ForceBalance version 1.1

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 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.

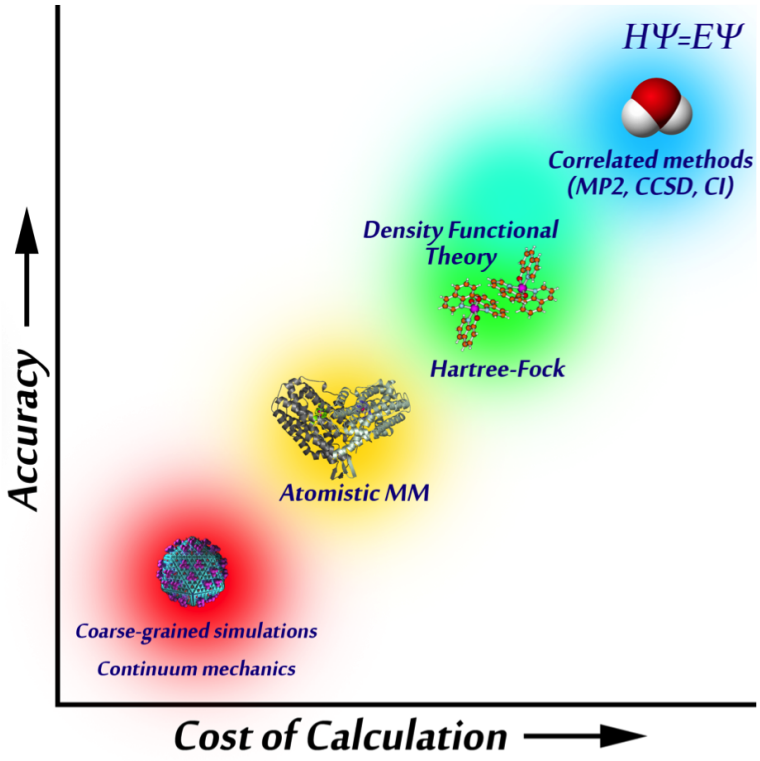

Figure 1: An arrangement of simulation methods by accuracy vs. computational cost.

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.

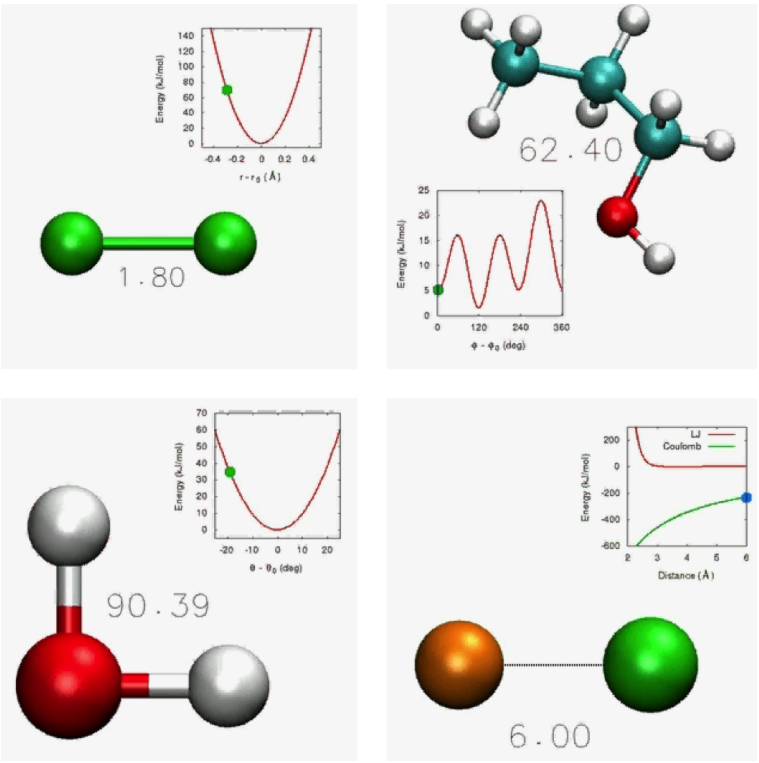

Figure 2: An illustration of some interactions typically found in classical force fields.

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 **target**. 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 a target, 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 targets 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 which minimizes the errors for all of the targets.

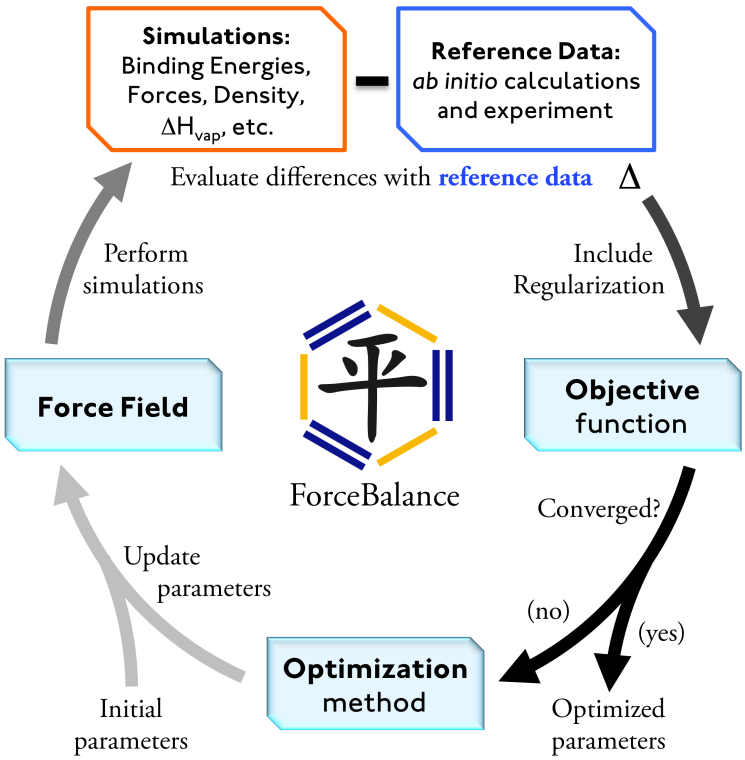

Figure 3: The division of the potential optimization problem into three parts; the force field, targets and optimization algorithm.

The problem is now split into three main components; the force field, the targets, 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.
- Arthur Vigil contributed the unit testing framework and many unit tests, significant improvements to the automatic documentation generation, and various code improvements.
- 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=$HOME/local/lib/libatlas.so`) before installing Scipy.
- `lxml` is a Python interface to the libxml2 XML parser. After much ado, I decided to use `lxml` instead of the `xml` module in Python's standard library for several reasons (`xml` contains only limited support for X-Path, scrambles the ordering of attributes in an element, etc.) The downside is that it can be harder to install. Installation instructions can be found on the `lxml website` but summarized here. The packages `libxml2` and `libxslt` need to be installed first, and in that order. On Ubuntu, run `sudo apt-get install libxml2-dev libxslt1-dev`. To compile from source, run `./configure --prefix=$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 containing all options, their default values, and a short description for each option.

3.1 Input file

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

```
$options
jobtype newton
forcefield water.itp
$end

$target
name cluster-02
type abinitio_gmx
$end

$target
name cluster-03
type abinitio_gmx
$end
```

Global options for a ForceBalance job are given in the `$options` section while the settings for each Target are given in the `$target` 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 target options to note are: `name` specifies the target name and `type` specifies the type of target (must correspond to a subdirectory in `targets/`). 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
  +- targets
  |   +- cluster-02
  |   |   +- settings (contains job settings)
  |   |   |   |- shot.mdp
  |   |   |   |- topol.top
  |   |   |- all.gro (contains geometries)
  |   |   |- qdata.txt (contains QM data)
  |   +- cluster-03
  |   |   +- settings
  |   |   |   |- shot.mdp
  |   |   |   |- topol.top
  |   |   |- all.gro
  |   |   |- qdata.txt
  |   +- <more target directories>
  |- 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 **targets** are fixed and cannot be changed. **forcefield** contains the force field files that you're optimizing, and **targets** contains all of the reference data as well as the input files for simulating that data using the force field. Each subdirectory in **targets** corresponds to a single target, and its contents depend on the specific kind of target and its corresponding `Target` class.

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 targets `cluster-02` and `cluster-03` match the `target` sections in the input file. There are two energy and force matching targets here; each directory contains the relevant geometries (in `all.gro`) and reference data (in `qdata.txt`).

3.3 Setting up the targets

There are many targets one can choose from.

- Energy and force matching - this is the oldest functionality and the most robust. 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 feature of ForceBalance is that targets 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 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 invite you to learn by example through looking at the files in the `studies` directory.

3.3.1 Energy 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 targets 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 gra

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 methods for parsing Q-Chem output files in the `molecule.py` class.

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 target.

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 target 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 to QM cluster calculations

After everything is installed, go to the `studies/001_water_tutorial` directory in the distribution. Extract the `targets.tar.bz2` archive file. Now execute:

```
ForceBalance.py very_simple.in
```

If the installation was successful, you will get an output file similar to `very_simple.out` . What follows is a description of the output file and what ForceBalance is actually doing.

ForceBalance begins by reading the force field files from the `forcefield` directory. The parameters to be optimized are specified in the parameter file by adding a special comment inside the file. For example, in the `water.itp` file, we specify that the two van der Waals parameters on oxygen are to be optimized, using the following syntax:

```
OW      8     15.99940     0.000       A    3.15365e-01  6.48520e-01 ; PARM 5 6
```

The comment `PARM 5 6` signals that "the parameters in fields 5 and 6 are to be optimized." The force field parser stores the physical value of the parameter and gives the parameter a name. These are printed out in the output file:

```
Reading force field from file: water.itp
#=========================================================#
#|  Starting parameter indices, physical values and IDs  |#
#=========================================================#
   0 [  3.1537e-01 ] : VDWS:OW
   1 [  6.4852e-01 ] : VDWT:OW
   2 [  9.5720e-02 ] : BONDSB:HWOW
   3 [  5.0242e+05 ] : BONDSK:HWOW
   4 [  1.0452e+02 ] : ANGLESB:HWOWHW
   5 [  6.2802e+02 ] : ANGLESK:HWOWHW
   6 [  5.2000e-01 ] : COUL:SOL-2 COUL:SOL-3
   7 [ -1.0400e+00 ] : COUL:SOL-4
   8 [  1.2800e-01 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
-----------------------------------------------------------
```

The next section it prints out are a set of rescaling factors which are important for various aspects of the optimization. They are discussed further in this `forum post`. For now it suffices to say that these values represent the natural size of the parameter, or more specifically how much the parameter is expected to vary.

```
#=========================================================#
#|     Rescaling Factors (Lower Takes Precedence):       |#
#=========================================================#
   BONDSB                               : 5.29177e-02
   BONDSK                               : 9.37583e+05
   VSITE3A                              : 5.29177e-02
   VSITE3B                              : 5.29177e-02
   ANGLESB                              : 5.72958e+01
   VDWS                                 : 5.29177e-02
   ANGLESK                              : 6.05928e+02
   COUL                                 : 1.00000e+00
   VDWT                                 : 2.47894e+00
-----------------------------------------------------------
```

Next, it prints out user-specified options that pertain to the force field the targets, the objective function and the optimizer. Options that are left at their default values (in this case, most) are not printed out. Use `verbose_options True` in the input file to print out all of the options.

Now for the good stuff - the optimizer begins. ForceBalance computes each target and prints out an indicator for each one, then provides a breakdown of the overall objective function:

```
#=========================================================#
#|                   Main Optimizer                      |#
#|        Newton-Raphson Mode (Adaptive Radius)          |#
#=========================================================#

#=====================================================================#
#|  Target: cluster-06 Type: AbInitio_GMX Objective = 1.12035e-01      |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                9.4124       27.3135     34.4605%
    Gradient (kJ/mol/A)           39.1963      119.0841     32.9148%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-12 Type: AbInitio_GMX Objective = 1.04039e-01      |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               15.2291       47.3455     32.1658%
    Gradient (kJ/mol/A)           38.5401      118.0240     32.6545%
---------------------------------------------------------------------
#=================================================================#
#|                  Objective Function Breakdown                 |#
#|   Target Name              Residual  x  Weight  =  Contribution  |#
#=================================================================#
cluster-06                    0.11203      0.500      5.60173e-02
cluster-12                    0.10404      0.500      5.20195e-02
Regularization                0.00000      1.000      0.00000e+00
Total                                               1.08037e-01
-----------------------------------------------------------------
  Step       |k|        |dk|       |grad|       -=X2=-     Delta(X2)    StepQual
     0   0.000e+00   0.000e+00   3.206e+00   1.08037e-01   0.000e+00      0.000
```

In this example job, the targets were QM energies and forces for clusters of 6 and 12 water molecules. In the initial step (using the default TIP4P parameters) and for the first target (6-mers), the RMS error for energies is 9.4124 kJ/mol (34% of the variance in the QM energies themselves), and the RMS error for atomistic forces is 32% (again, scaled to the variance of the QM forces). Similar information is printed out for the 12-mers. Each target contributes to the overall objective function, whose value is 1.080e-01. The parameters are at their initial values, which means that any penalty function will have a value of zero (the Regularization term).

Next, ForceBalance takes a step in the parameter space. The default algorithm (a variation of Newton-Raphson) uses first and second derivative information; the gradient is printed to the screen, as is the parameter step.

```
#=========================================================#
#|                    Total Gradient                     |#
#=========================================================#
   0 [ -1.36605285e-01 ] : VDWS:OW
   1 [ -2.24335748e-01 ] : VDWT:OW
   2 [ -3.14688760e+00 ] : BONDSB:HWOW
   3 [  3.54975985e-01 ] : BONDSK:HWOW
   4 [ -3.24607484e-01 ] : ANGLESB:HWOWHW
   5 [  8.92900123e-02 ] : ANGLESK:HWOWHW
   6 [ -7.50893285e-02 ] : COUL:SOL-2 COUL:SOL-3
   7 [ -2.44318391e-01 ] : COUL:SOL-4
   8 [ -2.23561237e-02 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
-----------------------------------------------------------

Levenberg-Marquardt: Newton-Raphson step found (length 1.000e-01),  0.92958359 added to Hessian diagonal
#=========================================================#
#|   Mathematical Parameters (Current + Step = Next)     |#
#=========================================================#
   0 [  0.0000e+00 + 2.3057e-02 =  2.3057e-02 ] : VDWS:OW
   1 [  0.0000e+00 + 3.7320e-02 =  3.7320e-02 ] : VDWT:OW
   2 [  0.0000e+00 + 5.7207e-03 =  5.7207e-03 ] : BONDSB:HWOW
   3 [  0.0000e+00 - 3.8228e-02 = -3.8228e-02 ] : BONDSK:HWOW
   4 [  0.0000e+00 + 8.6160e-03 =  8.6160e-03 ] : ANGLESB:HWOWHW
   5 [  0.0000e+00 - 7.4597e-02 = -7.4597e-02 ] : ANGLESK:HWOWHW
   6 [  0.0000e+00 - 7.4155e-03 = -7.4155e-03 ] : COUL:SOL-2 COUL:SOL-3
   7 [  0.0000e+00 + 2.9107e-02 =  2.9107e-02 ] : COUL:SOL-4
   8 [  0.0000e+00 + 6.3479e-03 =  6.3479e-03 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
-----------------------------------------------------------
#=========================================================#
#|     Physical Parameters (Current + Step = Next)       |#
#=========================================================#
   0 [  3.1537e-01 + 1.2201e-03 =  3.1659e-01 ] : VDWS:OW
   1 [  6.4852e-01 + 9.2513e-02 =  7.4103e-01 ] : VDWT:OW
   2 [  9.5720e-02 + 3.0273e-04 =  9.6023e-02 ] : BONDSB:HWOW
   3 [  5.0242e+05 - 3.5842e+04 =  4.6657e+05 ] : BONDSK:HWOW
   4 [  1.0452e+02 + 4.9366e-01 =  1.0501e+02 ] : ANGLESB:HWOWHW
   5 [  6.2802e+02 - 4.5200e+01 =  5.8282e+02 ] : ANGLESK:HWOWHW
   6 [  5.2000e-01 - 7.4155e-03 =  5.1258e-01 ] : COUL:SOL-2 COUL:SOL-3
   7 [ -1.0400e+00 + 2.9107e-02 = -1.0109e+00 ] : COUL:SOL-4
   8 [  1.2800e-01 + 3.3591e-04 =  1.2834e-01 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
-----------------------------------------------------------
```

Note that the step length is limited to a "trust radius" of 0.1 - this option is tunable. The step in parameter space is given in terms of the "mathematical parameters" - the internal optimization variables - and the "physical parameters" which are the actual values in the force field files. The mathematical parameters are mainly useful because they can be used to restart an optimization by creating a `read_mvals` section to the input file and pasting the lines from the output.

ForceBalance now computes the objective function again, using the new parameter values.

```
#=====================================================================#
#|  Target: cluster-06 Type: AbInitio_GMX Objective = 7.55909e-02     |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                8.0920       27.3135     29.6263%
    Gradient (kJ/mol/A)           30.1378      119.0841     25.3080%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-12 Type: AbInitio_GMX Objective = 7.17029e-02     |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               13.2306       47.3455     27.9447%
    Gradient (kJ/mol/A)           30.1330      118.0240     25.5312%
---------------------------------------------------------------------
#=============================================================================#
```

```
#|                          Objective Function Breakdown                          |#
#|   Target Name              Residual  x  Weight  =  Contribution (Current-Prev)  |#
#================================================================================#
cluster-06                     0.07559      0.500      3.77955e-02 ( -1.822e-02 )
cluster-12                     0.07170      0.500      3.58514e-02 ( -1.617e-02 )
Regularization                 0.00010      1.000      9.99999e-05 ( +1.000e-04 )
Total                                                  7.37469e-02 ( -3.429e-02 )
--------------------------------------------------------------------------------
  Step       |k|        |dk|       |grad|       -=X2=-     Delta(X2)    StepQual
     1   1.000e-01   1.000e-01   1.941e-01   7.37469e-02   3.429e-02      1.001
```

Using the new parameter values, the values for each target have gone down - that is to say, the force field now produces better agreement with the reference data. In the objective function breakdown, improvements (i.e. decreasing values) from the previous step are printed in green while increasing values are printed in red. The "Regularization" term is printed in red because the parameters have moved from their initial values, so the penalty function is now finite.

The last line reports:

- The step number (`Step`)
- The length of the parameter vector in the mathematical parameter space (`|k|`)
- The length of the most recent step (`|dk|`)
- The magnitude of the objective function gradient vector (`|grad|`)
- The objective function value (`-=X2=-`)
- The standard deviation of the objective function over a user-specified number of optimization steps
- The ratio of actual-to-predicted change in the objective function value. A `StepQual` value of 1.0 signifies that the trust radius can be increased.

Eventually, the optimization will converge. For this job (and when this documentation was written) it took five steps:

```
#=====================================================================#
#|  Target: cluster-06 Type: AbInitio_GMX Objective = 6.30676e-02      |#
#|                           Difference   Denominator     Percent     |#
#|  Physical Variable         (Calc-Ref)     RMS (Ref)   Difference   |#
#=====================================================================#
    Energy (kJ/mol)              7.9083       27.3135     28.9539%
    Gradient (kJ/mol/A)         24.1991      119.0841     20.3210%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-12 Type: AbInitio_GMX Objective = 5.89806e-02      |#
#|                           Difference   Denominator     Percent     |#
#|  Physical Variable         (Calc-Ref)     RMS (Ref)   Difference   |#
#=====================================================================#
    Energy (kJ/mol)             12.9053       47.3455     27.2578%
    Gradient (kJ/mol/A)         24.3021      118.0240     20.5908%
---------------------------------------------------------------------
#================================================================================#
#|                          Objective Function Breakdown                          |#
#|   Target Name              Residual  x  Weight  =  Contribution (Current-Prev)  |#
#================================================================================#
cluster-06                     0.06307      0.500      3.15338e-02 ( -4.544e-07 )
cluster-12                     0.05898      0.500      2.94903e-02 ( -7.454e-06 )
Regularization                 0.00170      1.000      1.70115e-03 ( +7.704e-06 )
Total                                                  6.27252e-02 ( -2.045e-07 )
--------------------------------------------------------------------------------
  Step       |k|        |dk|       |grad|       -=X2=-     Delta(X2)    StepQual
     5   4.124e-01   2.498e-03   2.130e-05   6.27252e-02   2.045e-07      1.029

Convergence criterion reached for gradient norm (1.00e-04)
```

```
@=========================================================@
@|            Final objective function value             |@
@|   Full:  6.272524e-02  Un-penalized:  6.102410e-02    |@
@=========================================================@
#=========================================================#
#|            Final optimization parameters:             |#
#|            Paste to input file to restart             |#
#=========================================================#
read_mvals
   0 [  3.3161e-02 ] : VDWS:OW
   1 [  4.3311e-02 ] : VDWT:OW
   2 [  5.5070e-03 ] : BONDSB:HWOW
   3 [ -4.5933e-02 ] : BONDSK:HWOW
   4 [  1.5497e-02 ] : ANGLESB:HWOWHW
   5 [ -3.7655e-01 ] : ANGLESK:HWOWHW
   6 [  2.4929e-03 ] : COUL:SOL-2 COUL:SOL-3
   7 [  1.1874e-02 ] : COUL:SOL-4
   8 [  1.5108e-01 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
/read_mvals
#=========================================================#
#|              Final physical parameters:               |#
#=========================================================#
   0 [  3.1712e-01 ] : VDWS:OW
   1 [  7.5589e-01 ] : VDWT:OW
   2 [  9.6011e-02 ] : BONDSB:HWOW
   3 [  4.5935e+05 ] : BONDSK:HWOW
   4 [  1.0541e+02 ] : ANGLESB:HWOWHW
   5 [  3.9986e+02 ] : ANGLESK:HWOWHW
   6 [  5.2249e-01 ] : COUL:SOL-2 COUL:SOL-3
   7 [ -1.0281e+00 ] : COUL:SOL-4
   8 [  1.3599e-01 ] : VSITE3B:SOL-4 VSITE3A:SOL-4
-----------------------------------------------------------

The final force field has been printed to the 'result' directory.
#=========================================================#
#|      Congratulations, ForceBalance has finished       |#
#|           Give yourself a pat on the back!            |#
#=========================================================#
```

As you can see, the objective function has decreased considerably since the previous step, and most of the improvement was due to reducing the error in the atomistic forces. In the `result` directory, you will find an updated `water.itp` file with the optimized parameter values.

This newly generated force field is a better fit to the reference data, but is it actually a better force field or did we just overfit the data? To answer this question, look at `validate.in` where the job type is set to `single`, and there are many more targets. In particular, we are now including QM energies and forces for many cluster sizes ranging from 2 to 12.

Take the `read_mvals` section from the output file of your previous run, paste it intointo the `$options` section of `validate.in`, and run `ForceBalance.py validate.in` . ForceBalance will now evaluate the objective function using the force field parameters from the previous optimization.

You should see the following output:

```
#=======================================================================#
#|  Target: cluster-02 Type: AbInitio_GMX Objective = 6.59279e-02      |#
#|                              Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
#=======================================================================#
    Energy (kJ/mol)                2.6852        8.9926     29.8605%
    Gradient (kJ/mol/A)           24.4491      120.0100     20.3725%
-----------------------------------------------------------------------
#=======================================================================#
#|  Target: cluster-03 Type: AbInitio_GMX Objective = 6.86838e-02      |#
#|                              Difference   Denominator     Percent   |#
#|  Physical Variable           (Calc-Ref)     RMS (Ref)   Difference  |#
```

```
#=====================================================================#
    Energy (kJ/mol)                4.1222       13.3676     30.8370%
    Gradient (kJ/mol/A)           24.1603      119.6514     20.1922%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-04 Type: AbInitio_GMX Objective = 6.99336e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                5.3337       17.0641     31.2567%
    Gradient (kJ/mol/A)           24.4117      121.2622     20.1313%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-05 Type: AbInitio_GMX Objective = 6.83413e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                6.4445       20.9275     30.7946%
    Gradient (kJ/mol/A)           24.2035      120.2407     20.1292%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-06 Type: AbInitio_GMX Objective = 6.30676e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                7.9083       27.3135     28.9539%
    Gradient (kJ/mol/A)           24.1990      119.0841     20.3209%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-07 Type: AbInitio_GMX Objective = 7.23291e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                9.0541       28.5018     31.7666%
    Gradient (kJ/mol/A)           24.5603      119.6730     20.5228%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-08 Type: AbInitio_GMX Objective = 6.47534e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)                9.9105       33.9780     29.1676%
    Gradient (kJ/mol/A)           24.5318      118.7419     20.6598%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-09 Type: AbInitio_GMX Objective = 6.31661e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               10.6633       37.0249     28.8003%
    Gradient (kJ/mol/A)           24.5632      119.9943     20.4703%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-10 Type: AbInitio_GMX Objective = 6.16248e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               11.3800       40.4430     28.1383%
    Gradient (kJ/mol/A)           24.6246      119.4050     20.6227%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-11 Type: AbInitio_GMX Objective = 5.88603e-02    |#
#|                            Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               12.7370       47.0656     27.0623%
    Gradient (kJ/mol/A)           24.6241      118.6740     20.7493%
---------------------------------------------------------------------
```

```
#=====================================================================#
#|  Target: cluster-12 Type: AbInitio_GMX Objective = 5.89805e-02      |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               12.9053       47.3455     27.2578%
    Gradient (kJ/mol/A)           24.3021      118.0240     20.5908%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-13 Type: AbInitio_GMX Objective = 5.94277e-02      |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               13.7931       50.5451     27.2887%
    Gradient (kJ/mol/A)           24.7275      119.3178     20.7241%
---------------------------------------------------------------------
#=====================================================================#
#|  Target: cluster-14 Type: AbInitio_GMX Objective = 5.53305e-02      |#
#|                             Difference   Denominator     Percent   |#
#|  Physical Variable          (Calc-Ref)     RMS (Ref)   Difference  |#
#=====================================================================#
    Energy (kJ/mol)               14.1450       54.7952     25.8143%
    Gradient (kJ/mol/A)           24.6526      119.0962     20.6997%
---------------------------------------------------------------------
#=====================================================================#
#|                  Objective Function Breakdown                     |#
#|   Target Name              Residual  x  Weight  =  Contribution   |#
#=====================================================================#
cluster-02                    0.06593      0.077      5.07138e-03
cluster-03                    0.06868      0.077      5.28337e-03
cluster-04                    0.06993      0.077      5.37951e-03
cluster-05                    0.06834      0.077      5.25702e-03
cluster-06                    0.06307      0.077      4.85135e-03
cluster-07                    0.07233      0.077      5.56378e-03
cluster-08                    0.06475      0.077      4.98103e-03
cluster-09                    0.06317      0.077      4.85893e-03
cluster-10                    0.06162      0.077      4.74037e-03
cluster-11                    0.05886      0.077      4.52772e-03
cluster-12                    0.05898      0.077      4.53697e-03
cluster-13                    0.05943      0.077      4.57136e-03
cluster-14                    0.05533      0.077      4.25619e-03
Regularization                0.00170      1.000      1.70118e-03
Total                                                 6.55802e-02
---------------------------------------------------------------------
```

As you can see, the agreement is comparable for all of the cluster sizes, and this effectively means that we were able to achieve an accurate fit to QM energies and forces for a wide range of cluster sizes using only the 6-mers and 12-mers.

A truly good force field needs to accurately reproduce experimental measurements, but these are more difficult to compute and optimize. ForceBalance provides methods for optimizing using experimental targets, but it is beyond the scope of this tutorial. However, hopefully this simple example helps to explain how force field optimization works within the framework of ForceBalance.

Feel free to explore using the other provided input files:

- `0.energy_force.in` uses all of the targets - cluster sizes 2 through 14 - in the optimization.
- `1.netforce_torque.in` includes net forces on water molecules and torques in the optimization.
- `2.L1_penalty.in` uses a L1 penalty function, effectively causing only some parameters to change and not others.
- `3.no_penalty.in` illustrates what happens when no penalty function is used at all.
- `4.change_factor.in` shows the effect of changing the rescaling factors.

- `5.gradient.in` performs a finite-difference check on the objective function gradient.

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!

- **Target** : A reference data set from high-level theoretical calculations or experimental measurements, paired with a procedure to simulate the same quantity using the force field. The objective function is the sum of one or more targets.

- **Force field** : This term is used interchangeably with empirical potential.

- **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 'Target options' which apply to one target within a job (described in the next section). The option index is generated by running make-option-index.py.

- **ADAPTIVE_DAMPING** (Float)

  One-line description : Damping factor that ties down the trust radius to trust0; decrease for a more variable step size.

  Default Value : 0.5

  Scope : Main optimizer (Optional)

  Full description : See documentation for adaptive_factor.

  Recommendation : A larger value will ensure that the trust radius never exceeds the original value by more than a small percentage. 0.5 is a reasonable value to start from.

- **ADAPTIVE_FACTOR** (Float)

  One-line description : The step size is increased / decreased by up to this much in the event of a good / bad step; increase for a more variable step size.

  Default Value : 0.25

  Scope : Main optimizer (Optional)

  Full description : Adaptive adjustment of the step size in trust-radius Newton Raphson. If the optimizer takes a good step, the step is increased as follows:

  ```
  trust += adaptive_factor*trust*np.exp(-adaptive_damping*(trust/self.trust0 - 1))
  ```

  Note that the adaptive_damping option makes sure that the trust radius increases by a smaller factor the further it deviates from the original trust radius (trust0). On the other hand, if the optimizer takes a bad step, the step is reduced as follows:

  ```
  trust = max(ndx*(1./(1+adaptive_factor)), self.mintrust)
  ```

Recommendation : 0.2 is a conservative value, 0.5 for big step size adjustments.

- **AMOEBA_POLARIZATION** (String)

  One-line description : The AMOEBA polarization type, either direct, mutual, or nonpolarizable.

  Default Value : direct

  Scope : Optimizations of the AMOEBA polarizable force field (Optional)

  Full description : When optimizing a force field with the AMOEBA functional form, set this variable to 'direct', 'mutual', or 'nonpolarizable'. At present this variable affects OpenMM API calls, but not TINKER input files.

  Recommendation : No recommendation, depends on the application.

- **ASYNCHRONOUS** (Bool)

  One-line description : Execute Work Queue tasks and local calculations asynchronously for improved speed

  Default Value : 0

  Scope : Targets that use Work Queue (Optional)

  Full description : When using Work Queue to distribute computationally intensive tasks (e.g. condensed phase simulations), it is computationally efficient to run the local jobs concurrently rather than wait for the tasks to finish. Setting this flag allows local evaluation of targets to proceed while the Work Queue runs in the background, which speeds up the calculation compared to waiting idly for the Work Queue tasks to complete.

  Recommendation : If using Work Queue to distribute tasks for some targets, set to True.

- **BACKUP** (Bool)

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

  Default Value : 1

  Scope : All force field optimizations (Optional)

- **CONSTRAIN_CHARGE** (Bool)

  One-line description : Specify whether to constrain the charges on the molecules.

  Default Value : 0

  Scope : Force fields with point charges (Optional)

  Full description : It is important for force fields with point charges to not change the overall charge on the molecule or ion. Setting this option will activate a linear transformation which projects out the direction in parameter space that changes the net charge.

  Recommendation : Either set to true and check your output carefully, or use "eval" statements in the force field file for finer control.

- **CONVERGENCE_GRADIENT** (Float)

  One-line description : Convergence criterion of gradient norm

  Default Value : 0.0001

  Scope : Main optimizer (Optional)

  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.

  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)

  One-line description : Convergence criterion of objective function (in MainOptimizer this is the stdev of X2 over [objective_history] steps)

  Default Value : 0.0001

  Scope : Main optimizer (Optional)

  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.

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

- **CONVERGENCE_STEP** (Float)

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

  Default Value : 0.0001

  Scope : Main optimizer (Optional)

  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.

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

- **EIG_LOWERBOUND** (Float)

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

  Default Value : 0.0001

  Scope : Main optimizer (Optional)

  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.

  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)

  One-line description : Error tolerance; the optimizer will only reject steps that increase the objective function by more than this number.

  Default Value : 0.0

  Scope : Main optimizer (Optional)

Full description : In some targets (e.g. condensed phase properties), the contribution to the objective function may contain statistical noise and cause the optimization step to be rejected. Introducing an error tolerance allows the optimization to continue despite some apparent roughness in the objective function surface.

Recommendation : Set to zero for targets that don't have statistical noise. Otherwise, choose a value based on the rough size of the objective function and the weight of the statistically noisy targets.

- **FFDIR** (String)

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

  Default Value : forcefield

  Scope : All force field optimizations (Optional)

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

- **FINITE_DIFFERENCE_H** (Float)

  One-line description : Step size for finite difference derivatives in many functions

  Default Value : 0.001

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

  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.

  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)

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

  Default Value : []

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

- **GMXPATH** (String)

  One-line description : Path for GROMACS executables (if not the default)

  Default Value : /usr/bin

  Scope : Targets that use GROMACS (***Required***)

  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'.

  Recommendation : Depends on your local installation and environment.

- **GMXSUFFIX** (String)

  One-line description : The suffix of GROMACS executables

  Default Value :

  Scope : Targets that use GROMACS (Optional)

  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.

  Recommendation : Depends on your local installation and environment.

- **HAVE_VSITE** (Bool)

  One-line description : Specify whether there are virtual sites in the simulation (being fitted or not). Enforces calculation of vsite positions.

  Default Value : 0

  (Needs full documentation)

- **JOBTYPE** (Allcap)

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

  Default Value : single

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

  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.

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

- **LM_GUESS** (Float)

  One-line description : Guess value for bracketing line search in trust radius algorithm

  Default Value : 1.0

  (Needs full documentation)

- **LOGARITHMIC_MAP** (Bool)

  One-line description : Optimize in the space of log-variables

  Default Value : 0

  (Needs full documentation)

- **MAXSTEP** (Int)

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

  Default Value : 100

  Scope : All iterative optimization jobs (Optional)

  Recommendation : At least 100 optimization steps are recommended.

- **MINTRUST** (Float)

  One-line description : Minimum trust radius (if the trust radius is tiny, then noisy optimizations become really gnarly)

  Default Value : 0.0

  (Needs full documentation)

- **NORMALIZE_WEIGHTS** (Bool)

  One-line description : Normalize the weights for the fitting targets

  Default Value : 1

  (Needs full documentation)

- **OBJECTIVE_HISTORY** (Int)

  One-line description : Number of good optimization steps to average over when checking the objective convergence criterion

  Default Value : 2

  (Needs full documentation)

- **PENALTY_ADDITIVE** (Float)

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

  Default Value : 0.0

  Scope : Objective function (Optional)

  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.

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

- **PENALTY_ALPHA** (Float)

  One-line description : Extra parameter for fusion penalty function. Dictates position of log barrier or L1-L0 switch distance

  Default Value : 0.001

  (Needs full documentation)

- **PENALTY_HYPERBOLIC_B** (Float)

  One-line description : Cusp region for hyperbolic constraint; for x=0, the Hessian is a/2b

  Default Value : 1e-06

  (Needs full documentation)

- **PENALTY_MULTIPLICATIVE** (Float)

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

  Default Value : 0.0

  Scope : Objective function (Optional)

  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.

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

- **PENALTY_TYPE** (String)

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

  Default Value : L2

  Scope : All force field optimizations (Optional)

  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.

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

- **PRINT_GRADIENT** (Bool)

  One-line description : Print the objective function gradient at every step

  Default Value : 1

  (Needs full documentation)

- **PRINT_HESSIAN** (Bool)

  One-line description : Print the objective function Hessian at every step

  Default Value : 0

  (Needs full documentation)

- **PRINT_PARAMETERS** (Bool)

  One-line description : Print the mathematical and physical parameters at every step

  Default Value : 1

  (Needs full documentation)

- **PRIORS** (Section)

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

  Default Value : OrderedDict()

  (Needs full documentation)

- **READ_MVALS** (Section)

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

  Default Value : None

  Scope : All force field optimizations (Optional)

  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
```

  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)

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

  Default Value : None

  Scope : All force field optimizations (Optional)

  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.

  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)

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

  Default Value : None

  Scope : Main optimizer (Optional)

  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.

- **RIGID_WATER** (Bool)

  One-line description : Perform calculations using rigid water molecules.

  Default Value : 0

  (Needs full documentation)

- **SCAN_VALS** (String)

  One-line description : Values to scan in the parameter space, given like this: -0.1:0.1:11

  Default Value : None

  Scope : scan_mvals and scan_pvals job types (Optional)

  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.

  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)

  One-line description : Parameter name to scan over (should convert to a numerical index)

  Default Value : []

  Scope : scan_mvals and scan_pvals job types (Optional)

  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.

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

- **SCANINDEX_NUM** (List)

  One-line description : Numerical index of the parameter to scan over

  Default Value : []

  Scope : scan_mvals and scan_pvals job types (Optional)

  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.

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

- **SEARCH_TOLERANCE** (Float)

  One-line description : Search tolerance; used only when trust radius is negative, dictates convergence threshold of nonlinear search.

  Default Value : 0.0001

  (Needs full documentation)

- **TINKERPATH** (String)

  One-line description : Path for TINKER executables (if not the default)

  Default Value :

  Scope : Targets that use TINKER (***Required***)

  Recommendation : Depends on your local installation and environment.

- **TRUST0** (Float)

  One-line description : Levenberg-Marquardt trust radius; set to negative for nonlinear search

  Default Value : 0.1

  Scope : Main optimizer (Optional)

  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.

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

- **USE_PVALS** (Bool)

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

  Default Value : 0

  (Needs full documentation)

- **VERBOSE_OPTIONS** (Bool)

  One-line description : Set to false to suppress printing options that are equal to their defaults

  Default Value : 0

  (Needs full documentation)

- **WQ_PORT** (Int)

  One-line description : The port number to use for Work Queue

  Default Value : 0

  (Needs full documentation)

- **WRITECHK** (String)

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

  Default Value : None

  Scope : Main optimizer (Optional)

  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.

  Recommendation : Writing the checkpoint file is highly recommended.

- **WRITECHK_STEP** (Bool)

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

  Default Value : 1

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

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

  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: Target options

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

- **ABSOLUTE** (Bool)

  One-line description : When matching energies in AbInitio, do not subtract the mean energy gap.

  Default Value : 0

  (Needs full documentation)

- **ALL_AT_ONCE** (Bool)

  One-line description : Compute all energies and forces in one fell swoop where possible(as opposed to calling the simulation code once per snapshot)

  Default Value : 1

  (Needs full documentation)

- **ANISOTROPIC_BOX** (Bool)

  One-line description : Enable anisotropic box scaling (e.g. for crystals or two-phase simulations) in external npt.py script

  Default Value : 0

  (Needs full documentation)

- **ATTENUATE** (Bool)

  One-line description : Normalize interaction energies using 1/(denom$**$2 + reference$**$2) only for repulsive interactions greater than denom.

  Default Value : 0

  (Needs full documentation)

- **BATCH_FD** (Bool)

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

  Default Value : 0

  Scope : All target types (Optional)

  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.

- **CAUCHY** (Bool)

  One-line description : Normalize interaction energies each using 1/(denom$**$2 + reference$**$2) which resembles a Cauchy distribution

  Default Value : 0

  (Needs full documentation)

- **COVARIANCE** (Bool)

  One-line description : Whether to use the quantum covariance matrix

  Default Value : 0

  Scope : Force and energy matching (Optional)

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.

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

- **DIPOLE_DENOM** (Float)

  One-line description : Dipole normalization (Debye) ; set to 0 if a zero weight is desired

  Default Value : 1.0

  (Needs full documentation)

- **DO_COSMO** (Bool)

  One-line description : Call Q-Chem to do MM COSMO on MM snapshots.

  Default Value : 0

  (Needs full documentation)

- **ENERGY** (Bool)

  One-line description : Enable the energy objective function

  Default Value : 1

  (Needs full documentation)

- **ENERGY_DENOM** (Float)

  One-line description : Energy normalization for binding energies in kcal/mol (default is to use stdev)

  Default Value : 1.0

  (Needs full documentation)

- **ENERGY_UPPER** (Float)

  One-line description : Upper energy cutoff (in kcal/mol); super-repulsive interactions are given zero weight

  Default Value : 30.0

  (Needs full documentation)

- **FD_PTYPES** (List)

  One-line description : The parameter types that are differentiated using finite difference

  Default Value : []

  Scope : All target types (Optional)

  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.

  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.

- **FDGRAD** (Bool)

  One-line description : Finite difference gradient of objective function w/r.t. specified parameters

  Default Value : 0

Scope : All target types (Optional)

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.

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)

  One-line description : Finite difference Hessian of objective function w/r.t. specified parameters

  Default Value : 0

  Scope : All target types (Optional)

  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.

  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)

  One-line description : Finite difference Hessian diagonals w/r.t. specified parameters (costs 2np times a objective calculation)

  Default Value : 0

  Scope : All target types (Optional)

  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.

  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)

  One-line description : Number of fitting atoms; defaults to all of them

  Default Value : 0

  Scope : Force and energy matching (Optional)

  Full description : Choose a subset of atoms 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. ;)

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

- **FORCE** (Bool)

  One-line description : Enable the force objective function

  Default Value : 1

  (Needs full documentation)

- **FORCE_CUDA** (Bool)

  One-line description : Force the external npt.py script to crash if CUDA Platform not available

  Default Value : 0

  (Needs full documentation)

- **FORCE_MAP** (String)

  One-line description : The resolution of mapping interactions to net forces and torques for groups of atoms. In order of resolution: molecule > residue > charge-group

  Default Value : residue

  (Needs full documentation)

- **FRAGMENT1** (String)

  One-line description : Interaction fragment 1: a selection of atoms specified using atoms and dashes, e.g. 1-6 to select the first through sixth atom (i.e. list numbering starts from 1)

  Default Value :

  (Needs full documentation)

- **FRAGMENT2** (String)

  One-line description : Interaction fragment 2: a selection of atoms specified using atoms and dashes, e.g. 7-11 to select atoms 7 through 11.

  Default Value :

  (Needs full documentation)

- **GAS_EQU_STEPS** (Int)

  One-line description : Number of time steps for the gas equilibration run, if different from default.

  Default Value : 0

  (Needs full documentation)

- **GAS_INTERVAL** (Float)

  One-line description : Time interval for saving coordinates for the gas production run (if zero, use default in external script.)

  Default Value : 0.0

  (Needs full documentation)

- **GAS_PROD_STEPS** (Int)

  One-line description : Number of time steps for the gas production run, if different from default.

  Default Value : 0

  (Needs full documentation)

- **GAS_TIMESTEP** (Float)

  One-line description : Time step size for the gas simulation (if zero, use default in external script.).

  Default Value : 0.0

  (Needs full documentation)

- **HVAP_SUBAVERAGE** (Bool)

  One-line description : Don't target the average enthalpy of vaporization and allow it to freely float (experimental)

  Default Value : 0

  (Needs full documentation)

- **LIQUID_EQU_STEPS** (Int)
  One-line description : Number of time steps for the liquid equilibration run.
  Default Value : 10000
  (Needs full documentation)

- **LIQUID_INTERVAL** (Float)
  One-line description : Time interval for saving coordinates for the liquid production run.
  Default Value : 0.05
  (Needs full documentation)

- **LIQUID_PROD_STEPS** (Int)
  One-line description : Number of time steps for the liquid production run.
  Default Value : 20000
  (Needs full documentation)

- **LIQUID_TIMESTEP** (Float)
  One-line description : Time step size for the liquid simulation.
  Default Value : 0.5
  (Needs full documentation)

- **MANUAL** (Bool)
  One-line description : Give the user a chance to fill in condensed phase stuff on the zeroth step
  Default Value : 0
  (Needs full documentation)

- **MASTERFILE** (String)
  One-line description : The name of the master file containing interacting systems
  Default Value : interactions.txt
  (Needs full documentation)

- **MDRUN_THREADS** (Int)
  One-line description : Set the number of threads used by remote Gromacs processes
  Default Value : 1
  (Needs full documentation)

- **MINIMIZE_ENERGY** (Bool)
  One-line description : Minimize the energy of the system prior to running dynamics
  Default Value : 1
  (Needs full documentation)

- **MTS_VVVR** (Bool)
  One-line description : Enable multiple-timestep integrator in external npt.py script
  Default Value : 0
  (Needs full documentation)

- **NAME** (String)

  One-line description : The name of the target, corresponding to the directory targets/name

  Default Value : None

  Scope : All targets (***Required***)

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

- **OPENMM_CUDA_PRECISION** (String)

  One-line description : Precision of local OpenMM calculation. Choose either single, double or mixed ; defaults to the OpenMM default.

  Default Value :

  (Needs full documentation)

- **OPTIMIZE_GEOMETRY** (Bool)

  One-line description : Perform a geometry optimization before computing properties

  Default Value : 1

  (Needs full documentation)

- **POLARIZABILITY_DENOM** (Float)

  One-line description : Dipole polarizability tensor normalization (cubic Angstrom) ; set to 0 if a zero weight is desired

  Default Value : 1.0

  (Needs full documentation)

- **QMBOLTZ** (Float)

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

  Default Value : 0.0

  Scope : Force and energy matching (Optional)

  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).

  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 there is a very large RMS energy difference between QM and MM.

- **QMBOLTZTEMP** (Float)

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

  Default Value : 298.15

  Scope : Force and energy matching (Optional)

  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.

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

- **QUADRUPOLE_DENOM** (Float)
  One-line description : Quadrupole normalization (Buckingham) ; set to 0 if a zero weight is desired
  Default Value : 1.0
  (Needs full documentation)

- **RESP** (Bool)
  One-line description : Enable the RESP objective function
  Default Value : 0
  (Needs full documentation)

- **RESP_A** (Float)
  One-line description : RESP "a" parameter for strength of penalty; 0.001 is strong, 0.0005 is weak
  Default Value : 0.001
  (Needs full documentation)

- **RESP_B** (Float)
  One-line description : RESP "b" parameter for hyperbolic behavior; 0.1 is recommended
  Default Value : 0.1
  (Needs full documentation)

- **RMSD_DENOM** (Float)
  One-line description : RMSD normalization for optimized geometries in Angstrom
  Default Value : 0.1
  (Needs full documentation)

- **RUN_INTERNAL** (Bool)
  One-line description : For OpenMM or other codes with Python interface: Compute energies and forces internally
  Default Value : 1
  (Needs full documentation)

- **SAMPCORR** (Bool)
  One-line description : Whether to use the archaic sampling correction
  Default Value : 0
  (Needs full documentation)

- **SELF_POL_ALPHA** (Float)
  One-line description : Polarizability parameter for self-polarization correction (in debye).
  Default Value : 0.0
  (Needs full documentation)

- **SELF_POL_MU0** (Float)
  One-line description : Gas-phase dipole parameter for self-polarization correction (in debye).
  Default Value : 0.0
  (Needs full documentation)

- **SHOTS** (Int)

  One-line description : Number of snapshots; defaults to all of the snapshots

  Default Value : -1

  Scope : Force and energy matching (Optional)

  Full description : This option allows you to choose a subset from the snapshots available in the force matching 'targets' 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

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

- **SLEEPY** (Int)

  One-line description : Wait a number of seconds every time this target is visited (gives me a chance to ctrl+C)

  Default Value : 0

  (Needs full documentation)

- **TYPE** (Allcap)

  One-line description : The type of fitting target, for instance AbInitio_GMX ; this must correspond to the name of a Target subclass.

  Default Value : None

  Scope : All targets (***Required***)

  Full description : This is the type of target that you are running. The current accepted values for the target type are given in the beginning of the objective.py file: COUNTERPOISE, ABINITIO_AMBER, ABINITIO_GMX, LIQUID_GMX, LIQUID_OPENMM, ABINITIO_OPENMM, ABINITIO_TINKER, BINDINGENERGY_TINKER, MONOMER_QTPIE, RDVR3_PSI4, INTERACTION_OPENMM, ABINITIO_INTERNAL, MOMENTS_TINKER, THCDF_PSI4, VIBRATION_TINKER, LIQUID_TINKER, INTERACTION_TINKER.

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

- **W_ALPHA** (Float)

  One-line description : Weight of thermal expansion coefficient

  Default Value : 1.0

  (Needs full documentation)

- **W_CP** (Float)

  One-line description : Weight of isobaric heat capacity

  Default Value : 1.0

  (Needs full documentation)

- **W_ENERGY** (Float)

  One-line description : Weight of energy

  Default Value : 1.0

  (Needs full documentation)

- **W_EPS0** (Float)

  One-line description : Weight of dielectric constant

  Default Value : 1.0

  (Needs full documentation)

- **W_FORCE** (Float)
  One-line description : Weight of atomistic forces
  Default Value : 1.0
  (Needs full documentation)

- **W_HVAP** (Float)
  One-line description : Weight of enthalpy of vaporization
  Default Value : 1.0
  (Needs full documentation)

- **W_KAPPA** (Float)
  One-line description : Weight of isothermal compressibility
  Default Value : 1.0
  (Needs full documentation)

- **W_NETFORCE** (Float)
  One-line description : Weight of net forces (condensed to molecules, residues, or charge groups)
  Default Value : 0.0
  (Needs full documentation)

- **W_RESP** (Float)
  One-line description : Weight of RESP
  Default Value : 0.0
  (Needs full documentation)

- **W_RHO** (Float)
  One-line description : Weight of experimental density
  Default Value : 1.0
  (Needs full documentation)

- **W_TORQUE** (Float)
  One-line description : Weight of torques (condensed to molecules, residues, or charge groups)
  Default Value : 0.0
  (Needs full documentation)

- **WAVENUMBER_TOL** (Float)
  One-line description : Frequency normalization (in wavenumber) for vibrational frequencies
  Default Value : 10.0
  (Needs full documentation)

- **WEIGHT** (Float)
  One-line description : Weight of the target (determines its importance vs. other targets)
  Default Value : 1.0

Scope : All target types (Optional)

Full description : This option specifies the weight that the target will contribute to the objective function. A larger weight for a given target means that the optimizer will prioritize it over the others. When several targets are used, the weight should be chosen carefully such that all targets 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.

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 target's contribution, and then add weights accordingly.

- **WHAMBOLTZ** (Bool)

  One-line description : Whether to use WHAM Boltzmann Weights

  Default Value : 0

  Scope : Force and energy matching (Optional)

  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.

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

- **WRITELEVEL** (Int)

  One-line description : Affects the amount of data being printed to the temp directory.

  Default Value : 1

  (Needs full documentation)

Figure 4: Logo.