Alexandria Chemistry Toolkit

Release 1.2b

David van der Spoel **Paul J. van Maaren**
Mohammad M. Ghahremanpour

Apr 02, 2026

CONTENTS:

CHAPTER

ONE

PREFACE

1.1 Before you begin...

This manual is a work in progress, covering the physical background of the force field development in the ACT, which is short for Alexandria Chemistry Toolkit, from potential energy functions to the molecular properties supported by the ACT, algorithms used for training, how to generate training data, installation and usage.

A pdf version of this manual is available.

Throughout this document we highlight *alexandria commands* like this. There are also hyperlinks in the document, both for the references and in the running text, often linking to Wikipedia.

The authors welcome suggestions for additions and improvements as well as corrections for factual errors. Please send your comments by e-mail to david.vanderspoel at icm.uu.se.

We kindly request that you cite the paper about ACT [1] and other relevant works if you use the software for a scientific publication of your own.

The ACT can be downloaded as free and open source software from Github.

1.2 About the Alexandria Chemistry Toolkit

The Alexandria Chemistry Toolkit (ACT) allows for global optimization of force fields for molecular simulation. That means that, in principle, a whole force field can be derived at once by training parameters to reproduce results from *ab initio* or density functional theory as well as experimental data when available.

The ACT software consists, at the time of writing about 107,000 lines of C++ code, of which about 35,000 lines were generated by the Mathematica ™ software (Section *Coulomb Interaction using Slater charges*). In addition about 7,500 lines of Python code are part of the ACT.

The ACT was developed in the lab of David van der Spoel at Uppsala University, Sweden. A list of contributors is given on the github page of the project.

1.3 Motivation for the ACT

The fundamental motivation for the development of the ACT are given in these reviews [2, 3, 4, 5, 6, 7, 8]. Milestones on the way to a new family of physics-based force fields are given below.:

- A long series of benchmarks of force fields [9, 10, 11, 12, 13, 14, 15].
- Databases of quantum-chemical data produced as part of the Alexandria project: enthalpies of formation and thermochemistry [16] applied to force fields here [13].

- The Alexandria Library with more quantum-chemistry data [17, 18].
- A phase-transferable force field for alkali halides [19] with applications towards prediction of melting points and conductivities for alkali halide liquids and mixtures [20, 21, 22, 23].
- A new potential for treating the angle in linear molecules [24].
- New models for noble gases including accurate simulations of the melting point of solid noble gases [25].
- Use of symmetry-adapted perturbation theory for a study of exchange around a water molecule [26].
- An evaluation of potentials for chemical bonds in molecules [27].

The remainder of this document contains first the User Manual, describing installation and practical usage of the ACT including the properties that can be predicted by the ACT software and Alexandria force fields.

The second part is the Reference Manual which presents, somewhat rudimentary at the time of writing, the physical foundation of the ACT models and the algorithms for training force fields.

Happy force field designing!

CHAPTER

TWO

USER MANUAL

2.1 Installation of the ACT

The Alexandria Chemistry Toolkit (ACT) relies on a number of libraries. Even though we tried to keep it to a minimum, some more or less standard libraries are needed. ACT should compile fine on any UNIX (including MacOs) or Linux machine. %Most of the libraries can be installed using Anaconda or even Miniconda which has the advantage of running in user-space entirely, that is you do not super-user access to install it.

2.1.1 Prerequisites

The following software packages are required for the ACT to work:

- C and C++ compilers supporting C++20 at least. On Linux a GNU c++ version newer than 12.0 is recommended.
- Some version of a library that supports the message passing interface (MPI) for parallel programming. A popular version is OpenMPI.
- The cmake tools (at least version 3.13.0) are needed for compiling the code.
- For linear algebra operations we use the Eigen library, version 5 or better.
- The RDKit library (at least version 2025.09.4)
- The Boost developer library of version 1.86.0 (matching RDKit)
- The LibXml2 is needed for processing the XML data files used by the ACT.
- Python, version 3.8 or better, and a number of Python libraries, namely, NumPy, Matplotlib, and PubChemPy.

The following libraries are optional only, but may be useful for developing the ACT:

- The Class Library for Numbers is used in an optional part of the code (Slater-distributed charges [17]) and can be omitted.
- The SQLite database engine is needed to process experimental data as well as quantum chemistry data from the Alexandria Library [17], available from Zenodo.

For developers, the following additional packages are needed:

- doxygen, it is used for generating software documentation,
- graphviz can be optionally added for generating graphs and tree-structures in the doxygen documentation,
- pygments, for source code listings,

- sphinx, which is used for building the manual, with
- sphinxcontrib-bibtex, for the references and
- nbsphinx for including Jupyter Notebooks.

2.1.2 Conda Environment

There are multiple ways to fulfill the prerequisites. The simplest way that should suffice on a single computer (i.e. not a cluster), is to first install miniconda on your computer, download the ACT conda environment file Yml and create and activate a new conda environment.:

```
conda create -n ACT
conda activate ACT
conda config --add channels anaconda
conda config --add channels conda-forge
conda install librdkit-dev>=2025.09.4 libboost-devel>=1.86.0 cmake eigen>=5␣
↪libxml2 numpy matplotlib pubchempy pillow
```

This should install the libraries mentioned above (note: it will take some time!). If you are installing ACT on a high-performance computing cluster, there likely is support for compilers and a MPI library already. If not, then add the *openmpi* package to your conda install line. Most Linux installations come bundled with the GNU compiler suite (GCC) and for macOS the Xcode package can be downloaded free of charge from Xcode. If you do not have a compiler, add *gcc* to the conda install line:

```
conda install gcc=14 gxx=14 openmpi=5
```

For developers, please additionally install these packages:

```
conda install doxygen graphviz pygments sphinx sphinxcontrib-bibtex nbsphinx
```

Attention

If you are installing ACT in a cluster we recommend to use the cluster-provided compilers and in particular the MPI library since it may be tuned to make optimal use of the communication hardware. High-performance computer centers typically provide compilers libraries using some kind of module system.

2.1.3 Running the Installation

Once you have download ACT either as a release version or by cloning the git repository, enter the directory where the ACT source code is installed, and issue the following commands:

```
mkdir build-Release
cd build-Release
cmake -DCMAKE_INSTALL_PREFIX=$HOME/tools ..
make -j 8 install
```

where the *CMAKE_INSTALL_PREFIX* flag points to the location where ACT will be installed and the *-j 8* flag instructs the make command to utilize 8 cores to speed up compilation.

In order to start using the software, run the following command:

```
source $HOME/tools/bin/ACTRC
```

or add it to your .bash_profile (or equivalent, for remote machines) or .bashrc (or equivalent, for local machines), and restart the shell or log in again. Then you can run the alexandria executable using:

```
alexandria -h
```

To make sure you do have the correct commands in your path, please try the command:

```
which alexandria
```

which should give you something like:

```
% which alexandria
~/tools/bin/alexandria
```

There are some building options available that are mainly of use for developers (see *cmake flags available to build the alexandria program.*). These options have to be specified using:

```
-DOPTION=Value
```

where *Value* can be *ON* or *OFF* or something more option specific.

Table 2.1: cmake flags available to build the alexandria program.

Flag	Description
CMAKE_BUILD_TYPE	Build Type: either Release (default) Debug (for use with a debugger) or ASAN (Adress Sanitizer, for uncovering memory leaks and debugging crashes).
CMAKE_INSTALL_PREFIX	Path where to install the ACT, see above
CMAKE_PREFIX_PATH	Path where cmake can look for libraries. Multiple paths can be specified, separated by semicolons, for instance -DCMAKE_PREFIX_PATH=${CONDA_PREFIX}/lib
ACT_CLN	Install ACT using the Class Library for Numbers for high precision calculations, used for Slater integrals. Default OFF, activated when ON.
ACT_BUILD_MANUAL	Whether or not to provision for building the manual. Requires installing developer tools, default OFF.
CMAKE_C_FLAGS	By default nothing is needed, but sometimes this linker flag helps to overcome problems: "-L${CONDA_PREFIX}/lib -Wl,-rpath,${CONDA_PREFIX}/lib"
CMAKE_CXX_FLAGS	See previous entry, you likely need to add both.

2.1.4 Troubleshooting Installation

Sometimes, the *cmake* process or building using *make* does not work as described above. For instance, there may be library mismatches like this:

```
undefined reference to `__cxa_call_terminate@CXXABI_1.3.15'
```

which is caused by the fact that conda libraries expect specific versions of system libraries, combined with the make process supplying a wrong version of that library. In that case you can instruct cmake to

change the order of libraries by adding the flags to the cmake command line according to Table 2.1.

2.1.5 Testing the ACT

To start testing, you first want to familiarize yourself with the test set. If the ACT is in your home directory, you can:

```
cd ACT/build-Release
```

Then you can build the test set using:

```
make tests
```

and run it using:

```
make test
```

which should give the following output:

```
Running tests...
Test project /Users/spoel/GG/ACT/build_Release_DOUBLE
      Start  1: TestUtilsUnitTests
 1/17 Test  #1: TestUtilsUnitTests ..............   Passed    3.10 sec
      Start  2: WangBuckinghamTests
 2/17 Test  #2: WangBuckinghamTests .............   Passed    0.33 sec
      Start 16: AlexandriaTests
```

(more tests):

```
16/17 Test #16: AlexandriaTests .................   Passed    8.80 sec
      Start 17: SobolTests
17/17 Test #17: SobolTests ......................   Passed    0.16 sec

100% tests passed, 0 tests failed out of 17
```

You can also run an individual test, like this bin/sobol-test which should give this output:

```
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from SobolTest
[ RUN      ] SobolTest.Test08
[       OK ] SobolTest.Test08 (0 ms)
[ RUN      ] SobolTest.Test09
[       OK ] SobolTest.Test09 (0 ms)
[----------] 2 tests from SobolTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
```

Note that these tests are run every time a change in the ACT source code is uploaded to github, to prevent errors in the code from being introduced.

2.2 Using the ACT

To use the ACT, you first need to install the ACT. Once you have finished that, please try the command:

```
alexandria -h
```

and:

```
alexandria help commands
```

the output of which is given in Table 2.2 .

Table 2.2: Commands available in the alexandria program.

Com man	Description
an- a- lyze	Analyze molecular or force field properties from a database and generate publication quality tables in LaTeX.
b2	Compute second virial coefficient as a function of temperature.
edit_	Manipulate and compare force field files in various ways and test whether reading and writing works.
edit_	Utility to merge a number of molecular property files and a SQLite database. Can also test reading and writing the molecular property file. It can also check the molecular property file for missing hydrogens and for whether it is possible to generate topologies for all compounds. Finally it can generate charges for all compounds.
gen_f	Generate a force field file from a user specification.
gen- top	Generate a molecular topology and coordinates based on structure files. Only inputs for OpenMM can be generated at this point in time.
ge- om- e- try_fi	Deduce bond/angle/dihedral distributions from a set of structures and add those to a force field file.
help	Print help information.
merg	Utility to merge a number of force field files and write a new file with average parameters. Can also write a LaTeX table.
min_	Generate inputs for an energy scan.
nma	Perform normal mode analysis and compute vibrational frequencies and thermochemistry properties.
sim- u- late	Perform a MD simulation and generate a trajectory.
train_	Train a force field to reproduce reference data.

Note that one can get detailed help for the alexandria modules using the -h flag, e.g.:

```
alexandria train_ff -h.
```

In addition to the *alexandria* program, the ACT contains a number of python scripts and utilities (Table 2.3), some of which are needed for the quantum chemistry calculations as well (see Sections *Generating SAPT data* and *Generating single molecule data*).

Table 2.3: Python utilities available in the ACT.

Command	Description
coords2molpro	Read a bunch of structure files containing molecule dimers A-B and write a molprop file.
dimer_scan	Compute dimer potentials for along a user-defined distance between two molecules.
donchev2mo	Convert dimer interactions from the Donchev paper [28] to a molprop file.
gauss2molpr	Read a Gaussian output file from the Alexandria library [17] to a molprop file.
generate_mp	Read and process many Gaussian output files from the Alexandria library [17] and generate a molprop file.
install_act	Script to install the ACT
molselect	Make a selection file based on compounds from the Alexandria Library.
ncia2molpro	Read an xyz file and write a molprop file.
plot_converg	Read an ACT training log file and plot the converge of the parameters as a function of generations in the evolution of the gene pool.
reshuffle_selection	Read a selection file and randomly assign Train status to entries, then write a new selection file.
view_fitness	Visualizes the fitness per generation of a GA/HYBRID by plotting the maximum, minimum, mean, and median, for an example see Fig. 2.1

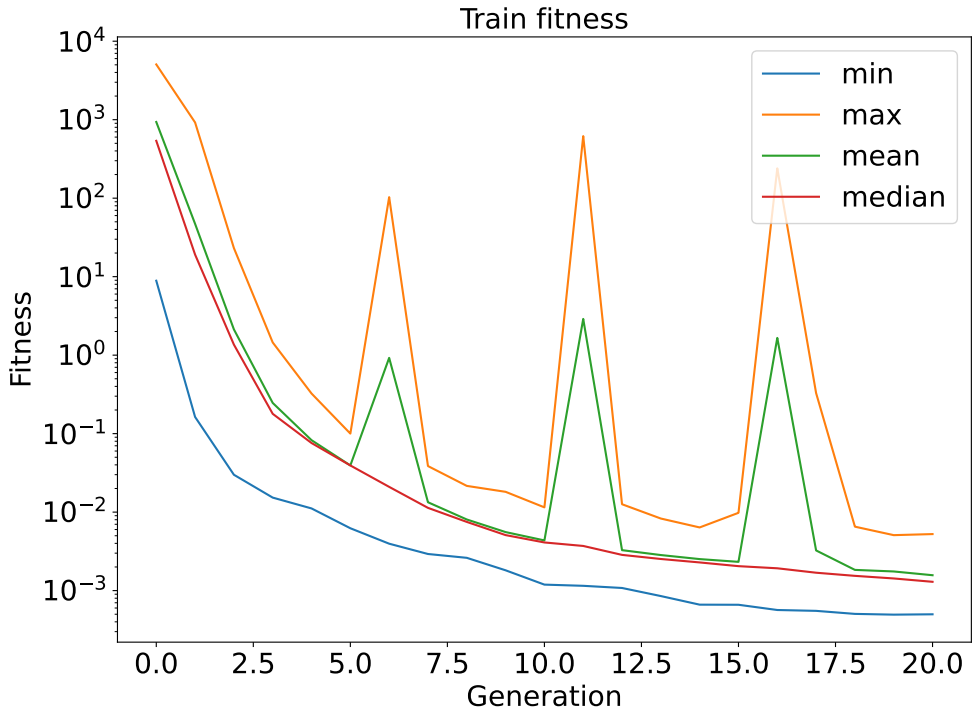

Fig. 2.1: Sample convergence plot from a HYBRID training, generated by the view_fitness script.

2.2.1 ACT File formats

The ACT contains a number of specific file formats, in principle all of them generated by the ACT itself through scripts (Table 2.3) or *alexandria* (Table 2.2). The files are listed below:

- Selection file, with *.dat* extension. List of monomers or dimers and designation on whether they are part of the train or test set. In case of dimers, the two compounds are separated by a hash (#) symbol.:

```
water#ethanol|Test
water#water|Train
ethanol#ethanol|Train
```

- Force field file, with *.xml* extension.
- Molprop file, with *.xml* extension.

2.2.2 Creating a new force field file from scratch

Start by copying the examples directory from the source catalog, and change directory to *examples/NonPol*.

Now, let's see what *alexandria gen_ff* has to offer:

```
alexandria gen_ff -h
```

(to see the output check the help text).

As we see there are many options, but for Alexandria force fields most can remain default. Make a fresh directory and use the one-line script:

./genff.sh

which gives output similar to:

```
There are 62 atom types in the force field with 42 properties.
There are 127 element properties

Thanks for using the Alexandria Chemistry Toolkit.
```

The script prints that it has read and processes a number of files from the ACT installation. These data files can be copied and modified by the user. The files should be relatively simple to understand. Installed files are in the share/act directory. Please note that the force field files use the eXtensible Markup Language that provides a structured way of storing data. Do not edit manually if at all possible.

This force field file gives us something to start with, but it is not complete yet. First, we have to add the possible bond lengths, bond angles etc., and those come from the Alexandria Library (see below). For now, we will use the provided file in XML/alcohol.xml. Thus, we need to run:

```
./geomff.sh
```

which should produce the following output:

```
   Welcome to the Alexandria Chemistry Toolkit

There are 5 molecules in the selection file SELECTIONS/alcohol-monomer.dat.
```

(continues on next page)

(continued from previous page)

```
There are 5 molprops. Will now sort them.
There were 0 double entries, leaving 5 after merging.
There were 5 total molecules before merging, 5 after.
Have generated 34 entries for bond charge correction (SQE algorithm).

Please check output in file geometry_ff.log.
```

This generates a ready-to-optimize force field file, myff2.xml and a log file that is good to inspect. It provides statistics over the geometry of compounds in the input xml file:

```
bond-c3_b~h_b len 108.8 sigma 0.3 (pm) N = 28
bond-c3_b~c3_b len 150.9 sigma 0.6 (pm) N = 7

angle-c3_b~c3_b~c3_b angle 112.1 sigma 0.6 (deg) N = 3
angle-c3_b~c3_b~o3_b angle 108.6 sigma 3.8 (deg) N = 7
```

which informs us, for instance, that there are 28 aliphatic c3-h bonds in the input with an average length of 108.8 $\pm$ 0.3 pm.

2.2.3 Train your first force field

Now you can run the first part of the example training, the intermolecular interactions, using:

```
./run_inter.sh
```

and the adventure has begun! This part will train the van der Waals parameters σ and ϵ (Eqn. (3.18)) as well as the charge distribution width ζ (Eqn. (3.2)) and the electronegativity χ and hardness η. The command will output some text to the terminal:

```
There are 16 threads/processes and 5 parameter types to optimize.
rank: 0/16 nodetype: Master superior: 0
nmiddlemen: 16 nhelper_per_middleman: 0
ordinal: 0 nhelper: 0
middlemen:   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

showing the genetic algorithm is using 16 individuals and the code is being run on 16 threads in parallel. Then, at the end you will see something like:

```
There were 39 EPOT-Train outliers for Train.
There were 29 EPOT-Test outliers for Test.

Please check output in file train_inter.log.
```

encouraging you to inspect the log file. It can be very helpful to check outliers that can help you to find inconsistencies in both the data and the model you are trying to train. After this training instance, a correlation plot *EPOT.xvg* will be produced (Fig. 2.2); you can use the:

```
./plot-epot.sh
```

script to make the plot yourself as well.

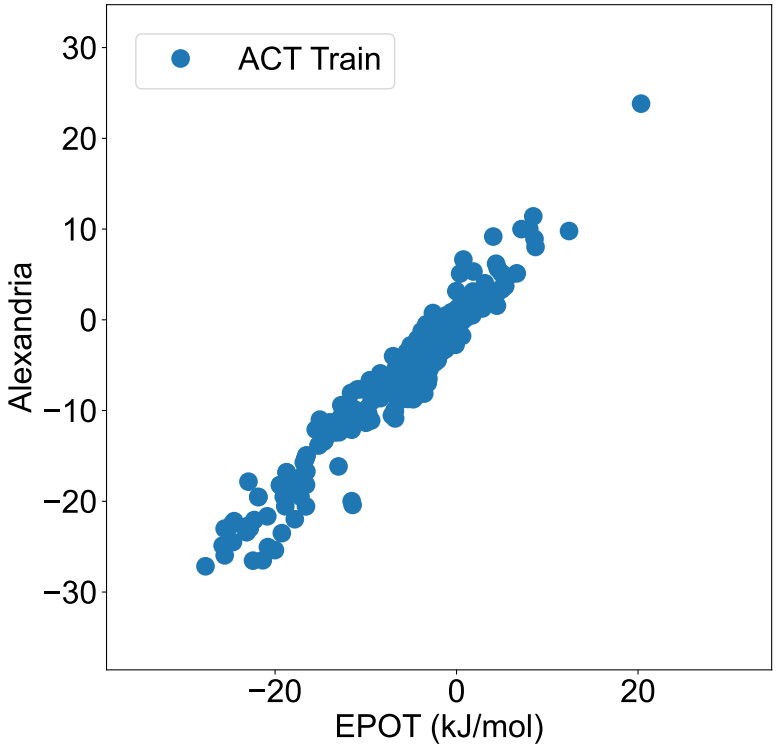

Fig. 2.2: Correlation between interaction energy from SAPT (x-axis) and training (y-axis).

You can inspect the different .xvg output files in the generated subdirectory *inds*. Fig. 2.3 shows how the total deviation from SAPT data, you can recreate the figure using the:

```
./plot-chi2.sh
```

script. Fig. 2.4 shows how well the Gaussian distribution widths converge.

Note that these graphs were made using the plotxvg script that is based on matplotlib.

After training the parameters governing intermolecular interactions, it is time to train the bonded forces. For this, we need to modify the selection of compounds to be monomers, we use the force field file that was trained on SAPT interaction energies (Train-inter.xml) and reference energies come from MP2 calculations.

```
./run_intra.sh
```

will do this training. As targets in this training we use both the intramolecular energies and the forces, however the deviations in the forces are weighted down by a factor of 0.1 because of the magnitude of the numbers. You will get similar output as for the intermolecular training:

```
There were 60 EPOT-Train outliers for Train.

Please check output in file train_intra.log.
```

Again, please inspect the contents of the log file carefully. A quick check of the intramolecular force field can be done by computing frequencies:

```
./run_nma.sh
```

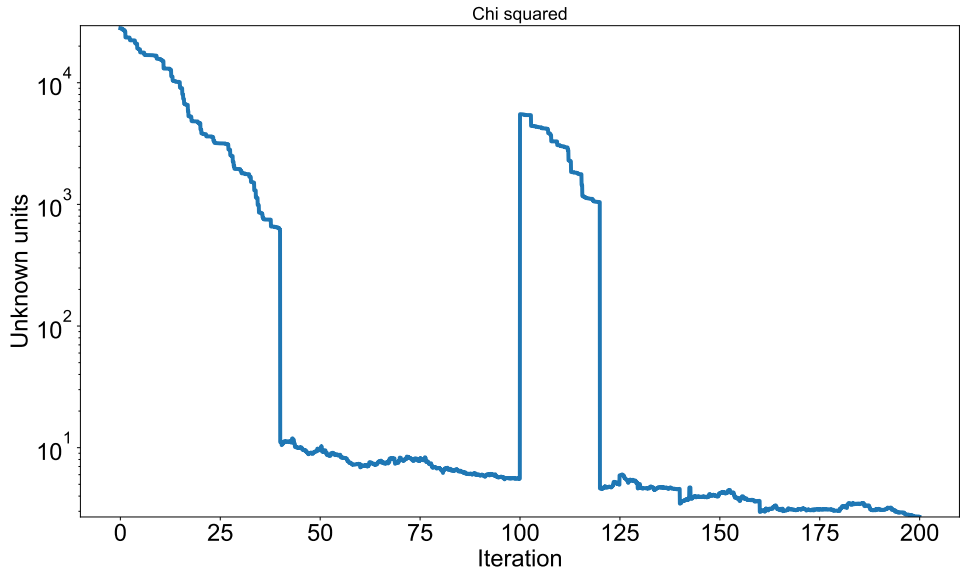

Fig. 2.3: Convergence of the χ^2 fitness value for the first individual in the example training. Note the jump at iteration 100, due to the catastrophe penalizer kicking in.

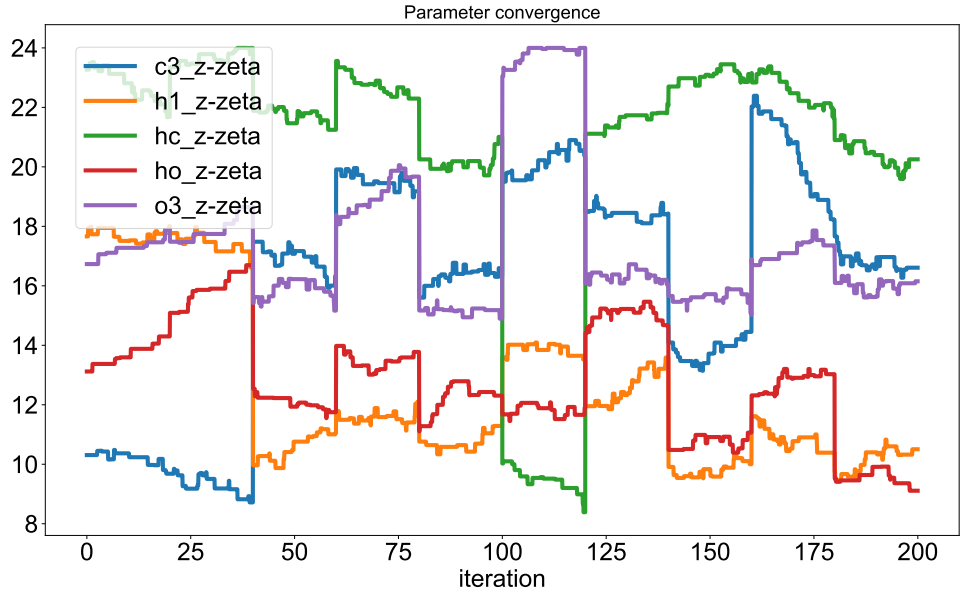

Fig. 2.4: Convergence of the Gaussian distribution widths ζ for five atom types. The jumps in the graphs are due to genetic algorithm performing crossover between individuals.

which will perform a normal mode analysis (Sec. *Normal Mode Analysis*) and compute an infrared spectrum (Sec. *Infrared Spectra*) based on your new force field. This should yield a spectrum like the one in Fig. 2.5.

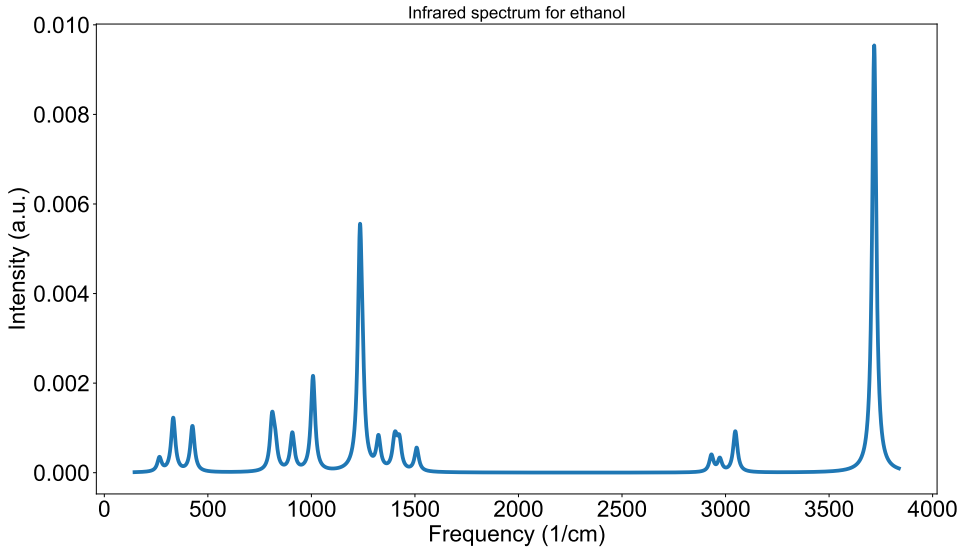

Fig. 2.5: Simulated infrared vibrational spectrum for ethanol based on the force field derived in this tutorial (please compare to Fig. 2.7).

As a final test of your new force field, we can use *alexandria simulate -minimize* to determine the optimized structure of a methanol dimer. For this, run:

```
./minimize.sh
```

and check the output structure, *after_em.pdb*. It should look like Fig. 2.6.

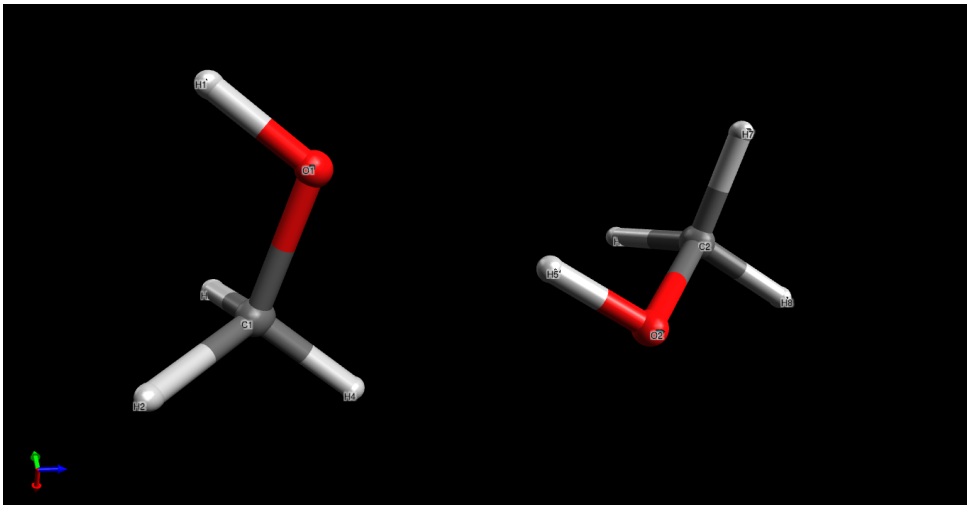

Fig. 2.6: Structure of a methanol dimer after minimization using the force field derived by the ACT in this tutorial. Visualization using Avogadro.

Please make sure to inspect all the scripts provided here to learn about the command line options to ACT modules, but be advised that there are many more options and possibilities.

To design a force field for your own system of choice you will have to provide quantum chemistry data (Sec. *Training Data*), and create selection files (Sec. *ACT File formats*). Most of the steps you have

gone through in this tutorial can be adapted to your need. For large trainings it is useful to study the parallellization features of the ACT (Sec. *Running training using parallel processing*).

2.2.4 Is my training hitting a wall?

To make training efficient, it is good to limit the search space. In some cases we can use chemical intuition to estimate the search space. For instance, we know that polarizability should be a positive number and that it is highly element dependent. For hydrogen, a likely range is $0.1 < \alpha < 0.5$ Å^3. Such boundaries are implemented in the force field file like this:

```
<parameterlist identifier="h_s">
  <parameter type="alpha" unit="Angstrom3" value="0.46"
  uncertainty="0.008" minimum="0.1" maximum="0.5"
  ntrain="300" mutability="Bounded" nonnegative="yes"/>
</parameterlist>
```

this example is after an optimization where the optimal value ended up being 0.46, based on a dataset comprising 300 hydrogen atoms. Note the nonnegative=”yes” which will prevent alexandria tools from making this parameter negative ever.

For many cases, we do not have an {em a-priori} intuition of what the values should be and we make a guess. Then, after a series of optimizations, we can check whether the bounds are OK, using an alexandria tool (assuming you have generated and trained a polarizable force field *.xml*):

```
alexandria edit_ff -ff ff.xml -ana EEM
<snip>
POLARIZATION n2_s alpha at maximum 1.5
POLARIZATION n4_s alpha at minimum 0.6
BONDCORRECTIONS c2_~c3_z hardness at minimum -0.4
BONDCORRECTIONS c2_z:n2_z hardness at minimum 0
BONDCORRECTIONS c3_z~c3_z hardness at minimum -0.4
```

Some of the parameters are indeed hitting the wall. To adjust the parameters we use the same tool again:

```
alexandria edit_ff -ff ff.xml -o new.xml -stretch -p alpha
<snip>
Thanks for using the Alexandria Chemistry Toolkit.
```

and then we check the resulting force field file in the same manner as before:

```
alexandria edit_ff -ff new.xml -ana EEM
<snip>
BONDCORRECTIONS c2_z~c3_z hardness at minimum -0.4
BONDCORRECTIONS c2_z:n2_z hardness at minimum 0
BONDCORRECTIONS c3_z~c3_z hardness at minimum -0.4
```

Now the polarizability minimum (for n4_s) and maximum (for n2_s) have been stretched. The same can now be done for hardness.

You can also do the reverse. When you are confident that the optimization is close to the final result, but want to randomize again without starting from scratch, you can set new minimum and maximum values for the parameters using, e.g.:

```
alexandria edit_ff -ff new.xml -o new.xml -p bondlength -limits 0.98
```

please check the on-line help (*alexandria edit_ff -h*) for more information.

2.2.5 Running training using parallel processing

The training process is heavy in computer time. Therefore, it has been parallellized using the MPI library. Since this is a prerequisite for compiling the ACT, you likely have it installed if you got this far. In fact this is used in the example scripts, where N is the number of cores you have available. Please consult with your cluster manager if in doubt. Clusters using the Slurm queueing system may need to use *srun* instead of *mpirun*. As far as we have tested [1], the code is quite efficient down to 4-5 molecules per core. In the above example there are 10 compounds in the training set, such that it is not worthwhile to use more than 2-3 cores, but do experiment with the number of cores.

2.3 MD Simulations with Alexandria Force Fields

2.3.1 Using ACT simulate

The command *alexandria simulate* will perform a simulation in the gas phase, i.e. without periodic boundary conditions. This obviously limits the usefulness but the OpenMM software can be used for condensed-phase simulations using ACT force fields [1, 25]. The *alexandria simulate* utility is aimed for validation of the potentials, for instance by evaluating whether the total energy is conserved, it can be verified that the force is the derivative of the potential.

The utility can also be used to minimize the energy of a molecule or small cluster

> alexandria simulate -minimize -ff actff.xml -f coords.pdb -charges mp2.xml -c afterem.pdb -g minimize.log

where *actff.xml* is the force field file, *coords.pdb* is a structure file (*xyz* and *sdf* are supported as well). The *alexandira -charges mp2.xml* indicates a molprop file (database) with monomeric (optimized) structures used for generating charges. The files *afterem.pdb* and *minimize.log* are output files.

2.3.2 The ACT-OpenMM interface

The OpenMM [29] software in its native form is controlled from Python scripts. This allows users great control over the simulations. OpenMM allows to specify user-defined energy functions, which we use to implement both Gaussian-distributed charges [19, 30] and many Van der Waals potentials [25] (see Section *Energy Function*). To facilitate this, a special python code was implemented that makes it relatively easy for the user to run simulations and minimization. The first step is to convert an ACT force field file (*actff.xml}) to one compatible with OpenMM (*openmmff.xml*):

```
alexandria gentop -ff actff.xml -openmm openmmff.xml
-charges mp2.xml -db "water methanol"
```

with two additional arguments. First, flag *-charges mp2.xml* indicates a molprop file (database) with monomeric (optimized) structures used for generating charges and, second, the flag *-db "water methanol"* instruct the code to produce an OpenMM topology for those compounds. If one or more of the compounds is not present in the database, a warning will be issued.

2.3.2.1 MD simulations using OpenMM

It is easy to perform simulations based on an Alexandria force field. For this, we rely on the OpenMM software that you need to install separately. Then you can use the ACT python interface to OpenMM, as in this simple script, that we will call run.py:

```
#!/usr/bin/env python3

from act_openmm import ActOpenMMSim

sim = ActOpenMMSim(pdbfile="file.pdb",
                   datfile="dimer.dat",
                   xmlfile="ff.xml",
                   txtfile="output2.txt",
                   verbose=True)
sim.run()
sim.log_to_xvg("energy.xvg", [ "Potential Energy (kJ/mole)"])
sim.log_to_xvg("temperature.xvg", [ "Temperature (K)" ])
sim.log_to_xvg("density.xvg", [ "Density (g/mL)" ])
```

Here, we first instantiate an ActOpenMMSim object, and pass it a structure file *file.pdb*, a simulation parameter file *dimer.dat* and a force field file *ff.xml*. We also instruct the code that output should be written to *output2.txt* and that we want a lot of information in that file. Then we can run it, that's all! The last three lines are for convenience of plotting the results. We can run this script using:

```
python ./run.py
```

or submitted to a cluster, preferably with GPUs available.

2.3.2.2 Energy Minimization using OpenMM

An energy minimization of the input structure can be done using this script:

```
#!/usr/bin/env python3

from act_openmm import ActOpenMMSim

sim = ActOpenMMSim(pdbfile="file.pdb",
                   datfile="dimer.dat",
                   xmlfile="ff.xml",
                   txtfile="output2.txt",
                   verbose=True)
sim.setup()
sim.minimize()
sim.write_coordinates("final.pdb")
```

which is run here using the same flags as the simulation. Note that details about simulations and minimization can be specified in the *dimer.dat* file.

2.4 Predicting Molecular Properties

The ACT can be used to perform MD simulations of clusters in the gas-phase using the *alexandria simulate* command. This module includes the possibility to perform energy minimizations with the flag *-minimize*. For simulations employing periodic boundaries the OpenMM package~ [29] should be used instead. For more details about MD simulations, see Section *MD Simulations with Alexandria Force Fields*.

Below we describe some of the properties that can be computed using the ACT.

2.4.1 Electrostatic Potential

The charge distribution ρ of a molecule is determined by the nuclear position of the N atoms at positions $\mathbf{x}$ and the electron density $n(\mathbf{r})$. The molecular electrostatic potential (MEP) at a point in space r' is thus given by

$$\Phi(\mathbf{r}') = \frac{1}{4\pi\varepsilon_0}\left[\sum_{i=1}^{N}\frac{z_i}{\|\mathbf{x}_i - \mathbf{r}'\|} - \int\frac{n(\mathbf{r})}{\|\mathbf{r} - \mathbf{r}'\|}\mathrm{d}r\right] \tag{2.1}$$

where N is the number of atoms, z_i are the nuclear charges, ε_0 is the permittivity of vacuum and integration is over the entire space. The minus sign before the integral is due to the negative charge of electrons.

Accurate knowledge of the MEP contributes to, for example, the understanding of interactions and function of biological macromolecules in solution~ [31]. For a molecule in the gas phase, Eqn. (2.1) can be evaluated using density functional theory and wave function quantum chemistry, albeit at a significant computational cost. Databases of such calculations for small molecules are available to facilitate reuse [8]. For large condensed-phase systems, however, it is common to apply classical force fields, where electrons are not taken into account explicitly. Instead, effective partial charges on atoms are used. The electronic degrees of freedom, charge polarization, is sometimes taken into account through induced point dipoles and higher electrostatic moments, or by using a core-shell model~ [32, 33, 34]. For additional background we refer to some excellent reviews~ [35, 36, 37].

The MEP can be used as a target in model development in the ACT, however we recommend against that for both fundamental and practical reasons~ [38].

2.4.2 Electrostatic Moments

If point $\mathbf{r}'$ in Eqn. (2.1) is outside the distribution of electron density and $\mathbf{r}' >> \mathbf{r}$, the electrostatic potential can be evaluated through the Taylor expansion of $|\mathbf{r} - \mathbf{r}'|^{-1}$ [39]:

$$\frac{1}{|\mathbf{r}' - \mathbf{r}|} \approx \frac{1}{r} + (\hat{\mathbf{r}}\cdot\mathbf{r})\frac{1}{r^2} + \frac{1}{2}\left[3(\hat{\mathbf{r}}\cdot\mathbf{r})^2 - r^2 I\right]\frac{1}{r^3} + \cdots \tag{2.2}$$

where $\hat{\mathbf{r}} = \mathbf{r}'/r$ and I is the identity matrix. By inserting Eqn. (2.2) into Eqn. (2.1), we get

$$\Phi(\mathbf{r}') \approx \frac{1}{4\pi\epsilon_0}\frac{Q}{r} + \frac{\boldsymbol{\mu_0}}{r^2} + \frac{\boldsymbol{\Theta_0}}{r^3} + \cdots$$

Q is the monopole moment, sometimes called the zeroth moment of the molecular electron density. In principle this is the total charge of the molecule. In what follows we combine both the atomic cores and the electron density into $n(r)$. Then, Q is given by

$$Q = \int n(\mathbf{r})d\mathbf{r}$$

$\mu_\mathbf{0}$ is the vector of the permanent dipole moment, which measures the polarity of the molecular electron density. $\mu_\mathbf{0}$ is given by

$$\mu_\mathbf{0} = \int n(\mathbf{r}) \left(\hat{\mathbf{r}} \cdot \mathbf{r}\right) d\mathbf{r}$$

$\mathbf{\Theta_0}$ is the tensor of the permanent quadrupole moment, which exhibits the deviation of the distribution of the molecular electron density from spherical symmetry. $\mathbf{\Theta_0}$ is given by

$$\mathbf{\Theta_0} = \frac{1}{2} \int n(\mathbf{r}) \left[3 \left(\hat{\mathbf{r}} \cdot \mathbf{r}\right)^2 - r^2 I\right] d\mathbf{r}$$

The quadrupole tensor can be written as a traceless 3×3 matrix in the Cartesian coordinate if one writes $\left(\hat{\mathbf{r}} \cdot \mathbf{r}\right)^2$ as:

$$\left(\hat{\mathbf{r}} \cdot \mathbf{r}\right)^2 = \hat{\mathbf{r}} \cdot (\mathbf{rr}) \cdot \hat{\mathbf{r}}$$

where $\mathbf{rr}$ is the outer product of vector $\mathbf{r}$ with itself. This results in a matrix that can be written in terms of the Cartesian components of the vector $\mathbf{r}$ as follows:

$$\mathbf{rr} = \begin{bmatrix} x^2 & xy & zx \\ yx & y^2 & yz \\ zx & zy & z^2 \end{bmatrix}$$

Finally, we get

$$\frac{1}{2} \left[3 \left(\hat{\mathbf{r}} \cdot \mathbf{r}\right)^2 - r^2 I\right] = \hat{\mathbf{r}} \cdot \frac{1}{2} \begin{bmatrix} 3x^2 - r^2 & 3xy & 3zx \\ 3yx & 3y^2 - r^2 & 3yz \\ 3zx & 3zy & 3z^2 - r^2 \end{bmatrix} \cdot \hat{\mathbf{r}} \tag{2.3}$$

2.4.3 Polarizability

The shape of the molecular electron density changes when it interacts with an external electric field; hence, the total energy of the molecule changes. The static response of a molecule to a homogeneous external electric field, ($\mathbf{F}$), can be studied by expanding its energy in a Taylor series [40, 41]:

$$E\left(\mathbf{F}\right) = E\left(\mathbf{0}\right) + \frac{\partial E}{\partial \mathbf{F}}\bigg|_{\mathbf{F}=0} \mathbf{F} + \frac{1}{2} \frac{\partial^2 E}{\partial \mathbf{F}^2}\bigg|_{\mathbf{F}=0} \mathbf{F}^2 + \frac{1}{6} \frac{\partial^3 E}{\partial \mathbf{F}^3}\bigg|_{\mathbf{F}=0} \mathbf{F}^3 + \cdots$$

where

$$-\frac{\partial E}{\partial \mathbf{F}}\bigg|_{\mathbf{F}=0} = \mu_\mathbf{0}$$

$$-\frac{\partial^2 E}{\partial \mathbf{F}^2}\bigg|_{\mathbf{F}=0} = \alpha$$

$$-\frac{\partial^3 E}{\partial \mathbf{F}^3}\bigg|_{\mathbf{F}=0} = \beta$$

where $\mu_\mathbf{0}$ is the vector of permanent dipole moment, α is the tensor of polarizability, which is the linear part of the response of the molecular electron density with respect to the external electric field, and $\boldsymbol{\beta}$ is the first hyperpolarizability [40].

Instead of expanding the energy, we can expand the dipole moment of a molecule in an external electric field [41], written as

$$\mu = \mu_0 + \alpha\mathbf{F} + \frac{1}{2}\beta\mathbf{F}^2 + \cdots$$

where $\alpha\mathbf{F}$ gives the vector of induced dipole moment, $\mu_\mathbf{1}$ [40]:

$$\mu_\mathbf{1} = \alpha\mathbf{F}$$

that can be written in matrix form as

$$\begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \end{bmatrix} = \begin{bmatrix} \alpha_{xx} & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & \alpha_{yy} & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & \alpha_{zz} \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{2.4}$$

From the polarizability tensor the polarizability isotropy [41, 42],

$$\bar{\alpha} = \frac{(\alpha_{xx} + \alpha_{yy} + \alpha_{zz})}{3}$$

and the polarizability anisotropy [42],

$$\Delta\alpha = \sqrt{[(\alpha_{xx} - \alpha_{yy})^2 + (\alpha_{xx} - \alpha_{zz})^2 + (\alpha_{yy} - \alpha_{zz})^2 + 6(\alpha_{xy}^2 + \alpha_{xz}^2 + \alpha_{yz}^2)]/2}$$

can be calculated.

2.4.4 Normal Mode Analysis

Please note that the text below is largely taken (with permission) from a paper by Henschel *et al.* [43].

The ACT contains the *alexandria nma* tool that performs a normal mode analysis to determine the vibrational frequencies of a compound. Vibrational frequencies are required to compute the IR spectra and thermochemistry of molecules. The normal modes of molecular vibrations can be obtained by eigenvalue decomposition of the Hessian matrix, whose elements are the second derivatives of the energy with respect to the atomic coordinates q.

$$H_{ij} = \frac{\partial^2 E}{\partial q_i \partial q_j}$$

where i and j run from 0 to $N - 1$, where N is the number of atoms in the molecule. If virtual sites v are used, for example, to model the σ-hole for halogen atoms, the energy, E, depends on the positions of both atoms and virtual sites; that is, $E = E(q_0, ..., q_{N-1}, v_0, ..., v_{M-1})$, where the positions of the M virtual sites, v, in the compound are a function of the atomic coordinates q.

The Hessian is computed numerically—the N atoms are moved independently in all three spatial dimensions, and the forces are computed. From these forces, the second derivative of the energy is then evaluated numerically. Note that the positions of the virtual sites are updated before each force calculation, which means that their influence on the Hessian is taken into account explicitly when computing H.

2.4.4.1 Infrared Spectra

For the calculation of a full IR spectrum, in addition to the vibrational frequencies, the intensities and the line shapes are required.

In case of the quantum chemical calculations both the eigenfrequencies and the corresponding IR intensities are produced by default when a frequency calculation is requested in, for instance, the Gaussian software~ [44]. The frequencies for about 5000 compounds are available from the Alexandria library~ [17] at the B3LYP/aug-cc-pvtz level of theory~ [45, 46, 47, 48].Details of the quantum chemical calculations from which the frequencies were obtained have been presented previously [16, 17].

For the force field calculations, the intensities I_n were derived from the transition dipole derivatives:

$$I_n = \sum_{k=1}^{3} \left(\frac{\partial p_k}{\partial Q_n} \right)^2 \tag{2.5}$$

where k iterates over cartesian dimensions, p is the dipole moment of the molecule, and Q_n the normal coordinate n. In order to take into account virtual sites v we note that

$$p_k = p_k(q_0, ..., q_{N-1}, v_0, ..., v_{M-1})$$

and rewrite equation (2.5) as:

$$I_n = \sum_{k=1}^{3} \left(\frac{\partial p_k}{\partial q_s} \frac{\partial q_s}{\partial Q_n} \right)^2$$

where s iterates over the N atomic coordinates. To make the calculation of intensities practical, the numerical derivative of the dipole moment with respect to the atomic coordinates $\frac{\partial p_k}{\partial q_s}$ is stored in a text file during the normal mode analysis and finally we note that the term $\frac{\partial q_s}{\partial Q_n}$ corresponds to one over component s of eigenvector n.

As an example, an infrared spectrum can be generated using the OPLS2020 force field [49] and a molprop file using:

```
alexandria nma -ff OPLS2020 -charges OPLS2020-charges -db ethanol -ir ir-
↪ethanol
```

yielding the spectrum in Fig. 2.7.

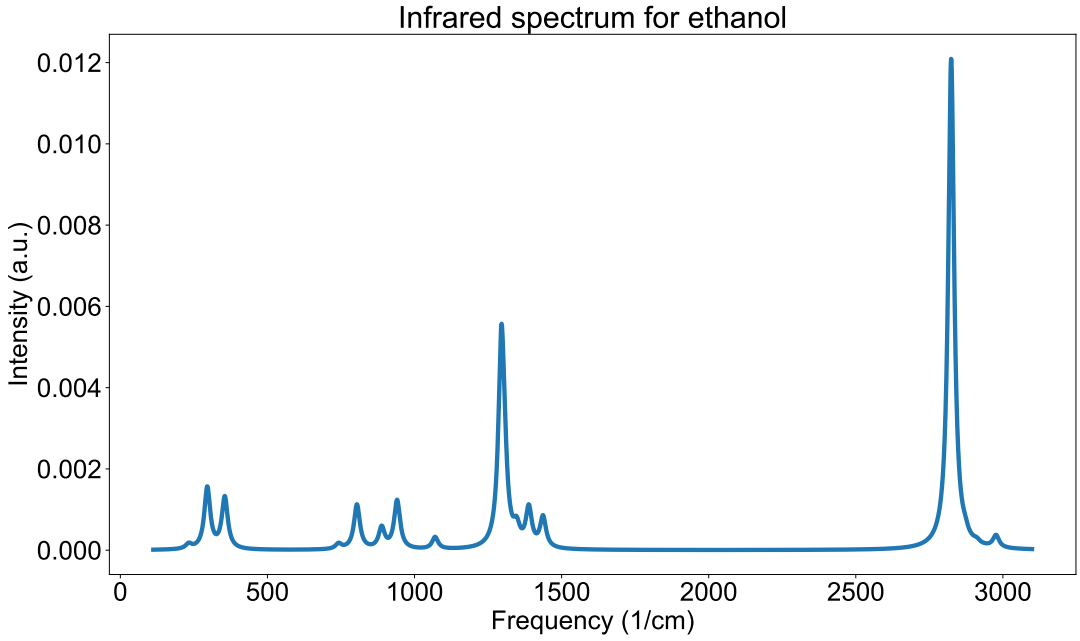

Fig. 2.7: Simulated infrared vibrational spectrum for ethanol, based on the OPLS2020 force field.

2.4.4.2 Thermochemistry

The Canonical ensemble $Q(N,V,T)$ can be used to compute the molecular Internal energy E ((2.6)), the standard Entropy S^o ((2.7)), and the Heat-Capacity C_v ((2.8)) at constant volume, using

$$E = RT^2 \left(\frac{\partial lnQ}{\partial T}\right)_{N,V} \tag{2.6}$$

$$S^o = RlnQ + RT \left(\frac{\partial lnQ}{\partial T}\right)_{N,V} \tag{2.7}$$

$$C_v = 2RT \left(\frac{\partial lnQ}{\partial T}\right)_{N,V} + RT^2 \left(\frac{\partial^2 lnQ}{\partial T^2}\right)_{N,V} \tag{2.8}$$

where R is the ideal gas constant and T the absolute temperature. For an ideal gas, $Q(N,V,T)$ can be decomposed into Partition-Function of different degrees of freedom: electronic (el), translational (tr), rotational (rot) and vibrational (vib) motions. Therefore, for a molecular ideal gas, $Q(N,V,T)$ can be expressed as:

$$Q(N,V,T) = \frac{(q_{el}q_{tr}q_{rot}q_{vib})^N}{N!} \tag{2.9}$$

The Rigid-Rotor and the Quantum-Harmonic-Oscillator can be used to approximate the contribution of the rotational and vibrational motions. The partition function of a rigid rotator is defined as

$$q_{rot} = \frac{T}{\sigma\Theta_{rot}}$$

where σ is the symmetry number, Θ_{rot} the rotational temperature defined as

$$\Theta_{rot} = \frac{h^2}{8\pi^2 I k_\beta}$$

where I is the moment of inertia, k_β the Boltzmann constant, and h the Planck constant. The partition function of a quantum harmonic oscillator is defined as

$$q_{vib} = \frac{e^{\frac{-\beta h\nu}{2}}}{1 - e^{-\beta h\nu}}$$

where ν is the vibrational frequency of the oscillator. If we define the vibrational temperature as $\Theta_{vib} = \frac{h\nu}{k_\beta}$, then we will have

$$q_{vib} = \frac{e^{\frac{\Theta_{vib}}{2T}}}{1 - e^{\frac{\Theta_{vib}}{T}}}$$

Applying Eqn. (2.9) to Eqn. (2.6) followed by the multiplication rule in logarithm yields:

$$E = E_{tr} + E_{rot} + E_{vib}$$

and similarly,

$$C_v = C_{tr} + C_{rot} + C_{vib}$$

$$S^o = S_{tr} + S_{rot} + S_{vib}$$

Thermochemical properties are computed automatically by the *alexandria nma* command. For more details, please see Van der Spoel *et al.* [13].

The *alexandria nma* used above to generate the infrared spectrum in Fig. 2.7 computes the thermochemical variables as well (Table 2.4). Information on calculation of the enthalpy of formation will be pubslished in the near future.

Table 2.4: Thermochemical values for ethanol at 298.15 K based on the OPLS2020 force field.

Property	Experiment	OPLS2020
S^o (J/mol K)	281 [50]	273.0
C_v (J/mol K)	57 [16]	62.2

2.4.5 Second Virial Coefficient

The second Virial-Coefficient is the second term in the Virial-Expansion that describes the deviation from the ideal gas law for real gases:

$$\frac{P}{RT\rho} = A + B_2(T)\rho + C_3(T)\rho^2 + ...$$

with P the pressure, R the gas constant, T the temperature and ρ the density.

$B_2(T)$ is a useful property gauging interactions in the gas phase because experimental values are available for close to 2000 compounds as a function of temperature. It is computed from an integral weighting the interaction between two molecules over three-dimensional space.

$$B_2^{cl}(T) = -\frac{1}{2}\int_0^\infty \left\langle e^{-\beta u_{12}(\mathbf{r})} - 1 \right\rangle d\mathbf{r}$$

where $u_{12}(\mathbf{r})$ is the interaction energy between two compounds (Eqn. (3.53)), $\beta = 1/k_BT$ and the integral is over all space and relative orientations of the compounds. If we sample these adequately (including at close, repulsive, distance) we can simplify the integral to a one-dimensional one:

$$B_2^{cl}(T) = -2\pi\int_0^\infty r^2 \left\langle e^{-\beta u_{12}(r)} - 1 \right\rangle dr$$

The above equation is entirely classical and quantum corrections have to be added according to:

$$B_2^{F}(T) = \frac{\hbar^2}{24(k_BT)^3}\sum_{j=1}^{2}\left[\frac{\left\langle \mathbf{F}_j^2 \right\rangle}{m_j}\right]$$

for the force on the compounds and

$$B_2^{\tau}(T) = \frac{\hbar^2}{24(k_BT)^3}\sum_{j=1}^{2}\left[\sum_{\alpha=x,y,z}\frac{\left\langle \tau_{j,\alpha}^2 \right\rangle}{I_{j,\alpha}}\right]$$

for the torque on the compounds, where m_j is the mass of the molecules j and $\mathbf{F}^2$ is the averaged square force on one molecule given by

$$\left\langle \mathbf{F}^2 \right\rangle = k_BT\int_0^\infty \left\langle e^{-\beta u_{12}(\mathbf{r})}\left[\nabla u_{12}(\mathbf{r})\right]^2 \right\rangle d\mathbf{r}$$

and where I is the moment of inertia of the molecule and τ^2 is the average square torque on one molecule defined by

$$\left\langle \tau_{j,\alpha}^2 \right\rangle = k_BT\int_0^\infty \left\langle e^{-\beta u_{12}(\mathbf{r})}\left[\nabla_\omega u_{12}(\mathbf{r})\right]_{j,\alpha}^2 \right\rangle d\mathbf{r}.$$

The change in $B_2(T)$ as a function of temperature can be used to scrutinize the repulsive and attractive components of the potential energy. $B_2(T)$ is negative at low temperatures due to attraction forces, while it becomes positive at higher temperatures as repulsion forces start to dominate, and passes through a maximum and eventually decreases at very high temperatures where repulsion force are fully dominant~ [51].

Code to compute the second virial coefficient is available in the *alexandria b2* command. Since the calculation is relatively expensive it has been implemented to use parallel processing using the message passing library. You can run it on a 16-core machine like:

```
mpirun -n 16 alexandria b2 -v 3 -g TIP4P -b2 TIP4P -ff TIP4P
-f water#water.pdb -T1 373.15 -T2 673.15 -dT 25.0
-maxdimer 32768
```

where TIP4P corresponds to the well-known water model~ [52], the T_1 and T_2 are the temperature limits (note gas-phase for water), dT is the temperature interval and maxdimer determines how many relative orientations will be evaluated. Due to the underlying algorithm for Quasi-Random numbers, this should be a power of two. The result is plotted in Fig. 2.8.

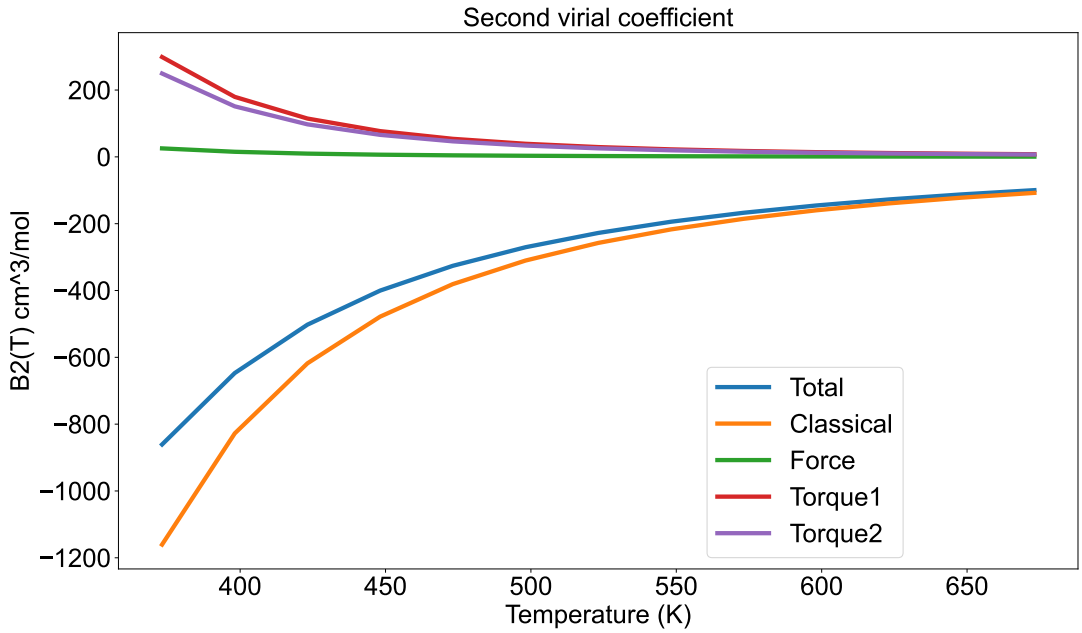

Fig. 2.8: Sample second virial coefficient and components for water using the TIP4P model.

CHAPTER

THREE

REFERENCE MANUAL

3.1 Energy Function

The ACT supports a range of different potentials for classical force field simulations. These potentials are described briefly in what follows.

3.1.1 Non-bonded interactions

3.1.1.1 Coulomb Interaction using Point charges

The Coulomb interaction can be computed with point charges:

$$V_{coul}(r_{ij}) \;=\; \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r r_{ij}}. \tag{3.1}$$

3.1.1.2 Coulomb Interaction using Gaussian charges

Alternatively, the Coulomb interaction can be described using Gaussian shielded charges

$$V_{coul}(r_{ij}) \;=\; \frac{q_i q_j \mathrm{erf}\left(\zeta_{ij} r_{ij}\right)}{4\pi\varepsilon_0\varepsilon_r r_{ij}}, \quad \zeta_{ij} = \frac{\zeta_i \zeta_j}{\sqrt{\zeta_i^2 + \zeta_j^2}}, \tag{3.2}$$

where ε_0 is the permittivity of vacuum, $q_{i,j}$ are the charges and $\zeta_{i,j}$ are the charge distribution widths (screening factors). Note that Eqn. (3.2) includes a relative dielectric constant ε_r that can be used to parameterize a non-polarizable force field using charge-scaling [53].

3.1.1.3 Coulomb Interaction using Slater charges

In addition, Slater distributed charges can be used as described in ref. [30], from which the text below is an adapted version.

The spherical Slater orbital wavefunction is given by

$$\psi_n(\mathbf{r}) \;=\; \sqrt{\frac{(2\zeta)^{2n+1}}{4\pi(2n)!}}\mathbf{r}^{n-1}\mathrm{e}^{-\zeta\mathbf{r}} \tag{3.3}$$

where n is the highest quantum number of the element and ζ is the exponent giving the width of the charge density. The distribution of charge can be described by the charge density, which is the square of the wave function ((3.3)). The Coulomb integral can then be written as [54, 55]:

$$J_{ij}(\mathbf{r}) \sim \int\int |\psi_n(\mathbf{r_i})|^2 \frac{q_i q_j}{|\mathbf{r_i} - \mathbf{r_j}|} |\psi_m(\mathbf{r_j})|^2 d\mathbf{r_i} d\mathbf{r_j} \tag{3.4}$$

where ψ_n and ψ_m are the Slater wave functions of atoms i and j with quantum numbers n and m and with partial charges q_i and q_j, respectively. Both (3.2) and (3.4) have a finite limit as $r \rightarrow 0$. As a result, these functions are well behaved at small r. A number of methods [56, 57] have been proposed to evaluate (3.4). Hentschke gives an analytical solution [55]:

$$\begin{aligned} J_{ij}(\mathbf{r}) = & \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{\|\mathbf{r_i}-\mathbf{r_j}\|}\frac{4\zeta_i^{2n+1}\zeta_j^{2m+1}}{(2n)!(2m)!}\frac{\partial^{2n-2}\partial^{2m-2}}{\partial\zeta_i^{2n-2}\partial\zeta_j^{2m-2}}\left(\frac{1}{\zeta_i^3\zeta_j^3}\right) \\ & \times\left[1-\frac{(3\zeta_i^2-\zeta_j^2)\zeta_j^4}{(\zeta_i-\zeta_j)^3(\zeta_i+\zeta_j)^3}e^{-2\zeta_i r_{ij}}-\frac{(\zeta_i^2-3\zeta_j^2)\zeta_i^4}{(\zeta_i-\zeta_j)^3(\zeta_i+\zeta_j)^3}e^{-2\zeta_j r_{ij}}\right. \\ & \left.-\frac{\zeta_i\zeta_j^4}{(\zeta_i-\zeta_j)^2(\zeta_i+\zeta_j)^2}r_{ij}e^{-2\zeta_i r_{ij}}-\frac{\zeta_i^4\zeta_j}{(\zeta_i-\zeta_j)^2(\zeta_i+\zeta_j)^2}r_{ij}e^{-2\zeta_j r_{ij}}\right]. \end{aligned} \tag{3.5}$$

Eqn. (3.5) was implemented in a Mathematica ™ program from which C++ code was generated for the analytical computation of J_{ij} and its analytical derivatives with respect to r, which are necessary for computing forces. Due to the nature of Eqn. (3.5), there are many terms with large powers, particularly for $n > 3$. Thus, the equations have to be implemented using the arbitrary precision arithmetic library Class Library for Numbers (CLN). to avoid numerical instabilities. It should be noted that the arbitrary precision library significantly increases the computational cost to analytically solve Eqn. (3.5)

3.1.1.4 Long range Coulomb interactions

It is good practice [58] to use the particle-mesh Ewald [59, 60] method to treat long-range electrostatics interactions in the condensed phase. In short, the method splits the Coulomb potential, either Eqn. (3.1), Eqn. (3.2) or something similar into a short and a long-range part as follows

$$V_{coul}(r_{ij}) \;=\; V_{coul}\,\mathrm{erfc}(\alpha r_{ij}) + V_{coul}\,\mathrm{erf}(\alpha r_{ij}) \tag{3.6}$$

using the fact that the complementary error function {rm erfc} equal 1 - {rm erf}. The first term here is the short-range part, that is computed in real space, and the second term is the long-range part that is computed in Fourier (reciprocal) space by moving charges on a regular grid [60]. The constant α determines how quickly the short-range potential decays to zero, and it can be computed from a user-specified relative error tolerance related to moving charges on a grid. Given the cut-off distance rc and an error tolerance ϵ, typically 10^{-4}, we have

$$\alpha \;=\; \frac{\sqrt{-\ln 2\epsilon}}{rc} \tag{3.7}$$

where ln is the natural logarithm. This shows that α has the dimension of 1 over distance in the units of rc. Fig. 3.1 shows how the division over short and long-range interactions works in practice, and how the long range contribution should be incorporated into simulations using a modified Coulomb function.

3.1.1.5 Polarization

Polarization can be treated explicitly as well in the ACT using the core-shell model [32, 33]

$$V_{pol}(r_{cs}) \;=\; \frac{1}{4\pi\varepsilon_0}\frac{q_s^2}{2\alpha_c}r_{cs}^2, \tag{3.8}$$

where q_s is the shell charge, α_c is the polarizability and r_{cs} the distance between core and shell particles.

3.1.1.6 Induction correction

A term to add extra attraction between atoms was proposed in ref. [61]

$$V_{ic}(r_{ij}) \;=\; -A_{ij}\mathrm{e}^{-b_{ij}r_{ij}} \tag{3.9}$$

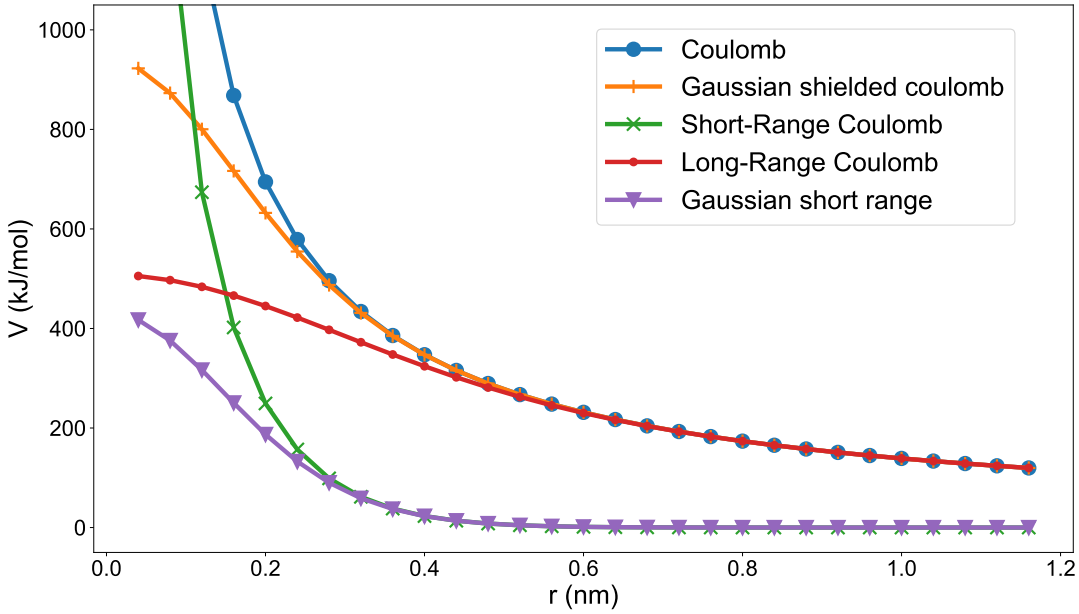

Fig. 3.1: Total Coulomb energy V as a function of distance r for two like unit charges, Gaussian-shielded Coulomb and short and long-range parts of the energy function. Ewald-tolerance ϵ 0.0001, rc 0.9 nm, α 3.24269/nm and Gaussian screening ζ 5/nm.

with A_{ij} and b_{ij} constants to be trained. The geometric combination rule is used for A_{ij} and the arithmetic combination rule for b_{ij}. It is not certain whether the exponential functional form is optimal to describe this interaction and there is no theory behind this.

3.1.1.7 Pauli Repulsion and London Dispersion

The repulsion and dispersion interactions acting between atoms i and j can be described by a number of potentials. In a recent study on noble gases [25] we found that the 14-7 potential due to Halgren [62] was one of the most accurate ones:

$$V_{14-7}(r_{ij}) = \epsilon_{ij}\left(\frac{1+\delta_{ij}}{\frac{r_{ij}}{\sigma_{ij}}+\delta_{ij}}\right)^7\left(\frac{1+\gamma_{ij}}{(\frac{r_{ij}}{\sigma_{ij}})^7+\gamma_{ij}}-2\right) \tag{3.10}$$

where ϵ_{ij} is the well depth at minimum, γ_{ij} and δ_{ij} are dimensionless numbers that were originally shared for all elements. However, in our previous work [25], we treated γ and δ as free atom-specific parameters subject to optimization and combination rules.

Furthermore, the generalized 4-parameter Buckingham potential with an adjustable long-range attraction is implemented as [63]

$$V_{GBH}(r_{ij}) = \epsilon_{ij}\frac{\delta_{ij}+2\gamma_{ij}+6}{2\gamma_{ij}}\frac{1}{1+(\frac{r}{\sigma_{ij}})^6}\left[\frac{6+\delta_{ij}}{\delta_{ij}+2\gamma_{ij}+6}e^{\gamma_{ij}(1-\frac{r}{\sigma_{ij}})}-1\right]-\frac{\epsilon_{ij}}{1+(\frac{r_{ij}}{\sigma})^\delta} \tag{3.11}$$

where γ and δ again are dimensionless constants.

Another potential that is supported is the buffered (Wang) Buckingham potential [64]:

$$V_{WBH}(r_{ij}) = \left(\frac{2\epsilon_{ij}}{1-\frac{3}{\gamma_{ij}+3}}\right)\left(\frac{\sigma_{ij}^6}{\sigma_{ij}^6+r_{ij}^6}\right)\left[\frac{3}{\gamma_{ij}+3}e^{\gamma_{ij}\left(1-\frac{r_{ij}}{\sigma_{ij}}\right)}-1\right] \tag{3.12}$$

where ϵ_{ij} is the well depth at minimum, σ_{ij} is the Van der Waals radius and γ_{ij} determines the steepness of the potential and r_{ij} is the distance between particles i and j. The buffered Buckingham potential mentioned above have been used to develop an accurate phase-transferable model for alkali-halides [19, 20, 21, 22, 23] and the original Buckingham potential [65] is implemented as

$$V_{BH}(r_{ij}) \; = \; A_{ij}\mathrm{e}^{-b_{ij}r_{ij}} - \frac{C_{ij}}{r_{ij}^6} \tag{3.13}$$

where A_{ij}, b_{ij} and C_{ij} are parameters to be trained. An alternative way of formulating the Buckingham potential [66] is

$$E_{MBH} = \frac{\epsilon}{1-\frac{6}{\gamma}}\left[\frac{6}{\gamma}e^{\gamma(1-\frac{r}{\sigma})} - \left(\frac{\sigma}{r}\right)^6\right] \tag{3.14}$$

where ϵ is the well-depth σ reflects the position of the minimum and γ is again a dimensionless constant. Although mathematically identical to Eqn. (3.13) we have shown that the potentials behave differently when combination rules are applied [25].

The well-known Tang-Toennies potential [67, 68] containing five parameters is implemented according to:

$$V_{TT}(x) = Ae^{-bx} - \sum_{n=3}^{5}\left[1 - e^{-bx}\sum_{k=0}^{2n}\frac{(bx)^k}{k!}\right]\frac{C_{2n}}{x^{2n}} \tag{3.15}$$

where repulsion and dispersion terms shared the parameter b. Here, all five parameters can be trained by the ACT. This potential was used in our recent work on noble gases published where we investigated combination rules [25].

In other works, the Tang-Toennies potential is extended with an additional parameter [69] giving the exponential functions a different decay length b:

$$V_{TT2}(x) = Ae^{-b_{exp}x} - \sum_{n=3}^{5}\left[1 - e^{-b_{disp}x}\sum_{k=0}^{2n}\frac{(b_{disp}x)^k}{k!}\right]\frac{C_{2n}}{x^{2n}} \tag{3.16}$$

and the ACT supports both these functions. Van Vleet *et al.* described a more accurate formula for the Pauli repulsion [70] that was applied to derive a force field later [71]. In this model the exchange repulsion is given by the Slater-ISA formula

$$V_{exch} \; = \; A\left(\frac{1}{3}(br)^2 + br + 1\right)\exp^{-br} \tag{3.17}$$

where it can be noted, that no additional parameters are needed for equation~:eq:*slater_isa* compared to the repulsion part in the Tang-Toennies potential (Eqn. (3.15)). In the ACT the Slater-ISA exchange can be combined with the damped dispersion part of the latter potential.

The Lennard-Jones 12-6 potential [72] is available as well:

$$V_{LJ}(r_{ij}) \; = \; 4\varepsilon_{ij}\left(\frac{\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{\sigma_{ij}^{6}}{r_{ij}^{6}}\right) \tag{3.18}$$

and, in addition, the 12-6-4 potential

$$V_{LJ}(r_{ij}) \; = \; 4\varepsilon_{ij}\left(\frac{\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{\sigma_{ij}^{6}}{r_{ij}^{6}}\right) - \frac{\gamma}{r^4} \tag{3.19}$$

which is meant to model ion-dipole interactions is supported too.

It is worth noting that in all these potentials (except Buckingham, Eqn. (3.13)) parameters describe both the exchange and dispersion interactions at the same time, which necessitates simultaneous training of these interaction. To account for anisotropy in exchange, a correction can be implemented using a virtual site particle on the atom that has a σ-hole interacting with other particles [26]:

$$V_{EXCH,Corr}(r_{ij}) = -A_{ij}\mathrm{e}^{-b_{ij}r_{ij}} \tag{3.20}$$

where A_{ij} and b_{ij} are parameters to be optimized. This correction term (Eqn. (3.20)) does not need to be applied to all possible atom pairs, therefore a new combination rule, dubbed "Kronecker" is introduced

$$A_{ij} \;=\; (1-\delta_{ij})\frac{A_i + A_j}{2}$$

where δ_{ij} is the Kronecker delta operating on particle types i and j. This means that only interactions between σ-holes (represented by a virtual site) and atoms are non-zero. To avoid parameter explosion, the atomic A_i are set to zero and only the virtual site A_j is non-zero. This is reasonable since the virtual site represents a property of the σ-hole of the atom it is connected to. The arithmetic combination rule is used for b_{ij}.

Multiple different combination rules can be used for parameters involved in van der Waals interactions, both within ACT and through the interface to OpenMM. For details, see the paper by Kříž *et al.* [25].

3.1.1.8 Treatment of long range dispersion

The different Van der Waals potentials described above can be used with Lennard-Jones PME [59, 73] if treated carefully. LJ-PME assumes a simple dispersion interaction given by

$$V_{LJPME}(r_{ij}) \;=\; -\frac{C_{ij}}{r_{ij}^6} \tag{3.21}$$

for atoms i and j, which differs from the potentials mentioned above. For LJ-PME, it is beneficial to use the geometric combination rule, hence

$$V_{LJPME}(r_{ij}) \;=\; -\frac{C_i C_j}{r_{ij}^6}$$

where C_i and C_j, in contrast to Eqn. (3.21), have units of distance to the third power.

To minimize the difference between the potential used and the LJ dispersion over the whole volume outside the cut-off, and to specify correct inputs to OpenMM, starting from e.g. Eqn. (3.12) we have to solve the following equation:

$$0 \;=\; \int_{rc}^{\infty} 4\pi r_{ij}^2 \left[\left(-\frac{2\epsilon_{ij}}{1-\frac{3}{\gamma_{ij}+3}}\right)\left(\frac{\sigma_{ij}^6}{\sigma_{ij}^6 + r_{ij}^6}\right) + \frac{C_i C_j}{r_{ij}^6}\right] \mathrm{d}r_{ij} \tag{3.22}$$

where we integrate from the cut-off rc to infinity. In this notation, we have to insert the combination rules for ϵ, σ and γ. For the special case that i = j, the constant C_{ii} can be derived using Mathematica ™ as

$$C_{ii} \;=\; \epsilon_i \frac{3+\gamma_i}{\gamma_i} rc^3 \sigma_i^3 \left(\pi - 2\arctan\left[\left(\frac{rc}{\sigma_i}\right)^3\right]\right)$$

which we can then convert back to a σ' compatible with standard Lennard-Jones by

$$\sigma_i' = \left(\frac{C_{ii}}{4\epsilon_i}\right)^{1/6}.$$

Note however, that we need to get effective σ' for all atoms, which means the problem turns into a minimization problem where

$$\varepsilon^2 = \sum_{i,j} \left[\int_{rc}^{\infty} 4\pi r_{ij}^2 \left[\left(-\frac{2\epsilon_{ij}}{1 - \frac{3}{\gamma_{ij}+3}} \right) \left(\frac{\sigma_{ij}^6}{\sigma_{ij}^6 + r_{ij}^6} \right) + \frac{C_i C_j}{r_{ij}^6} \right] \mathrm{d}r_{ij} \right]^2 \tag{3.23}$$

has to be minimized with respect to C_i independently for each set of combination rules. In the Alexandria alkali-halide model we used the Wang-Buckingham potential (Eqn. (3.12)) in conjunction with the combination rules due to Kong [74] in which σ_{ij} depends on all the σ, ϵ and γ. This makes it cumbersome to analytically derive integrals like Eqn. (3.23) for many combinations of potentials and combination rules.

Another problem is, that the parameters C_i that we need to compute the long-range dispersion interaction now depend on the cut-off, which means they should preferably not be computed beforehand. In summary, the C_i should be derived numerically given the potential and combination rules at the start of a simulation.

3.1.1.9 Combination Rules for Van der Waals Potentials

In a recent paper [25] we studied the effect of combination rules on potentials for noble gases and introduced two new combination rules (Eqn. (3.34) and Eqn. (3.35)). Here is a brief summary, copied (with permission) from that paper.

Combination rules reduce the number of parameters required for the pair-wise potentials introduced above because the parameters describing the interaction between dissimilar X–Y atoms are reconstructed from parameters of homodimers X–X and Y–Y. In this way, it is necessary only to fit atomic parameters on data for homodimers. Different sets of mathematical expression were considered, as detailed below.

The two simplest expressions that have historically been used as combination rules are the geometric [75] and arithmetic [76] averages, respectively:

$$X_{12} = \sqrt{x_1 x_2} \tag{3.24}$$

%was used for all three parameters ϵ, γ, σ in "geometric" rules. It is also used for ϵ in "arithmetic" rules and for σ in "Kong-Mason" rules [74].

$$X_{12} = \frac{x_1 + x_2}{2} \tag{3.25}$$

where x_1 and x_2 are the atomic parameters. Both these rules can in principle be applied to all parameters in the Van der Waals potentials described above and the rules are well-behaved mathematically.

Hogervorst introduced a set of combination rules for 12-6 Lennard-Jones and exp-6, modified Buckingham, potential (Eqn. (3.14)) [77]. He proposed using Eqn. (3.26) for ϵ for both potential forms. For the σ of 12-6 potential (Eqn. (3.18)), the arithmetic mean (Eqn. (3.25)) was used. In the case of the modified Buckingham potential (eqn. (3.14)), expression~:eq:*cr_sigma5*, which depends on the combined parameters $\epsilon_{1,2}$ and $\gamma_{1,2}$ was advocated for the σ, along with arithmetic mean for γ.

$$X_{12} = \frac{2x_1 x_2}{x_1 + x_2} \tag{3.26}$$

$$\sigma_{12}^6 = \sqrt{\frac{\epsilon_1 \gamma_1 \sigma_1^6}{\gamma_1 - 6} \frac{\epsilon_2 \gamma_2 \sigma_2^6}{\gamma_2 - 6}} \frac{(\gamma_{1,2} - 6)}{\gamma_{1,2} \epsilon_{1,2}} \tag{3.27}$$

Where the $\gamma^{1,2}$ used Eqn. (3.25) and $\epsilon^{1,2}$ Eqn. (3.26). It should be noted that equation (3.26) is ill-behaved if both x_1 and x_2 are zero, while Eqn. (3.27) is ill-behaved if either $\gamma_{1,2}$ or $\epsilon_{1,2}$ is zero. Yang *et*

al. [78] introduced an expression for the Morse potential [79], using Eqn. (3.26) for ϵ, while eqn. (3.28) is used for σ and γ.

$$X_{12} = \frac{x_1 x_2 (x_1 + x_2)}{x_1^2 + x_2^2} \tag{3.28}$$

The expression for γ proposed by Mason [80] for the exp-6 potential has the form %red{while used other rels for epsilon, sigma in alexandria}

$$\gamma_{12} = \sqrt{\sigma_1 \sigma_2} \left(\frac{\gamma_1}{2\sigma_1} + \frac{\gamma_2}{2\sigma_2} \right) \tag{3.29}$$

Waldman and Hagler [81] introduced expressions (3.30) for ϵ and (3.31) for σ
to reproduce experimental well-depths and interaction distances.

$$\epsilon_{12} = \sqrt{\epsilon_1 \epsilon_2} \frac{2\sigma_1^3 \sigma_2^3}{\sigma_1^6 + \sigma_2^6} \tag{3.30}$$

$$X_{12} = \left(\frac{X_1^6 + X_2^6}{2} \right)^{1/6} \tag{3.31}$$

Although Eqn. (3.31) was devised for σ we have evaluated it for other parameters here as well, hence the notation with X. Qi and coworkers advocated the use of buffered 14-7 Lennard-Jones potential (Eqn. (3.10)) due to Halgren [62], alongside combination expressions (3.30) for ϵ and (3.32) for σ: [82]

$$X_{12} = \frac{x_1^3 + x_2^3}{x_1^2 + x_2^2} \tag{3.32}$$

A further relation, the harmonic mean rule, was proposed by Halgren [62]:

$$X_{12} = \frac{4 x_1 x_2}{(x_1^{1/2} + x_2^{1/2})^2} \tag{3.33}$$

Finally, we introduce two new combination rules that we have not seen published previously. Since the σ in most potentials can be interpreted as a Van der Waals radius, we introduced a relation averaging third powers, corresponding to an atomic volume [25]:

$$\sigma_{12} = \left(\frac{\sigma_1^3 + \sigma_2^3}{2} \right)^{1/3}. \tag{3.34}$$

In addition, we applied the following rule for in particular ϵ since it yield an X_{12} that is smaller than the geometric one (Eqn. (3.24)):

$$X_{12} = \left(\frac{2}{x_1^{-2} + x_2^{-2}} \right)^{1/2} \tag{3.35}$$

The combination relations described above were permuted with each other into new combination rules. In this way, relations that depend on only one parameter type were used for any parameter. Relations depending on multiple parameters were used only for the specific parameter type combination they depend on (e.g. Eqn. (3.30) was only used for ϵ, using homodimer ϵ and σ). In our previous work on alkali halides [19] we used combination rules according to eqn. (3.27) for σ, eqn. (3.26) for ϵ and eqn. (3.25) for γ with the Wang-Buckingham potential (Eqn. (3.12)).

A number of these combination rules can be written using the generalized mean equation [83]

$$M_p(x_1, x_2, ..., x_N) = \left(\frac{1}{N} \sum_{i=1}^{N} x_i^p \right)^{1/p} \tag{3.36}$$

for $p \neq 0$. For $p = 0$, Eqn. (3.36) turns into the geometric rule. Table 3.1 lists other well-known combination rules and their respective exponent p. Hohm also describes combinations of the generalized mean and other similar expressions, but the most important observation he made was that the exponent p can be varied at will [83]. In the ACT it is possible to train this parameter along with the Van der Waals parameters.

Table 3.1: Correspondence between well-known combination rules and the generalized mean equation (3.36).

p	Name	Equation
-2	Inverse Square	(3.35)
-1	Hogervorst ϵ	(3.26)
-1/2	Harmonic Mean	(3.33)
0	Geometric	(3.24)
1	Arithmetic	(3.25)
3	Volumetric	(3.34)
6	Waldman-Hagler σ	(3.31)

3.1.2 Bonded interactions

3.1.2.1 Harmonic potential

Bond vibrations can be described using a harmonic term based on the bond length r_{ij}

$$V_b(r_{ij}) = \frac{k^b_{ij}}{2}\left(r_{ij} - r^0_{ij}\right)^2, \tag{3.37}$$

where k^b_{ij} is the force constant, and r^0_{ij} is the equilibrium bond length.

3.1.2.2 Morse potential

A Morse potential [79] can be used, with one addition:

$$V_M(r_{ij}) = D^e_{ij}\left[\mathrm{e}^{-2\beta_{ij}(r_{ij} - r^0_{ij})} - 2\mathrm{e}^{-\beta_{ij}(r_{ij} - r^0_{ij})}\right] + D^0_{ij} \tag{3.38}$$

The term D^e_{ij} roughly corresponds to a dissociation energy, however since there are Coulomb and/or Buckingham interactions between the atoms as well, the total "bond" potential is given by the sum of three terms and a correction term D^0_{ij} is needed to get the correct energy minimum.

3.1.2.3 Wei-Hua potential

In a very recent study we found the potential due to Hua [84, 85] to be the best compromise between accuracy of the vibrational frequencies and computational cost [27]. It is given by:

$$U(r) = D_e\left(\left[\frac{1 - e^{-b(r-r_e)}}{1 - ce^{-b(r-r_e)}}\right]^2 - 1\right) \tag{3.39}$$

where D_e is the well-depth, r_e the equilibrium bond length, and b and c are constants with $\|c\| < 1$.

3.1.2.4 Angle potential

Angle vibrations are described using a harmonic term based on the angle θ_{ijk}

$$V_a(\theta_{ijk}) \;=\; \frac{k^{\theta}_{ijk}}{2}\left(\theta_{ijk} - \theta^{0}_{ijk}\right)^2, \tag{3.40}$$

where k^{θ}_{ijk} is the force constant, and θ^{0}_{ijk} is the equilibrium angle.

3.1.2.5 Angle potential for linear compounds

The reference position, corresponding to a minimum energy structure, $\mathbf{x}^0_j$ for a central atom j in a linear triplet of atoms i, j, k is given by

$$\mathbf{x}^0_j \;=\; \mathrm{a}\,\mathbf{x}_i + (1 - \mathrm{a})\,\mathbf{x}_k \tag{3.41}$$

where a is a constant defined by the bond-lengths $i - j$ and $j - k$. In a group with bonds $i - j$ and $j - k$ with lengths b_{ij} and b_{jk} respectively, the constant is

$$a \;=\; \frac{b_{jk}}{b_{ij} + b_{jk}}. \tag{3.42}$$

If the order of atoms is flipped a will change to $1 - a$. The potential V_{lin} is then given by

$$V_{lin} \;=\; \frac{\mathrm{k}_{lin}}{2}\left(\mathbf{x}_j - \mathbf{x}^0_j\right)^2 \tag{3.43}$$

with k $_{\mathrm{lin}}$ the force constant [24].

3.1.2.6 Out-of-plane vibrations

Finally, out-of-plane vibrations are treated by another harmonic potential

$$V_i(\phi_{ijkl}) \;=\; \frac{k^{\phi}_{ijkl}}{2}\phi^2_{ijkl}, \tag{3.44}$$

where k^{ϕ}_{ijkl} is the force constant and ϕ_{ijkl} is defined by the angle between the two planes i, j, k and j, k, l. This potential was historically termed *improper dihedral*.

3.1.2.7 Torsion potential

A torsion potential is implemented using a Fourier series:

$$V_d(\phi_{ijkl}) \;=\; \sum_{n=0}^{5} c_n \cos^n(\pi + \phi_{ijkl}) \tag{3.45}$$

where c_n are constants and the torsion angle is defined as above. The constant π is added to be compatible with the Ryckaert-Bellemans potential [86] that is implemented in simulation codes like GROMACS [87] and OpenMM [88].

3.1.2.8 Proper dihedral

A simpler torsion potential is implemented for backward compatibility as:

$$V_d(\phi_{ijkl}) \;=\; \cos(n\phi_{ijkl} + \phi_0) \tag{3.46}$$

where n is the multiplicity (number of minima in 360 degrees) ϕ_0 is an offset angle.

3.1.3 Special potentials

The ACT includes a flat-bottom position restraint potential according to

$$V_{fbpr}(r) = \begin{cases} 0 & r \leq r_0 \\ \frac{k}{2}(r - r_0)^2 & r > r_0 \end{cases} \tag{3.47}$$

where k is the force constant and r_0 the radius of the sphere (centered at the origin) in which the potential is zero. The flat-bottom potential is activated by flags to the *alexandria simulate* command. It is useful mainly to keep molecules close to the origin and prevent them from flying into outer space.

3.1.4 Virtual Sites

A virtual site is an extra point, located at a defined position in a molecule. A variety of virtual site options is currently implemented within the ACT framework:

- a virtual site on top of an atom (VSITE1) [89]
- a virtual sites along the bond (VSITE2) for the description of anisotropic charge distribution and exchange [26] such as encountered in σ holes
- a virtual site on the bisector of a angle, like in the TIP4P water model [52] (VSITE3S) or in alcohol (VSITE3)
- off-plane virtual sites for modeling lone-pairs in sp^3 hybridized compounds, such as water (VSITE3SOUT, symmetric) or asymmetric for compounds like alcohols [26, 90].
- Four-particle virtual sites (VSITE4) to model a lone-pair on an amine group.

3.1.5 Total energy

The total energy E of a compound then follows from

$$E = V_{vdw}(r_{ij}) + V_{coul}(r_{ij}) + V_{pol}(r_{cs}) + V_b(r_{ij}) + V_a(\theta_{ijk}) + V_i(\phi_{ijkl}) + V_d(\phi_{ijkl}). \tag{3.48}$$

Finally, it should be noted that the number of excluded neighbors is user-configurable. That means that atoms that are covalently bonded can interact both through the Buckingham (or Lennard-Jones) and Coulomb potentials, and through the bonded potentials. The main reason for this is that the short-range Coulomb interactions yield polarization anisotropy that is difficult to reproduce by a non-interacting model. To make sure that the forces on the atoms in a molecule are zero in the reference minimum-energy structure from quantum chemistry, both bond lengths r^0_{ij} and angles θ^0_{ijk} can be treated as free parameters, that may differ substantially from the reference geometry. The number of exclusions can be selected separately for Coulomb and Van der Waals forces.

In total there are up to seven "atom" parameter types (ϵ, σ, γ, δ, ζ, q_s, and α) and 7-14 "bond" parameter types (k^b, D^e, r^0, r^{max}, k^θ, θ^0, k^{lin} and c_n) where n is the dihedral term index running from 0 to 6 to determine. All the atomic parameters are taken to be hybridization state dependent (corresponding to, for instance, sp ^{1}, sp ^{2} and sp ^{3} carbon atoms). %It is straightforward to add support for other potentials, such as proper dihedrals.

3.2 Force Field Training Algorithms

This chapter was written by Juli{'a}n Marrades and is based in part on his M.Sc. thesis :cite:p:`Marrades2022a`.

Within the *alexandria train_ff* module of the Alexandria Chemistry Toolkit you can choose among three algorithms to optimize force field parameters:

- Markov Chain Monte Carlo (flag *-optimizer MCMC*)
- Genetic Algorithm (flag *-optimizer GA*)
- Hybrid GA/MCMC (flag *-optimizer HYBRID*)

Let us see what they do behind the scenes and how to control them.

3.2.1 MCMC

Assume we want to set the values for five parameters (nParam = 5) by performing ten MCMC iterations (flag *-maxiter 10*). Then, the code "reads"

```
for (int i = 0; i < 10; i++)
  for (int j = 0; j < 5; j++)
    stepMCMC();
```

That is, we do 10 X 5 = 50 MCMC steps. What is a MCMC step though? In essence,

- *prevDev*: previous deviation from data
- Choose a parameter and alter it
- *newDev*: new deviation from data
- If *newDev* < *prevDev*, we accept the parameter change and continue into the next step. Else, apply Metropolis Criterion. That is, with some probability we accept the change. Otherwise, we restore the parameter to its previous value and proceed into the next step.

Now, let us add some more detail to the algorithm.

1. we already have the deviation from data of the previous step in *prevDev*.
2. we arbitrarily choose a parameter out of the *param* vector, say *param[i]*, which is bounded by its maximum *pmax[i]* and its minimum *pmin[i]*, yielding the range *prange[i] = pmax[i] - pmin[i]*. Here is where the *-step [0, 1]* flag comes into play by specifying a fraction.\
3. we draw a value of *delta*, which is uniformly distributed in *[- step * prange[i], + step * prange[i]]* and add it to the parameter value *param[i] += delta*. If the parameter has gone beyond its maximum, we set its value to the maximum. Same goes for the minimum.
4. we compute the deviation *newDev* from data of the modified parameter vector. If *newDev* < *prevDev*, we accept the change and finish the MCMC step. Otherwise, we apply the Metropolis Criterion and finish the step. Then go back to 1.

3.2.1.1 Metropolis Criterion

If Mathematics is your thing and you {em really} want to know the nitty-gritty stuff, you have thorough explanations of this method on href{https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm}{Wikipedia} and Shuyi Qin's Master thesis [91]. Here, we shall say that the Metropolis Criterion allows us to take steps that do not get us closer to a minimum, giving the opportunity to explore the parameter space and avoid local minima. How exactly?

The probability of accepting a "bad" parameter change is controlled by the flag *-temp T* flag and follows the equation

$$prob = exp(-deltaDev/T)$$

where deltaDev = newDev - prevDev. Note that the unit of temperature is the same as that of the deviation.

Given deltaDev, a higher temperature gives a higher probability of acceptance, as *-deltaDev/T* tends to *0* and *exp(0) = 1*. On the other hand, a lower T gives less chances of acceptance, as *-deltaDev/T* tends to $-\infty$ and $exp(-\infty) = 0$.

ACT gives us the possibility to lower the temperature (simulated annealing) during the MCMC optimization with the flag *-anneal [0, 1]* flag. If we use flag *-anneal 0.5*, the temperature will be flag *-temp T* until 50% of the iterations have been completed. Then, it will be linearly decreased until it reaches *0* on the last iteration.

There are two remarks to be made here:

- The temperature is kept constant during the *nParam* MCMC steps that take place for a given iteration.
- Since division by *0* is not defined, we set $T = 1e\text{-}6$ on the last iteration.

Fig. 3.2 shows a schematic example of the temperature over time when we use actflag{-maxiter 10 -temp 5 -anneal 0.5}.

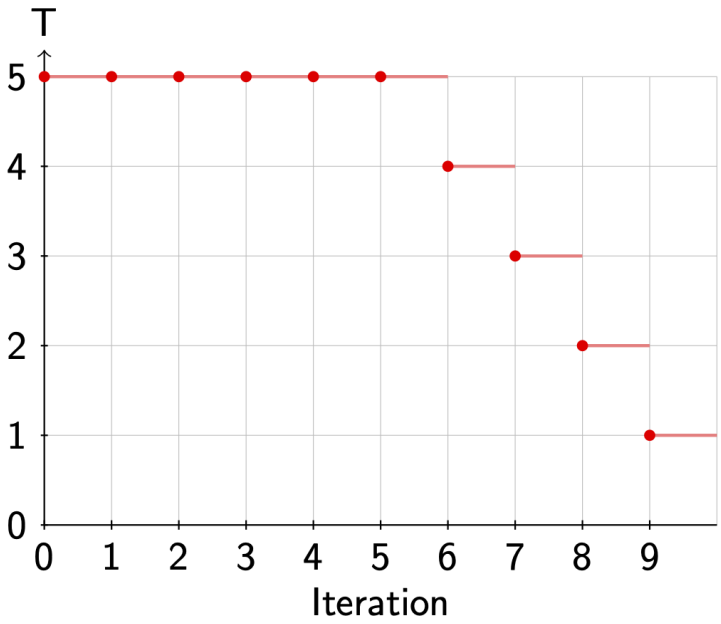

Fig. 3.2: Annealing during a MCMC run.

3.2.1.2 Multiple MCMC runs

We can optimize several candidate solutions in parallel using the *-pop_size* flag. If the *-random_init* flag is set (default), each candidate solution will be initialized arbitrarily and respecting parameter ranges. If *-norandom_init* is employed, the candidate solution(s) will be initialized as specified by the force field file(s) provided by the user.

3.2.2 Genetic Algorithm

Quoting WikipediaGA

> In computer science and operations research, a genetic algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate

high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.

In this section, we describe our implementation of a genetic algorithm for force field parameterization. If this is the first time you hear about genetic algorithms and want to acquaint yourself, we recommend you to read Chapter 2 of Steven F. van Dijk's PhD thesis [92] and/or head over to Youtube.

Our implementation is based upon a class definition and an external function for computing random numbers (Listing *Class definition.*).

Listing 3.1: Class definition.

```
class Genome
{
    double params[];
    double deviation;
    double probability;
};

// Returns a sample of a random variable that is
// uniformly distributed in [0, 1]
double random();
```

then the genome evolution boils down to the code in Listing *Schematic of the evolution algorithm*

Listing 3.2: Schematic of the evolution algorithm

```
void evolve(int popSize, int nElites, int prCross, int prMut)
{
    Genome oldPop[popSize] = initialize(popSize); // Initialize
    Genome newPop[];
    computeDeviation(oldPop); // Deviation from data
    int generation = 0;
    do
    {
        sort(oldPop); // Sorting
        if (penalize(oldPop, generation)) // Penalty?
        {
            // Recompute deviation and sort again
            computeDeviation(oldPop);
            sort(oldPop);
        }
        generation++;
        computeProbabilities(oldPop); // Selection probabilities
        // Elitism
        for (int i = 0; i < nElites; i++)
            newPop.add(oldPop[i]);
        // Rest of population
        for (int i = nElites; i < popSize; i += 2)
        {
            Genome parent1, parent2 = select(oldPop); // Selection
            Genome child1, child2;
```

(continues on next page)

(continued from previous page)

```
            if (random() <= prCross) // Crossover?
                child1, child2 = crossover(parent1, parent2);
            else
                child1, child2 = parent1, parent2;
            // Mutation
            for (Genome child : {child1, child2})
            {
                mutate(child, prMut);
                newPop.add(child); // Add to new population
            }
        }
        computeDeviation(newPop); // Deviation from data
        oldPop = newPop; // Swap populations
        newPop.clear();  // Erase all genomes in the population
    }
    // Termination
    while (!terminate(oldPop, generation));
}
```

flag *popSize* and flag *nElites* are assumed to be even. Let us explore the different stages of the process.

3.2.2.1 Initialization

This stage fills the *params* field in the *Genome* class, generating popSize (flag *-pop_size*) genomes.

If we are using flag *-norandom_init*, the genomes will be initialized as specified by the force field file(s) provided. If we are employing flag *-random_init*, each genome will be initialized by setting a random value for each parameter, uniformly distributed over the allowed range.

3.2.2.2 Deviation from data

Here we fill the *deviation* field in for each *Genome* in the population by computing the deviation from the dataset.

3.2.2.3 Sorting

Sorting is not a mandatory step but may be required depending on the GA components selected by the user. We sort the population in ascending order of *deviation*. Whether we sort or not is controlled by the flag *-sort* flag.

3.2.2.4 Penalties

At this stage, we may alter the population if certain conditions are met, with the main goal of preventing premature convergence and enforcing solution diversity.

> Covering such a small portion of the space you are. Broaden your search, you should. - Yoda

To that end, we have a function *penalize()* which returns *true* if the population was penalized and *false* otherwise.

For now, there are two components in this function:

- **Volume.** This option enables flag *-sort*. If the volume spanned by the population divided by the total volume of the parameter space is smaller than flag *-vfp_vol_frac_limit [0, 1]*|, the {em worst} fraction flag *-vfp_pop_frac [0, 1]* of genomes in the population will be randomly reinitialized. If flag *-log_volume* is used, volumes will be computed in logarithmic scale. * But wait, then the volume could be negative! Yes, we have to fix that!*

  Death comes equally to us all, and makes us all equal when it comes. - John Donne

- **Catastrophe.** Each flag *-cp_gen_interval* generations, a fraction flag *-cp_pop_frac [0, 1]* of the genomes in the population will be randomly reinitialized. Genomes to reinitialize are arbitrarily chosen.

3.2.2.5 Selection probabilities

We provide three options for computing selection probabilities:

1. Rank (flag *-prob_computer RANK*)
2. Fitness (flag *-prob_computer FITNESS*)
3. Boltzmann (flag *-prob_computer BOLTZMANN*)

The sum of the selection probabilities of the genomes in the population, is 1.

3.2.2.6 Rank

This option enables flag *-sort*. The selection probability depends exclusively on the index (rank) of genome in the population (Listing *Calculation of the probability from the order of probabilities.*).

Listing 3.3: Calculation of the probability from the order of probabilities.

```
for (int i = 0; i < popSize; i++)
{
  oldPop[i].probability = (popSize - i) / (popSize * (popSize + 1) / 2);
}
```

That is, the lower the index, the higher the probability of being selected. The independence of the *deviation* avoids the possible phenomena of a genome with a very high selection probability dominating the population.

3.2.2.7 Fitness

The selection probability is inversely proportional to the *deviation* (Listing *Calculation of the probability from the deviations from data.*).

Listing 3.4: Calculation of the probability from the deviations from data.

```
double total = 0;
double inverses[popSize];
double epsilon = 1e-4;
for (int i = 0; i < popSize; i++)
```

(continues on next page)

(continued from previous page)

```
    inverses[i] = 1 / (epsilon + oldPop[i].deviation);
    total += inverses[i];
for (int i = 0; i < popSize; i++)
    oldPop[i].probability = inverses[i] / total;
```

3.2.2.8 Boltzmann

The *temperature* parameter is specified by the flag *-boltz_temp* flag and controls the smoothing of the selection probabilities (Listing *Use of Boltzmann-weighting when calculating the probability*). A higher value will avoid polarization in the probability values and vice versa. The flag *-boltz_anneal* flag allows us to decrease the temperature over time and has the same logic as flag *-anneal*, except that it targets the Boltzmann selection temperature and operates based on the maximum amount of generations.

Listing 3.5: Use of Boltzmann-weighting when calculating the probability

```
double total = 0;
double exponentials[popSize];
double epsilon = 1e-4;
for (int i = 0; i < popSize; i++)
    exponentials[i] = exp(1 / (epsilon + oldPop[i].deviation) / temperature);
    total += exponentials[i];
for (int i = 0; i < popSize; i++)
    oldPop[i].probability = exponentials[i] / total;
```

3.2.2.9 Elitism

In order to avoid losing the best candidate solutions found so far, the GA will move the top nElites flag *-n_elites* genomes, {em unchanged}, into the new population. That means, the genome will not undergo crossover nor mutation.

When flag *-n_elites* > *0*, flag *-sort* will be enabled.

3.2.2.10 Selection

Once the selection probabilities are computed, the population becomes a href{https://en.wikipedia.org/wiki/Probability_density_function}{probability density function} from which we can sample genomes based on their *probability*.

As of now, only a vanilla selector is available. It only looks at the probability and can select the same genome to be *parent1* and *parent2*.

3.2.2.11 Crossover

With certain probability *prCross*, controlled by the flag *-pr_cross* flag, two parents will combine their parameters to form two children. Right now, only an N-point crossover is available, where N is defined by the flag *-n_crossovers* flag.

If $N = 1$, we arbitrarily select on crossover point (v) and:

```
   v                        v
|X|X|X|X|X|X|X|X|  --> |X|X|Y|Y|Y|Y|Y|Y|
|Y|Y|Y|Y|Y|Y|Y|Y|  --> |Y|Y|X|X|X|X|X|X|
```

If $N = 2$, we arbitrarily select two crossover points (v) and:

```
  v     v                    v     v
|X|X|X|X|X|X|X|X|  --> |X|Y|Y|Y|X|X|X|X|
|Y|Y|Y|Y|Y|Y|Y|Y|  --> |Y|X|X|X|Y|Y|Y|Y|
```

If $N = 3$, we arbitrarily select three crossover points (v) and:

```
      v     v     v           v     v     v
|X|X|X|X|X|X|X|X|  --> |X|X|Y|Y|Y|X|X|X|Y|
|Y|Y|Y|Y|Y|Y|Y|Y|  --> |Y|Y|X|X|X|Y|Y|Y|X|
```

If $N = 4 \ldots$ you get the idea (hopefully).

3.2.2.12 Mutation

Given the mutation probability *prMut*, controlled by flag *-pr_mut*, we iterate through the parameters. If the probability is met, we alter the parameter, otherwise, we leave it unchanged.

```
for (int i = 0; i < nParams; i++)
{
  if (random() <= prMut)
  {
    changeParam(params, i);
  }
}
```

The parameter is changed in the same way as in MCMC, by a fraction of its allowed range and not allowing values outside of it. The fraction in this case is controlled by the *-percentage* flag, which has the same meaning as *-step*, but applies to GA instead of MCMC.

3.2.2.13 Termination

This stage decides whether the GA evolution should continue or halt. We allow the user to tweak the termination criteria with several flags:

1. flag *-max_generations*: evolution will halt after so many generations.
2. flag *-max_test_generations* (disabled by default): evolution will halt if in the last flag *-max_test_generations* the best *deviation* of the test set found so far has not improved.

3.2.3 HYBRID

Even though this optimizer has a very fancy name, it is just a GA with MCMC as its mutator engine. When MCMC acts as a mutator, it will always alter the genomes independently of flag *-pr_mut*. Also, it is important to note that the simulated annealing by default is applied independently in each MCMC run. For instance, in case we would use flag *-max_generations 2 -maxiter 10 -temp 5 -anneal 0.5*, Fig. 3.3 shows the temperature during the MCMC part.

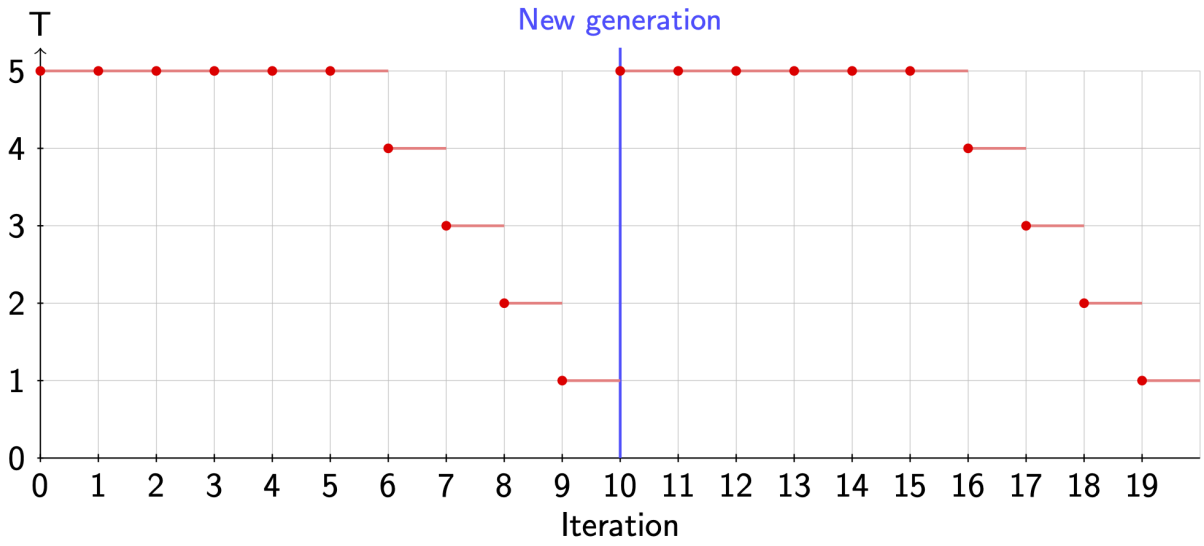

Fig. 3.3: Annealing in the hybrid algorithm

However, when using the flag flag *-anneal_globally*, the starting temperature of the annealing will be decreased in steps at the beginning of each generation.

3.3 Force field design

3.3.1 Physical Background

The Alexandria force-fields have a functional form consisting of van der Waals (*vdw*), electrostatics (*coul*), polarization (*pol*) and bonded terms including a radial (*b*), angular (*a*), out-of-plane dihedral (*i*) and torsion (*d*) terms.

$$E = V_{vdw}(r_{ij}) + V_{coul}(r_{ij}) + V_{pol}(r_{cs}) + V_b(r_{ij}) + V_a(\theta_{ijk}) + V_i(\phi_{ijkl}) + V_d(\phi_{ijkl}) \tag{3.49}$$

where r_{ij} refers to the distance between atoms i and j, r_{cs} to the distance between a core and a shell (Drude) particle, θ_{ijk} to the angle given by three atoms i, j and k, and ϕ_{ijkl} to a torsional angle between the planes given by atoms i, j, k and atoms j, k, l. For each of the terms, multiple functional forms are available, so that within the Alexandria framework, different force-fields, including previously published ones, can be reconstructed and compared to one another in a systematic manner [6]. For details on these potentials we refer to Chapter *Energy Function*.

In total there are seven *atom* parameter types and 7-14 *bond* parameter types. A variety of virtual sites can be used, like those used in water models [34, 93] or to model anisotropy due to σ-holes on halogen atoms or water [26].

3.3.2 Determining atom types

When importing structure files (SDF, PDB, or XYZ), ACT uses RDKit for the initial chemical perception, including sanitization and aromaticity assignment [94].

Attention

If bond information is present in the PDB or SDF file, it will be used by ACT. That means the user is responsible for providing correct information, unless the compound is a special case, see below. If

there is no bond information in the PDB (or a XYZ) file, it will be generated using RDKit. There is no guarantee that the output from RDKit is correct in all cases, however. Please check your output.

Atom typing in ACT is based on SMARTS patterns stored in *share/atom_bond.xml* [95]. The distributed *atom_bond.xml* file assigns atom types for carbon, nitrogen, oxygen, phosphorus, and sulfur according to their hybridization state, adding 1,2,3 behind the element symbol. For all other elements, atom types are assigned based on formal charge. For neutral atoms, the default atom type is the lowercase element symbol.

The file also contains special cases, such as resonance structures or situations where RDKit assigns bond orders or hybridizations that differ from those required by the ACT force fields. For example, sulfate (SO_4^{2-}) is assigned with one sulfur as s3 and four oxygens with mixed types (o- and o2), which does not reflect the resonance equivalence of the oxygens. These special rules are typically placed at the top of the file.

You may provide your own *atom_bond.xml*. ACT always checks the current working directory first (the filename must be exactly *atom_bond.xml*), and if no file is found there, it falls back to the shared directory. The file is ordered from most specific to most general rules (see Table 3.2 and Table 3.3).

ACT provides two Jupyter notebooks to assist with inspecting and developing SMARTS patterns:

- *atom_bond_demo.ipynb*: Provide a specific molecule to obtain the assigned bonds and atom types, together with a labeled 3D visualization.
- *test_smarts.ipynb*: Provide a molecule and a SMARTS pattern and the notebook returns the matching atoms and a 2D depiction with highlighted hits.

Users are strongly encouraged to verify that the chemical interpretation is appropriate for their specific use case. If you need assistance creating new SMARTS patterns, please open an issue on GitHub.

The code to determine atomtypes is implemented in both the C++ code and in a Python file *get_mol_dict.py* that is part of the ACT distribution. The latter can be used in conjunction with the generating quantion chemistry data in the SaptACT repository.

Table 3.2: Special-case SMARTS patterns used by ACT

Molecule/Rule	SMARTS / recursive SMARTS	atom_type(s)	q	bo
sulfate	[#16](=[#8])(=[#8])(-[#8-])-[#8-]	s3, o2, o2, o2, o2	-2	1.5
phosphate2	[#8-]-[#15](-[#8-])(-[#8-])=[#8]	o3, p3, o3, o3, o3	-3	1.5
phosphate	[#15D4]([#8D1])([#8D1])([#8-,#8D1])([#8-,#8D1])	p3, o3, o3, o3, o3	-3	1.5
phosphate2 (variant 2)	[#8-]-[#15](-[#8-])(-[#8-])	o3, p3, o3, o3	-2	1.5
carboxylate	[#6X3](-[#8-])=[#8]	c2, o2, o2	-1	1.5
phosphate3	[#8-]-[#15](-[#8-])	o3, p3, o3	-1	1
nitro1	[#7+](-[#8-])=[#8]	n2, o2, o2	0	1.5
nitro2	[#7+](-[#8])=[#8]	n2, o2, o2	0	1.5
water	[#8](-[#1])-[#1]	ow, hw, hw	0	1

Table 3.3: Atom type SMARTS patterns used by ACT

Atom type rule	SMARTS / recursive SMARTS	atom_typ
gaff h5 ring 2EWG	[#1X1;$(*-[c](~[#7,#8,#16,#17,#35,#53])~[#7,#8,#16,#17,#35,#53])]	h5
gaff h4 ring 1EWG	[#1X1;$(*-[c]~[#7,#8,#16,#17,#35,#53])]	h4
gaff h3 chain 3EWG	[#1X1;$(*-[C]([#7,#8,#16,#17,#35,#53,F,Cl, | h3 | 0	
gaff hc tertcarbon like	[#1X1;$(*-[#6X4]([#6])([#6])[#1])]	hc
gaff h2 C eq 2EWG	[#1X1;$(*-[C](=[#7,#8,#9,#16])([#7,#8,#9,#16]))]	h2
gaff h2 chain 2EWG	[#1X1;$(*-[C]([#7,#8,#9,#16,#17,#35,#53])[#7,#8,#9,#16,#17,#35,#53])]	h2
gaff h1 chain 1EWG	[#1X1;$(*-[C]-[#7,#8,#9,#16,#17,#35,#53])]	h1
gaff h1 chain 1EWG double	[#1X1;$(*-[C]=[#7,#8,#9,#16])]	h1
gaff ha conjugated CX3 eq C	[#1X1;$(*-[#6X3]=[#6])]	ha
gaff ha triplebond	[#1X1;$(*-[#6X2]#[#6])]	ha
gaff hc aliphatic C	[#1X1;$(*-[#6X4])]	hc
hcl	[#1X1;$(*-Cl)]	hcl
hbr	[#1X1;$(*-Br)]	hbr
gaff ho oxygen	[#1X1;$(*-O)]	ho
gaff hn nitrogen	[#1X1;$(*-N)]	hn
gaff hn aromatic n	[#1X1;$(*-n)]	hn
gaff hs sulfur	[#1X1;$(*-S)]	hs
gaff hp phosphorus	[#1X1;$(*-P)]	hp
hf	[#1X1;$(*-F)]	hf
hi	[#1X1;$(*-I)]	hi
gaff ha generic	[#1X1]	ha
gaff hc any	[#1]	hc
p sp	[#15;^1]	p1
p sp2	[#15;^2]	p2
p sp3	[#15;^3]	p3
p sp3 sp3d	[#15;^4]	p3
p sp3 sp3d2	[#15;^5]	p3
s sp	[#16;^1]	s1
s sp2	[#16;^2]	s2
s sp3	[#16;^3]	s3
s sp3 sp3d	[#16;^4]	s3
s sp3 sp3d2	[#16;^5]	s3
c sp	[#6;^1]	c1
c sp2	[#6;^2]	c2
c sp3	[#6;^3]	c3
c sp3 sp3d	[#6;^4]	c3
c sp3 sp3d2	[#6;^5]	c3
n sp3 +1	[#7+;^3]	n4
n sp	[#7;^1]	n1
n sp2	[#7;^2]	n2
n sp3	[#7;^3]	n3
n sp3 sp3d	[#7;^4]	n3
n sp3 sp3d2	[#7;^5]	n3
o sp	[#8;^1]	o1
o sp2	[#8;^2]	o2
o sp3	[#8;^3]	o3

Table 3.3 – continued from previous page

Atom type rule	SMARTS / recursive SMARTS	atom_typ
o sp3 sp3d	[#8;^4]	o3
o sp3 sp3d2	[#8;^5]	o3
li +1	[Li+]	Li+
li 0	[Li]	li
na +1	[Na+]	Na+
na 0	[Na]	na
k +1	[K+]	K+
k 0	[K]	k
rb +1	[Rb+]	Rb+
rb 0	[Rb]	rb
cs +1	[Cs+]	Cs+
cs 0	[Cs]	cs
be +2	[Be+2]	Be2+
be 0	[Be]	be
mg +2	[Mg+2]	Mg2+
mg 0	[Mg]	mg
ca +2	[Ca+2]	Ca2+
ca 0	[Mg]	mg
al +3	[Al+3]	Al3+
al 0	[Al]	al
si -4	[Si-4]	Si4-
si +4	[Si+4]	Si4+
si 0	[Si]	si
p -3	[P-3]	P3-
p +3	[P+3]	P3+
p +5	[P+5]	P5+
f -1	[F-]	F-
cl -1	[Cl-]	Cl-
cl +1	[Cl+]	Cl+
cl +3	[Cl+3]	Cl3+
cl +5	[Cl+5]	Cl5+
cl +7	[Cl+7]	Cl7+
cl 0	[Cl]	cl
br -1	[Br-]	Br-
br +1	[Br+]	Br+
br +3	[Br+3]	Br3+
br +5	[Br+5]	Br5+
br +7	[Br+7]	Br7+
i -1	[I-]	I-
i +1	[I+]	I+
i +3	[I+3]	I3+
i +5	[I+5]	I5+
i +7	[I+7]	I7+
br 0	[Br]	br
he 0	[He]	he
ne 0	[Ne]	ne
ar 0	[Ar]	ar
kr +2	[Kr+2]	Kr2+

Table 3.3 – continued from previous page

Atom type rule	SMARTS / recursive SMARTS	atom_typ
kr 0	[Kr]	kr
xe 0	[Xe]	xe
xe +2	[Xe+2]	Xe2+
xe +4	[Xe+4]	Xe4+
xe +6	[Xe+6]	Xe6+
rn 0	[Rn]	rn
rn +2	[Rn+2]	Rn2+

3.3.3 Determining partial charges

The electrostatic potential (ESP, Section *Electrostatic Potential*) has historically been used to determine partial charges [96] and the ACT supports training models to reproduce the ESP. However, in a very recent paper we have shown that fitting charges to reproduce the ESP in a limited volume around a compound is fundamentally flawed due to lack of information [38]. If the purpose is to build models that reproduce electrostatic interactions, this can be done directly by training models to reproduce SAPT energy components. Here, the split-charge equilibration (SQE) algorithm [97] is used to generate the effective partial charge on each atom in a molecule. SQE, in turn, is based on the electronegativity equalization method (EEM), as developed by Rapp{'e} and Goddard [98]. In brief, EEM uses the atomic hardness η and electronegativity χ to determine the atomic charges in a molecule from a Taylor expansion of the molecular energy in terms of charges. The SQE algorithm introduces a correction to the atomic electronegativities for bonded atoms $\Delta\chi$ as well as a bond hardness $\Delta\eta$. With this addition, charge can *flow* through bonds only, which overcomes issues with over-polarization in the EEM [99].

The ACT code implements the possibility to generate charges for compounds in dimers or clusters where charge transfer between compounds is disallowed which is a reasonable approximation since charge transfer has been shown to have limited impact on the binding energy of non-covalent complexes [100]. For the SQE algorithm two atomic parameters (χ and η) as well as two bond parameter types ($\Delta\chi$ and $\Delta\eta$) need to be determined and the ACT can train SQE parameters to reproduce electrostatic and induction energies [38]. For background information we refer the reader to an excellent review by Jensen [101], but below follows a break-down of using SQE with shells or virtual sites.

3.3.4 Charge Equilibration with Shells or Virtual Sites

Among the approaches to modeling the charge-dependent component of a force field, those rooted in the chemical potential equalization principle are especially notable, as the principle stems directly from density functional theory [102]. The first computational implementation of the chemical potential equalization principle was the electronegativity equalization method (EEM) [98, 103]. However, due to limitations of this model, Chelli *et al.* proposed the atom-atom charge transfer (AACT) model [104]. Later, Nistor and co-workers combined the EEM and AACT approaches into a single framework, the split-charge equilibration (SQE) model, which fulfills the essential criteria for a successful charge-transfer potential [97, 99]. The ACT implements both the EEM and the SQE as algorithms for determining partial charges.

In brief, EEM minimizes an empirical model of the intramolecular electrostatic energy (computed from the atomic electronegativity χ_i and atomic hardness η_i) with respect to the atomic partial charges q_i, where i are the atoms. This method comprises a second order expansion of the molecular energy E_{EEM} in terms of the partial charges q_i:

$$E_{\mathrm{EEM}}(q_1, q_2, \ldots, q_N) = \sum_{i=1}^{N} \left[\chi_i q_i + \frac{1}{2}\eta_i q_i^2 + \frac{1}{2} \sum_{\substack{l=1 \\ l \neq i}}^{N} q_i q_l J_{il} \right], \tag{3.50}$$

where N is the number of atoms, χ_i are the atomic electronegativities, η_i the atomic hardness, and J_{il} the Coulomb interaction between atoms. The factor $\frac{1}{2}$ before the Coulomb matrix is to avoid double counting.

In this work, the SQE method is used, which addresses a shortcoming of the EEM, namely that molecules tend to get over-polarized [99]. For more background, we refer to the recent review on charge flow models by Jensen [101].

Verstraelen and co-workers proposed the following variant of the molecular energy:

$$E_{\mathrm{SQE}} = E_{\mathrm{EEM}} + \sum_{i,j}^{M} \left(\frac{1}{2} \Delta\eta_{ij} p_{ij}^2 + \Delta\chi_{ij}(q_i - q_j) \right) \tag{3.51}$$

where p_{ij} corresponds to the (intramolecular) charge transfer over bonds, $\Delta\eta_{ij}$ is the bond hardness and $\Delta\chi_{ij}$ is the bond electronegativity correction. Therefore, the charge variables q_i are replaced by charge-transfer variables p_{ij} which are related by

$$q_i = \frac{q_{\mathrm{tot}}}{N} + \sum_{\substack{i,j \\ \text{bonds}}} p_{ij}, \tag{3.52}$$

where q_{tot} is the net charge on the compound and $p_{ij} = -p_{ji}$. Although it is trivial to determine the partial charges q_i from the charge transfer p_{ij}, the reverse is not necessarily true. As outlined by Chen *et al.* [105], the problem can be solved by expressing the energy in terms of the charge transfer variables. By substituting $J_{ii} = \eta_i$ in Eq. (3.50), inserting Eq. (3.50) into Eq. (3.51) and introducing M_x as the number of bonds for species x, we obtain:

$$\begin{aligned} E_{\mathrm{SQE}} = \sum_{n=1}^{N} \Bigg[& \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_n} p_{nm} \right) \left(\chi_n + \frac{1}{2}\eta_n \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_n} p_{nm} \right) \right. \\ & \left. + \frac{1}{2} \sum_{\substack{l=1 \\ l \neq n}}^{N} \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_l} p_{lm} \right) J_{nl} \right) \Bigg] \\ & + \sum_{i,j}^{M} \left[\frac{1}{2}\zeta_{ij} p_{ij}^2 + \Delta\chi_{ij} \left(\left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_i} p_{im} \right) - \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_j} p_{jm} \right) \right) \right]. \end{aligned}$$

The next step is to determine the p_{ij} that minimize E_{SQE}. Since all summations run over atoms i, j, k, l, we take the derivative with respect to p_{ij} and equate it to zero:

$$\begin{aligned} 0 &= \frac{\partial E_{\mathrm{SQE}}}{\partial p_{ij}} \\ &= \left(\chi_i - \chi_j + \frac{q_{\mathrm{tot}}}{N}(\eta_i - \eta_j) + \sum_{k=1}^{M_i} \Delta\chi_{ik} - \sum_{k=1}^{M_j} \Delta\chi_{jk} \right) \\ &\quad + \frac{1}{2} \left(\sum_{l=1}^{N} J_{il} \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{m=1}^{M_l} p_{lm} \right) - \sum_{l=1}^{N} J_{li} \left(\frac{q_{\mathrm{tot}}}{N} + \sum_{k=1}^{M_i} p_{ik} \right) \right) + p_{ij}\zeta_{ij}, \end{aligned}$$

using the identity of the Coulomb-matrix elements ($J_{ij} = J_{ji}$), the terms involving the atomic hardness η_x are incorporated into the diagonals of the J_{xy} matrix, excluding the contribution from the total charge q_{tot}. Note that the q_{tot} terms in the two sums cancel. The first term in Eq. (3.53) is the difference in electronegativity between atoms i and j sharing the bond plus the correction; the second term represents the difference in electrostatic potentials at the atoms, and the third term accounts for the interaction

between i and j times the charge transfer. This results in a coupled set of equations, written in matrix form as

$$\mathbf{M}\,\mathbf{P} = \mathbf{R},$$

where $\mathbf{M}$ is a square matrix of dimension equal to the number of bonds, $\mathbf{P}$ is the vector of the charge transfers for all bonds, and $\mathbf{R}$ is the right-hand side of the equations. The matrix elements are given by

$$M_{ij,kl} = J_{ik} - J_{il} - J_{jk} + J_{jl} + \delta_{ij,kl}\Delta\eta_{ij}$$

where $\delta_{ij,kl}$ is one if bond ij is identical to kl and zero otherwise. The right-hand side is defined by the electronegativity terms according to

$$R_{ij} = \chi_j - \chi_i + \sum_{k=1}^{M_j}\Delta\chi_{jk} - \sum_{k=1}^{M_i}\Delta\chi_{ik} + \frac{q_{\mathrm{tot}}}{N}\left(\sum_{l=1}^{N}J_{jl} - \sum_{l=1}^{N}J_{il}\right).$$

The charges of the shells (and virtual sites) are treated as constant in this algorithm, meaning that q_{tot} becomes the sum of the charges of the shells and virtual sites and the total charge of the compound. During force field training, all of these charges can be modified alongside the SQE parameters. As noted in the paper describing the ACT software [1], the SQE algorithm may not be flexible enough to reproduce optimal charge distributions, and other algorithms [101, 106] may need to be implemented in future versions of the software.

3.4 Force Field Training Targets

A number of different targets can be used for training force fields in the ACT. Below, we mention the most important ones including, where needed, the technical details necessary to appreciate the methods. For information on obtaining or generating training data we refer to Section *Training Data*.

Computing interaction energies and components of interaction energies is a crucial part of force field development. In the ACT we have hitherto used data from symmetry-adapted perturbation theory (SAPT) calculations [107, 108]. It is not entirely trivial to match the energy components from SAPT to force field terms, however.

3.4.1 Algorithm to compute energy components in ACT

The component of the interaction energy of a dimer can be computed from the difference between dimer and monomers A and B [109]:

$$E_x^{inter}(AB) = E_x^{total}(AB) - (E_x(A) + E_x(B)) \tag{3.53}$$

where x is exchange or dispersion and $total$ indicates that the energy includes both the intra- and intermolecular interactions. To compute the electrostatics or induction energy of a dimer in the gas phase, the ACT first computes the relaxed energy of the two monomers A and B, that is, the energy of the shell particles is minimized with respect to their positions, yielding $E_x(A)$ and $E_x(B)$. The rationale for this is that SAPT computes electrostatic energies between unperturbed monomers based on the response/relaxation of monomer Hartree-Fock (HF) orbitals in the electric field of the interacting partner. Then, the energies of the dimer AB are computed in three steps:

1. electrostatics is computed with shells located in the relaxed monomer positions, yielding $E_{elec}^{total}(AB)$,
2. the shells of compound A are allowed to relax (further) in the electric field from compound B, while shells of compound B remain at their monomer positions, and vice versa, yielding the second order relaxation $E_{induc}^{inter(2)}$ (see ref. [61] for details),

3. the shells are allowed to relax completely, yielding the total E_{induc} from which the higher order terms, named $E_{induc}^{inter(3)}$ here for convenience, can be derived by subtracting $E_{induc}^{inter(2)}$. According to ref. [61], parameters of models corresponding to the higher other terms, including, potentially, charge transfer, can be trained to the δ_{HF} contribution of the SAPT induction energy. Here, we have added the exponential term proposed by McDaniel and Schmidt for this purpose (section *Induction correction*).

The terms below can be compared directly to SAPT:

$$E_{elec}^{inter}(AB) = E_{elec}^{total}(AB) - (E_{ei}(A) + E_{ei}(B)) \tag{3.54}$$

$$E_{induc}^{inter(2)}(AB) = \left(E_{ei}^{total}\|_A + E_{ei}^{total}\|_B\right) - 2E_{elec}^{total}(AB) \tag{3.55}$$

$$E_{induc}^{inter(3)}(AB) = E_{ei}^{total}(AB) - E_{induc}^{(2)}(AB) - E_{elec}^{total}(AB) \tag{3.56}$$

where ei is short for $elec + induc$ and the notation $\|_X$ indicates that the shells of compound X are kept fixed in the relaxed monomer conformation. If we sum Eqns. (3.54)- (3.56) we recover Eqn. (3.53) where x equals ei. Eqn. (3.54) corresponds to the electrostatics in SAPT.

To summarize, ACT computes the induction term in two parts: a second order term $E_{induc}^{inter(2)}(AB)$ (Eqn. (3.55)) and a third-and-higher-order term $E_{induc}^{inter(3)}(AB)$ (Eqn. (3.56)), the sum of which corresponds to the total induction from SAPT.

McDaniel and Schmidt [61] proposed that Eqn. (3.56) should be equal to the δ_{HF} + δ_{MP2} terms from SAPT while Eqn. (3.55) would correspond to the polarization energy. They then continue to suggest how this can be implemented in a force field:

$$E_{induc} = E_{shell} + E_{pol} + E_{\delta HF}$$

where E_{shell} is Eqn. (3.8) and

$$E_{pol} = A_{ij}^{ind}\exp(-b_{ij}r_{ij})$$

where r_{ij} is the interatomic distance and b_{ij} a constant, and

$$E_{\delta HF} = A_{ij}^{\delta HF}\exp(-b_{ij}r_{ij})$$

where both A_{ij}^{ind} and $A_{ij}^{\delta_{HF}}$ are determined from a negative geometric combination rule

$$A_{ij} = -\sqrt{A_i A_j}$$

meaning these terms are always attractive. These potentials use the b_{ij} that is used in the exchange energy (using a Buckingham potential, Eqn. (3.13)). Whether this is the most appropriate way of splitting terms and reproducing SAPT energies remains to be determined.

In the output the ACT training module *alexandria train_ff* there are two terms related to induction. The term INDUCTIONCORRECTION refers to Eqn. (3.56). If that is present, the term INDUCTION refers to Eqn. (3.55), if not it refers to the sum of the two.

3.4.2 Monomer Energies and Forces

Typically, a series of single-point quantum calculations are done at a user-defined level of theory. These calculations can then be converted to a molprop file. More information to come.

3.4.3 Other Properties

In principle, all the molecular properties mentioned in Section *Predicting Molecular Properties* can be used for training, but it is highly recommended to leave some properties for validation.

3.5 Parameters for Force Field Training

Each potential available in the ACT has its own set of parameters. Those are listed in Table 3.4 for non-bonded parameters, Table 3.5 for bonded parameters. Parameters for virtual sites are given in Table 3.6 and for charge generation algorithms in Table 3.7. In total, well over 80 parameter categories can be trained using the ACT.

Table 3.4: Force field parameters related to non-bonded potentials that can be trained using ACT. Note that the parameter names are case-sensitive and, in some cases, they correspond to a Greek symbol in the equation.

Potential	Equation	Parameters
COULOMB_GAUSSIAN	(3.2)	zeta
COULOMB_SLATER	(3.5)	zeta
POLARIZATION	(3.8)	alpha
MACDANIEL_SCHMIDT	(3.9)	a1dexp bdexp
LJ14_7	(3.10)	sigma epsilon gamma delta
GENERALIZED_BUCKINGHAM	(3.11)	rmin epsilon gamma delta
WANG_BUCKINGHAM	(3.12)	sigma epsilon gamma
BUCKINGHAM	(3.13)	A b C
TANG_TOENNIES	(3.15)	Att btt c6tt c8tt c10tt
TT2	(3.16)	Att2b bExchtt2b c6tt2b c8tt2b c10tt2b
SLATER_ISA	(3.17)	A b
LJ12_6	(3.18)	sigma epsilon
LJ12_6_4	(3.19)	sigma epsilon gamma
BORN_MAYER	(3.20)	A b
Generalized mean	(3.36)	exponent

Table 3.5: Force field parameters related to bonded potentials that can be trained using ACT. Note that the parameter names are case-sensitive and, in some cases, they correspond to a Greek symbol in the equation.

Potential	Equation	Parameters
HARMONIC_BONDS	(3.37)	kb bondlength bondenergy
MORSE_BONDS	(3.38)	beta De bondlength D0
HUA_BONDS	(3.39)	De bondlength b c
HARMONIC_ANGLES	(3.40)	kt angle
LINEAR_ANGLES	(3.41)	klin a
HARMONIC_DIHEDRALS	(3.44)	kimp
FOURIER_DIHEDRALS	(3.45)	c0 c1 c2 c3 c4 c5
PROPER_DIHEDRALS	(3.46)	kp mult phi0

Table 3.6: Parameters related to virtual sites that can be trained using ACT. Note that the parameter names are case-sensitive. Equations will be added later.

Virtual Site	Equation	Parameters
VSITE1		vs1a
VSITE2		vs2a
VSITE2FD		vs2fd_a
VSITE3		vs3a vs3b
VSITE3S		vs3a
VSITE3FD		vs3fd_a vs3fd_b
VSITE3FAD		vs3fad_a vs3fad_b
VSITE3OUT		vs3out_a vs3out_b vs3out_c
VSITE4		vs4a vs4b vs4c
VSITE4S		vs4sa vs4sb
VSITE4S3		vs4s3a

Table 3.7: Parameters related to charge algorithms that can be trained using ACT. Note that the parameter names are case-sensitive. Equations will be added later.

Algorithm	Equation	Parameters
EEM	(3.50)	eta chi
SQE	(3.51)	eta chi delta_eta delta_chi

3.6 Training Data

3.6.1 Using existing data

A recent review from the ACT developers~ [8] discusses the different available quantum chemistry data sets. Some of these can be used in the ACT, for instance the coupled-cluster dimer data set due to Donchev *et al.* [28], the Non-covalent interaction atlas from the Czech group led by Řezáč~ [110] and some more. The SAPT dataset on protein side-chain analogs and backbone analogs by Burns~ [111] can in principle be used in the ACT as well.

In principle, the ANI-1 dataset~ [112] can be used to provide off-equilibrium energies of small compounds, but in the first version it was limited to compound containing the elements C, H, N, O only.

3.6.2 Alexandria Library

The Alexandria Library contains energies at optimized conformations of about 5000 compounds, thermochemistry, electric multipoles and electrostatic potential at grid points around molecules~ [17, 18]. It is possible to train electrostatic models using data from the library.

3.6.3 Donchev data set

To use the dataset due to Donchev *et al.* [28], we provide a script that reads the comma-separated value file provided by those authors. Please refer to their article for download information. Then refer to the built-in help in the script for more guidance by executing:

```
donchev2molprop -h
```

and investigating the output.

3.6.4 Non-covalent interaction atlas

The Non-covalent interaction atlas~ [110] can be used in a similar manner. At the time of writing it can be href{http://www.nciatlas.org}{found here}. To convert the data to ACT files, start by:

```
ncia2molprop -h
```

and investigate your options.

3.6.5 ACT data

Quantum chemical data used in ACT-related publications will be uploaded to a sharing site.

3.6.6 Generating SAPT data

A recent review described the available data sets available for machine learning or force field training~ [8]. There is however a lack of certain data, or data sets are incomplete. For this reason there is a git repository SaptACT that provides script to run SAPT calculations using the Psi4 software~ [113] including tools to convert the output to ACT compatible inputs (see below). The steps needed to prepare data for training in ACT are as follows:

- Clone the SaptACT repository using *git*
- Prepare selection of compounds, e.g. water, methanol, ethanol, and make sure that monomer structures for these compounds are present in the *xyz/monomers* catalog.
- Generate a dimer selection file using the *gen_dimers.py* script in the SaptACT repository:

  ```
  ./gen_dimers.py -sel water methanol ethanol -o alcohol.dat
  ```

  which will generate a file containing:

  ```
  water#water
  water#methanol
  water#ethanol
  methanol#methanol
  methanol#ethanol
  ethanol#ethanol
  ```

  where the # symbol separates the monomers. Note that the order on the command line determines the order in the selection file.

- Perform SAPT calculations using Psi4 by generating randomly oriented dimers at a series of distances defined by scaling the shortest distance between any pair of atoms in the generated orientations. In this case, six distances varying from 0.9 to 1.4 times the sum of the van der Waals radii of the nearest atoms will be generated.:

  ```
  ./run_calcs.py -dimers alchol.dat -ndist 6 -mindist 0.9 -maxdist 1.4 -
  ↪norient 10
  ```

Note, that in the above command 6 x 10 calculations are started for each dimer in *alcohol.dat*, that is 360 calculations in total. You will obviously need a compute cluster to do these calculations, in particular if you select a more accurate SAPT level of theory~ [108] than the default (sapt2+/aug-cc-pvdz).

3.6.7 Generating single molecule data

Single molecules off-equilibrium energies and forces are needed to parameterize the intramolecular potential functions. Scripts are available that will take a structure of a monomer, perform a 50 ps MD simulation in the gas phase at elevated temperature using the GAFF force field~ [114] and the GROMACS software~ [115]. Then, conformations are extracted from the simulation trajectory and these are subjected to quantum chemistry calculations using Psi4~ [113].

3.6.8 Conversion to ACT molprop files

The ACT uses the href{https://en.wikipedia.org/wiki/XML{eXtensible Markup Language} to store both data for training and force field files. Once your SAPT calculations are finished you need to perform the following steps to generate the ACT input:

- Convert the Psi4 outputs to compact and human-readable href{https://en.wikipedia.org/wiki/JSON}{json} files using another script in SaptACT:

```
./generate_json.py
```

  which will traverse the output directories, find Psi4 outputs, and if there is not already a *results.json* file, will generate it. Please familiarize yourself with the content of these files.

- When the json files have been generated, you are ready to convert them to a molprop file:

```
./write_molprop.py -o molprop.xml
```

  please inspect the output file from this script to verify that your calculations are present.

Both these scripts have more flags that can be useful, please investigate those using the *-h* option.

CHAPTER

FOUR

INTERACTIVE NOTEBOOKS

4.1 Charge Methods

A number of different algorithms are available in ACT.

4.1.1 Electronegativity equalization method

The electronegativity equalization method (EEM) is a second order expansion of the molecular energy E_{EEM} in terms of the partial charges q_i:

$$E_{EEM}(q_1, q_2, ..., q_N) = \sum_{i=1}^{N} \left[\chi_i q_i + \frac{1}{2}\eta_i q_i^2 + \frac{1}{2} \sum_{l \neq i}^{N} q_i q_l J_{il} \right]$$

where N is the number of atoms, χ_i are the atomic electronegativities, η_i the atomic hardness and J_{il} the Coulomb interaction between atoms. The factor 1/2 before the Coulomb matrix is to avoid double counting. In equilibrium, the chemical potential is the same for each atom, χ_{eq}. The EEM can be implemented by taking the derivative of E_{EEM} with respect to q_i and equating it to zero:

$$0 = \frac{\partial E_{EEM}}{\partial q_i} = \chi_{eq} = \chi_i + q_i \eta_i + \frac{1}{2} \sum_{l \neq i}^{N} q_l J_{il}$$

which leads to a set of $N + 1$ equations that is linear in the q_i and that can be solved using linear algebra tools. After taking the derivative with respect to q_i a matrix equation is obtained:

$$\begin{bmatrix} \eta_1 & J_{12} & J_{13} & ... & J_{1N} & -1 \\ J_{21} & \eta_2 & J_{23} & ... & J_{2N} & -1 \\ ... & ... & ... & ... & ... & ... \\ J_{N1} & J_{N2} & J_{N3} & ... & \eta_N & -1 \\ 1 & 1 & 1 & ... & 1 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ ... \\ q_N \\ \chi_{eq} \end{bmatrix} = \begin{bmatrix} -\chi_1 \\ -\chi_2 \\ ... \\ -\chi_N \\ q_{tot} \end{bmatrix}$$

Note the last column in the matrix is there to make sure that the electronegativity for all atoms is the same (χ_{eq}), while the last row is there to make sure the total charge q_{tot} is maintained. A good reference for the method is the classic paper by Rappe and Goddard.

In the code below we test the method on a methanol molecule, using parameters from Verstraelen et al. as well as experimental data from Cordero et al..

```
import numpy as np
from scipy.linalg import lstsq
import math
from charge_utils import *
from test_systems import *
```

(continues on next page)

(continued from previous page)

```
def solveEEM(names, coords, qtotal, model, verbose=False):
    # Compute Coulomb
    J = calcJEEM(names, coords, model)
    N = coords.shape[0]

    # Right hand side of the equation
    rhs = np.zeros(N+1,dtype=float)
    chi = get_chi(model)
    for i in range(N):
        if names[i] in chi:
            rhs[i] = -chi[names[i]]
        else:
            print("No chi for %s" % names[i])
            exit(1)
    rhs[N] = qtotal
    q = np.linalg.solve(J, rhs)
    if verbose:
        print("J = \n{}".format(J))
        print("rhs = {}".format(rhs))
        print("q = {}".format(q))
        y = np.dot(J,q)
        print("y = {}".format(y))
    return q

def run_compound(molname, verbose, alexandria=False):
    mol = get_system(molname)
    if not mol:
        print("No test system %s defined" % molname)
    else:
        print("\n%s" % molname)
        q = solveEEM(mol["names"], mol["coords"], mol["qtotal"], verbose,␣
↪alexandria)
        for i in range(mol["coords"].shape[0]):
            print("q[%s] = %g" % (mol["names"][i], q[i]))
        printDipole(q, mol["coords"], mol["atomnr"], False)

verbose = False

for compound in [ "carbon-monoxide", "water", "methanol", "acetate" ]:
    run_compound(compound, "ACM-g", verbose)
```

```
carbon-monoxide
q[c2] = 0.100795
q[o] = -0.100795
Dipole = 0.0580969 Debye

water
q[hw] = 0.130157
q[ow] = -0.260313
```

(continues on next page)

(continued from previous page)

```
q[hw] = 0.130157
Dipole = 0.731574 Debye

methanol
q[oh] = -0.252355
q[hp] = 0.13162
q[c3] = -0.151608
q[h1] = 0.0878756
q[h1] = 0.087886
q[h1] = 0.0965812
Dipole = 1.22594 Debye

acetate
q[c3] = -0.254055
q[cm] = -0.108471
q[om] = -0.322013
q[om] = -0.322018
q[hc] = 0.00241518
q[hc] = 0.00166854
q[hc] = 0.00247338
Dipole = 0.841977 Debye
```

Note the positive charge on O and negative charge on one of the H with the EEM chi and eta values from Verstraelen. These charges are too small since the experimental dipole for methanol is about 1.69 Debye.

4.1.2 Split charge equilibration

The split charge equilibration (SQE) method postulates that partial charges q_i on atoms can only arise from transfer of charge p_{ij} between bonded atoms i and j only, according to:

$$q_i = \frac{q_{tot}}{N} + \sum_{i,j}^{bonds} p_{ij}$$

where q_{tot} is the net charge on the compound and $p_{ij} = -p_{ji}$. Although it is trivial to determine the partial charges q_i from the charge transfer p_{ij}, the reverse is not necessarily true. To exemplify this we write the equations for methanol:

$$\begin{bmatrix} 0 & p_{12} & p_{13} & p_{14} & p_{15} & 0 \\ -p_{12} & 0 & 0 & 0 & 0 & 0 \\ -p_{13} & 0 & 0 & 0 & 0 & 0 \\ -p_{14} & 0 & 0 & 0 & 0 & 0 \\ -p_{15} & 0 & 0 & 0 & 0 & p_{56} \\ 0 & 0 & 0 & 0 & 0 - p_{56} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{bmatrix}$$

With these equations we can express the p_{ij} in terms of the charges after manual Gaussian elimination as follows:

$$\begin{bmatrix} p_{12} \\ p_{13} \\ p_{14} \\ p_{15} \\ p_{56} \end{bmatrix} = \begin{bmatrix} -q_2 \\ -q_3 \\ -q_4 \\ -q_5 - q_6 \\ -q_6 \end{bmatrix}$$

with q_1 by definition $q_{tot} - q_2 - q_3 - q_4 - q_5 - q_6$ (with q_{tot} = 0). However, expressing the p_{ij} in q_i does not guarantee that the charge transfer through space is zero. Therefore it is necessary to do the reverse. As outlined by Chen et al. the problem can be solved by expressing the energy in terms of the charge transfer variables.

Verstraelen and co-workers used the following variant of the molecular energy:

$$E_{SQE} = E_{EEM} + \sum_{i,j}^{M} \left(\tfrac{1}{2}\zeta_{ij} p_{ij}^2 + \Delta\chi_{ij}(q_i - q_j) \right)$$

In this formulation, the energy is not just a function of the charges, but also of the charge transfer over M bonds. ζ_{ij} is the bond hardness and $\Delta\chi_{ij}$ is the electronegativity correction, both of these are bond-specific. We therefore replace the charge variable by charge transfer variables and substitute $J_{ii} = \eta_i$, and for clarity we replace the summation over bonds by a summation over atom pairs, which is the same thing noting that only p_{ij} for chemical bonds are non-zero (note that we have to introduce a factor 1/2 to avoid double counting). If we introduce M_x as the number of bonds for species x we obtain

$$E_{SQE} = \sum_{n=1}^{N} \left[\left(\frac{q_{tot}}{N} + \sum_{m=1}^{M_n} p_{nm} \right) \left(\chi_n + \tfrac{1}{2}\eta_n \left(\frac{q_{tot}}{N} + \sum_{m=1}^{M_n} p_{nm} \right) + \tfrac{1}{2} \sum_{l \neq n}^{N} \left[\frac{q_{tot}}{N} + \sum_{m=1}^{M_l} p_{lm} \right] J_{nl} \right) \right] + \sum_{i,j}^{M} \left[\tfrac{1}{2}\zeta_{ij} p_{ij}^2 + \Delta\chi_{ij} \left(\left[\frac{q_{tot}}{N} + \sum_{m=1}^{M_i} p_{im} \right] - \left[\frac{q_{tot}}{N} + \sum_{m=1}^{M_j} p_{jm} \right] \right) \right]$$

Our task now is to find the p_{ij} that minimize E_{SQE}. Please note that the all summations are over atoms i, j, k, l. Therefore, we take the derivative and equate it to zero:

$$0 = \frac{\partial E_{SQE}}{\partial p_{ij}} = \left(\chi_i - \chi_j + \frac{q_{tot}}{N}(\eta_i - \eta_j) + \sum_{k=1}^{M_i} \Delta\chi_{ik} - \sum_{k=1}^{M_j} \Delta\chi_{jk} \right) + \frac{1}{2} \left(\sum_{l=1}^{N} J_{il} \left(\frac{q_{tot}}{N} + \sum_{m=1}^{M_l} p_{lm} \right) - \sum_{i=1}^{N} J_{li} \left(\frac{q_{tot}}{N} + \sum_{k=1}^{M_i} p_{ik} \right) \right) + p_{ij}\zeta_{ij}$$

using the identity of the Coulomb-matrix elements ($J_{ij} = J_{ji}$) and where the terms involving the atomic hardness η_x are included on the diagonals of the J_{xy} matrix, except for the contribution of the total charge q_{tot}. Note that the q_{tot} terms in the two sums cancel. The first term is the difference in electronegativity between atoms i and j sharing the bond plus the correction, the second term represents the difference in electrostatic potentials at the atoms, the third term is the interaction between i and j times the charge transfer. This leads to a coupled set of equations, written in matrix form as:

$$\mathbf{M}P = R$$

where $\mathbf{M}$ is a square matrix of dimension number of bonds, P is a vector containing the charge transfers for all bonds and R the right hand side of the equations. If the summations are all written out we obtain the result published by Chen et al. for QTPIE and similar algorithms:

$$M_{ij,kl} = J_{ik} - J_{il} - J_{jk} + J_{jl}$$

for off-diagonal elements of the matrix. For the diagonal elements, the bond hardness ζ_{ij} needs to be added. The right hand side is given by the electronegativity terms according to:

$$R_{ij} = \chi_j - \chi_i + \sum_{k=1}^{M_j} \Delta\chi_{jk} - \sum_{k=1}^{M_i} \Delta\chi_{ik} + \frac{q_{tot}}{N} \left(\sum_{l=1}^{N} J_{jl} - \sum_{l=1}^{N} J_{il} \right).$$

The equations are implemented in Python below.

```
import numpy as np
from scipy.linalg import lstsq
```

(continues on next page)

(continued from previous page)

```
import math
from charge_utils import *
from test_systems import *

def calcRHS(names, bonds, JEEM, qtotal, model):
    Natoms = len(names)
    Nbond = len(bonds)
    RHS = np.zeros(Nbond, dtype=float)
    ##### calc corrected chi ####
    chi_corr = []
    chi = get_chi(model)
    for i in range(Natoms):
        chi_corr.append(chi[names[i]])
        for k in range(Nbond):
            ai  = names[bonds[k][0]]
            aj  = names[bonds[k][1]]
            if bonds[k][0] == i:
                chi_corr[i] += getDeltaChi(ai, aj, model)
            if bonds[k][1] == i:
                chi_corr[i] -= getDeltaChi(ai, aj, model)
        for l in range(Natoms):
            chi_corr[i] += JEEM[i][l]*(qtotal/Natoms)
    #### calc EN diff for the bonds ####
    for i in range(Nbond):
        RHS[i] = chi_corr[bonds[i][1]] - chi_corr[bonds[i][0]]
    return RHS

def solveSQE(names, coords, bonds, verbose, qtotal, model):
    JEEM, JSQE = calcJSQE(names, coords, bonds, verbose, model)
    rhs  = calcRHS(names, bonds, JEEM, qtotal, model)
    if verbose:
        print("RHS: {}".format(rhs))
    pij  = np.linalg.solve(JSQE, rhs)
    if verbose:
        print("pij: {}".format(pij))
    q =  np.zeros(coords.shape[0], dtype=float)
    for n in range(len(bonds)):
        bi = bonds[n][0]
        bj = bonds[n][1]
        q[bi] += pij[n]
        q[bj] -= pij[n]
    Natoms = names.shape[0]
    for i in range(Natoms):
        q[i] += qtotal/Natoms
    print("qSQE: {}".format(q))
    return q

def run_compound(molname, verbose, shells):
    mol = get_system(molname)
```

(continues on next page)

(continued from previous page)

```
    print("%s" % molname)
    for bonds in mol["bonds"]:
        q = solveSQE(mol["names"], mol["coords"], bonds, verbose, mol["qtotal
↪"], "ACS-g")
    printDipole(q, mol["coords"], mol["atomnr"], False)

verbose = False
shells = False

for compound in [ "methanol", "water", "acetate", "carbon-monoxide" ]:
    run_compound(compound, verbose, shells)
```

```
methanol
qSQE: [-0.50802387  0.33691938 -0.33742533  0.16436908  0.16438559  0.
↪17977516]
qSQE: [-0.50802387  0.33691938 -0.33742533  0.16436908  0.16438559  0.
↪17977516]
qSQE: [-0.50802387  0.33691938 -0.33742533  0.16436908  0.16438559  0.
↪17977516]
qSQE: [-0.50802387  0.33691938 -0.33742533  0.16436908  0.16438559  0.
↪17977516]
qSQE: [-0.50802387  0.33691938 -0.33742533  0.16436908  0.16438559  0.
↪17977516]
Dipole = 2.07033 Debye
water
qSQE: [ 0.11909703 -0.23819405  0.11909703]
qSQE: [ 0.11909703 -0.23819405  0.11909703]
qSQE: [ 0.11909703 -0.23819405  0.11909703]
Dipole = 0.669411 Debye
acetate
qSQE: [-0.48654202 -0.0231567  -0.24282282 -0.24282009 -0.00136991 -0.00196462
 -0.00132384]
qSQE: [-0.48654202 -0.0231567  -0.24282282 -0.24282009 -0.00136991 -0.00196462
 -0.00132384]
Dipole = 1.41698 Debye
carbon-monoxide
No hardness information for bond c2-o
No deltaChi information for bond c2-o
No deltaChi information for bond c2-o
qSQE: [ 0.77205802 -0.77205802]
No hardness information for bond o-c2
No deltaChi information for bond o-c2
No deltaChi information for bond o-c2
qSQE: [ 0.77205802 -0.77205802]
Dipole = 0.445003 Debye
```

4.2 Atom Typing using RDKit

CHAPTER

FIVE

DEVELOPER MANUAL

Please checkout the Doxygen documentation.

BIBLIOGRAPHY

[1] David van der Spoel, Julián Marrades, Kristian Kříž, A. Najla Hosseini, Alfred T. Nordman, João Paulo Ataide Martins, Marie-Madeleine Walz, Paul J. van Maaren, and Mohammad M. Ghahremanpour. Evolutionary machine learning of physics-based force fields in high-dimensional parameter-space. *Digit. Discovery*, 4:1925–1935, 2025. doi:10.1039/d5dd00178a.

[2] David van der Spoel and Erik Lindahl. Brute-Force Molecular Dynamics Simulations of Villin Headpiece: Comparison with NMR Parameters. *J. Phys. Chem. B*, 107(40):11178–11187, 2003. doi:10.1021/jp034108n.

[3] Oliver F Lange, David van der Spoel, and Bert L de Groot. Scrutinizing Molecular Mechanics Force Fields on the Submicrosecond Timescale with NMR Data. *Biophys. J.*, 99:647–655, 2010. doi:10.1016/j.bpj.2010.04.062.

[4] Haiyang Zhang, Chunhua Yin, Yang Jiang, and David van der Spoel. Force field benchmark of amino acids: i. hydration and diffusion in different water models. *J. Chem. Inf. Model.*, 58:1037–1052, 2018.

[5] Zahedeh Bashardanesh and David van der Spoel. Impact of dispersion coefficient on simulations of proteins and organic liquids. *J. Phys. Chem. B.*, 122:8018–8027, 2018. doi:10.1021/acs.jpcb.8b05770.

[6] David van der Spoel. Systematic design of biomolecular force fields. *Curr. Opin. Struct. Biol.*, 67:18–24, 2021. doi:10.1016/j.sbi.2020.08.006.

[7] David van der Spoel, Jin Zhang, and Haiyang Zhang. Quantitative predictions from molecular simulations using explicit or implicit interactions. *Wiley Interdiscip. Rev.-Comput. Mol. Sci.*, 12:e1560, 2022. doi:{10.1002/wcms.1560}.

[8] Kristian Kříž, Lisa Schmidt, Alfred T. Andersson, Marie-Madeleine Walz, and David van der Spoel. An imbalance in the force: the need for standardised benchmarks for molecular simulation. *J. Chem. Inf. Model.*, 63:412–431, 2023. doi:10.1021/acs.jcim.2c01127.

[9] C Caleman, P J van Maaren, M Hong, J S Hub, L T Costa, and D van der Spoel. Force field benchmark of organic liquids: density, enthalpy of vaporization, heat capacities, surface tension, compressibility, expansion coefficient and dielectric constant. *J. Chem. Theory Comput.*, 8:61–74, 2012. doi:10.1021/ct200731v.

[10] Nina M. Fischer, Paul J. van Maaren, Jonas C. Ditz, Ahmet Yildirim, and David van der Spoel. Properties of liquids in molecular dynamics simulations with explicit long-range Lennard-Jones interactions. *J. Chem. Theory Comput.*, 11:2938–2944, 2015. doi:10.1021/acs.jctc.5b00190.

[11] Jin Zhang, Badamkhatan Tuguldur, and David van der Spoel. Force field benchmark II: Gibbs energy of solvation of organic molecules in organic liquids. *J. Chem. Inf. Model.*, 55:1192–1201, 2015. doi:10.1021/acs.jcim.5b00106.

[12] Jin Zhang, Badamkhatan Tuguldur, and David van der Spoel. Correction to force field benchmark II: Gibbs energy of solvation of organic molecules in organic liquids. *J. Chem. Inf. Model.*, 56:819–820, 2016. doi:10.1021/acs.jcim.6b00081.

[13] David van der Spoel, Mohammad Mehdi Ghahremanpour, and Justin Lemkul. Small molecule thermochemistry: a tool for empirical force field development. *J. Phys. Chem. A*, 122:8982–8988, 2018. doi:10.1021/acs.jpca.8b09867.

[14] Lisa Schmidt, David van der Spoel, and Marie-Madeleine Walz. Probing phase transitions in organic crystals using atomistic md simulations. *ACS Phys Chem Au*, 3:84–93, 2023. doi:10.1021/acsphyschemau.2c00045.

[15] A. Najla Hosseini and David van der Spoel. Simulations of amyloid-forming peptides in the crystal state. *Prot. J.*, 42:192–204, 2023. doi:10.1007/s10930-023-10119-3.

[16] Mohammad Mehdi Ghahremanpour, Paul J. van Maaren, Jonas Ditz, Roland Lindh, and David van der Spoel. Large-scale calculations of gas phase thermochemistry: enthalpy of formation, standard entropy and heat capacity. *J. Chem. Phys.*, 145:114305, 2016. doi:10.1063/1.4962627.

[17] Mohammad Mehdi Ghahremanpour, Paul J. van Maaren, and David van der Spoel. The Alexandria library: a quantum chemical database of molecular properties for force field development. *Sci. Data*, 5:180062, 2018. doi:10.1038/sdata.2018.62.

[18] Mohammad Mehdi Ghahremanpour, Paul J. van Maaren, and David van der Spoel. Alexandria library [data set]. Zenodo. 2017. URL: https://dx.doi.org/10.5281/zenodo.1004711.

[19] Marie Madeleine Walz, Mohammad M. Ghahremanpour, Paul J. van Maaren, and David van der Spoel. Phase-transferable force field for alkali halides. *J. Chem. Theory Comput.*, 14:5933–5948, 2018. doi:10.1021/acs.jctc.8b00507.

[20] Marie-Madeleine Walz and David van der Spoel. Molten alkali halides - temperature dependence of structure, dynamics and thermodynamics. *Phys. Chem. Chem. Phys.*, 21:8516–18524, 2019. doi:10.1039/C9CP03603B.

[21] Marie-Madeleine Walz and David van der Spoel. Systematically improved melting point prediction: a detailed physical simulation model is required. *Chem. Comm.*, 55:12044–12047, 2019. doi:10.1039/C9CC06177K.

[22] Marie-Madeleine Walz and David van der Spoel. Direct link between structure, dynamics and thermodynamics in molten salts. *J. Phys. Chem. C.*, 123:25596–25602, 2019. doi:10.1021/acs.jpcc.9b07756.

[23] Marie-Madeleine Walz and David van der Spoel. Microscopic origin of conductivity in molten salts unraveled by computer simulations. *Commun. Chem.*, 4(9):1–10, 2021. doi:10.1038/s42004-020-00446-2.

[24] David van der Spoel, Henning Henschel, Paul J. van Maaren, Mohammad M. Ghahremanpour, and Luciano T. Costa. A potential for molecular simulation of compounds with linear moieties. *J. Chem. Phys.*, 153(8):084503, 2020. doi:10.1063/5.0015184.

[25] Kristian Kříž, Paul J. van Maaren, and David van der Spoel. Impact of combination rules, level of theory and potential function on the modelling of gas and condensed phase properties of noble gases. *J. Chem. Theory Comput.*, 20:2362–2376, 2024. doi:10.1021/acs.jctc.3c01257.

[26] Kristian Kříž and David van der Spoel. Quantification of anisotropy in exchange and dispersion interactions: a simple model for physics-based force fields. *J. Phys. Chem. Lett.*, 15:9974–9978, 2024. doi:10.1021/acs.jpclett.4c02034.

[27] Paul J. van Maaren and David van der Spoel. Quantitative evaluation of anharmonic bond potentials for molecular simulations. *Digit. Discov.*, 4:824–830, 2025. doi:10.1039/D4DD00344F.

[28] Alexander G. Donchev, Andrew G. Taube, Elizabeth Decolvenaere, Cory Hargus, Robert T. McGibbon, Ka-Hei Law, Brent A. Gregersen, Je-Luen Li, Kim Palmo, Karthik Siva, Michael Bergdorf, John L. Klepeis, and David E. Shaw. Quantum chemical benchmark databases of gold-standard dimer interaction energies. *Sci. Data*, 8(1):55, 2021. doi:10.1038/s41597-021-00833-x.

[29] Peter Eastman, Raimondas Galvelis, Raúl P. Peláez, Charlles R. A. Abreu, Stephen E. Farr, Emilio Gallicchio, Anton Gorenko, Michael M. Henry, Frank Hu, Jing Huang, Andreas Krämer, Julien Michel, Joshua A. Mitchell, Vijay S. Pande, João PGLM Rodrigues, Jaime Rodriguez-Guerra, Andrew C. Simmonett, Sukrit Singh, Jason Swails, Philip Turner, Yuanqing Wang, Ivy Zhang, John D. Chodera, Gianni De Fabritiis, and Thomas E. Markland. Openmm 8: molecular dynamics simulation with machine learning potentials. *J. Phys. Chem. B.*, 128:109–116, 2024. doi:10.1021/acs.jpcb.3c06662.

[30] Mohammad Mehdi Ghahremanpour, Paul J. van Maaren, Carl Caleman, Geoffrey R. Hutchison, and David van der Spoel. Polarizable drude model with s-type gaussian or slater charge density for general molecular mechanics force fields. *J. Chem. Theory Comput.*, 14:5553–5566, 2018. doi:10.1021/acs.jctc.8b00430.

[31] Daniel S D Larsson and David van der Spoel. Screening for the location of RNA using the chloride ion distribution in simulations of virus capsids. *J. Chem. Theory Comput.*, 8:2474–2483, 2012. doi:10.1021/ct3002128.

[32] B G Dick and A W Overhauser. Theory of the dielectric constants of alkali halide crystals. *Phys. Rev.*, 112:90–103, 1958. doi:10.1103/PhysRev.112.90.

[33] P C Jordan, P J van Maaren, J Mavri, D van der Spoel, and H J C Berendsen. Towards Phase Transferable Potential Functions: Methodology and Application to Nitrogen. *J. Chem. Phys.*, 103:2272–2285, 1995. doi:10.1063/1.469703.

[34] P J van Maaren and D van der Spoel. Molecular dynamics simulations of water with a novel shell-model potential. *J. Phys. Chem. B.*, 105:2618–2626, 2001. doi:10.1021/jp003843l.

[35] P. Dauber-Osguthorpe and A.T. Hagler. Biomolecular force fields: where have we been, where are we now, where do we need to go and how do we get there? *J. Comput. Aid. Mol. Des.*, 33:133–203, 2019. doi:10.1007/s10822-018-0111-4.

[36] A.T. Hagler. Force field development phase II: relaxation of physics-based criteria... or inclusion of more rigorous physics into the representation of molecular energetics. *J. Comput. Aided Mol. Des.*, 33:205–264, 2019. doi:10.1007/s10822-018-0134-x.

[37] Zhifeng Jing, Chengwen Liu, Sara Y. Cheng, Rui Qi, Brandon D. Walker, Jean-Philip Piquemal, and Pengyu Ren. Polarizable force fields for biomolecular simulations: recent advances and applications. *Ann. Rev. Biophys.*, 48(1):371–394, 2019. doi:10.1146/annurev-biophys-070317-033349.

[38] A. Najla Hosseini, Kristian Kříž, and David van der Spoel. Beyond partitioning: using force field science to evaluate electrostatics models. *J. Chem. Theory Comput.*, 22: , 2026. doi:10.1021/acs.jctc.6c00039.

[39] Herman J. C. Berendsen. *Simulating the physical world*. Cambridge University Press, Cambridge, 2007.

[40] Frank Jensen. *Introduction to computational chemistry*. John Wiley & Sons Ltd, West Sussex, UK, 2007.

[41] P Calaminici, K Jug, and M Köster. Density funtional calculations of molecular polarizabilities anf hyperpolarizabilities. *J. Chem. Phys.*, 109(18):7756, 1998. doi:10.1063/1.477421.

[42] A. J Stone. *The Theory of Intermolecular Forces*. Oxford University Press, Great Clarendon Street, Oxford, ox2 6dp, UK, 2013.

[43] Henning Henschel, Alfred T. Andersson, Willem Jespers, Mohammad Mehdi Ghahremanpour, and David van der Spoel. Theoretical infrared spectra: quantitative similarity measures and force fields. *J. Chem. Theory Comput.*, 16(5):3307–3315, 2020. doi:10.1021/acs.jctc.0c00126.

[44] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox. Gaussian 16 Revision A.03. 2016. Gaussian Inc. Wallingford CT.

[45] A D Becke. Density-functional exchange-energy approximation with correct asymptotic-behavior. *Phys. Rev. A*, 38:3098–3100, 1988. doi:10.1103/PhysRevA.38.3098.

[46] R A Kendall, T H Dunning, Jr., and R J Harrison. Electron affinities of the first-row atoms revisited. Systematic basis sets and wave functions. *J. Chem. Phys.*, 96:6796–6806, 1992. URL: http://dx.doi.org/10.1063/1.462569.

[47] D E Woon and T H Dunning, Jr. Benchmark calculations with correlated molecular wave functions. I. Multireference configuration interaction calculations for the second row diatomic hydrides. *J. Chem. Phys.*, 99:1914–1929, 1993. doi:10.1063/1.464303.

[48] D E Woon and T H Dunning, Jr. Gaussian basis sets for use in correlated molecular calculations. III. The atoms aluminum through argon. *J. Chem. Phys.*, 98:1358–1371, 1993.

[49] William L. Jorgensen, Mohammad M. Ghahremanpour, Anastasia Saar, and Julian Tirado-Rives. OPLS/2020 force field for unsaturated hydrocarbons, alcohols, and ethers. *J. Phys. Chem. B.*, 128:250–262, 2023. doi:10.1021/acs.jpcb.3c06602.

[50] John Rumble. *CRC Handbook of Chemistry and Physics 103rd edition*. CRC Press, Gaitherburg, MD, 2022. URL: http://hbcp.chemnetbase.com/.

[51] I. Amdur and E. A. Mason. Properties of gases at very high temperatures. *Phys. Fluids*, 1:370–383, 09 1958. doi:10.1063/1.1724353.

[52] W L Jorgensen, J Chandrasekhar, J D Madura, R W Impey, and M L Klein. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.*, 79:926–935, 1983. doi:10.1063/1.445869.

[53] Brian J. Kirby and Pavel Jungwirth. Charge scaling manifesto: a way of reconciling the inherently macroscopic and microscopic natures of molecular simulations. *J. Phys. Chem. Lett.*, 10:7531–7536, 2019. doi:10.1021/acs.jpclett.9b02652.

[54] S W Rick, S J Stuart, and B J Berne. Dynamical fluctuating charge force fields: Application to liquid water. *J. Chem. Phys.*, 101:6141–6156, 1994. doi:10.1063/1.468398.

[55] R Hentschke, E M Aydt, B Fodi, and E Schöckelmann. *Molekulares Modellieren mit Kraftfeldern*. Bergische Universität Wuppertal, Wuppertal, Germany, 2004. URL: http://constanze.materials.uni-wuppertal.de.

[56] Anders Öhrn, Jose M. Hermida-Ramon, and Gunnar Karlström. Method for slater-type density fitting for intermolecular electrostatic interactions with charge overlap. i. the model. *J. Chem. Theory Comput.*, 12(5):2298–2311, 2016.

[57] I I Guseinov. Analytical evaluation of two-centre coulomb, hybrid and one-electron integrals for slater-type orbitals. *J. Phys. B: Atom. Molec. Phys.*, 3(11):1399, 1970. doi:10.1088/0022-3700/3/11/001.

[58] D van der Spoel and P J van Maaren. The origin of layer structure artifacts in simulations of liquid water. *J. Chem. Theory Comput.*, 2:1–11, 2006. doi:10.1021/ct0502256.

[59] T Darden, D York, and L Pedersen. Particle mesh Ewald: An N-log(N) method for Ewald sums in large systems. *J. Chem. Phys.*, 98:10089–10092, 1993. doi:10.1063/1.464397.

[60] U Essmann, L Perera, M L Berkowitz, T Darden, H Lee, and L G Pedersen. A smooth particle mesh Ewald method. *J. Chem. Phys.*, 103:8577–8592, 1995. doi:10.1063/1.470117.

[61] Jesse G. McDaniel and J.R. Schmidt. Physically-motivated force fields from symmetry-adapted perturbation theory. *J. Phys. Chem. A.*, 117(10):2053–2066, 2013. doi:10.1021/jp3108182.

[62] Thomas A Halgren. The representation of van der Waals (vdW) interactions in molecular mechanics force fields: potential form, combination rules, and vdW parameters. *J. Amer. Chem. Soc.*, 114:7827–7843, 1992. doi:10.1021/ja00046a032.

[63] Jasper C Werhahn, Evangelos Miliordos, and Sotiris S Xantheas. A new variation of the buckingham exponential-6 potential with a tunable, singularity-free short-range repulsion and an adjustable long-range attraction. *Chem. Phys. Lett.*, 619:133–138, 2015. doi:10.1016/j.cplett.2014.11.051.

[64] Lee-Ping Wang, Jiahao Chen, and Troy Van Voorhis. Systematic Parametrization of Polarizable Force Fields from Quantum Chemistry Data. *J. Chem. Theory Comput.*, 9(1):452–460, 2013. doi:10.1021/ct300826t.

[65] R A Buckingham. The Classical Equation of State of Gaseous Helium, Neon and Argon. *Proc. R. Soc. London Ser. A*, 168:264–283, 1938. doi:10.1098/rspa.1938.0173.

[66] Edward A. Mason. Transport Properties of Gases Obeying a Modified Buckingham (Exp-Six) Potential. *J. Chem. Phys.*, 22:169–186, 2004. doi:10.1063/1.1740026.

[67] K T Tang and J P Toennies. An improved simple-model for the vanderwaals potential based on universal damping functions for the dispersion coefficients. *J. Chem. Phys.*, 80:3726–3741, 1984. doi:10.1063/1.447150.

[68] K. T. Tang and J. P. Toennies. The van der Waals potentials between all the rare gas atoms from He to Rn. *J. Chem. Phys.*, 118(11):4976–4983, 03 2003. URL: https://doi.org/10.1063/1.1543944, arXiv:https://pubs.aip.org/aip/jcp/article-pdf/118/11/4976/19209973/4976_1_online.pdf, doi:10.1063/1.1543944.

[69] Xiaowei Sheng, J. Peter Toennies, and K. T. Tang. Conformal analytical potential for all the rare gas dimers over the full range of internuclear distances. *Phys. Rev. Lett.*, 125:253402, 2020. doi:10.1103/PhysRevLett.125.253402.

[70] Mary J. Van Vleet, Alston J. Misquitta, Anthony J. Stone, and J. R. Schmidt. Beyond born-mayer: improved models for short-range repulsion in ab initio force fields. *J. Chem. Theory Comput.*, 12(8):3851–3870, 2016. doi:10.1021/acs.jctc.6b00209.

[71] Mary J. Van Vleet, Alston J. Misquitta, and J. R. Schmidt. New angles on standard force fields: toward a general approach for treating atomic-level anisotropy. *J. Chem. Theory Comput.*, 14(2):739–758, 2018. PMID: 29266931. doi:10.1021/acs.jctc.7b00851.

[72] J E Jones. On the determination of molecular fields. -II. From the equation of state of a gas. *Proc. Royal Soc. Lond. Ser. A*, 106:463–477, 1924. doi:10.1098/rspa.1924.0082.

[73] Christian L. Wennberg, Teemu Murtola, Berk Hess, and Erik Lindahl. Lennard-Jones lattice summation in bilayer simulations has critical effects on surface tension and lipid properties. *J. Chem. Theory Comput.*, 9:3527–3537, 2013. doi:10.1021/ct400140n.

[74] Chang Lyoul Kong and Manoj R Chakrabarty. Combining rules for intermolecular potential parameters. iii. application to the exp 6 potential. *J. Phys. Chem.*, 77(22):2668–2670, 1973. doi:10.1021/j100640a019.

[75] D Berthelot. Sur le mélange des gaz. *C. R. Hebd. Seances Acad. Sci.*, 126:1703–1855, 1898.

[76] H. A. Lorentz. Ueber die anwendung des satzes vom virial in der kinetischen theorie der gase. *Annalen der Physik*, 248(1):127–136, 1881. doi:10.1002/andp.18812480110.

[77] W Hogervorst. Transport and equilibrium properties of simple gases and forces between like and unlike atoms. *Physica*, 51(1):77–89, 1971. doi:10.1016/0031-8914(71)90138-8.

[78] Li Yang, Lei Sun, and Wei-Qiao Deng. Combination Rules for Morse-Based van der Waals Force Fields. *J. Phys. Chem. A*, 122:1672–1677, 2018. doi:10.1021/acs.jpca.7b11252.

[79] P M Morse. Diatomic molecules according to the wave mechanics. II. Vibrational levels. *Phys. Rev.*, 34:57–64, 1929. doi:10.1103/PhysRev.34.57.

[80] Edward A. Mason. Forces between Unlike Molecules and the Properties of Gaseous Mixtures. *J. Chem. Phys.*, 23:49–56, 1955. doi:10.1063/1.1740561.

[81] M Waldman and A T Hagler. New combining rules for rare gas van der waals parameters. *J. Comput. Chem.*, 14:1077–1084, 1993. doi:10.1002/jcc.540140909.

[82] Rui Qi, Qiantao Wang, and Pengyu Ren. General van der Waals potential for common organic molecules. *Bioorg. Med. Chem.*, 24:4911–4919, 2016. doi:10.1016/j.bmc.2016.07.062.

[83] Uwe Hohm. A continuous description of different means with application to mixing rules. *ACS Omega*, 11:10641–10648, 2026. doi:10.1021/acsomega.5c12377.

[84] W Hua. 4-parameter exactly solvable potential for diatomic-molecules. *Phys. Rev. A*, 42(5):2524–2529, SEP 1 1990. doi:10.1103/PhysRevA.42.2524.

[85] Wei Hua. Four-parameter potential and its bound-state matrix elements. *J. Phys. B: Atom., Molec. Opt. Phys.*, 23:2521, aug 1990. doi:10.1088/0953-4075/23/15/019.

[86] J P Ryckaert and A Bellemans. Molecular Dynamics of Liquid n-Butane near its Boiling Point. *Chem. Phys. Lett.*, 30:123–125, 1975. doi:10.1016/0009-2614(75)85513-8.

[87] David van der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E Mark, and Herman J C Berendsen. GROMACS: Fast, Flexible and Free. *J. Comput. Chem.*, 26:1701–1718, 2005. doi:10.1002/jcc.20291.

[88] Peter Eastman and Vijay S Pande. Efficient Nonbonded Interactions for Molecular Dynamics on a Graphics Processing Unit. *J. Comput. Chem.*, 31:1268–1272, 2010. doi:10.1002/jcc.21413.

[89] David van der Spoel and A. Najla Hosseini. Point + Gaussian charge model for electrostatic interactions derived by machine learning. *Phys. Chem. Chem. Phys.*, 27:13817–13820, 2025. doi:10.1039/D5CP01254F.

[90] M W Mahoney and W L Jorgensen. A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. *J. Chem. Phys.*, 112:8910–8922, 2000. doi:10.1063/1.481505.

[91] Shuyi Qin. Sensitivity analysis in high-dimensional space. Master's thesis, Uppsala University, Dept. of Information Technology, 2021. URL: https://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A1614100&dswid=6669.

[92] Steven F. van Dijk. *Genetic Algorithms for Map Labeling*. PhD thesis, Utrecht University, Dept. of Mathematics and Computer Science, 2001. URL: https://dspace.library.uu.nl/bitstream/handle/1874/864/full.pdf?sequence=1.

[93] G Lamoureux, A D MacKerell, and B Roux. A simple polarizable model of water based on classical Drude oscillators. *J. Chem. Phys.*, 119:5185–5197, 2003. doi:10.1063/1.1598191.

[94] Greg Landrum and RDKit contributors. Rdkit/rdkit: 2025_09_4 (q3 2025) release. 2025. doi:10.5281/zenodo.18098214.

[95] Daylight Chemical Information Systems, Inc. Smarts — a language for describing molecular patterns. Accessed: 2026-01-26. URL: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html.

[96] Brent H Besler, Kenneth M Merz Jr., and Peter A Kollman. Atomic charges derived from semiempirical methods. *J. Comput. Chem.*, 11:431–439, 1990. doi:10.1002/jcc.540110404.

[97] Toon Verstraelen, Veronique Van Speybroeck, and Michel Waroquier. The electronegativity equalization method and the split charge equilibration applied to organic systems: Parametrization, validation, and comparison. *J. Chem. Phys.*, 131:044127, 2009. doi:10.1063/1.3187034.

[98] A K Rappé and W A Goddard III. Charge Equillibration for Molecular Dynamics Simulations. *J. Phys. Chem.*, 95:3358–3363, 1991. doi:10.1021/j100161a070.

[99] Razvan A. Nistor, Jeliazko G. Polihronov, Martin H. Müser, and Nicholas J. Mosey. A generalization of the charge equilibration method for nonmetallic materials. *J. Chem. Phys.*, 125(9):094108, 2006. doi:10.1063/1.2346671.

[100] Chunhua Yin, Ziheng Cui, Yang Jiang, David van der Spoel, and Haiyang Zhang. Role of host-guest charge transfer in cyclodextrin complexation: a computational study. *J. Phys. Chem. C.*, 123:17745–17756, 2019. doi:10.1021/acs.jpcc.9b05399.

[101] Frank Jensen. Unifying charge-flow polarization models. *J. Chem. Theory Comput.*, 19:4047–4073, 2023. doi:10.1021/acs.jctc.3c00341.

[102] Peter Itskowitz and Max L Berkowitz. Chemical potential equalization principle: direct approach from density functional theory. *J. Phys. Chem. A*, 101(31):5687–5691, 1997. doi:10.1021/jp963962u.

[103] W J Mortier, S K Ghosh, and S Shankar. Electronegativity-equalization method for the calculation of atomic charges in molecules. *J. Amer. Chem. Soc.*, 108:4315–4320, 1986. doi:10.1021/ja00275a013.

[104] Riccardo Chelli, Piero Procacci, Roberto Righini, and Salvatore Califano. Electrical response in chemical potential equalization schemes. *J. Chem. Phys.*, 111(18):8569–8575, 1999. doi:10.1063/1.480198.

[105] Jiahao Chen, Dirk Hundertmark, and Todd J. Martínez. A unified theoretical framework for fluctuating-charge models in atom-space and in bond-space. *J. Chem. Phys.*, 129(21):214113, 2008. doi:10.1063/1.3021400.

[106] T. Verstraelen, P. W. Ayers, V. Van Speybroeck, and M. Waroquier. Acks2: atom-condensed kohn-sham dft approximated to second order. *J. Chem. Phys.*, 138:074108, 2013. doi:10.1063/1.4791569.

[107] Bogumil Jeziorski, Robert Moszynski, and Krzysztof Szalewicz. Perturbation theory approach to intermolecular potential energy surfaces of van der waals complexes. *Chem. Rev.*, 94(7):1887–1930, 1994. doi:10.1021/cr00031a008.

[108] Trent M. Parker, Lori A. Burns, Robert M. Parrish, Alden G. Ryno, and C. David Sherrill. Levels of symmetry adapted perturbation theory (sapt). I. efficiency and performance for interaction energies. *J. Chem. Phys.*, 140(9):094106, 2014. doi:10.1063/1.4867135.

[109] Grzegorz Chałasiński and Małgorzata M Szcześniak. State of the art and challenges of the ab initio theory of intermolecular interactions. *Chem. Rev.*, 100(11):4227–4252, 2000. doi:10.1021/cr990048z.

[110] Jan Řezáč. Non-covalent interactions atlas benchmark data sets: hydrogen bonding. *J. Chem. Theory Comput.*, 16:2355–2368, 2020. doi:10.1021/acs.jctc.9b01265.

[111] Lori A. Burns, John C. Faver, Zheng Zheng, Michael S. Marshall, Daniel G. A. Smith, Kenno Vanommeslaeghe, Alexander D. MacKerell, Kenneth M. Merz, and C. David Sherrill. The BioFragment Database (BFDb): An open-data platform for computational chemistry analysis of noncovalent interactions. *J. Chem. Phys.*, 147:161727, 2017. doi:10.1063/1.5001028.

[112] Justin S. Smith, Olexandr Isayev, and Adrian E. Roitberg. ANI-1, a data set of 20 million calculated off-equilibrium conformations for organic molecules. *Sci. Data*, 4:170193, 2017. doi:10.1038/sdata.2017.193.

[113] Justin M. Turney, Andrew C. Simmonett, Robert M. Parrish, Edward G. Hohenstein, Francesco A. Evangelista, Justin T. Fermann, Benjamin J. Mintz, Lori A. Burns, Jeremiah J. Wilke, Micah L. Abrams, Nicholas J. Russ, Matthew L. Leininger, Curtis L. Janssen, Edward T. Seidl, Wesley D. Allen, Henry F. Schaefer, Rollin A. King, Edward F. Valeev, C. David Sherrill, and T. Daniel Crawford. Psi4: an open-source ab initio electronic structure program. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 2:556–565, 2011. doi:10.1002/wcms.93.

[114] J Wang, R M Wolf, J W Caldwell, P A Kollman, and D A Case. Development and testing of a general AMBER force field. *J. Comput. Chem.*, 25:1157–1174, 2004. doi:10.1002/jcc.20035.

[115] Sander Pronk, Szilárd Páll, Roland Schulz, Per Larsson, Pär Bjelkmar, Rossen Apostolov, Michael R Shirts, Jeremy C Smith, Peter M Kasson, David van der Spoel, Berk Hess, and Erik Lindahl. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, 29:845–854, April 2013. doi:10.1093/bioinformatics/btt055.