FESetup1.2 Documentation

Release 1.2.1

Hannes Loeffler

Oct 20, 2022

Contents

1 Introduction **3**
- 1.1 Capabilities and Description . . . 3
- 1.2 General . . . 4
- 1.3 MD engines . . . 4
- 1.4 Maximum Common Substructure Search . . . 5
- 1.5 Adding hydrogens . . . 5
- 1.6 Charge parameterisation . . . 6

2 Installation **7**
- 2.1 Dependencies . . . 8

3 Running FESetup **9**
- 3.1 Command line . . . 9
- 3.2 Input format . . . 9
- 3.3 Explicit tagging mechanism . . . 10

4 Input Options **13**
- 4.1 Full option list . . . 13

5 Indices and tables **19**

Contents:

CHAPTER 1

Introduction

FESetup automates the setup of relative alchemical free energy (AFE) simulations such as thermodynamic integration (TI) and free energy perturbation (FEP). Post–processing methods like MM–PBSA and LIE are supported as well. FESetup can also be used for general simulation setup ("equilibration") through an abstract MD engine (currently supported MD engines are AMBER, GROMACS, NAMD and DL_POLY). For relative AFE simulation the mapping of corresponding atoms between the two free energy states, that is their topological similarity, is computed via a maximum common substructure search (MCSS). This enables a maximal single topology description of the perturbed molecule pair. Ligand molecules can automatically be parameterised using the AMBER GAFF/AM1-BCC method. Supported force fields for biomolecules are all the modern AMBER force fields.

The AFE simulation packages that are currently supported are Sire, AMBER, GROMACS and CHARMM/PERT. All these codes implement AFE simulation by making use of a hybrid single/dual topology description of the perturbed region i.e. the mapped region (single topology) can be used simultaneously with an un–mapped, duplicated region (dual topology) from each state. There is also some support for NAMD's purely dual topology implementation but this requires an additional PDB file to mark appearing/vanishing atoms and possibly relative restraints to keep ligands spatially in place and/or together.

FESetup particularly aims at automation where it makes sense and is possible, ease of use and robustness of the code. Users are very welcome to discuss on our forum, report issues and request new features. The software is licensed under the GPL2 and such is a community effort: user contributions in any form are highly encouraged!

The basis of the current code was a collection of Python and shell scripts written previously by Julien Michel and Christopher Woods. The FESetup1.2 code base was mainly developed by Hannes Loeffler (STFC) with contributions from the original developers.

Please cite DOI: 10.1021/acs.jcim.5b00368 when you use FESetup.

1.1 Capabilities and Description

The sections here give a brief overview of FESetup from a general point of view i.e. topics like background information, concepts, implementation ideas and philosophies, etc. but also the limits of the current software.

1.2 General

FESetup is currently (July 2017) mainly geared at automated setup for receptor-ligand simulations. But in principle any biomolecular system can be set up for simulation provided the system is supported by the force field (AMBER and AMBER/GAFF) and the ligand is not covalently bound and can reasonably be parameterised with AM1/BCC. The latter implies that the ligand is a relatively "small" organic molecule as typically considered in the drug development process.

Core functionality of FESetup are the parameterisation of (possibly large numbers of) ligands and the "morphing" of a pair of ligands. Morphing describes how a molecule's force field parameters are transformed (or "perturbed") into a set of parameters descriptive for a second, topological related molecule. As such FESetup is a perturbed topology creator. Ligand parameterisation enables standard MD simulation including post-processing methods like MM-PBSA for simplified free energy estimates. The topology file describing the morphing of the ligand pair enables alchemical free energy simulation like Thermodynamic Integration (TI) or Free Energy Perturbation (FEP, also called EXP for exponential formula). Currently, FESetup supports such simulation setups for Sire, AMBER, Gromacs and the PERT module in CHARMM. In the case of AMBER the dual topology softcore approach as well as the older explicit dummy atom approach, as implemented in both sander and pmemd, are supported.

FESetup reads a series of coordinates for ligands and proteins. The ligands can then be parameterised with the AM1/BCC method and can be combined with a protein (or other molecule supported by the AMBER force fields or with user supplied parameters) into a complex. This requires the complex to be set up properly beforehand, e.g. the complex may have been created via a previous docking run or by any other suitable method. Protein and ligand coordinates must be supplied separately. While the protein is expected in PDB format the ligand can be in any format supported by OpenBabel (but must have 3D coordinates). A caveat though is that OpenBabel does not read total charge information from all file formats that support this and thus the most suitable formats are either PDB or SDF for the ligand if a charge other than zero is needed. Internally the coordinates will be converted to the mol2 format. If morph pairs are requested, FESetup will compute the "difference" between the two ligand molecules using a maximum common substructure search (MCSS) algorithm (from RDKit) as a distance metric and use this information to create appropriate topology files and control files for alchemical free energy simulation.

1.3 MD engines

FESetup makes use of an abstract MD engine, currently for the purpose of "equilibration" of simulation systems. This means that various popular MD simulation software packages (at the moment: AMBER, DL_POLY, NAMD and GROMACS) can be used in a transparent fashion, that is without the need to know the specific control and command structures of a particular simulation software. This is, of course, somewhat limited by MD software not always providing fully equivalent features as in other software packages. But for the general purpose of carrying out setup processes like minimization, heating, pressurizing, restraint release, etc. this is sufficient and sensible starting structures for production MD simulation can be obtained without problems.

The MD engine mechanism allows the user to choose the specific binary (program) to be used for MD/minimisation, e.g. with AMBER that could be any variant of sander or pmemd, with GROMACS any variant of mdrun, etc. Parallel versions are supported to a certain extent too as described in the following. Multi-threaded binaries like mdrun with thread-MPI or the multi-core version of NAMD will run on any single machine or "node". MPI versions like the AMBER binaries do also work but currently there is no support for job schedulers like PBS, LSF, etc.

Some of the limitations of abstraction are that software like GROMACS or NAMD appear to always need a minimization step of a solvated system prior to MD whereas AMBER is less sensitive to this. FESetup currently includes standard input configuration settings for minimization, random velocity assignment, heating, constant T and pressurizing. For all these, restraints can be defined by either keyword or an AMBER mask. All corresponding run-time control parameters are created through templates and each of these steps is carried out as an individual simulation run. This is also done for NAMD despite it having full scripting capabilities of its own. However, the abstract interface hides the details of the actual MD engine away and thus is much easier to use. In addition, the interface is unified i.e. it appears

the same independent of the MD engine chosen. AMBER and GROMACS do no have built-in scripting facilities and thus the separate steps are needed especially the step-wise release of restraints.

1.4 Maximum Common Substructure Search

Morph pairs are generated through a graph based maximum common substructure search (MCSS) algorithm, in particular the implementation in RDKit. By comparing the graphs of two ligands the maximum match between their atoms, based on the topology i.e. connectivity, are found. Currently we do not distinguish between atom types or bond types, i.e. the graph is unlabeled. Rings, on the other hand, are required to match other rings and cannot be broken. All atoms not part of the MCS mapping (equivalent atoms between the two ligands are recorded), are considered to be dummy atoms and coordinates are created accordingly from internal coordinates (this is not needed in the AMBER dual topology softcore apprach).

There are, however, a few caveats to keep in mind. First, the comparison is carried out in graph space which is essentially only two dimensional. Thus 3D features are not retained unless otherwise specified. This includes the substitution patterns of stereogenic centres, e.g. the absolute configuration can be reverted with the scheme described above. Symmetry related issues, e.g. molecules may have multiple MCSs but a "random" one is assigned to a given morph pair, may appear too. Retaining specific binding modes is another problem which may be caused by symmetry but other origins, e.g. flipping by 180 degrees or parts of a molecules preferring another binding pocket when decorated differently, are possible too. FESetup allows the user to provide explicit atom tagging of individual atoms to overcome these problems albeit at the expense of automation. Tagging however allows the user to overwrite the default behaviour of FESetup and can thus guide the mapping to their own preferences.

Second, the time behaviour of MCSS algorithms may pose problems. In the worst case a search would be exponential in time due to the need of an exhaustive search (in practice, the MCSS algorithms have clever shortcuts but there is no universal algorithm available because NP-complete) which implies that for every additional atom the search time would be doubled. We have found several examples where the MCS search can range from many minutes to hours to days (in a very large system of a molecular weight of nearly 1500 with the old Python implementation of fmcs). In practice, however, the MCS appears to be found within just a minute or so (more careful testing needed) and we have not found any problem cases yet where the MCS would not be as expected from visual inspection. FESetup provides a setting to limit the time spend on the MCS search.

1.5 Adding hydrogens

This is a particular serious matter and attitudes among researchers vary. We strongly recommend that the user makes sure that hydrogens are added to the ligand and protonation state as well as tautomeric state are fully determined before FESetup is run. It is the user's responsibility to get the chemistry right.

In principle, there is a setting in FESetup to allow addition of hydrogens in a simple valence filling fashion. But we have found that this does not always work properly with OpenBabel, e.g. the N9 (binding to the ribose) in nucleosides appears to be always perceived as being located within a double-bonded or aromatic ring. This results in addition of a hydrogen and a charge of +1 on N9. However, charge parameterisation is achieved through sqm, a semi-empirical tool in the AmberTools, which crucially depends on having the charge properly assigned, or otherwise may terminate with an error or compute grossly wrong results.

Finding protonation and tautomeric states is even much more difficult because they depend strongly on the environment. There is currently no support for assignment of such states in FESetup. Any future work in this direction will have to await very thorough investigations into what is possible and what not.

1.6 Charge parameterisation

The parameterisation steps in FESetup currently carries out charge calculation for the AMBER/GAFF force field at the AM1/BCC level with the help of the AmberTools toolchain antechamber. This method derives Mulliken charges at the semiempirical AM1 level of theory (via sqm) and then applies bond charge corrections (BCC) to these Mulliken charges to finally obtain charges almost equivalent to the higher level HF 6-31G* RESP charge derivation scheme. A subsequent run through parmchk(2) will assign missing bonded parameters through a similarity search in the first attempt or, if this cannot be achieved, an empirical approach is followed. If both methods fail the correpsonding force field parameters will be set to zero. This is actually exploited in the case of the dummy atoms as the corresponding atom type does not have an equivalent in the data base (the code relies on zero force field parameters for identifying dummy atoms at various places). Lennard-Jones parameters and mixing rules will be applied as per the initial perception of GAFF atom types (atomtype and bondtype tools).

There are a few notable limitations. Sqm carries out geometry minimizations in vacuum. This can lead to distorted structures when highly charged groups are present, e.g. zwitterions. We have also found proton "shifts" between the two ends of a molecule and "decarboxylation", both in very rare cases. FESetup provides a (undocumented) option to carry out geometry minimizations with the help of an implicit solvent model (through the QM/MM feature in sander without an actual MM part). But we did not really find an improvement for the aforementioned problems except for more stable SCF convergence and a smoother geometry optimisation.

Larger molecules should probably be considered to be broken into sensible smaller fragments similar to how the biomolecules force field has been parameterised. This is beyond the scope of FESetup currently, however. More elaborate schemes, on the other hand. like the original RESP method or possibly specialised software like R.E.D. may be incorporated in the future.

CHAPTER 2

Installation

Linux, Intel 32bit and 64bit

Run the installer package from www.ccpbiosim.ac.uk/software in a convenient location (N will stand for the release you have downloaded). You can run the installer with "–help" to see further options. Here we describe interactive installation i.e. when run without any command line options. The installer requires the xz compression tool to be installed on your system.

> cd /where/I/want/it # replace the path to whatever you like > ./FESetupN_Linux.sh # extract all files into FESetupN/

The installer will automatically detect which version to extract (either 32 or 64 bit). You will be asked to provide paths to AMBER, GROMACS, NAMD and DL_POLY. It is strongly recommended to choose 'N' (the default, so just press Enter) when the first question suggests to use your existing $AMBERHOME. Choose the internal path as suggested in the following question to avoid modifications to the original AmberTools installation. Press Enter to accept defaults or to set an empty path if you do not have a certain MD package. You will also be asked for a Python 2.7 interpreter (the default is "python", assumed to be located in the $PATH but read the note below). You will find the FESetup script in ./FESetupN/FesetupMM/bin after successful installation (MM is either 32 or 64 depending on your hardware). Check that FESetup is working. You can do this by running the test set from our first tutorial or just run the FESetup script without any command line parameters. This will write the default input parameters to your terminal. You can copy/link the FESetup script to your PATH e.g. /usr/local/bin if you like, or create an alias to point to the script.

Our packages are self-contained and come with all relevant tools from AmberTools 16 including sander (pmemd still requires a full AMBER license). To carry out standard MD simulations, in particular equilibration of your system, the abstract MD engine supports AMBER (both sander and pmemd), NAMD, GROMACS and DL_POLY. Please note that currently we do not directly support NAMD's alchemical free energy methods though there is support for dual topology runs with AMBER inputs in NAMD (an additional PDB file is required to mark appearing/vanishing atoms, see NAMD manual). Standard MD is supported for NAMD though.

Also note that you should use the standard Python interpreter (e.g. the one that comes as a package with your OS distribution or you download and compile from python.org). Python versions that come with a package management systems of their own may break the assumptions that our installer makes with regards to shared libraries. Specifically, anaconda appears to mess with the library search path and seems to disregard the setup in the FESetup script.

2.1 Dependencies

FESetup depends on various third-party software. All of these are included in the installer package. Here a list of dependencies for those who want to compile everything themselves. Not listed are some dependencies which can be installed through the operating sytem's package management software. Some secondary dependencies are listed too. Debian based systems have most libraries, toolkits and tools pre-compiled and ready to install through their package managment system.

Python 2.7

Sire/corelib 0.0.1, Sire/Python2 0.0.1: Qt4, Boost, GSL, BLAS/LAPACK, pcre3

Ambertools 16

OpenBabel 2.3.x: eigen, swig, xml2

RDKit 2016: numpy, Boost

CHAPTER 3

Running FESetup

The script FESetup in the release is the command line tool for the end-user. This shell script sets a few environment variables and eventually calls dGprep.py. dGprep.py is the actually code running the routines for AFE setup.

Note: this manual describes options used as of Release 1.2.

3.1 Command line

Default values for all key–value pairs are written to stdout for each of the four sections when FESetup is called without any command-line parameters. Calling FESetup with '–help' gives information of all possible command line parameters. Currently all options are for information purposes only so will not affect the setup in any way. The option –tracebacklimit is really only of use for debugging. As noted above FESetup is the front-end script to the Python code dGprep.py.

```
> FESetup --help

usage: dGprep.py [-h] [-v] [--tracebacklimit N] [infile]

positional arguments:
 infile  input file in INI format, if not given then just output defaults

optional arguments:
 -h, --help          show this help message and exit
 -v, --version       full version information
 --tracebacklimit N  set the Python traceback limit (for debugging)
```

3.2 Input format

The input file format for the FESetup script (dGprep.py) is an INI like format, popular in the MS-DOS/Windows world. It is not exactly the same format but a simplified version of it.

The input file may contain four sections where the section names are delimited with brackets:

The four sections of the INI file

1. [globals] global settings, the section name is optional if it is the first section in the file
2. [ligand] settings for the ligand
3. [protein] settings for the protein
4. [complex] settings for the complex

Each section consists of various key-value pairs which are two strings separated with an equal sign ("="). The key must not contain any whitespace.:

```
# a typical key-value pair
morph_pairs = p-aminophenol > o-cresol
```

Lines may be continued with an initial whitespace (normal space, TAB) on the following line:

```
# multiple continuation lines
morph_pairs = ethane > methanol, ethane > tbutane, ethane > propane,
              tbutane > propane, tbutane > acetone,
              propane > acetone, propane > methane
```

However, list pairs must always appear on the same line because each line is parsed individually. So the following will cause an error:

```
# this will result in an error
morph_pairs = ethane > methanol, ethane > tbutane, ethane >
              propane,  # the string "propane" must appear in the previous line!
```

Comments are either empty lines or lines starting with '#' or ';' (leading whitespace is removed). Inline comments are allowed too provided the comment character is preceded by a space. Otherwise the string is part of the preceding string, e.g:

```
basedir = smallmols # this is an inline comment
basedir = smallmol#foo # the valid string and directory name 'smallmol#foo'
```

3.3 Explicit tagging mechanism

In some cases it may be necessary to explicitly map certain atoms to one another e.g. to preserve a certain spatial rearrangment as for binding modes. This means that the default mapping algorithm (maximum common substructure search, MCSS) can be given additional hints. In the following example the benzofuran is oriented in the way outlined by forcing a certain mapping. As should be evident there are twelve ways of mapping the two molecules but if e.g. the binding mode is known a prior the user has here a chance to preserve it!:

```
[ligand]
basedir = smallmols
morph_pairs = benzol > benzofuran /1=3/2=2  # indices start from 1
```

An alternative mechanism is to create a special file in basedir (as set in the [ligand] section). The name of the file must be in the form ligand1~ligand2.map, e.g. if the input reads:

```
[ligand]
basedir = smallmols
morph_pairs = benzol > benzofuran
```

then the map file must be in smallmols i.e. smallmols/benzol~benzofuran.map .

The map file has a two column format indicating which atom index in the initial state maps to which atom index in the final state. Atom indexes start from 1:

```
# example mapping file benzol~benzofuran.map in the basedir smallmols/
# explicitly map the following atom indexes onto each other
1 3 # this mapping and...
2 2 # ...this one will fix the orientation of benzofuran in space
```

CHAPTER 4

Input Options

4.1 Full option list

4.1.1 Options unique to each section

The following tables list all options unique to each section. Note that empty strings (denoted as 'none' in the table) means that the user has to use appropriate values. 'molecules' will be overwritten i.e. ignored when 'morph_pairs' are used in [ligand]. The default for complex building is to combine every protein with every ligand. If you do no want that, you must explicitly list all pairs using the 'pairs' key. Please note that your file and directory names must not contain the characters ':' (colon), '>' (right angle bracket), '"' (double quote) and '~' (tilde). The comma ',' is permited as long as the filename is enclosed in double quotes, e.g.

morph_pairs = "1,2-dichloroethane" > E-dichloroethene, E-dichloroethene > "1,2-dichloroethane"

[globals]

Key	Values, default listed first	Type	Explanation
AFE.type	Sire, sander/dummy, sander/softcore, gromacs, pmemd/softcore, pmemd/dummy charmm/pert	string	free energy type, determines which MD package the input files are created for (for backwards compatability AMBER = sander/dummy and AMBER/softcore = sander/softcore)
AFE.separate_vdw_elec	True, False	bool	separate the Coulomb (charge) transformation from the vdW+bonded transformation
forcefield	amber, ff14SB, tip3p, hfe	list of strings	ff family, subtype of ff, water ff, divalent ion set
ff_addons	empty	list of strings	additional force fields like GLYCAM_06j-1 or lipid14
gaff	gaff1, gaff2	string	Choice for the small molecules forcefield, either GAFF 1.x or GAFF2.x
logfile	dGprep.log	string	name of the debug log file
mdengine	amber, sander; amber,pmemd; gromacs, mdrun namd, namd2	list of 2 strings	program for minimisation and MD, the first in the list is the MD package, the second is the actual binary
mdengine.prefix	empty	string	the string preceding the mdengine binary command, e.g. mpirun -np 4 (for MPI programs)
mdengine.postfix	empty	string	the string following the mdengine binary command, e.g. +p2 +isomalloc_sync (for namd multicore)
parmchk_version	2, 1	integer	parmchk version
mcs.timeout	60.0	float	**timeout in seconds for fmcs,** 0 means no timeout
remake	False, True	bool	remake already done molecules (excluding morphs)
overwrite	False, True	bool	by default no files are ever overwritten in the _ directories, use this to change this behaviour
user_params	False, True	bool	read user force field parameter files, i.e. all .frmod, .preb and .lib (OFF format) files are read in

[ligand]

Key	Values, default listed first	Type	Explanation
basedir	none, must be set by user	string	base directory to find ligands
file.name	ligand.pdb	string	ligand input file name
file.format	none, determined from extension of filename	string	format of file.name, can be used to overwrite if file extension is different from actual file format
ions.conc	0.0	float	sets the NaCl concentration in mol/l
ions.dens	1.0	float	density for which the ion concentration is wanted
calc_charge	False, True	bool	Force calculation of molecule's formal charge, required e.g. for mol2 format for which Openbabel computes the charge only in select cases.
conf_search.conv	1e-06	float	conformation search option
conf_search.confsteps	250	integer	conformation search option
conf_search.ffield	mmff94	string	conformation search option
conf_search.gensteps	5	integer	conformation search option
conf_search.numconf	0	integer	conformation search option
conf_search.steep_conv	0.0001	float	conformation search option
conf_search.steps	100 steps	integer	conformation search option
molecules	none, must be set by user	list of strings	list of molecules
morph.absolute	False, True	bool	write absolute transformation MORPH.pert files for Sire
morph_pairs	none, must be set by user	list of strings	list of pairs in the form lig1 > lig2, overwrites 'molecules', do not use '>' in file names
neutralize	False, True	bool	neutralize the solvation box by adding minimum couterions required
skip_param	False, True	bool	skip the parameterisation step, useful in conjunction with user_params or ff_addons see [globals]

[protein]

Key	Values, default listed first	Type	Explanation
align_axes	False, True	bool	align protein along principal axes before hydrating
basedir	none, must be set by user	string	base directory to proteins
ions.conc	0.0	float	sets the NaCl concentration in mol/l
ions.dens	1.0	float	density for which the ion concentration is wanted
molecules	none, must be set by user	list of strings	list of molecules
neutralize	False, True	bool	neutralize the solvation box by adding minimum couterions required
propka	False, True	bool	use ProPKA to protonate protein
propka.pH	7.0	float	pH for ProPKA

[complex]

Key	Values, default listed first	Type	Explanation
align_axes	False, True	bool	align protein along principal axes before hydrating
ions.conc	0.0	float	sets the NaCl concentration in mol/l
ions.dens	1.0	float	density for which the ion concentration is wanted
neutralize	False, True	bool	neutralize the solvation box by adding minimum couterions required
flatten_rings	False, True	bool	make aromatic rings fully planar, for MC with Sire
pairs	none, must be set by user	list of strings	**list of pairs in the form protein:ligand, do n** ':' in file names

4.1.2 The minimisation and MD options

The following options are the minimsation and MD options for molecule setup common to ligands, proteins and complexes. To allow minimisation and MD 'box.type' has to be set explicilty which also creates a water box. If 'box.type' is not set by the user then no box will be created and minimisation or MD will not be carried out. To actually run a minimsation or simulation you will need to set any of the '.nsteps' keys to a value larger than 0. The only difference is relaxation where setting 'md.relax.nrestr' to a value larger than 0 will trigger restraint relaxation. The order of simulation protocols is fixed as heating (md.heat.*), constant volume and temperature (md.constT.*), pressurising = density adjustment (md.press.*), relaxation at NpT conditions (md.relax.*). If any of those steps are not needed set '.nsteps' to 0 but be aware that there are no further sanity checks. The MD protocol can be preceded by a minimisation step (min.*).

Key	Values, default listed first	Type	Explanation
box.type	empty string = no box created, rectangular, octahedron (limited support)	string	creates a box of water
box.length	10.0	float	the distance in Ångström between solute and the box edges, NOTE: the TIP3P box will create a system of low density and thus this distance will decrease on pressuring the sytem.
min.ncyc	0	integer	number of steepest decent steps in minimisation
.nsteps	0	integer	number of steps; e.g. min.nsteps
.restr_force	10.0	float	restraint force; e.g. md.heat.restr_force
.restraint	protein, backbone, heavy, notligand, notsolvent	string	restraint type, if other string then in the list it is the restraintmask for sander; e.g. md.constT.restraint
.T	300.0	float	temperature; e.g. md.press.T
.p	1.0	float	pressure; e.g. md.relax.p
md.relax.nrestr	0	integer	number of relaxation steps, needed to trigger restraint relaxation

CHAPTER 5

Indices and tables