ForceBalance version 0.11.0

Generated by Doxygen 1.7.6.1

Contents

1 Main Page 1

1.1 Preface: How to use this document . . . 1

1.2 Introduction . . . 2

1.2.1 Background: Empirical Potentials . . . 2

1.2.2 Purpose and brief description of this program . . . 4

1.3 Credits . . . 5

2 Installation 6

2.1 Installing ForceBalance . . . 6

2.1.1 Prerequisites . . . 6

2.1.2 Installing . . . 7

2.2 Create documentation . . . 8

2.3 Installing GROMACS-X2 . . . 9

2.3.1 Prerequisites for GROMACS-X2 . . . 9

3 Usage 10

3.1 Input file . . . 10

3.2 Directory structure . . . 11

3.3 Setting up the fitting simulations . . . 11

3.3.1 and force matching . . . 12

3.3.2 potentials . . . 12

3.4 Running the optimization . . . 13

4 Tutorial 13

4.1 Fitting a TIP4P potential using two fitting simulations . . . 13

5 Glossary 14

5.1 Scientific concepts . . . 14

5.2 Option index: General options . . . 15

5.3 Option index: Simulation options . . . 21

6 Namespace Index 26

6.1 Namespace List . . . 26

7 Class Index 27

7.1 Class List . . . 27

8 Namespace Documentation 27

8.1 ForceBalance Namespace Reference . . . 27
8.1.1 Detailed Description . . . 27
8.1.2 Function Documentation . . . 27
8.2 GenerateQMData Namespace Reference . . . 28
8.2.1 Detailed Description . . . 28
8.2.2 Function Documentation . . . 28
8.3 MakeInputFile Namespace Reference . . . 28
8.3.1 Detailed Description . . . 29
8.3.2 Function Documentation . . . 29
8.4 ParseInputFile Namespace Reference . . . 29
8.4.1 Detailed Description . . . 29
8.4.2 Function Documentation . . . 29

9 Class Documentation **29**

9.1 SelfConsistentCycle.ForceBalance_SCF Class Reference . . . 29
9.1.1 Detailed Description . . . 30
9.2 TagMol2.MolG Class Reference . . . 30
9.2.1 Detailed Description . . . 30
9.2.2 Member Function Documentation . . . 30
9.3 CallGraph.node Class Reference . . . 32
9.3.1 Detailed Description . . . 32
9.4 SelfConsistentCycle.SCF_Simulation Class Reference . . . 32
9.4.1 Detailed Description . . . 33

1 Main Page

1.1 Preface: How to use this document

The documentation for ForceBalance exists in two forms: a web page and a PDF manual. They contain equivalent content. The newest versions of the software and documentation, along with relevant literature, can be found on the SimTK website.

Users of the program should read the *Introduction, Installation*, *Usage*, and *Tutorial* sections on the main page.

Developers and contributors should read the Introduction chapter, including the *Program Layout* and *Creating - Documentation* sections. The *API documentation*, which describes all of the modules, classes and functions in the program, is intended as a reference for contributors who are writing code.

ForceBalance is a work in progress; using the program is nontrivial and many features are still being actively developed. Thus, users and developers are highly encouraged to contact me through the SimTK website, either by sending me email or posting to the public forum, in order to get things up and running.

Thanks!

Lee-Ping Wang

1.2 Introduction

Welcome to ForceBalance! :)

This is a *theoretical and computational chemistry* program primarily developed by Lee-Ping Wang. The full list of people who made this project possible are given in the Credits.

The function of ForceBalance is *automatic force field optimization*. Here I will provide some background, which for the sake of brevity and readability will lack precision and details. In the future, this documentation will include literature citations which will guide further reading.

1.2.1 Background: Empirical Potentials

In theoretical and computational chemistry, there are many methods for computing the potential energy of a collection of atoms and molecules given their positions in space. For a system of *N* particles, the potential energy surface (or *potential* for short) is a function of the *3N* variables that specify the atomic coordinates. The potential is the foundation for many types of atomistic simulations, including molecular dynamics and Monte Carlo, which are used to simulate all sorts of chemical and biochemical processes ranging from protein folding and enzyme catalysis to reactions between small molecules in interstellar clouds.

The true potential is given by the energy eigenvalue of the time-independent Schrodinger's equation, but since the exact solution is intractable for virtually all systems of interest, approximate methods are used. Some are *ab initio* methods ('from first principles') since they are derived directly from approximating Schrodinger's equation; examples include the independent electron approximation (Hartree-Fock) and perturbation theory (MP2). However, most methods contain some tunable constants or *empirical parameters* which are carefully chosen to make the method as accurate as possible. Three examples: the widely used B3LYP approximation in density functional theory (DFT) contains three parameters, the semiempirical PM3 method has 10-20 parameters per chemical element, and classical force fields have hundreds to thousands of parameters. All such formulations require an accurate parameterization to properly describe reality.

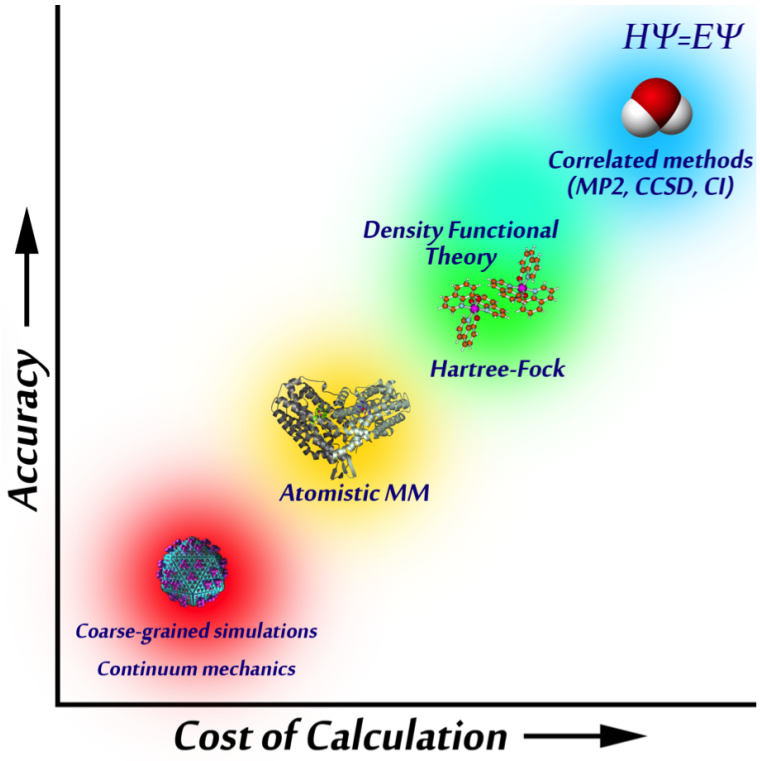


Figure 1: An arrangement of simulation methods by accuracy vs. computational cost.

The main audience of ForceBalance is the scientific community that uses and develops classical force fields. These force fields do not use the Schrodinger's equation as a starting point; instead, the potential is entirely specified using elementary mathematical functions. Thus, the rigorous physical foundation is sacrificed but the computational cost is reduced by a factor of millions, enabling atomic-resolution simulations of large biomolecules on long timescales and allowing the study of problems like protein folding.

In classical force fields, relatively few parameters may be determined directly from experiment - for instance, a chemical bond may be described using a harmonic spring with the experimental bond length and vibrational frequency. More often there is no experimentally measurable counterpart to a parameter - for example, electrostatic interactions are often described as Coulomb interactions between pairs of atomic point "partial charges", but the fractional charge assigned to each atom has no rigorous experimental of theoretical definition. To complicate matters further, most molecular motions arise from a combination of interactions and are sensitive to many parameters at once - for example, the dihedral interaction term is intended to govern torsional motion about a bond, but these motions are modulated by the flexibility of the nearby bond and angle interactions as well as the nonbonded interactions on either side.

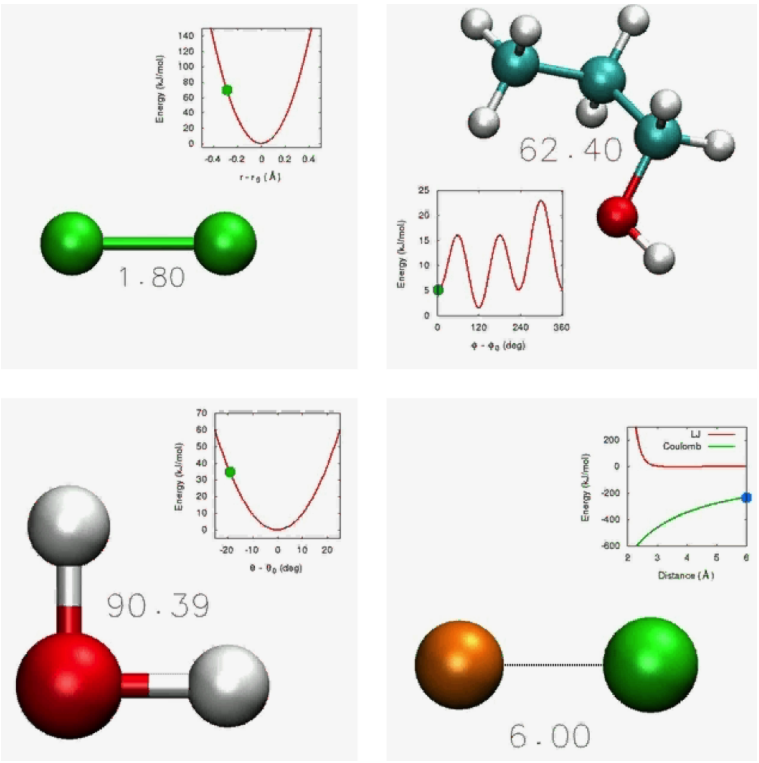


Figure 2: An illustration of some interactions typically found in classical force fields.

For all of these reasons, force field parameterization is difficult. In the current practice, parameters are often determined by fitting to results from other calculations (for example, restrained electrostatic potential fitting (RESP) for determining the partial charges) or chosen so that the simulation results match experimental measurements (for example, adjusting the partial charges on a solvent molecule to reproduce the bulk dielectric constant.) Published force fields have been modified by hand over decades to maximize their agreement with experimental observations (for example, adjusting some parameters in order to reproduce particular protein NMR structure) at the expense of reproducibility.

1.2.2 Purpose and brief description of this program

Given this background, I can make the following statement. **The purpose of ForceBalance is to create force fields by applying a highly general and systematic process with explicitly specified input data and optimization methods, paving the way to higher accuracy and improved reproducibility.**

At a high level, ForceBalance takes an empirical potential and a set of reference data as inputs, and tunes the parameters such that the simulations are able to reproduce the data as accurately as possible. Examples of reference data include energy and forces from high-level QM calculations, experimentally known molecular properties (e.g. polarizabilities and multipole moments), and experimentally measured bulk properties (e.g. density and dielectric constant).

ForceBalance presents the problem of potential optimization in a unified and easily extensible framework. Since there are many empirical potentials in theoretical chemistry and similarly many types of reference data, significant effort is taken to provide an infrastructure which allows a researcher to fit any type of potential to any type of reference data.

Conceptually, a set of reference data (usually a physical quantity of some kind), in combination with a method for computing the corresponding quantity with the force field, is called a **fitting simulation**. For example:

- A force field can predict the density of a liquid by running NPT molecular dynamics, and this computed value can be compared against the experimental density.

- A force field can be used to evaluate the energies and forces at several molecular geometries, and these can be compared against energies and forces from higher-level quantum chemistry calculations using these same geometries. This is known as **force and energy matching**.

- A force field can predict the multipole moments and polarizabilities of a molecule isolated in vacuum, and these can be compared against experimental measurements.

Within the context of a fitting simulation, the accuracy of the force field can be optimized by tuning the parameters to minimize the difference between the computed and reference quantities. One or more fitting simulations can be combined to produce an aggregate **objective function** whose domain is the **parameter space**. This objective function, which typically depends on the parameters in a complex way, is minimized using nonlinear optimization algorithms. The result is a force field with high accuracy in all of the fitting simulations.

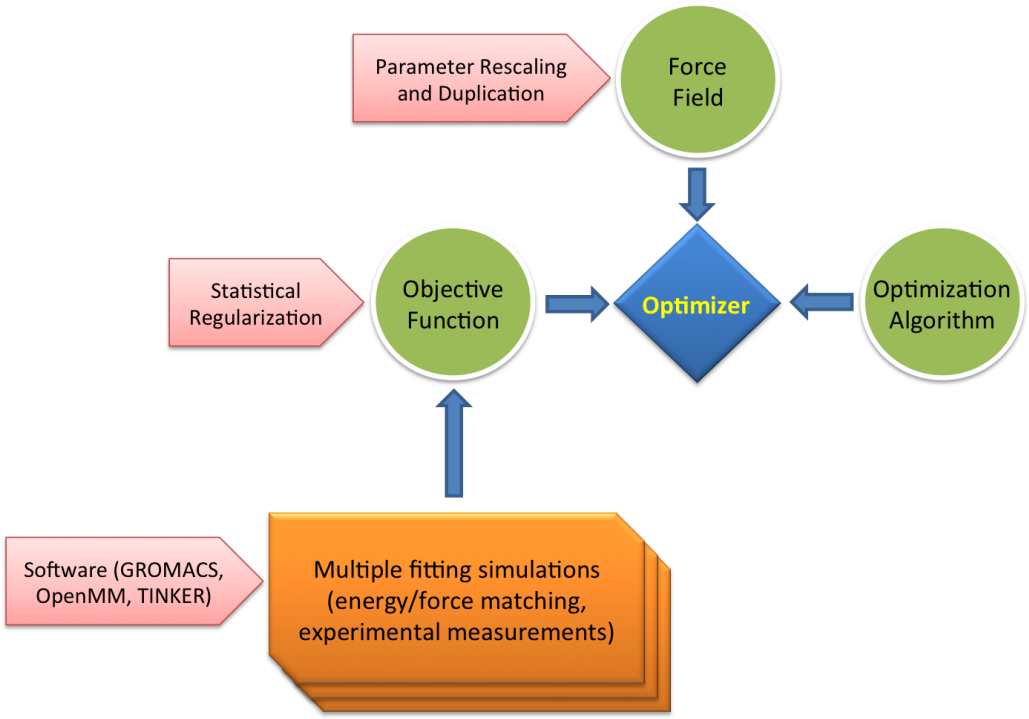


Figure 3: The division of the potential optimization problem into three parts; the force field, fitting simulations and optimization algorithm.

The problem is now split into three main components; the force field, the fitting simulations, and the optimization algorithm. ForceBalance uses this conceptual division to define three classes with minimal interdependence. Thus, if a researcher wishes to explore a new functional form, incorporate a new type of reference data or try a new optimization algorithm, he or she would only need to contribute to one branch of the program without having to restructure the entire code base.

The scientific problems and concepts that this program is based upon are further described in my Powerpoint presentations and publications, which can be found on the SimTK website.

1.3 Credits

- Lee-Ping Wang is the principal developer and author.

- Troy Van Voorhis provided scientific guidance and many of the central ideas as well as financial support.
- Jiahao Chen contributed the call graph generator, the QTPIE fluctuating-charge force field (which Lee-Ping implemented into GROMACS), the interface to the MOPAC semiempirical code, and many helpful discussions.
- Matt Welborn contributed the parallelization-over-snapshots functionality in the general force matching module.
- Vijay Pande provided scientific guidance and financial support, and through the SimBios program gave this software a home on the Web at the SimTK website.
- Todd Martinez provided scientific guidance and financial support.

2 Installation

This section covers how to install ForceBalance.

Currently only Linux is supported, though installation on other Unix-based systems (e.g. Mac OS) should also be straightforward.

Importantly, note that *ForceBalance does not contain a simulation engine*. Instead it interfaces with simulation software like GROMACS, TINKER, AMBER or OpenMM; reference data is obtained from experimental measurements (consult the literature) or from quantum chemistry software (for example, NWChem or Q-Chem).

Several interfaces to existing software packages are provided. However, if you use ForceBalance for a research project, you should be prepared to write some simple Python code to interface with a software package of your choice. If you choose to do so, please contact me as I would be happy to include your contribution in the main distribution.

2.1 Installing ForceBalance

ForceBalance is packaged as a Python module. Here are the installation instructions.

A quick preface: Installing software can be a real pain. I tried to make ForceBalance easy to install by providing clear instructions and minimizing the number of dependencies; however, complications and challenges during installation happen all the time. If you are running into installation problems or having trouble resolving a dependency, please contact me.

2.1.1 Prerequisites

ForceBalance requires the following software packages:

- Python version 2.7
- NumPy version 1.5
- SciPy version 0.9

The following packages are required for certain functionality:

- lxml version 2.3.4 - Python interface to libxml2 for parsing OpenMM force field files
- cctools version 3.4.1 - Cooperative Computing Tools from Notre Dame for distributed computing

The following packages are used for documentation:

- Doxygen version 1.7.6.1
- Doxypy plugin for Doxygen
- LaTeX software such as TeXLive

2.1.2 Installing

To install the package, first extract the tarball that you downloaded from the webpage using the command:

```
tar xvzf ForceBalance-[version].tar.gz
```

Alternatively, download the newest Subversion revision from the SimTK website:

```
svn checkout https://simtk.org/svn/forcebalance
```

Upon extracting the distribution you will see this directory structure:

```
 <root>
   +- bin
   |   |- <Executable scripts>
   +- src
   |   |- <ForceBalance source files>
   +- ext
   |   |- <Extensions; self-contained software packages that are used by ForceBalance>
   +- studies
   |   +- <ForceBalance example jobs>
   +- doc
   |   +- callgraph
   |   |   |- <Stuff for making a call graph>
   |   +- Images
   |   |   |- <Images for the website and PDF manual>
   |   |- make-all-documentation.sh (Create the documentation)
   |   |- <Below are documentation chapters in Doxygen format>
   |   |- introduction.txt
   |   |- installation.txt
   |   |- usage.txt
   |   |- tutorial.txt
   |   |- glossary.txt
   |   |- <The above files are concatenated into mainpage.py>
   |   |- make-all-documentation.sh (Command for making all documentation)
   |   |- make-option-index.py (Create the option index documentation chapter)
   |   |- header.tex (Customize the LaTex documentation)
   |   |- add-tabs.py (Adds more navigation tabs to the webpage)
   |   |- DoxygenLayout.xml (Removes a navigation tab from the webpage)
   |   |- doxygen.cfg (Main configuration file for Doxygen)
   |   |- ForceBalance-Manual.pdf (PDF manual, but the one on the SimTK website is probably newer)
   |- PKG-INFO (Auto-generated package information)
   |- README.txt (Points to the SimTK website)
   |- setup.py (Python script for installation)
```

To install the code into your default Python location, run this (you might need to be root):

```
python setup.py install
```

You might not have root permissions, or you may want to install the package somewhere other than the default location. You can install to a custom location (for example, to /home/leeping/local) by running:

```
python setup.py install --prefix=/home/leeping/local
```

Assuming your Python version is 2.7, the executable scripts will be placed into `/home/leeping/local/bin` and the module will be placed into `/home/leeping/local/lib/python2.7/site-packages/forcebalance`.

Note that Python does not always recognize installed modules in custom locations. Any one of the three below options will work for adding custom locations to the Python search path for installed modules:

```
ln -s /home/leeping/local /home/leeping/.local
```

```
export PYTHONUSERBASE=/home/leeping/local
```

```
export PYTHONPATH=$PYTHONPATH:/home/leeping/local/lib/python2.7
```

As with the installation of any software, there are potential issues with dependencies (for example, scipy and lxml.) One way to resolve dependencies is to use the Enthought Python Distribution (EPD), which contains all of the required packages and is free for academic users. Install EPD from the Enthought website. Configure your environment by running both of the commands below (assuming Enthought was installed to `/home/leeping/opt/epd-7.3.2` , with the python executable in the `bin` subdirectory):

```
export PATH=/home/leeping/opt/epd-7.3.2/bin:$PATH
export PYTHONUSERBASE=/home/leeping/opt/epd-7.3.2
```

Once you have done this, the Numpy, Scipy and lxml dependency issues should be resolved and ForceBalance will run without any problems.

Here are a list of installation notes (not required if you install ForceBalance into the Enthought Python Distribution). These notes assume that Python and other packages are installed into $HOME/local.

- The installation of Numpy, Scipy and lxml may be facilitated by installing the `pip` package - simply run a command like `pip install numpy`.
- Scipy requires a BLAS (Basic Linear Algebra Subroutines) library to be installed. On certain Linux distributions such as Ubuntu, the BLAS libraries and headers can be found on the repository (run `sudo apt-get install libblas-dev`). Also, BLAS is provided by libraries such as ATLAS (Automatically Tuned Linear Algebra Software) or the Intel MKL (Math Kernel Library) for Intel processors. To compile Scipy with Intel's MKL, follow the guide on Intel's website. To use ATLAS, install the package from the ATLAS website and set the ATLAS environment variable (for example, `export ATLAS=$HOME/local/lib/libatlas.so`) before installing Scipy.
- `lxml` is a Python interface to the libxml2 XML parser. After much ado, I decided to use `lxml` instead of the `xml` module in Python's standard library for several reasons (`xml` contains only limited support for XPath, scrambles the ordering of attributes in an element, etc.) The downside is that it can be harder to install. - Installation instructions can be found on the `lxml` website but summarized here. The packages `libxml2` and `libxslt` need to be installed first, and in that order. On Ubuntu, run `sudo apt-get install libxml2-dev libxslt1-dev`. To compile from source, run `./configure --prefix=$HOME/local --with-python=$HOME/local`. Then run `make` followed by `make install`. Python itself needs to be compiled with `--enable-shared` for this to work. Finally, download and unzip `lxml`, then run `python setup.py install --prefix=$HOME/local .`

2.2 Create documentation

This documentation is created by Doxygen with the Doxypy plugin. To create new documentation or expand on what's here, follow the examples in the source code or visit the Doxygen home page.

To create this documentation from the source files, go to the `doc` directory in the distribution and run `doxygen doxygen.cfg` to generate the HTML documentation and LaTeX source files. Run the `add-tabs.py` script to generate the extra navigation tabs for the HTML documentation. Then go to the `latex` directory and type in `make` to build the PDF manual (You will need a LaTeX distribution for this.) All of this is automated by running `make-all-documentation.sh`.

2.3 Installing GROMACS-X2

GROMACS-X2 is not required for ForceBalance and is currently deprecated. Installation is not recommended. This section is retained for your information and in case I choose to revive the software.

I have provided a specialized version of GROMACS (dubbed version 4.0.7-X2) on the SimTK website which interfaces with ForceBalance through the abinitio_gmxx2 module. Although interfacing with unmodified simulation software is straightforward, GROMACS-X2 is optimized for force field optimization and makes things much faster.

GROMACS-X2 contains major modifications from GROMACS 4.0.7. Most importantly, it enables computation of the objective function and its analytic derivatives for rapid energy and force matching. There is also an implementation of the QTPIE fluctuating-charge polarizable force field, and the beginnings of a GROMACS/Q-Chem interface (carefully implemented but not extensively tested). Most of the changes were added in several new source files (less than ten): `qtpie.c`, `fortune.c`, `fortune_utils.c`, `fortune_vsite.c`, `fortune_nb_utils.c`, `zmatrix.c` and their corresponding header files, and `fortunerec.h` for the force matching data structure. The name 'fortune' derives from back when this code was called ForTune.

The force matching functions are turned on by calling `mdrun` with the command line argument `'-fortune'` ; without this option, there should be no impact on the performance of normal MD simulations.

ForceBalance interfaces with GROMACS-X2 through the functions in `abinitio_gmxx2.py` ; the objective function and derivatives are computed and printed to output files. The interface is defined in `fortune.c` on the GROMACS side. ForceBalance needs to know where the GROMACS-X2 executables are located, and this is specified using the `gmxpath` option in the input file.

2.3.1 Prerequisites for GROMACS-X2

GROMACS-X2 needs the base GROMACS requirements and several other libraries.

- FFTW version 3.3
- GLib version 2.0
- Intel MKL library

GLib is the utility library provided by the GNOME foundation (the folks who make the GNOME desktop manager and GTK+ libraries). GROMACS-X2 requires GLib for its hash table (dictionary).

GLib and FFTW can be compiled from source, but it is much easier if you're using a Linux distribution with a package manager. If you're running Ubuntu or Debian, run `sudo apt-get install libglib2.0-dev libfftw3-dev`; if you're using CentOS or some other distro with the yum package manager, run `sudo yum install glib2-devel.x86_64 fftw3-devel.x86_64` (or replace `x86_64` with `i386` if you're not on a 64-bit system.

GROMACS-X2 requires the Intel Math Kernel Library (MKL) for linear algebra. In principle this requirement can be lifted if I rewrite the source code, but it's a lot of trouble, plus MKL is faster than other implementations of BLAS and LAPACK.

The Intel MKL can be obtained from the Intel website, free of charge for noncommercial use. Currently GROMACS-X2 is built with MKL version 10.2, which ships with compiler version 11.1/072 ; this is not the newest version, but it can still be obtained from the Intel website after you register for a free account.

After installing these packages, extract the tarball that you downloaded from the website using the command:

```
tar xvjf gromacs-[version]-x2.tar.bz2
```

The directory structure is identical to GROMACS 4.0.7, but I added some shell scripts. `Build.sh` will run the configure script using some special options, compile the objects, create the executables and install them; you will probably need to modify it slightly for your environment. The comments in the script will help further with installation.

Don't forget to specify the install location of the GROMACS-X2 executables in the ForceBalance input file!

3 Usage

This page describes how to use the ForceBalance software.

A good starting point for using this software package is to run the scripts contained in the `bin` directory on the example jobs in the `studies` directory.

`ForceBalance.py` is the main executable script for force field optimization. It requires an input file and a Directory structure. `MakeInputFile.py` will create an example input file that contains all options, their default values, and a short description for each option. There are plans to automatically generate the correct input file from the provided directory structure, but for now the autogenerated input file provides the hardcoded default options.

3.1 Input file

A typical input file for ForceBalance might look something like this:

```
$options
jobtype newton
gmxpath /home/leeping/opt/gromacs-4.5.5/bin
forcefield water.itp
penalty_additive       0.0
penalty_multiplicative 0.01
convergence_objective 1e-6
convergence_step 1e-6
print_parameters no
print_gradient no
print_hessian no
$end

$simulation
simtype abinitio_gmx
name water12_a
weight 1
w_energy 1.0
w_force 1.0
$end

$simulation
simtype abinitio_gmx
name water12_b
weight 1
w_energy 1.0
w_force 1.0
$end
```

Global options for a ForceBalance job are given in the `$options` section while the settings for each fitting simulation are given in the `$simulation` sections. These are the only two section types.

The most important general options to note are: `jobtype` specifies the optimization algorithm to use and `forcefield` specifies the force field file name (there may be more than one of these). The most important simulation options to note are: `simtype` specifies the type of fitting simulation and `name` specifies the simulation name (must correspond to a subdirectory in `simulations/`). All options are explained in the Option Index.

3.2 Directory structure

The directory structure for our example job would look like:

```
<root>
  +- forcefield
  |   |- water.itp
  +- simulations
  |   +- water12_gen1
  |   |   +- settings
  |   |   |   |- shot.mdp
  |   |   |   |- topol.top
  |   |   |- all.gro (containing 300 geometries)
  |   |   |- qdata.txt
  |   +- water12_gen2
  |   |   |   |- shot.mdp
  |   |   |   |- topol.top
  |   |   |- all.gro (containing 300 geometries)
  |   |   |- qdata.txt
  |- input.in
  +- temp
  |   |- iter_0001
  |   |- iter_0002
  |   |   |- <files generated during runtime>
  +- result
  |   |- water.itp (Optimized force field, generated on completion)
  |- input.in (ForceBalance input file)
```

The top-level directory names **forcefield** and **simulations** are fixed and cannot be changed. **forcefield** contains the force field files that you're optimizing, and **simulations** contains all of the fitting simulations and reference data. Each subdirectory in **simulations** corresponds to a single fitting simulation, and its contents depend on the specific kind of simulation and its corresponding `FittingSimulation` subclass.

The **temp** directory is the temporary workspace of the program, and the **result** directory is where the optimized force field files are deposited after the optimization job is done. These two directories are created if not already there.

Note the force field file, `water.itp` and the two fitting simulations `water12_gen1` and `water12_gen2` correspond to the entries in the input file. There are two energy and force matching simulations here; each directory contains the relevant geometries (in `all.gro`) and reference data (in `qdata.txt`).

3.3 Setting up the fitting simulations

There are many fitting simulations one can choose from.

- Energy and force matching - this is the oldest functionality. Enabled in GROMACS, OpenMM, TINKER, AMBER.
- Electrostatic potential fitting via the RESP method. Enabled in GROMACS and AMBER.
- High-performance interaction energies - intended for the same two fragments in many conformations. Enabled in GROMACS.
- General binding energies - intended for highly diverse collections of complexes and fragments. Enabled in TINKER.

- Normal mode frequencies. Enabled in TINKER.
- Condensed-phase properties; currently enabled only for density and enthalpy of vaporization of water. Enabled in OpenMM.
- Basis set coefficient fitting; enabled in psi4 (experimental)

One salient quality of ForceBalance is that fitting simulations can be linearly combined to produce an aggregate objective function. For example, our recently developed polarizable water model contains energy and force matching, binding energies, normal mode frequencies, density, and enthalpy of vaporization. With the AMOEBA functional form and 19 adjustable parameters, we developed a highly accurate model that fitted all of these properties to very high accuracy.

Due to the diverse (and somewhat esoteric) nature of these calculations, they need to be set up in a specific way that is recognized by ForceBalance. The setup is different for each type of simulation, and we encourage you to learn by example through looking at the files in the `studies` directory.

3.3.1 and force matching

In these relatively simple simulations, the objective function is computed from the squared difference in the potential energy and forces (gradients) between the force field and reference (QM) method, evaluated at a number of stored geometries called *snapshots*. The mathematics are implemented in `abinitio.py` while the interfaces to simulation software exist in derived classes in `gmxio.py`, `tinkerio.py`, `amberio.py` and `openmmio.py`.

All energy and force matching simulations require a *coordinate trajectory file* (`all.gro`) and a *quantum data file* (`qdata.txt`). The coordinate trajectory file contains the Cartesian coordinates of the snapshots, preferably in the file format of the simulation software used (`all.gro` is the most extensively tested.) The quantum data file is formatted according to a very simple specification:

```
JOB 0
COORDS x1 y1 z1 x2 y2 z2 ... (floating point numbers in Angstrom)
ENERGY (floating point number in Hartree)
FORCES fx1 fy1 fz1 ... (floating point numbers in Hartree/bohr - this is a misnomer because they are actually gr

JOB 1
...
```

The coordinates in the quantum data file should be consistent with the coordinate trajectory file, although ForceBalance will use the latter most of the time. It is easy to generate the quantum data file from parsing the output of quantum chemistry software. ForceBalance contains example methods for parsing Q-Chem output files.

In addition to `all.gro` and `qdata.txt` , the simulation setup files are required. These contain settings needed by the simulation software for the calculation to run. For Gromacs calculations, a topology (.top) file and a run parameter file (.mdp) are required. These should be placed in the `settings` subdirectory within the directory belonging to the fitting simulation.

As a side note: If you wish to tune a number in the .mdp file, simply move it to the `forcefield` directory and specify it as a force field file. ForceBalance will now be able to tune any highlighted parameters in the file, although it will also place copies of this file in all simulation directories within `temp` while the program is running.

3.3.2 potentials

ForceBalance contains methods for evaluating electrostatic potentials given a collection of point charges. At this time, this functionality is very experimental and risky to use for systems containing more than one molecule. This is because ForceBalance evaluates the electrostatic potentials internally, and we don't have an infrastructure for building a full topology consisting of many molecules. Currently, we assume that the electrostatic potential fitting contains only one molecule.

Once again, the coordinate trajectory file and quantum data files are used to specify the calculation. However, now the coordinates for evaluating the potential, and the reference potential values, are included:

```
JOB 0
COORDS x1 y1 z1 x2 y2 z2 ...
ENERGY e
FORCES fx1 fy1 fz1 ...
ESPXYZ ex1 ey1 ez1 ex2 ey2 ez2 ...
ESPVAL ev1 ev2 ev3 ..
```

3.4 Running the optimization

To run ForceBalance, make sure the calculation is set up properly (refer to the above sections), and then type in:

ForceBalance.py input.in

In general it's impossible to set up a calculation perfectly the first time, in which case the calculation will crash. ForceBalance will try to print helpful error messages to guide you toward setting up your calculation properly.

Further example inputs and outputs are given in the Tutorial section.

4 Tutorial

This is a tutorial page, but if you haven't installed ForceBalance yet please go to the Installation page first.

It is very much in process, and there are many more examples to come.

4.1 Fitting a TIP4P potential using two fitting simulations

After everything is installed, go to the `test` directory in the distribution and run:

```
cd 001_water12_tip4p/
OptimizePotential.py 01_bfgs_from_start.in | tee my_job.out
```

If the installation was successful, you will get an output file similar to `01_bfgs_from_start.out` . `OptimizePotential.py` begins by taking the force field files from the `forcefield` directory and the fitting simulations / reference data from the `simulations` directory. Then it calls GROMACS-X2 to compute the objective function and its derivatives, uses the internal optimizer (based on BFGS) to take a step in the parameter space, and repeats the process until convergence criteria were made.

At every step, you will see output like:

```
   Step        |k|         |dk|        |grad|        -=X2=-      Stdev(X2)
     35    6.370e-01    1.872e-02    9.327e-02    2.48773e-01    1.149e-04

 Sim: water12_sixpt   E_err(kJ/mol)=     8.8934 F_err(%)=    29.3236
 Sim: water12_fourpt  E_err(kJ/mol)=    14.7967 F_err(%)=    39.2558
```

The first line reports the step number, the length of the parameter displacement vector, the gradient of the objective function, the objective function itself, and the standard deviation of the last ten *improved* steps in the objective function. There are three kinds of convergence criteria - the step size, the gradient, and the objective function itself; all of them can be specified in the input file.

The next two lines report on the two fitting simulations in this job, both of which use force/energy matching. First, note that there are two fitting simulations named `water12_sixpt` and `water12_fourpt`; the names are because one set of geometries was sampled using a six-site QTPIE force field, and the other was sampled using the TIP4P force field. However, the TIP4P force field is what we are fitting for this ForceBalance job. This shows how only one force field or parameter set is optimized for each ForceBalance job, but the method for sampling the configuration space is completely up to the user. The geometries can be seen in the `all.gro` files, and the reference data is provided in `qdata.txt` . Note that the extra virtual sites in `water12_sixpt` have been replaced with a single TIP4P site.

`E_err` and `F_err` report the RMS energy error in kJ/mol and the percentage force error; note the significant difference in the quality of agreement! This illustrates that the quality of fit depends not only on the functional form of the potential but also the configurations that are sampled. `E_err` and `F_err` are 'indicators' of our progress - that is, they are not quantities to be optimized but they give us a mental picture of how we're doing.

The other input files in the directory use the same fitting simulations, but they go through the various options of reading/writing checkpoint files, testing gradients and Hessians by finite difference, and different optimizers in SciPy. Feel free to explore some optimization jobs of your own - for example, vary the weights on the fitting simulations and see what happens. You will notice that the optimizer will try very hard to fit one simulation but not the other.

5 Glossary

This is a glossary page containing scientific concepts for the discussion of potential optimization, as well as the (automatically generated) documentation of ForceBalance keywords.

5.1 Scientific concepts

- **Empirical parameter** : Any adjustable parameter in the empirical potential that affects the potential energy, such as the partial charge on an atom, the equilibrium length of a chemical bond, or the fraction of Hartree-Fock exchange in a density functional.

- **Empirical Potential** : A formula that contains empirical parameters and computes the potential energy of a collection of atoms. Note that in ForceBalance this is used very loosely; even a DFT functional may contain many empirical parameters, and ForceBalance has the ability to optimize these as well!

- **Fitting simulation** : A simulation protocol that allows a force field to predict a physical quantity, paired with some reference data. The accuracy of the force field is given by its closeness

- **Force field** : This term is used interchangeably with empirical potential; it is more prevalent in the biomolecular simulation community.

- **Functional form** : The mathematical functions in the force field. For instance, a CHARMM-type functional form has harmonic interactions for bonds and angles, a cosine expansion for the dihedrals, Coulomb interactions between point charges and Lennard-Jones terms for van der Waals interactions.

- **Reference data** : In general, any accurately known quantity that the force field is optimized to reproduce. - Reference data can come from either theory or experiment. For instance, energies and forces from a high-level QM method can be used as reference data (for instance, a CHARMM-type force field can be fitted to reproduce forces from a DFT or MP2 calculation), or a force field can be optimized to reproduce the experimental density of a liquid, its enthalpy of vaporization or the solvation free energy of a solute.

5.2 Option index: General options

This section contains a listing of the general options available when running a ForceBalance job, which go into the $options section. The general options are global for the ForceBalance job, in contrast to 'Simulation options' which apply to one fitting simulation within a job (described in the next section). The option index is generated by running make-option-index.py.

- **ADAPTIVE_DAMPING** (Float)

  Need to document

- **ADAPTIVE_FACTOR** (Float)

  Need to document

- **AMOEBA_POLARIZATION** (String)

  Need to document

- **BACKUP** (Bool)

  Scope : All force field optimizations (Optional)

  One-line description : Write temp directories to backup before wiping them

  Default Value : 1

- **CONSTRAIN_CHARGE** (Bool)

  Need to document

- **CONVERGENCE_GRADIENT** (Float)

  Scope : Main optimizer (Optional)

  One-line description : Convergence criterion of gradient norm

  Full description : The main optimizer will quit when the objective function gradient falls below this number. Since this is a newly implemented option, I can't say when this option will fail.

  Default Value : 0.0001

  Recommendation : Leave at the default, or set to several orders of magnitude below a typical value of the gradient (perhaps the gradient at the start of the optimization.)

- **CONVERGENCE_OBJECTIVE** (Float)

  Scope : Main optimizer (Optional)

  One-line description : Convergence criterion of objective function (in MainOptimizer this is the stdev of x2 over [objective_history] steps)

  Full description : The main optimizer will quit when the last ten good values of the objective function have a standard deviation that falls below this number. We use the last ten good values (instead of the latest change in the objective function), otherwise this condition would be triggered by taking tiny steps.

  Default Value : 0.0001

  Recommendation : Decrease this value if it's being triggered by small step sizes.

- **CONVERGENCE_STEP** (Float)

  Scope : Main optimizer (Optional)

  One-line description : Convergence criterion of step size (just needs to fall below this threshold)

Full description : The main optimizer will quit when the step size falls below this number. This happens if we are approaching a local minimum, or if the optimizer is constantly taking bad steps and the trust radius is reduced until it falls below this number. In the latter case, this usually means that the derivatives are wrong.

Default Value : 0.0001

Recommendation : Make sure that this value is much smaller than trust0.

- **EIG_LOWERBOUND** (Float)

  Scope : Main optimizer (Optional)

  One-line description : Minimum eigenvalue for applying steepest descent correction in the MainOptimizer

  Full description : The main optimizer will misbehave if there are negative or very small eigenvalues in the objective function Hessian. In the former case the optimizer will travel toward a saddle point (or local maximum), and in the latter case the matrix inversion will fail because of the matrix singularity. If the smallest eigenvalue is below this value, then a multiple of the identity matrix is added to the Hessian to increase the smallest eigenvalue to at least this value.

  Default Value : 0.0001

  Recommendation : Shouldn't have to worry about this setting, unless the optimizer appears to be taking bad steps or inverting nearly singular matrices.

- **ERROR_TOLERANCE** (Float)

  Need to document

- **FFDIR** (String)

  Scope : All force field optimizations (Optional)

  One-line description : Directory containing force fields, relative to project directory

  Default Value : forcefield

  Recommendation : Unless you're using a nonstandard location for force field files, you probably shouldn't change this.

- **FINITE_DIFFERENCE_H** (Float)

  Scope : fdcheck_G or fdcheck_H job types, or whenever the objective function is evaluated using finite difference (Optional)

  One-line description : Step size for finite difference derivatives in many functions (get_(G/H) in fitsim, FDCheck-G)

  Full description : When the objective function derivatives are checked using finite difference, or when the objective function derivative requires finite difference, this is the step size that is used (in the mathematical space). The actual parameter in the force field is changed by this amount times the rescaling factor.

  Default Value : 0.01

  Recommendation : 1e-2 to 1e-4; run FDCheckG to see if derivatives are accurate; if derivatives are inaccurate then adjust accordingly. If the objective function itself requires finite difference, there will still be a difference because FDCheckG(H) uses an accurate seven-point (five-point) stencil. Make sure that the derivatives agree before settling on a value to use.

- **FORCEFIELD** (List)

  Scope : All force field optimizations (***Required***)

  One-line description : The names of force fields, corresponding to directory forcefields/file_name.(itp,xml,prm,frcmod,mol2)

  Default Value : []

- **GMXPATH** (String)

  Scope : Fitting simulations that use GROMACS (GROMACS-X2 for ForceEnergyMatch_GMX) (***Required***)

  One-line description : Path for GROMACS executables

  Full description : Specify the path where GROMACS executables are installed, most likely ending in 'bin'. Note that executables are only installed 'bin' if the program is installed using 'make install'; this will NOT be the case if you simply ran 'make'.

  Default Value : None

  Recommendation : Depends on your local installation and environment.

- **GMXSUFFIX** (String)

  Scope : Fitting simulations that use GROMACS (Optional)

  One-line description : The suffix of GROMACS executables

  Full description : Depending on how GROMACS is configured and installed, a suffix may be appended to executable names. If there is a suffix, it needs to be specified here (or else ForceBalance will not find the GROMACS executable and it will crash.

  Default Value :

  Recommendation : Depends on your local installation and environment.

- **HAVE_VSITE** (Bool)

  Need to document

- **JOBTYPE** (Allcap)

  Scope : All force field optimizations (***Required***)

  One-line description : The job type, defaults to a single-point evaluation of objective function

  Full description : Here you may specify the type of ForceBalance job. This ranges from gradient-based and stochastic optimizations to simple scans over the parameter space and finite difference checking of gradients.

  Default Value : single

  Recommendation : See the Optimizer class documentation for which optimizer is best suited for you.

- **LOGARITHMIC_MAP** (Bool)

  Need to document

- **MAXSTEP** (Int)

  Scope : All iterative optimization jobs (Optional)

  One-line description : Maximum number of steps in an optimization

  Default Value : 100

  Recommendation : At least 100 optimization steps are recommended.

- **MINTRUST** (Float)

  Need to document

- **NORMALIZE_WEIGHTS** (Bool)

  Need to document

- **OBJECTIVE_HISTORY** (Int)

  Need to document

- **PENALTY_ADDITIVE** (Float)

  Scope : Objective function (Optional)

  One-line description : Factor for additive penalty function in objective function

  Full description : Add a penalty to the objective function (e.g. L2 or L1 norm) with this prefactor. Using an additive penalty requires an assessment of the order of magnitude of the objective function, but it is closer to the statistical concept of ridge or LASSO regression.

  Default Value : 0.0

  Recommendation : No recommendation; run a single-point calculation to choose a prefactor. Consider 0.01 for an objective function of order 1.

- **PENALTY_HYPERBOLIC_B** (Float)

  Need to document

- **PENALTY_MULTIPLICATIVE** (Float)

  Scope : Objective function (Optional)

  One-line description : Factor for multiplicative penalty function in objective function

  Full description : Multiply the objective function by (1+X) where X is this value. Using an multiplicative penalty works well for objective functions of any size but it is not equivalent to statistical regularization methods.

  Default Value : 0.0

  Recommendation : A value of 0.01 tends to keep the length of the parameter vector from exceeding 1.

- **PENALTY_TYPE** (String)

  Scope : All force field optimizations (Optional)

  One-line description : Type of the penalty, L2 or Hyp in the optimizer

  Full description : To prevent the optimization from changing the parameters too much, an additional penalty is applied to the objective function that depends linearly (L1) or quadratically (L2) on the norm of the parameter displacement vector. L1 corresponds to LASSO regularization while L2 is known as Tikhonov regularization or ridge regression.

  Default Value : L2

  Recommendation : L2; tested and known to be working. Implementation of L1 in progress.

- **PRINT_GRADIENT** (Bool)

  Need to document

- **PRINT_HESSIAN** (Bool)

  Need to document

- **PRINT_PARAMETERS** (Bool)

  Need to document

- **PRIORS** (Section)

  Need to document

- **READ_MVALS** (Section)

  Scope : All force field optimizations (Optional)

  One-line description : Paste mathematical parameters into the input file for them to be read in directly

  Full description : Read in mathematical parameters before starting the optimization. There is a standardized syntax, given by:

```
read_mvals
 0 [ -2.9766e-01 ] : VDWSOW
 1 [  2.2283e-01 ] : VDWTOW
 2 [ -1.1138e-03 ] : BONDSBHWOW
 3 [ -9.0883e-02 ] : BONDSKHWOW
 \read_mvals
```

  Default Value : None

  Recommendation : If you run the main optimizer, it will print out this block at the very end for you to use and/or modify.

- **READ_PVALS** (Section)

  Scope : All force field optimizations (Optional)

  One-line description : Paste physical parameters into the input file for them to be read in directly

  Full description : Read in physical parameters before starting the optimization. There is a standardized syntax, given by:

```
read_pvals
  0 [  2.9961e-01 ] : VDWSOW
  1 [  1.2009e+00 ] : VDWTOW
  2 [  9.5661e-02 ] : BONDSBHWOW
  3 [  4.1721e+05 ] : BONDSKHWOW
  \read_pvals
```

  These are the actual numbers that go into the force field file, so note the large changes in magnitude.

  Default Value : None

  Recommendation : If you run the main optimizer, it will print out this block at the very end for you to use and/or modify.

- **READCHK** (String)

  Scope : Main optimizer (Optional)

  One-line description : Name of the restart file we read from

  Full description : The main optimizer has the ability to pick up where it left off by reading / writing checkpoint files. Here you may specify the checkpoint file to read in from a previous optimization run. This is equivalent to reading in stored parameter values, except the gradient and Hessian (which contains memory from previous steps) is recorded too.

  Default Value : None

- **SCAN_VALS** (String)

  Scope : scan_mvals and scan_pvals job types (Optional)

  One-line description : Values to scan in the parameter space for job type "scan[mp]vals", given like this: -0.1:0.1:11

  Full description : This specifies a range of parameter values to scan in a uniform grid. scan_mvals works in the mathematical parameter space while scan_pvals works in the physical parameter space. The syntax is lower:step:upper . Both lower and upper limits are included in the range.

  Default Value : None

  Recommendation : For scan_mvals, a range of values between -1 and +1 is recommended; for scan_pvals, choose values close to the physical parameter value.

- **SCANINDEX_NAME** (List)

  Scope : scan_mvals and scan_pvals job types (Optional)

  One-line description : Parameter name to scan over (should convert to a numerical index) in job type "scan[mp]vals"

  Full description : ForceBalance assigns to each adjustable parameter a 'parameter name'. By specifying this option, this tells the parameter scanner to locate the correct parameter with the specified name and then scan over it.

  Default Value : []

  Recommendation : Look at the printout from a single-point job to determine the parameter names.

- **SCANINDEX_NUM** (List)

  Scope : scan_mvals and scan_pvals job types (Optional)

  One-line description : Numerical index of the parameter to scan over in job type "scan[mp]vals"

  Full description : ForceBalance assigns to each adjustable parameter a 'parameter number' corresponding to its position in the parameter vector. This tells the parameter scanner which number to scan over.

  Default Value : []

  Recommendation : Look at the printout from a single-point job to decide which parameter number you wish to scan over.

- **TINKERPATH** (String)

  Scope : Fitting simulations that use TINKER (***Required***)

  One-line description : Path for TINKER executables

  Default Value : None

  Recommendation : Depends on your local installation and environment.

- **TRUST0** (Float)

  Scope : Main optimizer (Optional)

  One-line description : Trust radius for the MainOptimizer

  Full description : The main optimizer uses a trust radius which 'adapts' (i.e. increases or decreases) based on whether the last step was a good or bad step. 'trust0' provides the starting trust radius, and the trust radius is not allowed to increase too much from trust0.

  Default Value : 0.1

  Recommendation : Increase from the default if the optimizer takes many good steps but takes too long; decrease if the optimizer takes many bad steps.

- **WRITECHK** (String)

  Scope : Main optimizer (Optional)

  One-line description : Name of the restart file we write to (can be same as readchk)

  Full description : The main optimizer has the ability to pick up where it left off by reading / writing checkpoint files. Here you may specify the checkpoint file to write after the job is finished.

  Default Value : None

  Recommendation : Writing the checkpoint file is highly recommended.

- **WRITECHK_STEP** (Bool)

  Scope : Main optimizer when 'writechk' is turned on (Optional)

  One-line description : Write the checkpoint file at every optimization step

  Full description : Write a checkpoint file every single step, not just after the job is finished.

  Default Value : 1

  Recommendation : Useful if you want to quit an optimization before it finishes and restart, but make sure you don't overwrite existing checkpoint files by accident.

5.3 Option index: Simulation options

This section contains a listing of the simulation options available when running a ForceBalance job, which go into the $sim_opts section. There can be multiple $sim_opts sections in a ForceBalance input file, one for each fitting simulation.

- **ALL_AT_ONCE** (Bool)

  Need to document

- **BATCH_FD** (Bool)

  Scope : All fitting simulations (Optional)

  One-line description : Whether to batch and queue up finite difference jobs, defaults to False

  Full description : This is a stub for future functionality. When the flag is switched on, the jobs corresponding to finite difference derivatives are evaluated in parallel on a distributed computing platform.

  Default Value : 0

- **COVARIANCE** (Bool)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Whether to use the quantum covariance matrix (ab initio), defaults to False

  Full description : The components of the energy and force contribution to the objective function are rescaled to be on the same footing when the objective function is optimized. This can be done by dividing each component by its variance, or by multiplying the energy-force polytensor by the inverse of the quantum energy-force covariance matrix. The latter method was proposed as a way to emphasize intermolecular interactions but it is unproven.

  Default Value : 0

  Recommendation : No recommendation; turn the covariance off if the number of snapshots is not much larger than the number of coordinates.

- **DO_COSMO** (Bool)

  Need to document

- **ENERGY** (Bool)

  Need to document

- **ENERGY_DENOM** (Float)

  Need to document

- **FD_PTYPES** (List)

  Scope : All fitting simulations (Optional)

  One-line description : The parameter types that need to be differentiated using finite difference

  Full description : To compute the objective function derivatives, some components may require numerical finite difference in the derivatives. Here you may specify the parameter types that finite difference is applied to, or write 'ALL' to take finite-difference derivatives in all parameter types.

  Default Value : []

  Recommendation : If you aren't sure, either use 'ALL' to do finite difference in each component (this is costly), or run a fdcheckG(H) job with this option set to 'NONE' to check which analytic derivatives are missing. Usually analytic derivatives will be missing in anything but FORCEENERGYMATCH_GMXX2 jobs.

- **FDGRAD** (Bool)

  Scope : All fitting simulations (Optional)

  One-line description : Finite difference gradients

  Full description : When this option is enabled, finite difference gradients will be enabled for selected parameter types (using the fd_ptypes option). Gradients are computed using two-point finite difference of the objective function.

  Default Value : 1

  Recommendation : If analytic derivatives are implemented (and correct), then they are much faster than finite difference derivatives. Run the 'fdcheckG' routine with this option set to Off to check which finite difference derivatives you need.

- **FDHESS** (Bool)

  Scope : All fitting simulations (Optional)

  One-line description : Finite difference Hessian diagonals (costs np times a gradient calculation)

  Full description : When this option is enabled, finite difference Hessians will be enabled for selected parameter types (using the fd_ptypes option). Hessians are computed using two-point finite difference of the gradient.

  Default Value : 0

  Recommendation : Run the 'fdcheckH' routine with this option set to Off to check which finite difference Hessian elements you need. Note that this requires a very large number of objective function evaluations, so use sparingly.

- **FDHESSDIAG** (Bool)

  Scope : All fitting simulations (Optional)

  One-line description : Finite difference Hessian diagonals (cheap; costs 2np times a objective calculation)

  Full description : When this option is enabled, finite difference gradients and Hessian diagonal elements will be enabled for selected parameter types (using the fd_ptypes option). This is done using a three-point finite difference of the objective function.

  Default Value : 1

  Recommendation : Use this as a substitute for 'fdgrad'; it doubles the cost but provides more accurate derivatives plus the Hessian diagonal values (these are very nice for quasi-Newton optimizers like BFGS).

- **FITATOMS** (Int)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Number of fitting atoms (ab initio); defaults to all of them

  Full description : Choose a subset of atoms from the force matching simulation to fit forces to. This is useful in situations where it is undesirable to fit the forces on part of the system (e.g. the part that is described by another

force field.) Currently, you are only allowed to choose from the atoms in the front of the trajectory; soon this will be expanded for random flexibility (see 'shots'). However, random coordinate selections are not allowed. ;)

Default Value : 0

Recommendation : Situation-dependent; this should be based on the part of the simulation that you're fitting, or leave blank if you're fitting the whole system.

- **FORCE** (Bool)

  Need to document

- **MASTERFILE** (String)

  Need to document

- **NAME** (String)

  Scope : All fitting simulations (***Required***)

  One-line description : The name of the simulation, which corresponds to the directory simulations/dir_name

  Default Value : None

  Recommendation : Choose a descriptive name and make sure all fitting simulations have different names.

- **QMBOLTZ** (Float)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Fraction of Quantum Boltzmann Weights (ab initio), 1.0 for full reweighting, 0.5 for hybrid

  Full description : When Boltzmann sampling is used to gather snapshots for force/energy matching, there is a potential ambiguity regarding which ensemble one should sample from (either the force field's ensemble or the QM calculation's ensemble. The QM ensemble may be sampled using MM-sampled snapshots by reweighting; this tuning parameter specifies the fraction of QM Boltzmann weight to include. Note that when two ensembles are different, reweighting will decrease the statistical significance of the number of snapshots (i.e. there is less InfoContent).

  Default Value : 0.0

  Recommendation : If you want to reweight your snapshots entirely to the QM ensemble, choose 1.0; for hybrid weights, use 0.5. Avoid if the fitting simulation has a very large RMS energy difference between QM and MM.

- **QMBOLTZTEMP** (Float)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Temperature for Quantum Boltzmann Weights (ab initio), defaults to room temperature

  Full description : The reweighting of an ensemble involves an exponential of (DE)/kT, so there is a massive degradation of sample quality if (DE) is large. This option allows you to change the temperature in the denominator, which is unphysical (but it does decrease the effect of moving toward the QM ensemble.

  Default Value : 298.15

  Recommendation : Irrelevant if 'qmboltz' is set to zero. Leave at the default value unless you're performing experiments.

- **RESP** (Bool)

  Need to document

- **RESP_A** (Float)

  Need to document

- **RESP_B** (Float)

  Need to document

- **RMSD_DENOM** (Float)

  Need to document

- **RUN_INTERNAL** (Bool)

  Need to document

- **SAMPCORR** (Bool)

  Scope : Force and energy matching simulations that use GROMACS-X2 (Optional)

  One-line description : Whether to use the (archaic) sampling correction (ab initio), defaults to False

  Full description : Every time the force field parameters are updated, the ensemble is different. In principle this applies to not only the self-consistent optimization cycles (which include re-running dynamics, QM calculations etc) but also the numerical optimization itself. When this option is turned on, the Boltzmann weights of the snapshots are updated in every step of the optimization and the derivatives are modified accordingly. My investigations reveal that this makes the force field more accurate in energy minima and less accurate for barriers, which was not very useful. I haven't touched the 'sampling corrected' code in a long time; thus this option is vestigial and may be removed in the future.

  Default Value : 0

  Recommendation : Off.

- **SHOTS** (Int)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Number of snapshots (ab initio); defaults to all of the snapshots

  Full description : This option allows you to choose a subset from the snapshots available in the force matching 'simulations' directory. The subset is simply taken from the front of the trajectory. In the future this option will be expanded to allow a random selection of snapshots, or a specific selection

  Default Value : -1

  Recommendation : 100-10,000 snapshots are recommended. Note that you need at least 3x (number of atoms) if the covariance matrix is turned on.

- **SIMTYPE** (Allcap)

  Scope : All fitting simulations (***Required***)

  One-line description : The type of fitting simulation, for instance AbInitio_GMXX2

  Full description : This is the type of fitting simulation that you are running. The current accepted values for the fitting simulation are given in the SimTab.py file: COUNTERPOISE, LIQUID_OPENMM, ABINITIO_OPENMM, ABINITIO_GMXX2, THCDF_PSI4, INTERACTIONS_TINKER, ABINITIO_AMBER, ABINITIO_GMX, INTERACTION_GMX, ABINITIO_TINKER, ABINITIO_INTERNAL, VIBRATION_TINKER.

  Default Value : None

  Recommendation : Choose the appropriate type, and if the fitting simulation is missing, feel free to implement your own (or ask me for help).

- **USE_PVALS** (Bool)

  Scope : All fitting simulations (Optional)

  One-line description : Bypass the transformation matrix and use the physical parameters directly

Full description : When this option is enabled, the coordinate transformation in parameter space will be bypassed, and parameters passed into the 'get' subroutines will be plugged directly into the force field files. This option is turned on automatically if we are running a 'scan_pvals' job. Note that the coordinate transformation is essential in multi-parameter optimizations and preserves the condition number of the Hessian, so turning it off should generally be avoided.

Default Value : 0

Recommendation : This option should almost always be off unless the user really knows what he/she is doing.

- **W_ENERGY** (Float)

  Need to document

- **W_FORCE** (Float)

  Need to document

- **W_HVAP** (Float)

  Need to document

- **W_RESP** (Float)

  Need to document

- **W_RHO** (Float)

  Need to document

- **WAVENUMBER_TOL** (Float)

  Need to document

- **WEIGHT** (Float)

  Scope : All fitting simulations (Optional)

  One-line description : Weight of the simulation (vs. other simulations)

  Full description : This option specifies the weight that the fitting simulation will contribute to the objective function. A larger weight for a given fitting simulation means that the optimizer will prioritize it over the others. When several fitting simulations are used, the weight should be chosen carefully such that all fitting simulations contribute a finite amount to the objective function. Note that the choice of weight determines the final outcome of the force field, although we hope not by too much.

  Default Value : 1.0

  Recommendation : It is important to specify something here (giving everything equal weight is unlikely to work.) Run a single-point objective function evaluation with all weights set to one to get a handle on the natural size of each fitting simulation's contribution, and then add weights accordingly.

- **WHAMBOLTZ** (Bool)

  Scope : Force and energy matching simulations (Optional)

  One-line description : Whether to use WHAM Boltzmann Weights (ab initio), defaults to False

  Full description : In self-consistent energy/force matching projects, the data from previous cycles can be reused by applying the Weighted Histogram Analysis Method (WHAM). However, the WHAM data is currently generated by external scripts that haven't made it into this distribution yet. In the future, generation of WHAM data will be incorporated into this program automatically.

  Default Value : 0

  Recommendation : Leave off unless you have an externally generated wham-master.txt and wham-weights.txt files.

- **WQ_PORT** (Int)

  Need to document

Figure 4: Logo.

6 Namespace Index

6.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

ForceBalance
Executable script for starting ForceBalance 27

GenerateQMData
Executable script for generating QM data for force, energy, electrostatic potential, and other ab initio-based fitting simulations 28

MakeInputFile
Executable script for printing out an example input file with defaults and documentation 28

ParseInputFile
Read in ForceBalance input file and print it back out 29

7 Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

SelfConsistentCycle.ForceBalance_SCF 29

TagMol2.MolG 30

CallGraph.node
Data structure for holding information about python objects 32

SelfConsistentCycle.SCF_Simulation 32

8 Namespace Documentation

8.1 ForceBalance Namespace Reference

Executable script for starting ForceBalance.

Functions

- def Run_ForceBalance
  Create instances of ForceBalance components and run the optimizer.
- def **main**

8.1.1 Detailed Description

Executable script for starting ForceBalance.

8.1.2 Function Documentation

8.1.2.1 def ForceBalance.Run_ForceBalance (*input_file*)

Create instances of ForceBalance components and run the optimizer.

The triumvirate, trifecta, or trinity of components are:

- The force field
- The objective function
- The optimizer Cipher: "All I gotta do here is pull this plug... and there you have to watch Apoc die" Apoc: "TRINITY" ∗chunk∗

The force field is a class defined in forcefield.py. The objective function is a combination of fitting simulation classes and a penalty function class. The optimizer is a class defined in this file.

Definition at line 28 of file ForceBalance.py.

8.2 GenerateQMData Namespace Reference

Executable script for generating QM data for force, energy, electrostatic potential, and other ab initio-based fitting simulations.

Functions

- def even_list

  Creates a list of number sequences divided as easily as possible.
- def **generate_snapshots**
- def **drive_msms**
- def **create_esp_surfaces**
- def **do_quantum**
- def **gather_generations**
- def **Generate**
- def **main**

Variables

- dictionary **VdW99** = {'H' : 1.00, 'C' : 1.70, 'N' : 1.625, 'O' : 1.49, 'F' : 1.56, 'P' : 1.871, 'S' : 1.782, 'I' : 2.094, 'Cl' : 1.735, 'Br': 1.978}

8.2.1 Detailed Description

Executable script for generating QM data for force, energy, electrostatic potential, and other ab initio-based fitting simulations.

8.2.2 Function Documentation

8.2.2.1 def GenerateQMData.even_list (*totlen, splitsize*)

Creates a list of number sequences divided as easily as possible.

Intended for even distribution of QM calculations. However, this might become unnecessary if we always create one directory per calculation (we need it to be this way for Q-Chem anyhow.)

Definition at line 32 of file GenerateQMData.py.

8.3 MakeInputFile Namespace Reference

Executable script for printing out an example input file with defaults and documentation.

Functions

- def main

  Print out all of the options available to ForceBalance.

8.3.1 Detailed Description

Executable script for printing out an example input file with defaults and documentation. At the current stage, this script simply prints out all of the default options, but in the future we may want to autogenerate the input file. This would make everyone's lives much easier, don't you think? :)

8.3.2 Function Documentation

8.3.2.1 def MakeInputFile.main ( )

Print out all of the options available to ForceBalance.

Definition at line 18 of file MakeInputFile.py.

8.4 ParseInputFile Namespace Reference

Read in ForceBalance input file and print it back out.

Functions

- def main

  Input file parser for ForceBalance program.

8.4.1 Detailed Description

Read in ForceBalance input file and print it back out.

8.4.2 Function Documentation

8.4.2.1 def ParseInputFile.main ( )

Input file parser for ForceBalance program.

We will simply read the options and print them back out.

Definition at line 14 of file ParseInputFile.py.

9 Class Documentation

9.1 SelfConsistentCycle.ForceBalance_SCF Class Reference

Public Member Functions

- def **__init__**
- def **DetermineState**
- def **Cycle**

Public Attributes

- **Mao**
- **root**
- **forcefield**

9.1.1 Detailed Description

Definition at line 82 of file SelfConsistentCycle.py.

The documentation for this class was generated from the following file:

- forcebalance/bin/SelfConsistentCycle.py

9.2 TagMol2.MolG Class Reference

Public Member Functions

- def **__eq__**
- def __hash__

  The hash function is something we can use to discard two things that are obviously not equal.
- def L

  Return a list of the sorted atom numbers in this graph.
- def AStr

  Return a string of atoms, which serves as a rudimentary 'fingerprint' : '99,100,103,151' .
- def e

  Return an array of the elements.
- def ef

  Create an Empirical Formula.
- def x

  Get a list of the coordinates.

9.2.1 Detailed Description

Definition at line 51 of file TagMol2.py.

9.2.2 Member Function Documentation

9.2.2.1 def TagMol2.MolG.__hash__ (*self*)

The hash function is something we can use to discard two things that are obviously not equal.

Here we neglect the hash.

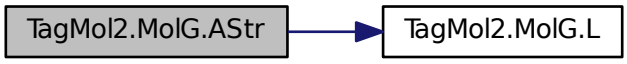
Definition at line 57 of file TagMol2.py.

9.2.2.2 def TagMol2.MolG.AStr (*self*)

Return a string of atoms, which serves as a rudimentary 'fingerprint' : '99,100,103,151' .

Definition at line 65 of file TagMol2.py.

Here is the call graph for this function:



9.2.2.3 def TagMol2.MolG.e (*self*)

Return an array of the elements.

For instance ['H' 'C' 'C' 'H'].

Definition at line 69 of file TagMol2.py.

Here is the call graph for this function:

9.2.2.4 def TagMol2.MolG.L (*self*)

Return a list of the sorted atom numbers in this graph.

Definition at line 61 of file TagMol2.py.

9.2.2.5 def TagMol2.MolG.x (*self*)

Get a list of the coordinates.

Definition at line 79 of file TagMol2.py.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- forcebalance/bin/TagMol2.py

9.3 CallGraph.node Class Reference

Data structure for holding information about python objects.

Public Member Functions

- def __init__
  Constructor.

Public Attributes

- **oid**
- **name**
- **parent**
- **dtype**

9.3.1 Detailed Description

Data structure for holding information about python objects.

Definition at line 20 of file CallGraph.py.

The documentation for this class was generated from the following file:

- forcebalance/doc/callgraph/CallGraph.py

9.4 SelfConsistentCycle.SCF_Simulation Class Reference

Public Member Functions

- def **__init__**
- def **SetupGeneration**
- def **RunDynamics**
- def **BuildMSM**

Public Attributes

- **name**
- **root**

9.4.1 Detailed Description

Definition at line 24 of file SelfConsistentCycle.py.

The documentation for this class was generated from the following file:

- forcebalance/bin/SelfConsistentCycle.py