ForceBalance version 0.11.0

Generated by Doxygen 1.7.6.1

Contents

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 potential optimization*. It addresses the problem of parameterizing empirical

potential functions, colloquially called *force fields*. 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, the majority of 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.

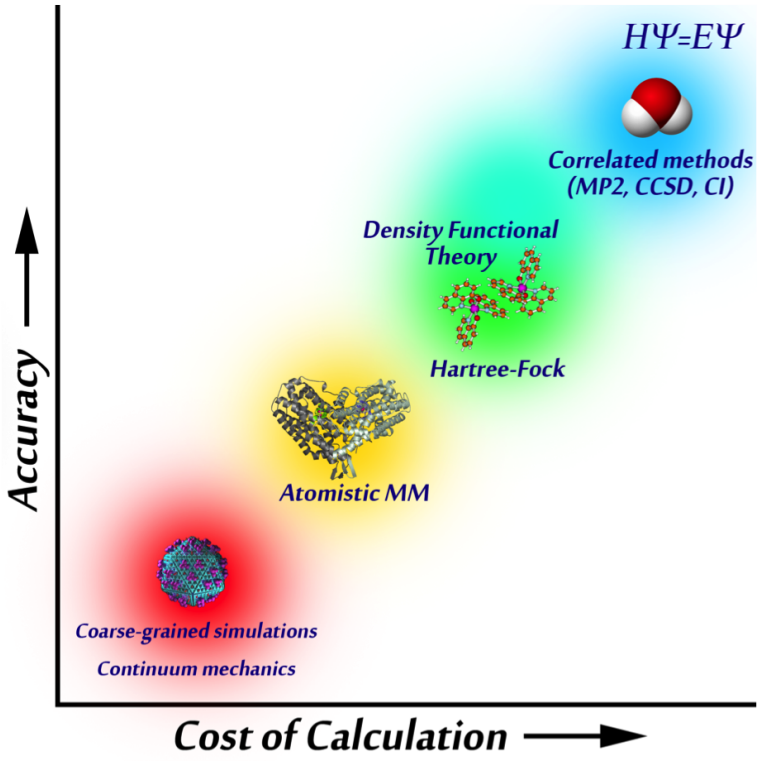


Figure 1: An arrangement of simulation methods by accuracy vs. computational cost.

A major 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 interactions between pairs of point charges on atomic centers, 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.

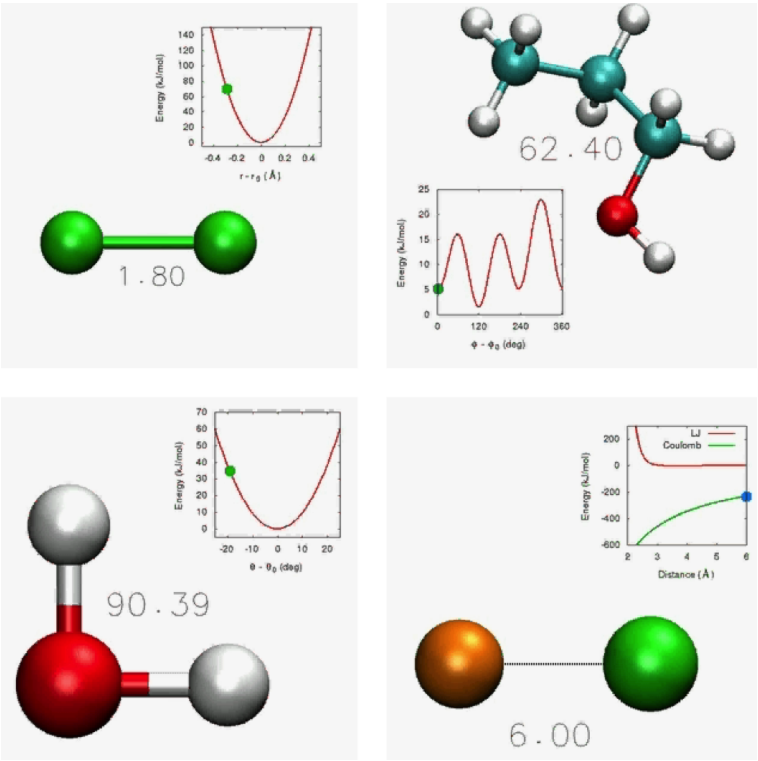


Figure 2: An illustration of some interactions typically found in classical force fields.

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 to reproduce experimental measurements which depend indirectly on the parameters (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 and predictive power.

1.2.2 Purpose and brief description of this program

Given this background, I can make the following statement. **ForceBalance aims to advance the methods of empirical potential development by applying a highly general and systematic process with explicitly specified input data and mathematical optimization algorithms, paving the way to higher accuracy potentials, improved**

reproducibility of potential development, and well-defined scopes of validity and error estimation for the parameters.

At a high level, ForceBalance takes an empirical potential and a set of reference data as inputs, and tunes the parameters such that the reference data is reproduced 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 empirical potential, 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.

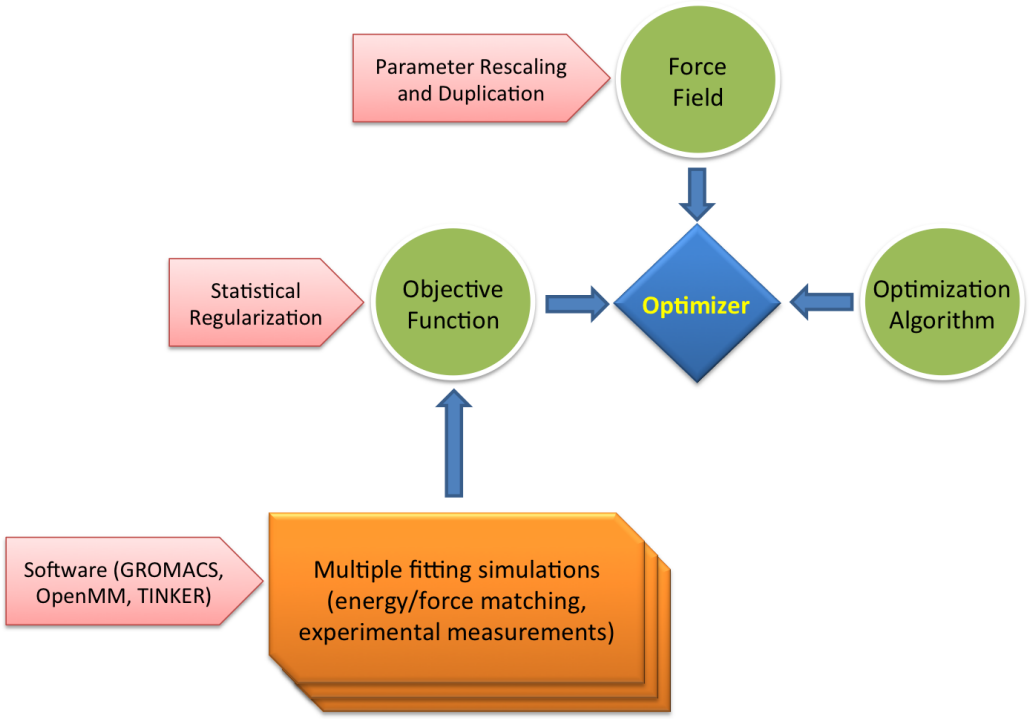


Figure 3: The division of the potential optimization problem into three parts; the force field, fitting simulations and optimization algorithm.

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 and its companion software GROMACS-X2.

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

Importantly, note that *ForceBalance is not a simulation software package*. During force field optimizations, ForceBalance interfaces with simulation software like GROMACS, TINKER or OpenMM; reference data is obtained from experimental measurements (consult the literature), or from simulation / quantum chemistry software (for example, NWChem or Q-Chem).

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. Soon, I will also implement functions for grid-scale computation of reference energies and forces using Q-Chem (a commercial software). However, you should be prepared to write some simple code to interface with a fitting simulation or quantum chemistry software 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 an ordinary Python module, so if you know how to install Python modules, you shouldn't have any trouble with this.

2.1.1 Prerequisites

ForceBalance requires the following software packages:

- Python version 2.7
- NumPy version 1.5
- SciPy version 0.9
- The lxml package (depends on libxml2 and libxslt)

A few more packages are required if you want to Create documentation. Here is a list of Python packages and software:

- Doxygen version 1.7.6.1
- Doxypy plugin for Doxygen
- LaTeX software like 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
```

Upon extracting the distribution you will notice this directory structure:

```
<root>
  +- bin
  |   |- <Executable scripts>
  +- src
  |   |- <Python module files>
  +- studies
  |   +- <ForceBalance example jobs>
  +- doc
  |   +- callgraph
  |   |   |- <Stuff for making a call graph>
  |   +- Images
  |   |   |- <Images for the website and PDF manual>
  |   |- mainpage.py (Contains most user documentation and this text)
  |   |- 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
```

Alternatively, you can do a local install by running:

```
python setup.py install --prefix=/home/your_username/local_directory
```

where you would of course replace your_username and local_directory with your username and preferred install location. The executable scripts will be placed into `/home/your_username/local_directory/bin` and the module will be placed into `/home/your_username/local_directory/lib/python[version]/site-packages/forceb`

Note that if you do a local installation, for Python to recognize the newly installed module you may need to append your PYTHONPATH environment variable using a command like the one below:

```
export PYTHONPATH=$PYTHONPATH:/home/your_username/local_directory/lib/python[version]
```

As with the installation of any Python software, there are potential issues with dependencies (for example, scipy and lxml.) Please let me know if you encounter issues with dependencies.

2.2 Installing GROMACS-X2

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.

2.2.1 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!

2.3 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 on 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 might need a LaTeX distribution for this.)

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 in the `bin` directory of the distribution.

`ForceBalance.py` is the executable script that performs 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 only provides the hardcoded default options.

3.1 Input file

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

```
$options
jobtype                 bfgs
gmxpath                 /home/leeping/opt/gromacs-4.0.7-x2/bin
forcefield              water.itp
penalty_multiplicative  0.01
convergence_objective   1e-6
convergence_step        1e-6
convergence_gradient    1e-4
$end

$simulation
simtype                 forceenergymatch_gmx
name                    water12_gen1
weight                  1
efweight                0.5
shots                   300
fd_ptypes               VSITE
fdhessdiag              1
covariance              0
$end

$simulation
simtype                 forceenergymatch_gmx
name                    water12_gen2
weight                  1
efweight                0.5
shots                   300
fd_ptypes               VSITE
fdhessdiag              1
covariance              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. At this time, 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
  |   |   |- all.gro (containing 300 geometries)
  |   |   |- qdata.txt
  |   |   |- shot.mdp
  |   |   |- topol.top
  |   +- water12_gen2
  |   |   |- all.gro (containing 300 geometries)
  |   |   |- qdata.txt
  |   |   |- shot.mdp
```

```
|   |   |- topol.top
|- input_file.in
```

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 they're 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`).

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.

Figure 4: Logo.

6 Namespace Index

6.1 Namespace List

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

7 Class Index

7.1 Class List

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

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.

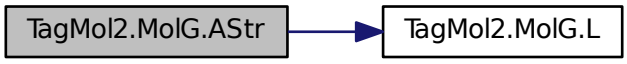
Definition at line 57 of file TagMol2.py.

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