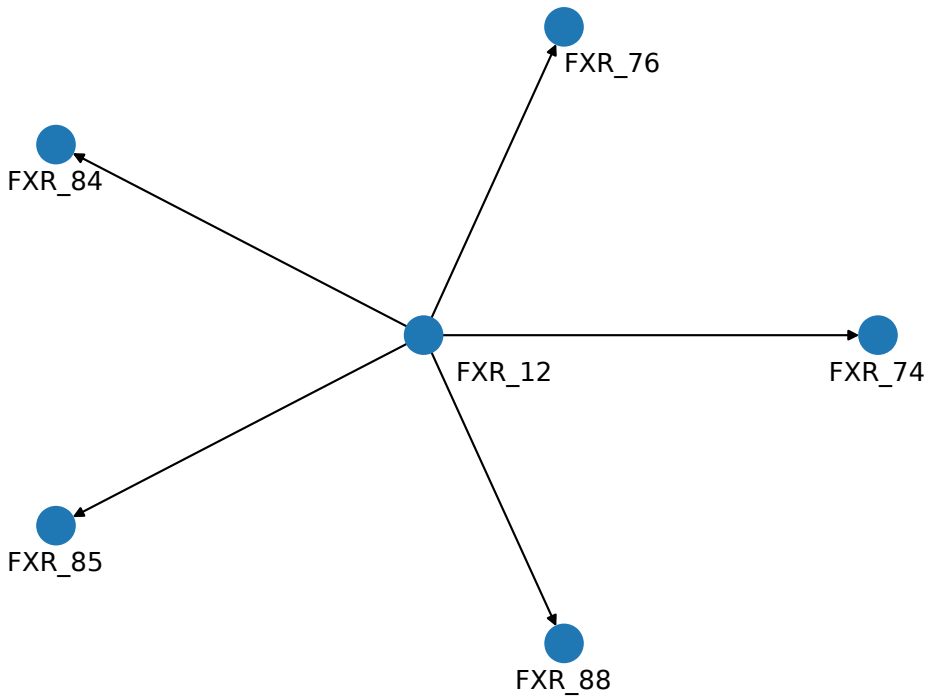
NOTE: the automated map generation is optional and a custom map can be supplied directly to the second module.

NOTE: the map generator will build connected graphs, but this is no requirement to PyAutoFEP to work; a disconnected map can be used, but not all analysis will be available.

The perturbation map generator can prepare three types of perturbation maps. In this tutorial the simpler one, the star map, will be used. A star map is a perturbation map where all the ligands are connected to a central one, so that the minimum possible number of perturbations is generated. During the setup of the perturbation map, PyAutoFEP will calculate the Maximum Common Substructure (MCS) between the center ligand and all others, and this info will be saved to be reused later on. MCS and other data will be saved to a progress file, a Python3 pickle file that can be read by PyAutoFEP.

To generate a star map using **FXR_12** as the center molecule, the following command can be used. It will read all `.mol` files in `lig_data`, calculate MCS and save the perturbation map to `progress.pkl`. A graphical depiction of the map will be saved to `best_graph.svg`.

```
# Generate a star map
> generate_perturbation_map.py --map_type=star --map_bias=FXR_12 --input lig_data/*.mol
```



Star map generated by *generate_perturbation_map.py*

Page 4 – Prepare inputs

The main part of PyAutoFEP, `prepare_dual_topology.py`, creates the dual topologies, setup the MD systems and prepare GROMACS inputs. A directory structure run scripts will be generated, so that everything can be run using a single script. The run script can either directly run GROMACS gmx mdrun or submit jobs to a scheduler, such as Slurm or PBS.

In order to prepare the complex legs, `prepare_dual_topology.py` requires input poses for the ligands in respect to the FXR. In general, such poses can be obtained by unconstrained docking, constrained core docking, or by from experimental structures. In this tutorial, however, a simpler approach will be used. PyAutoFEP can prepare input poses by superimposing the input ligands to a reference small molecule structure, as long the series and the reference molecule share a common core.

NOTE: when generating poses via superimposition, PyAutoFEP will not take the macromolecule into account and a simple conformational sampling strategy will be used, so superimposition is only viable when the perturbations are small and rigid. For more diverse series, a core constrained docking strategy is likely a better choice.

Another relevant point is the type of script generated to minimize, equilibrate and run the perturbations. The default option will make a common, serial bash script that will serially run each perturbation leg. To simplify the submission to job schedulers, PyAutoFEP can make scripts that will submit job instead of directly running them.

NOTE: using a serial bash script is likely a bad idea, even for the few perturbations in this tutorial. PyAutoFEP manual describes how setup `prepare_dual_topology.py` to prepare a run script to submit jobs to SLURM, Torque or PBS.

Finally, in this part of the tutorial, instead of using the command line arguments, a configuration file will be used. This simplifies the input especially when many options have to be set. PyAutoFEP can read ini files as configuration and all arguments available in the command-line are also available as a configuration option (See manual for further info).

These are the contents of the configuration file:

```
# This is the configuration section for prepare_dual_topology.py (which is the only section in this file, but the [ section ] is mandatory)
[prepare_dual_topology]

# Read ligands and topologies from this folder
input_ligands = lig_data

# Read the macromolecule structure from this file
structure = receptor_data/5q17_processed.pdb

# This is the force field directory, it will be copied to each perturbation dir and used to prepare the MD systems
extradirs = oplsaam.ff

# Options controlling the core-constrained superimposition
# First select the use of it instead of reading all ligand poses
pose_loader = superimpose

# Use this pose as the reference for the superimposition
poses_reference_pose_superimpose = receptor_data/9mv.pdb
```

```
# Name of the output. This will be a self-extracting bash file
perturbations_dir = tutorial

# Sets the path to GROMACS executable in the run node, uncomment and modify if needed.
# gmx_bin_run = /usr/local/bin/gromacs

# Options controlling the output, see manual for more info. Uncomment as needed
# FEP legs are to be submitted to a slurm scheduler
# output_scripttype = slurm
# Run these commands at the beginnig of the jog (useful to load modules, importing libs)
# output_runbefore = module load python3; module load cuda
# Fine-tune job resources
# output_resources = all_cpus:24; all_gpus:2; all_time: 24
# Use a python file instead of a binary during the collect step
# output_collecttype = python
```

The prepare *prepare_dual_topology.py* script can, then, be run using:

```
# Prepare the systems
> prepare_dual_topology.py --config_file=step2.ini
```

prepare_dual_topology.py will run for a moment and create a **tutorial.bin** file.

Page 5 – Run FEP

After **`tutorial.bin`** is copied over to the cluster or machine in which the calculation will be run, it must be extracted.

```
# The default is a self-extracting script, but see manual of other options
> bash ./tutorial.bin
```

A `tutorial` directory will be created with a complete directory structure to run all FEP legs and collect the data. In this directory there is a a convenience script, **`tutorial.bin`**, to run everything (in case `output_scripttype = bash` was used, the default), or submit all the legs as jobs (in case `output_scripttype = slurm`).

```
# Run or submit all
> cd tutorial
> ls # Contents may vary
FXR_12-FXR_74  FXR_12-FXR_84  FXR_12-FXR_88             pack.sh       runall.sh
FXR_12-FXR_76  FXR_12-FXR_85  collect_results_from_xvg  progress.pkl

> bash runall.sh
```

NOTE: the MD required to collect data to FEP will likely take some hours in a cluster node. User is advised to make sure enough computational resources are available.

Executing **`runall.sh`** will run MD directly using a serial bash script. Regardless of using a job scheduler or not, the run process is similar. Each λ window for each system will be minimized and equilibrated. Then, using GROMACS multirun, the MD for all λ windows will be run. After the MD completion, GROMACS rerun will be used to reevaluate the trajectories using the hamiltonians of all λ windows. Finally, MD data will be collected using `collect_results_from_xvg`.

NOTE: see the PyAutoFEP manual for a through description on the run process.

After the script run to completion or all jobs finish, a new file, `tutorial.tgz`, will be present in the `tutorial` directory. This file contains all the data PyAutoFEP needs for the analysis and should be copied back from the cluster or remote machine, in case either one is used.

Page 6 – Analysis

PyAutoFEP streamlines the analysis of the FEP data, calculates the ΔΔG and generate several useful plots for each perturbation. Some of the analysis done by PyAutoFEP uses `alchemical-analysis.py`[13]. User is strongly advised to refer to this reference for a excelent discussion about analysis of free energy calculations.

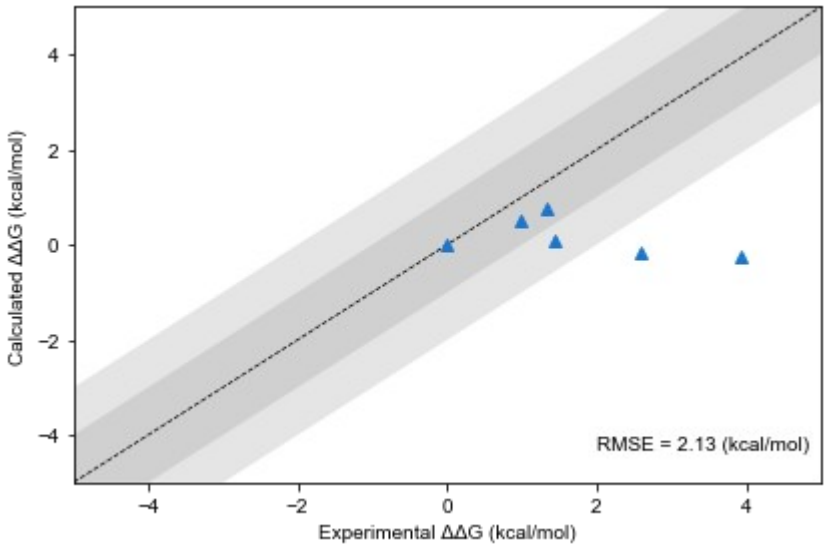
NOTE: PyAutoFEP uses pymbar through alchemlyb to obtain ΔΔG. Make sure you are using pymbar >= 3.0.4 as newer versions have a bug that can cause errors (see https://github.com/choderalab/pymbar/issues/419).

The only file required for running *`analyze_results.py`* is the `tutorial.tgz` output. The analysis can be run with the following command:

```
# Run all analysis, use kcal/mol as unit, and calculate ΔΔG relative to the center
# molecule FXR_12 (--center_molecule=FXR_12 is redundant in the case and could be omitted)
> analyze_results.py --input tutorial.tgz --units kcal --output_uncompress_directory
tutorial --center_molecule=FXR_12

# analyse_results.py will print some data, including the formatted pairwise ΔΔG
```

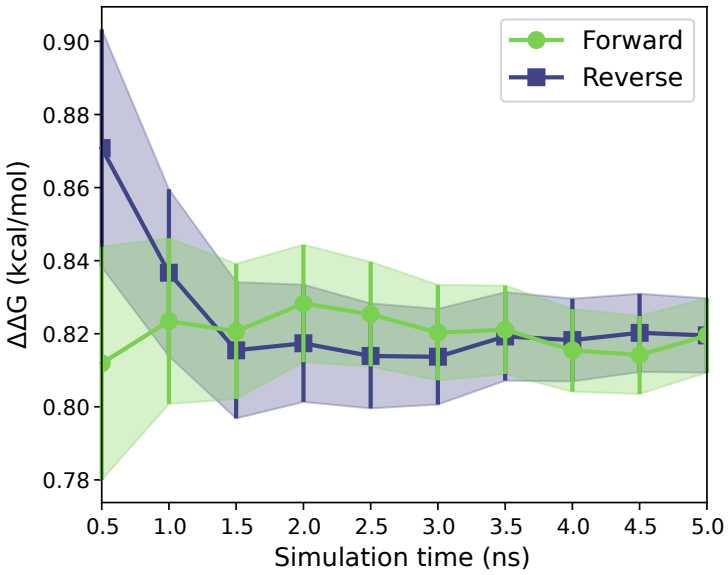
This will create a `tutorial` directory containing some files and a directory for each perturbation. The ΔΔG of all molecules in respect to the center molecule will be written to `ddg_to_center.csv`. From this file, a plot correlating the experimental and calculated can be generated. A example of how to do that can be find in the `gc2_data/PyAutoFEP tutorial plots.ipynb` Jupyter notebook, which uses experimental data from `gc2_data/experimental_affinity.csv`.



Correlation plot from `gc2_data/PyAutoFEP tutorial plots.ipynb`. Dark gray area is the 1 kcal/mol error range. Light gray is the 2 kcal/mol error range. The diagonal dashed line is the theoretical Y=X correlation.

In each perturbation directory, a `water` and a `protein` directory contains data for each leg. This can be used to assess the quality of the prediction. For instance, the `ddg_vs_time.svg` file will contain a plot of the estimated ΔΔG when the trajectory is evaluated up to a certain time. This plot can be used to

evaluate the convergency of the FEP, as a well-converged estimative will have stable estimated ΔΔG near the complete simulation. Also the estimated ΔΔG using the reversed trajectory should be similar to the regular (forward) ones for longer simulation times. Simulations can be subject to extension to reach convergency.



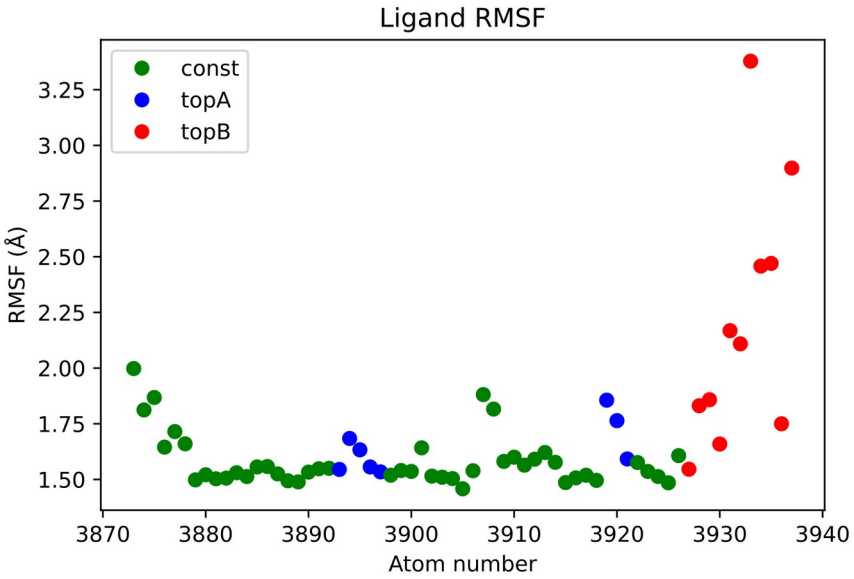
ΔΔG estimated using a truncated trajectory at sequential times. Note that the predicted values for the forward and reversed trajectories are very similar to each other and stable over time after 1.5 ns.

The `overlap_matrix.svg` will contain a estimative of the overlap between states. A good overlap requires that the first upper and lower diagonal to be at least 0.03[13]. A different λ scheme can help improve overlapping (see the manual to know more about selecting λ schemes in PyAutoFEP).

λ	0	1	2	3	4	5	6	7	8	9	10	11
0	.17	.16	.16	.16	.16	.01	.01	.03	.03	.03	.03	.03
1	.16	.16	.16	.16	.16	.01	.01	.03	.03	.03	.03	.03
2	.16	.16	.16	.16	.16	.01	.01	.03	.03	.03	.03	.03
3	.16	.16	.16	.16	.16	.01	.01	.03	.03	.03	.03	.03
4	.16	.16	.16	.16	.16	.01	.01	.03	.03	.03	.03	.03
5	.01	.01	.01	.01	.01	.58	.37					
6	.01	.01	.01	.01	.01	.37	.54	.01	.01	.01	.01	.01
7	.03	.03	.03	.03	.03		.01	.17	.16	.16	.16	.16
8	.03	.03	.03	.03	.03		.01	.16	.16	.16	.16	.16
9	.03	.03	.03	.03	.03		.01	.16	.16	.16	.16	.16
10	.03	.03	.03	.03	.03		.01	.16	.16	.16	.17	.17
11	.03	.03	.03	.03	.03		.01	.16	.16	.16	.17	.17

An overlap matrix.

Furthermore, a directory named `analysis` will be present along with the directories containing the data for each lambda MD (`lambdaX`, where X is a integer). Inside this `analysis` directory, for complex perturbations, there will be xvg files containing RMSD, RMSF, SASA and distance data for the ligand and protein. These can be used to inspect features of the trajectory and assess ligand stability. Note that the all the intermediate states (ie, all but the end points) are non-physical, so the user is advised to be cautious when interpreting data for such states.



Ligand RMSF for the state λ = 0. Note that topology B atoms (in red) are non-interacting particles and its RMSF have no physical meaning. (*Plot prepared using Python3, numpy and matplotlib*).

References

(1) Abraham, M. J.; Murtola, T.; Schulz, R.; Páll, S.; Smith, J. C.; Hess, B.; Lindahl, E. GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers. *SoftwareX* **2015**, *1–2*, 19–25. https://doi.org/10.1016/j.softx.2015.06.001.

(2) Terakawa, T.; Kameda, T.; Takada, S. On Easy Implementation of a Variant of the Replica Exchange with Solute Tempering in GROMACS. *J. Comput. Chem.* **2011**, *32* (7), 1228–1234. https://doi.org/10.1002/jcc.21703.

(3) Wang, L.; Friesner, R. A.; Berne, B. J. Replica Exchange with Solute Scaling: A More Efficient Version of Replica Exchange with Solute Tempering (REST2). *Journal of Physical Chemistry B* **2011**, *115* (30), 9431–9438. https://doi.org/10.1021/jp204407d.

(4) Mi, L.-Z.; Devarakonda, S.; Harp, J. M.; Han, Q.; Pellicciari, R.; Willson, T. M.; Khorasanizadeh, S.; Rastinejad, F. Structural Basis for Bile Acid Binding and Activation of the Nuclear Receptor FXR. *Molecular Cell* **2003**, *11* (4), 1093–1100. https://doi.org/10.1016/S1097-2765(03)00112-6.

(5) Fang, S.; Suh, J. M.; Reilly, S. M.; Yu, E.; Osborn, O.; Lackey, D.; Yoshihara, E.; Perino, A.; Jacinto, S.; Lukasheva, Y.; Atkins, A. R.; Khvat, A.; Schnabl, B.; Yu, R. T.; Brenner, D. A.; Coulter, S.; Liddle, C.; Schoonjans, K.; Olefsky, J. M.; Saltiel, A. R.; Downes, M.; Evans, R. M. Intestinal FXR Agonism Promotes Adipose Tissue Browning and Reduces Obesity and Insulin Resistance. *Nature Medicine* **2015**, *21* (2), 159–165. https://doi.org/10.1038/nm.3760.

(6) Gaieb, Z.; Liu, S.; Gathiaka, S.; Chiu, M.; Yang, H.; Shao, C.; Feher, V. A.; Walters, W. P.; Kuhn, B.; Rudolph, M. G.; Burley, S. K.; Gilson, M. K.; Amaro, R. E. D3R Grand Challenge 2: Blind Prediction of Protein–Ligand Poses, Affinity Rankings, and Relative Binding Free Energies. *Journal of Computer-Aided Molecular Design* **2018**, *32* (1), 1–20. https://doi.org/10.1007/s10822-017-0088-4.

(7) Vanommeslaeghe, K.; Hatcher, E.; Acharya, C.; Kundu, S.; Zhong, S.; Shim, J.; Darian, E.; Guvench, O.; Lopes, P.; Vorobyov, I.; Mackerell, A. D. CHARMM General Force Field: A Force Field for Drug-like Molecules Compatible with the CHARMM All-atom Additive Biological Force Fields. *Journal of Computational Chemistry* **2010**, *31* (4), 671–690. https://doi.org/10.1002/jcc.21367.

(8) Dodda, L. S.; Cabeza de Vaca, I.; Tirado-Rives, J.; Jorgensen, W. L. LigParGen Web Server: An Automatic OPLS-AA Parameter Generator for Organic Ligands. *Nucleic Acids Res* **2017**, *45* (W1), W331–W336. https://doi.org/10.1093/nar/gkx312.

(9) Roos, K.; Wu, C.; Damm, W.; Reboul, M.; Stevenson, J. M.; Lu, C.; Dahlgren, M. K.; Mondal, S.; Chen, W.; Wang, L.; Abel, R.; Friesner, R. A.; Harder, E. D. OPLS3e: Extending Force Field Coverage for Drug-Like Small Molecules. *J. Chem. Theory Comput.* **2019**, *15* (3), 1863–1874. https://doi.org/10.1021/acs.jctc.8b01026.

(10) Sousa da Silva, A. W.; Vranken, W. F. ACPYPE - AnteChamber PYthon Parser InterfacE. *BMC research notes* **2012**, *5* (1), 367. https://doi.org/10.1186/1756-0500-5-367.

(11) Dolinsky, T. J.; Nielsen, J. E.; McCammon, J. A.; Baker, N. A. PDB2PQR: An Automated Pipeline for the Setup of Poisson-Boltzmann Electrostatics Calculations. *Nucleic Acids Research* **2004**, *32* (Web Server), W665–W667. https://doi.org/10.1093/nar/gkh381.

(12) Olsson, M. H. M.; Søndergaard, C. R.; Rostkowski, M.; Jensen, J. H. PROPKA3: Consistent Treatment of Internal and Surface Residues in Empirical PKa Predictions. *Journal of Chemical Theory and Computation* **2011**, *7* (2), 525–537. https://doi.org/10.1021/ct100578z.

(13) Klimovich, P. V.; Shirts, M. R.; Mobley, D. L. Guidelines for the Analysis of Free Energy Calculations. *Journal of Computer-Aided Molecular Design* **2015**, *29* (5), 397–411. https://doi.org/10.1007/s10822-015-9840-9.