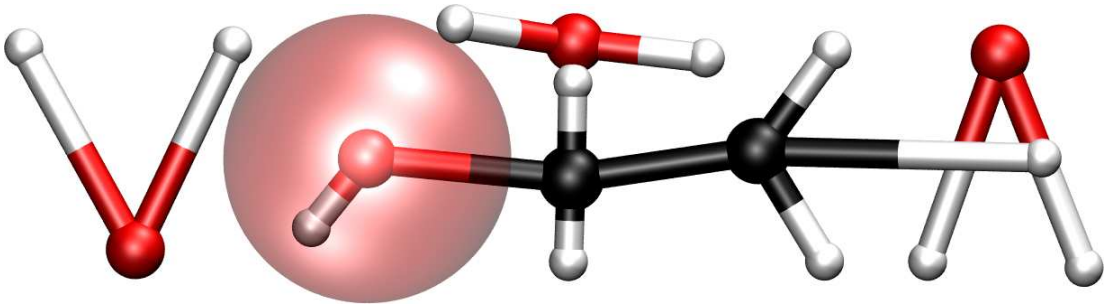
VOTCA
USER MANUAL



Versatile **O**bject-oriented **T**oolkit for **C**oarse-graining **A**pplications

Modular C++ kernel
Scripting for iterative workflow
Simple integration of other simulation packages

Iterative Boltzmann inversion
Inverse Monte Carlo
Force matching

September 18, 2019
Version: 1.6-dev (gitid: ef47114)

Programs version: 1.6-dev gitid: ef47114

www.votca.org

Disclamer

This manual is not complete. The best way to start using the software is to look at provided tutorials. The reference section is generated automatically from the source code, so please make sure that your software and manual versions match.

Citations

Development of this software depends on academic research grants. If you are using the package, please cite the following papers

[1] Relative entropy and optimization-driven coarse-graining methods in VOTCA,
S.Y. Mashayak, Mara Jochum, Konstantin Koschke, N.R. Aluru, Victor Rühle, and Christoph Junghans,
PLoS one 10, e131754 (2015)

[2] Hybrid approaches to coarse-graining using the VOTCA package: liquid hexane,
Victor Rühle and Christoph Junghans,
Macromol. Theory Simul. 20, 472 (2011)

[3] Versatile Object-oriented Toolkit for Coarse-graining Applications
Victor Rühle, Christoph Junghans, Alexander Lukyanov, Kurt Kremer, and Denis Andrienko
J. Chem. Theor. Comp. 5, 3211, 2009

Development

The core development is currently taking place at the Los Alamos National Laboratory and Max Planck Institute for Polymer Research, Mainz, Germany.

Copyright

Contents

1 Introduction **1**

2 Theoretical background **3**
2.1 Mapping . . . 3
2.2 Boltzmann inversion . . . 4
2.2.1 Separation of bonded and non-bonded degrees of freedom . . . 5
2.3 Iterative methods . . . 6
2.4 Iterative Boltzmann Inversion . . . 6
2.5 Inverse Monte Carlo . . . 6
2.5.1 Regularization of Inverse Monte Carlo . . . 7
2.6 Force Matching . . . 7
2.7 Relative Entropy . . . 8

3 Input files **11**
3.1 Mapping files . . . 11
3.2 Verification of a mapping . . . 12
3.3 Advanced topology handling . . . 12
3.4 Trajectories . . . 15
3.5 Setting files . . . 15
3.6 Table formats . . . 16

4 Preparing coarse-grained runs **17**
4.1 Generating a topology file for a coarse-grained run . . . 17
4.2 Post-processing of the potential . . . 17
4.2.1 Clipping of poorly sampled regions . . . 18
4.2.2 Resampling . . . 18
4.2.3 Extrapolation . . . 18
4.2.4 Exporting the table . . . 18
4.2.5 An example on non-bonded interactions . . . 19
4.3 Alternatives . . . 20

5 Boltzmann Inversion **21**
5.1 Generating exclusion lists . . . 21
5.2 Statistical analysis . . . 22
5.2.1 Distribution functions and tabulated potentials . . . 22
5.2.2 Correlation analysis . . . 23

6 Force matching **25**
6.1 Program input . . . 25
6.2 Program output . . . 26
6.3 Integration and extrapolation of .force files . . . 26

7 Iterative methods **27**
7.1 Iterative workflow control . . . 27
7.1.1 Preparing the run . . . 28
7.1.2 Starting the iterative process . . . 29
7.1.3 Restarting and continuing . . . 29
7.2 Iterative Boltzmann Inversion . . . 31
7.2.1 Input preparation . . . 31
7.3 Inverse Monte Carlo . . . 31
7.3.1 General considerations . . . 31
7.3.2 Correlation groups . . . 31
7.3.3 Regularization . . . 32
7.4 Relative Entropy . . . 32
7.4.1 Potential function and parameters . . . 32
7.4.2 Update scaling parameter . . . 33
7.4.3 Statistical averaging of parameters . . . 33
7.4.4 General considerations . . . 33
7.5 Pressure correction . . . 34
7.5.1 Simple pressure correction . . . 34
7.5.2 Advanced pressure correction . . . 34
7.6 Kirkwood-Buff correction . . . 35
7.7 Runtime optimization . . . 35
7.8 Coordination Iterative Boltzmann Inversion . . . 36

8 DL_POLY interface **37**
8.1 General remarks on using VOTCA with DL_POLY . . . 37

9 Advanced topics **39**
9.1 Customization . . . 39
9.2 Used external packages . . . 40
9.2.1 GroMaCS . . . 40
9.2.2 ESPResSo . . . 40
9.2.3 DL_POLY . . . 40
9.2.4 Gnuplot . . . 40
9.2.5 GNU Octave . . . 40
9.2.6 LAMMPS . . . 41
9.2.7 Matlab . . . 41
9.2.8 NumPy . . . 41

10 Reference **43**
10.1 Programs . . . 43
10.1.1 csg_boltzmann . . . 43
10.1.2 csg_call . . . 43
10.1.3 csg_density . . . 44
10.1.4 csg_dlptopol . . . 44
10.1.5 csg_dump . . . 45
10.1.6 csg_fmatch . . . 45
10.1.7 csg_gmxtopol . . . 45
10.1.8 csg_imcrepack . . . 46
10.1.9 csg_inverse . . . 46
10.1.10 csg_map . . . 46
10.1.11 csg_property . . . 47
10.1.12 csg_resample . . . 47
10.1.13 csg_reupdate . . . 47
10.1.14 csg_stat . . . 48

10.2 Mapping file . . . 48
10.3 Topology file . . . 49
10.4 Settings file . . . 50
10.5 Scripts . . . 58
10.5.1 add_POT.pl . . . 62
10.5.2 add_pot_generic.sh . . . 62
10.5.3 calc_density_generic.sh . . . 62
10.5.4 calc_kbint.sh . . . 62
10.5.5 calc_pressure_gromacs.sh . . . 63
10.5.6 calc_pressure_lammps.sh . . . 63
10.5.7 calc_rdf_generic.sh . . . 63
10.5.8 calc_target_rdf_generic.sh . . . 63
10.5.9 clean_generic.sh . . . 64
10.5.10 cma_processor.py . . . 64
10.5.11 configuration_compare.py . . . 64
10.5.12 convergence_check_default.sh . . . 64
10.5.13 dist_adjust.pl . . . 64
10.5.14 dist_boltzmann_invert.pl . . . 64
10.5.15 dpot_crop.pl . . . 65
10.5.16 dummy.sh . . . 65
10.5.17 functions_common.sh . . . 65
10.5.18 functions_dlpoly.sh . . . 66
10.5.19 functions_genericsim.sh . . . 67
10.5.20 functions_gromacs.sh . . . 67
10.5.21 imc_purify.sh . . . 67
10.5.22 imc_stat_generic.sh . . . 68
10.5.23 initialize_step_generic.sh . . . 68
10.5.24 initialize_step_genericsim.sh . . . 68
10.5.25 initialize_step_optimizer.sh . . . 68
10.5.26 initialize_step_re.sh . . . 68
10.5.27 inverse.sh . . . 69
10.5.28 kbibi_ramp_correction.pl . . . 69
10.5.29 linsolve.m . . . 69
10.5.30 linsolve.octave . . . 69
10.5.31 linsolve.py . . . 69
10.5.32 lj_126.pl . . . 70
10.5.33 merge_tables.pl . . . 70
10.5.34 optimizer_parameters_to_potential.sh . . . 70
10.5.35 optimizer_prepare_state.sh . . . 70
10.5.36 optimizer_state_to_mapping.sh . . . 70
10.5.37 optimizer_state_to_potentials.sh . . . 71
10.5.38 optimizer_target_density.sh . . . 71
10.5.39 optimizer_target_pressure.sh . . . 71
10.5.40 optimizer_target_rdf.sh . . . 71
10.5.41 post_add.sh . . . 72
10.5.42 post_add_single.sh . . . 72
10.5.43 post_update_generic.sh . . . 72
10.5.44 post_update_generic_single.sh . . . 72
10.5.45 post_update_re_single.sh . . . 72
10.5.46 postadd_acc_convergence.sh . . . 72
10.5.47 postadd_average.sh . . . 73
10.5.48 postadd_compress.sh . . . 73
10.5.49 postadd_convergence.sh . . . 73
10.5.50 postadd_copyback.sh . . . 73

10.5.51 postadd_dummy.sh . . . 73
10.5.52 postadd_overwrite.sh . . . 74
10.5.53 postadd_plot.sh . . . 74
10.5.54 postadd_shift.sh . . . 74
10.5.55 postupd_addlj.sh . . . 74
10.5.56 postupd_cibi_correction.sh . . . 74
10.5.57 postupd_extrapolate.sh . . . 75
10.5.58 postupd_kbibi_correction.sh . . . 75
10.5.59 postupd_pressure.sh . . . 75
10.5.60 postupd_scale.sh . . . 76
10.5.61 postupd_smooth.sh . . . 76
10.5.62 postupd_splinesmooth.sh . . . 76
10.5.63 potential_extrapolate.sh . . . 76
10.5.64 potential_shift.pl . . . 77
10.5.65 potential_to_dlpoly.sh . . . 77
10.5.66 potential_to_generic.sh . . . 77
10.5.67 potential_to_gromacs.sh . . . 77
10.5.68 potential_to_lammps.sh . . . 78
10.5.69 potentials_to_dlpoly.sh . . . 78
10.5.70 potentials_to_generic.sh . . . 78
10.5.71 pre_update_re.sh . . . 79
10.5.72 prepare_generic.sh . . . 79
10.5.73 prepare_generic_single.sh . . . 79
10.5.74 prepare_ibm.sh . . . 79
10.5.75 prepare_imc.sh . . . 79
10.5.76 prepare_optimizer.sh . . . 80
10.5.77 prepare_optimizer_single.sh . . . 80
10.5.78 prepare_re.sh . . . 80
10.5.79 pressure_cor_simple.pl . . . 80
10.5.80 pressure_cor_wjk.pl . . . 80
10.5.81 resample_target.sh . . . 80
10.5.82 run_genericsim.sh . . . 81
10.5.83 run_gromacs.sh . . . 81
10.5.84 simplex_downhill_processor.pl . . . 81
10.5.85 solve_matlab.sh . . . 81
10.5.86 solve_numpy.sh . . . 82
10.5.87 solve_octave.sh . . . 82
10.5.88 table_average.sh . . . 82
10.5.89 table_change_flag.sh . . . 82
10.5.90 table_combine.pl . . . 82
10.5.91 table_dummy.sh . . . 83
10.5.92 table_extrapolate.pl . . . 83
10.5.93 table_functional.sh . . . 83
10.5.94 table_get_value.pl . . . 84
10.5.95 table_integrate.pl . . . 84
10.5.96 table_linearop.pl . . . 84
10.5.97 table_scale.pl . . . 84
10.5.98 table_smooth.pl . . . 84
10.5.99 table_switch_border.pl . . . 85
10.5.100 table_to_tab.pl . . . 85
10.5.101 table_to_xvg.pl . . . 85
10.5.102 tables_jackknife.pl . . . 85
10.5.103 tag_file.sh . . . 85
10.5.104 update_ibi.sh . . . 85

10.5.105 update_ibi_pot.pl 86
10.5.106 update_ibi_single.sh 86
10.5.107 update_ibm.sh 86
10.5.108 update_imc.sh 86
10.5.109 update_optimizer.sh 86
10.5.110 update_optimizer_single.sh 87
10.5.111 update_re.sh 87

Chapter 1

Introduction

Versatile Object-oriented Toolkit for Coarse-graining Applications, or VOTCA, is a package which helps to systematically coarse-grain various systems [3]. This includes deriving the coarse-grained potentials, assessing their quality, preparing input files required for coarse-grained simulations, and analyzing the latter.

A typical coarse-graining workflow includes *sampling* of the system of interest, *analysis* of the trajectory using a specific *mapping* and a coarse-graining *method* to derive coarse-grained potentials and, in case of iterative methods, running coarse-grained simulations and iteratively *refining* the coarse-grained potentials.

In most cases, coarse-graining requires canonical sampling of a reference (high resolution) system. In addition, iterative methods require canonical sampling of the coarse-grained system. The sampling can be done using either molecular dynamics (MD), stochastic dynamics (SD), or Monte Carlo (MC) techniques. The latter are implemented in many standard simulation packages. Rather than implementing its own MD/SD/MC modules, VOTCA allows swift and flexible integration of existing programs in such a way that sampling is performed by the program of choice. At the moment, an interface to GROMACS [4] simulation package is provided. The rest of the analysis needed for systematic coarse-graining is done using the package tools.

The workflow can be exemplified on coarse-graining of a propane liquid. A single molecule of propane contains three carbon and eight hydrogen atoms. A united atom coarse-grained representation of a propane molecule has three beads and two bead types, A and B, with three and two hydrogens combined with the corresponding atom, as shown in fig. 1.1. This representation defines the mapping operator, as well as the bonded coarse-grained degrees of freedom, such as the bond b and the bond angle θ. Apart from the bonded interactions, u_b and u_θ, beads belonging to different molecules have non-bonded interactions, u_{AA}, u_{AB}, u_{BB}. The task of coarse-graining is then to derive a potential energy surface u which is a function of all coarse-grained degrees of freedom. Note that, while the atomistic bond and angle potentials are often chosen to be simple harmonic functions, the coarse-grained potentials cannot be expressed in terms of simple analytic functions. Instead, tabulated functions are normally used.

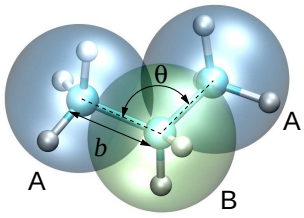


Figure 1.1: Three-bead coarse-grained model of propane.

The coarse-graining *method* defines criteria according to which the potential energy surface is constructed. For example, for the bond b and the angle θ Boltzmann Inversion can be used. In this case a coarse-grained potential will be a potential of mean force. For the non-bonded degrees of freedom, the package provides Iterative Boltzmann Inversion (IBI) or Inverse Monte Carlo (IMC) methods. In this case the radial distribution functions of the coarse-grained model will match those of the atomistic model. Alternatively, Force Matching (FM) (or multiscale coarse-graining) can be used, in which case the coarse-grained potential will approximate the many-body potential of mean force. The choice of a particular method is system-specific and requires a thorough consistency

check. It is important to keep in mind that coarse-graining should be used with understanding and caution, methods should be crossed-checked with each other as well as with respect to the reference system.

The package consists of two parts: a C++ kernel and a scripting engine. The kernel is capable of processing atomistic topologies and trajectories and offers a flexible framework for reading, manipulating and analyzing topologies and generated by MD/SD/MC sampling trajectories. It is modular: new file formats can be integrated without changing the existing code. Currently, an interface for GROMACS [4] topologies and trajectories is provided. The kernel also includes various coarse-graining tools, for example calculations of probability distributions of bonded and non-bonded interactions, correlation and autocorrelation functions, and updates for the coarse-grained pair potential.

The scripting engine is used to steer the iterative procedures. Here the analysis tools of the package used for sampling (e.g. GROMACS tools) can be integrated into the coarse-graining workflow, if needed. The coarse-graining workflow itself is controlled by several Extensible Markup Language (XML) input files, which contain mapping and other options required for the workflow control. In what follows, these input files are described.

Before using the package, do not forget to initalize the variables in the bash or csh (tcsh)

```
source <csg-installation>/bin/VOTCARC.bash
source <csg-installation>/bin/VOTCARC.csh
```

More details as well as several examples can be found in ref. [3]. Please cite this paper if you are using the package. Tutorials can be found on the VOTCA homepage WWW.VOTCA.ORG.

Chapter 2

Theoretical background

2.1 Mapping

The mapping is an operator that establishes a link between the atomistic and coarse-grained representations of the system. An atomistic system is described by specifying the values of the Cartesian coordinates and momenta

$$\boldsymbol{r}^n = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n\}, \tag{2.1}$$
$$\boldsymbol{p}^n = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}. \tag{2.2}$$

of the n atoms in the system.[1] On a coarse-grained level, the coordinates and momenta are specified by the positions and momenta of CG sites

$$\boldsymbol{R}^N = \{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_N\}, \tag{2.3}$$
$$\boldsymbol{P}^N = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_N\}. \tag{2.4}$$

Note that capitalized symbols are used for the CG sites while lower case letters are used for the atomistic system.

The mapping operator $\boldsymbol{c}_I$ is defined by a matrix for each bead I and links the two descriptions

$$\boldsymbol{R}_I = \sum_{i=1}^{n} c_{Ii}\boldsymbol{r}_i, \tag{2.5}$$

$$\boldsymbol{P}_I = M_I\dot{\boldsymbol{R}}_I = M_I\sum_{i=1}^{n} c_{Ii}\dot{\boldsymbol{r}}_i = M_I\sum_{i=1}^{n}\frac{c_{Ii}}{m_i}\boldsymbol{p}_i. \tag{2.6}$$

for all $I = 1, \ldots, N$.

If an atomistic system is translated by a constant vector, the corresponding coarse-grained system is also translated by the same vector. This implies that, for all I,

$$\sum_{i=1}^{n} c_{Ii} = 1. \tag{2.7}$$

In some cases it is useful to define the CG mapping in such a way that certain atoms belong to several CG beads at the same time [6]. Following ref. [5], we define two sets of atoms for each of the N CG beads. For each site I, a set of *involved* atoms is defined as

$$\mathcal{I}_I = \{i | c_{Ii} \neq 0\}. \tag{2.8}$$

[1] In what follows we adopt notations of ref. [5].

An atom i in the atomistic model is involved in a CG site, I, if and only if this atom provides a nonzero contribution to the sum in eq. 2.6.

A set of *specific* atoms is defined as

$$\mathcal{S}_I = \{i | c_{Ii} \neq 0 \text{ and } c_{Ji} = 0 \text{ for all } J \neq I\}. \tag{2.9}$$

In other words, atom i is specific to site I if and only if this atom is involved in site I and is not involved in the definition of any other site.

The CG model will generate an equilibrium distribution of momenta that is consistent with an underlying atomistic model if all the atoms are *specific* and if the mass of the I^{th} CG site is given by [5]

$$M_I = \left(\sum_{i \in \mathcal{I}_I} \frac{c_{Ii}^2}{m_i} \right)^{-1} . \tag{2.10}$$

If all atoms are specific and the center of mass of a bead is used for mapping, then $c_{Ii} = \frac{m_i}{M_I}$, and the condition 2.10 is automatically satisfied.

2.2 Boltzmann inversion

Boltzmann inversion is mostly used for *bonded* potentials, such as bonds, angles, and torsions [7]. Boltzmann inversion is structure-based and only requires positions of atoms.

The idea of Boltzmann inversion stems from the fact that in a canonical ensemble *independent* degrees of freedom q obey the Boltzmann distribution, i. e.

$$P(q) = Z^{-1} \exp\left[-\beta U(q)\right] \ , \tag{2.11}$$

where $Z = \int \exp\left[-\beta U(q)\right] \mathrm{d}q$ is a partition function, $\beta = 1/k_{\mathrm{B}}T$. Once $P(q)$ is known, one can obtain the coarse-grained potential, which in this case is a potential of mean force, by inverting the probability distribution $P(q)$ of a variable q, which is either a bond length, bond angle, or torsion angle

$$U(q) = -k_{\mathrm{B}}T \ln P(q) \ . \tag{2.12}$$

The normalization factor Z is not important since it would only enter the coarse-grained potential $U(q)$ as an irrelevant additive constant.

Note that the histograms for the bonds $H_r(r)$, angles $H_\theta(\theta)$, and torsion angles $H_\varphi(\varphi)$ have to be rescaled in order to obtain the volume normalized distribution functions $P_r(r)$, $P_\theta(\theta)$, and $P_\varphi(\varphi)$, respectively,

$$P_r(r) = \frac{H_r(r)}{4\pi r^2} \ , \ P_\theta(\theta) = \frac{H_\theta(\theta)}{\sin\theta} \ , \ P_\varphi(\varphi) = H_\varphi(\varphi) \ , \tag{2.13}$$

where r is the bond length r, θ is the bond angle, and φ is the torsion angle. The bonded coarse-grained potential can then be written as a sum of distribution functions

$$\begin{aligned} U(r,\theta,\varphi) &= U_r(r) + U_\theta(\theta) + U_\varphi(\varphi) \ , \\ U_q(q) &= -k_{\mathrm{B}}T \ln P_q(q), \ q = r, \theta, \varphi \ . \end{aligned} \tag{2.14}$$

On the technical side, the implementation of the Boltzmann inversion method requires *smoothing* of $U(q)$ to provide a continuous force. Splines can be used for this purpose. Poorly and unsampled regions, that is regions with high $U(q)$, shall be *extrapolated*. Since the contribution of these regions to the canonical density of states is small, the exact shape of the extrapolation is less important.

Another crucial issue is the cross-correlation of the coarse-grained degrees of freedom. Independence of the coarse-grained degrees of freedom is the main assumption that allows factorization of the probability distribution and the potential, eq. 2.14. Hence, one has to carefully check whether

this assumption holds in practice. This can be done by performing coarse-grained simulations and comparing cross-correlations for all pairs of degrees of freedom in atomistic and coarse-grained resolution, e. g. using a two-dimensional histogram, analogous to a Ramachandran plot. [2]

2.2.1 Separation of bonded and non-bonded degrees of freedom

When coarse-graining polymeric systems, it is convenient to treat bonded and non-bonded interactions separately [7]. In this case, sampling of the atomistic system shall be performed on a special system where non-bonded interactions are artificially removed, so that the non-bonded interactions in the reference system do not contribute to the bonded interactions of the coarse-grained model.

This can be done by employing exclusion lists using **csg_boltzmann** with the option `--excl`. This is described in detail in sec. 5.1.

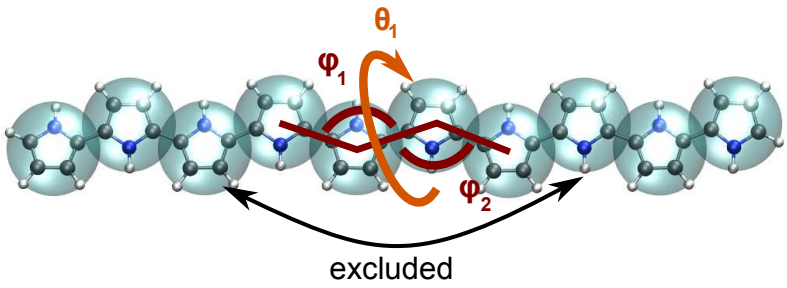


Figure 2.1: Example of excluded interactions.

[2] Checking the linear correlation coefficient does not guarantee statistical independence of variables, for example $c(x, x^2) = 0$ if x has a symmetric probability density $P(x) = P(-x)$. This case is often encountered in systems used for coarse-graining.

2.3 Iterative methods

Iterative workflow control is essential for the IBI and IMC methods. The general idea of iterative workflow is sketched in fig. 2.2. A run starts with an initial guess during the global initialization phase. This guess is used for the first sampling step, followed by an update of the potential. The update itself often requires additional postprocessing such as smoothing, interpolation, extrapolation or fitting. Different methods are available to update the potential, for instance Iterative Boltzmann Inversion (see next section 2.4) or Inverse Monte Carlo (see section 2.5). The whole procedure is then iterated until a convergence criterion is satisfied.

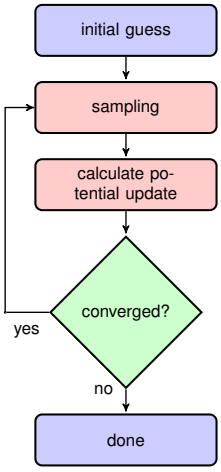


Figure 2.2: Blockscheme of an iterative method.

2.4 Iterative Boltzmann Inversion

Iterative Boltzmann inversion (IBI) is a natural extension of the Boltzmann inversion method. Since the goal of the coarse-grained model is to reproduce the distribution functions of the reference system as accurately as possible, one can also iteratively refine the coarse-grained potentials using some numerical scheme.

In IBI the potential update ΔU is given by [8]

$$U^{(n+1)} = U^{(n)} + \lambda \Delta U^{(n)} \,, \tag{2.15}$$

$$\Delta U^{(n)} = k_\mathrm{B} T \ln \frac{P^{(n)}}{P_\mathrm{ref}} = U^\mathrm{ref}_\mathrm{PMF} - U^{(n)}_\mathrm{PMF} \,. \tag{2.16}$$

Here $\lambda \in (0, 1]$ is a numerical factor which helps to stabilize the scheme.

The convergence is reached as soon as the distribution function $P^{(n)}$ matches the reference distribution function P_ref, or, in other words, the potential of mean force, $U^{(n)}_\mathrm{PMF}$, converges to the reference potential of mean force.

IBI can be used to refine both bonded and non-bonded potentials. It is primarily used for simple fluids with the aim to reproduce the radial distribution function of the reference system in order to obtain non-bonded interactions. On the implementation side, IBI has the same issues as the inverse Boltzmann method, i. e. smoothing and extrapolation of the potential must be used.

2.5 Inverse Monte Carlo

Inverse Monte Carlo (IMC) is an iterative scheme which additionally includes cross correlations of distributions. A detailed derivation of the IMC method can be found in ref. [9].

The potential update ΔU of the IMC method is calculated by solving a set of linear equations

$$\langle S_\alpha \rangle - S^\mathrm{ref}_\alpha = A_{\alpha\gamma} \Delta U_\gamma \,, \tag{2.17}$$

where

$$A_{\alpha\gamma} = \frac{\partial \langle S_\alpha \rangle}{\partial U_\gamma} = \beta \left(\langle S_\alpha \rangle \langle S_\gamma \rangle - \langle S_\alpha S_\gamma \rangle \right) \,,$$

and S the histogram of a coarse-grained variable of interest. For example, in case of coarse-graining of the non-bonded interactions which depend only on the distance r_{ij} between particles i and j and assuming that the interaction potential is short-ranged, i.e. $U(r_{ij}) = 0$ if $r_{ij} \geq r_\mathrm{cut}$, the average value of S_α is related to the radial distribution function $g(r_\alpha)$ by

$$\langle S_\alpha \rangle = \frac{N(N-1)}{2} \frac{4\pi r_\alpha^2 \Delta r}{V} g(r_\alpha) \,, \tag{2.18}$$

where N is the number of atoms in the system ($\frac{1}{2}N(N-1)$ is then the number of all pairs), Δr is the grid spacing, r_{cut}/M, V is the total volume of the system. In other words, in this particular case the physical meaning of S_α is the number of particle pairs with interparticle distances $r_{ij} = r_\alpha$ which correspond to the tabulated value of the potential U_α.

2.5.1 Regularization of Inverse Monte Carlo

To get a well defined cross correlation matrix, $A_{\alpha\gamma}$, enough sampling is needed. If there is not enough smapling or the initial potential guess is far from the real solution of the inverse problem, the algorithm might not converge to a stable solution. To overcome this instability problem one could reformulate equation 2.18 by addition of a penalty term. In this case the potential update is computed as follows:[10]

$$\Delta U_\gamma = \arg\min \|A_{\alpha\gamma}\Delta U_\gamma - \left(\langle S_\alpha\rangle - S_\alpha^{\text{ref}}\right)\|^2 + \lambda\|R\Delta U_\gamma\|^2 \tag{2.19}$$

Equation 2.19 is known as Tikhonov regularization, where R is the regularization operator, which here is the identity matrix and $\lambda > 0$ is the regularization parameter. The optimal choice for λ can only be determined if the exact solution of the inverse problem is known, which in practice is not the case. To get a good initial guess on the magnitude of the regularization parameter a singular value decomposition of the matrix $A_{\alpha\gamma}$ might help. A good λ parameter should dominate the smallest singular values (squared) but is itself small compared to the larger ones.[11]

2.6 Force Matching

Force matching (FM) is another approach to evaluate corse-grained potentials [12–14]. In contrast to the structure-based approaches, its aim is not to reproduce various distribution functions, but instead to match the multibody potential of mean force as close as possible with a given set of coarse-grained interactions.

The method works as follows. We first assume that the coarse-grained force-field (and hence the forces) depends on M parameters $g_1, ..., g_M$. These parameters can be prefactors of analytical functions, tabulated values of the interaction potentials, or coefficients of splines used to describe these potentials.

In order to determine these parameters, the reference forces on coarse-grained beads are calculated by summing up the forces on the atoms

$$\boldsymbol{F}_I^{\text{ref}} = \sum_{j\in\mathcal{S}_\mathcal{I}} \frac{d_{Ii}}{c_{Ii}} \boldsymbol{f}_j(\boldsymbol{r}^n), \tag{2.20}$$

where the sum is over all atoms of the CG site I (see. sec. 2.1). The d_{Ij} coefficients can, in principle, be chosen arbitrarily, provided that the condition $\sum_{i=1}^{n} d_{Ii} = 1$ is satisfied [5]. If mapping coefficients for the forces are not provided, it is assumed that $d_{Ij} = c_{Ij}$ (see also sec. 3).

By calculating the reference forces for L snapshots we can write down $N \times L$ equations

$$\boldsymbol{F}_{Il}^{\text{cg}}(g_1, \ldots, g_M) = \boldsymbol{F}_{il}^{\text{ref}},\ I = 1, \ldots, N,\ l = 1, \ldots, L\ . \tag{2.21}$$

Here $\boldsymbol{F}_{Il}^{\text{ref}}$ is the force on the bead I and $\boldsymbol{F}_{Il}^{\text{cg}}$ is the coarse-grained representation of this force. The index l enumerates snapshots picked for coarse-graining. By running the simulations long enough one can always ensure that $M < N \times L$. In this case the set of equations 2.21 is overdetermined and can be solved in a least-squares manner.

$\boldsymbol{F}_{il}^{\text{cg}}$ is, in principle, a non-linear function of its parameters $\{g_i\}$. Therefore, it is useful to represent the coarse-grained force-field in such a way that equations (2.21) become linear functions of $\{g_i\}$. This can be done using splines to describe the functional form of the forces [13]. Implementation details are discussed in ref. [3].

Note that an adequate sampling of the system requires a large number of snapshots L. Hence, the applicability of the method is often constrained by the amount of memory available. To remedy

the situation, one can split the trajectory into blocks, find the coarse-grained potential for each block and then perform averaging over all blocks.

2.7 Relative Entropy

Relative entropy is a method which quantifies the extent of the configurational phase-space overlap between two molecular ensembles [15]. It can be used as a measure of the discrepancies between various properties of the CG system's and the target all-atom (AA) ensemble. It has been shown by Shell S. [16] that one can minimize the relative entropy metric between the model CG system and the target AA system to optimize CG potential parameters such that the CG ensemble would mimic the target AA ensemble.

Relative entropy, S_{rel}, is defined as [16]

$$S_{\text{rel}} = \sum_i p_{\text{AA}}(r_i) \ln \left(\frac{p_{\text{AA}}(r_i)}{p_{\text{CG}}\left(M(r_i)\right)} \right) + \langle S_{\text{map}} \rangle_{\text{AA}}, \tag{2.22}$$

where the sum is over all the configurations of the reference AA system, $r = \{r_i\}(i = 1, 2, ...)$, M is the mapping operation to generate a corresponding CG configuration, R_I, from a AA configuration, r_i, i.e., $R_I = M(r_i)$, p_{AA} and p_{CG} are the configurational probabilities based on the AA and CG potentials, respectively, and $\langle S_{\text{map}} \rangle_{\text{AA}}$ is the mapping entropy due to the average degeneracy of AA configurations mapping to the same CG configuration, given by

$$S_{\text{map}}(R_I) = \ln \sum_i \delta_{R_I, M(r_i)}, \tag{2.23}$$

where δ is the Kronecker delta function. Physically, S_{rel} can be interpreted as the likelihood that one test configuration of the model CG ensemble is representative of the target AA ensemble, and when the likelihood is a maximum, S_{rel} is at a minimum. Hence, the numerical minimization of S_{rel} with respect to the parameters of the CG model can be used to optimize the CG model.

In a canonical ensemble, substituting canonical configurational probabilities into eq. 2.22, the relative entropy simplifies to

$$S_{\text{rel}} = \beta \langle U_{\text{CG}} - U_{\text{AA}} \rangle_{\text{AA}} - \beta \left(A_{\text{CG}} - A_{\text{AA}} \right) + \langle S_{\text{map}} \rangle_{\text{AA}}, \tag{2.24}$$

where $\beta = 1/k_{\text{B}}T$, k_{B} is the Boltzmann constant, T is the temperature, U_{CG} and U_{AA} are the total potential energies from the CG and AA potentials, respectively, A_{CG} and A_{AA} are the configurational part of the Helmholtz free energies from the CG and AA potentials, respectively, and all the averages are computed in the reference AA ensemble.

Consider a model CG system defined by the CG potentials between various CG sites such that the CG potentials depend on the parameters $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, ...\lambda_n\}$. Then $\boldsymbol{\lambda}$ are optimized by the relative entropy minimization. We use the Newton-Raphson strategy for the relative entropy minimization described in ref. [17]. In this strategy, the CG potential parameters, $\boldsymbol{\lambda}$, are refined iteratively as

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k - \chi \mathbf{H}^{-1} \cdot \nabla_\lambda S_{\text{rel}}, \tag{2.25}$$

where k is the iteration index, $\chi \in (0...1)$ is the scaling parameter that can be adjusted to ensure convergence, $\nabla_\lambda S_{\text{rel}}$ is the vector of the first derivatives of S_{rel} with respect to $\boldsymbol{\lambda}$, which can be computed from eq. 2.24 as

$$\nabla_\lambda S_{\text{rel}} = \beta \left\langle \frac{\partial U_{\text{CG}}}{\partial \lambda} \right\rangle_{\text{AA}} - \beta \left\langle \frac{\partial U_{\text{CG}}}{\partial \lambda} \right\rangle_{\text{CG}}, \tag{2.26}$$

and $\mathbf{H}$ is the Hessian matrix of S_{rel} given by

$$\begin{aligned}\mathbf{H}_{ij} &= \beta\left\langle\frac{\partial^2 U_{\text{CG}}}{\partial\lambda_i\partial\lambda_j}\right\rangle_{\text{AA}} - \beta\left\langle\frac{\partial^2 U_{\text{CG}}}{\partial\lambda_i\partial\lambda_j}\right\rangle_{\text{CG}} \\ &\quad +\beta^2\left\langle\frac{\partial U_{\text{CG}}}{\partial\lambda_i}\frac{\partial U_{\text{CG}}}{\partial\lambda_j}\right\rangle_{\text{CG}} \\ &\quad -\beta^2\left\langle\frac{\partial U_{\text{CG}}}{\partial\lambda_i}\right\rangle_{\text{CG}}\left\langle\frac{\partial U_{\text{CG}}}{\partial\lambda_j}\right\rangle_{\text{CG}}. \end{aligned} \tag{2.27}$$

To compute $\nabla_\lambda S_{\text{rel}}$ and $\mathbf{H}$ from eq. 2.26 and 2.27, we need average CG energy derivatives in the AA and CG ensembles. For two-body CG pair potentials, u_{CG}, between CG sites, the ensemble averages of the CG energy derivatives can be computed as

$$\begin{aligned}\left\langle\left(\frac{\partial^a U_{\text{CG}}}{\partial\lambda^a}\right)^b\right\rangle_{\text{AA}} &= \left\langle\left(\sum_{i<j}\frac{\partial^a u_{\text{CG}}(r_{ij})}{\partial\lambda^a}\right)^b\right\rangle_{\text{AA}} \\ \left\langle\left(\frac{\partial^a U_{\text{CG}}}{\partial\lambda^a}\right)^b\right\rangle_{\text{CG}} &= \left\langle\left(\sum_{i<j}\frac{\partial^a u_{\text{CG}}(r_{ij})}{\partial\lambda^a}\right)^b\right\rangle_{\text{CG}}, \end{aligned} \tag{2.28}$$

where the sum is performed over all the CG site pairs (i, j), a stands for the 1^{st}, 2^{nd},... derivatives and b stands for the different powers, i.e., $b = 1, 2, ...$ For the averages in the AA ensemble, first a single AA system simulation can be performed and RDFs between the CG sites in the AA ensemble can be saved, then the average CG energy derivatives in AA ensemble can be computed by processing the CG RDFs in the AA ensemble using the CG potentials at each iteration. For the averages in the CG ensemble, since the CG ensemble changes with the CG parameters, $\boldsymbol{\lambda}$, a short CG simulation is performed at each iteration to generate corresponding CG configurations.

Comparisons between relative entropy and other coarse-graining methods are made in ref. [18] and [17]. Chaimovich and Shell [17] have shown that for certain CG models relative entropy minimization produces the same CG potentials as other methods, e.g., it is equivalent to the IBI when CG interactions are modeled using finely tabulated pair additive potentials, and to the FM when a CG model is based on $N-$body interactions, where N is the number of degrees of freedom in the CG model. However, there are some advantages of using relative entropy based coarse-graining. Relative entropy method allows to use analytical function forms for CG potentials, which are desired in theoretical treatments, such as parametric study of CG potentials, whereas, methods, like IBI, use tabulated potentials. Recently Lyubartsev et. al [19] have shows how to use IMC with an analytical function form, too. BI, IBI, and IMC methods are based on pair correlations and hence, they are only useful to optimize 2-body CG potentials, whereas, relative entropy uses more generic metric which offers more flexibility in modeling CG interactions and not only 2-body, but also 3-body (for example see ref. [20]) and N-body CG potentials can be optimized. In addition to the CG potential optimization, the relative entropy metric can also be used to optimize an AA to CG mapping operator.

Chapter 3

Input files

3.1 Mapping files

Mapping relates atomistic and coarse-grained representations of the system. It is organized as follows: for each molecule *type* a mapping file is created. When used as a command option, these files are combined in a list separated by a semicolon, e. g. `--cg "protein.xml;solvent.xml"`.

Each mapping file contains a *topology* of the coarse-grained molecule and a list of *maps*. Topology specifies coarse-grained beads and bonded interactions between them. Each coarse-grained bead has a name, type, a list of atoms which belong it, and a link to a map. A map is a set of weights c_{Ii} for an atom i belonging to the bead I. It is used to calculate the position of a coarse-grained bead from the positions of atoms which belong to it. Note that c_{Ii} will be automatically re-normalized if their sum is not equal to 1, i. e. in the case of a center-of-mass mapping one can simply specify atomic masses. A complete reference for mapping file definitions can be found in sec. 10.2.

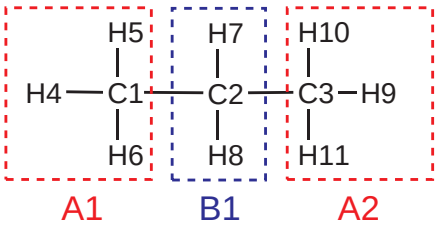


Figure 3.1: Atom labeling and mapping from an all-atom to a united atom representation of a propane molecule.

As an example, we will describe here a mapping file of a united atom model of a propane molecule, chemical structure of which is shown in fig. 1.1. In this coarse-grained model two bead types (A,B) and three beads (A1, B1, A2) are defined, as shown in fig. 3.1. We will use centers of mass of the beads as coarse-grained coordinates.

Extracts from the `propane.xml` file of the tutorial are shown below. The name tag indicates the molecule name in the coarse-grained topology. The ident tag must match the name of the molecule in the atomistic representation. In the topology section all beads are defined by specifying bead name (A1, B1, A2), type, and atoms belonging to this bead in the form `residue id:residue name:atom name`. For each bead a map has to be specified, which is defined later in maps section. Note that bead type and map can be different, which might be useful in a situation when chemically different beads (A1, B1) are assigned to the same bead type. After defining all beads the bonded interactions of the coarse-grained molecule must be specified in the cg_bonded section. This is done by using the identifiers of the beads in the coarse-grained model. Finally, in the mapping section, the mapping coefficients are defined. This includes a weighting of the atoms in the topology section. In particular, the number of weights given should match the number of beads.

3.2 Verification of a mapping

Note that the ident tag should match the molecule name in the reference system. A common mistake is that beads have wrong names. In this case, the **csg_dump** tool can be used in order to identify the atoms which are read in from a topology file `.tpr`. This tool displays the atoms in the format `residue id:residue name:atom name`. For multicomponent systems, it might happen that molecules are not identified correctly. The workaround for this case is described in sec. 3.3.

To compare coarse-grained and atomistic configurations one can use a standard visualization program, e. g. `vmd`. When comparing trajectories, one has to be careful, since `vmd` opens both a `.gro` and `.trr` file. The first frame is then the `.gro` file and the rest is taken from the `.trr` file. The coarse-grained trajectory contains only the frames of the trajectory. Hence, the first frame of the atomistic run has to be removed using the `vmd` menu.

3.3 Advanced topology handling

A topology is completely specified by a set of beads, their types, and a list of bonded interactions. VOTCA is able to read topologies in the GROMACS `.tpr` format. For example, one can create a coarse-grained topology based on the mapping file and atomistic GROMACS topology using **csg_gmxtopol**.

```
csg_gmxtopol --top topol.tpr --cg propane.xml --out out.top
```

In some cases, however, one might want to use a .pdb, H5MD or .dump file which does not contain all information about the atomistic topology. In this case, additional information can be supplied in the XML mapping file.

A typical example is lack of a clear definition of molecules, which can be a problem for simulations with several molecules with multiple types. During coarse-graining, the molecule type is identified by a name tag as names must be clearly identified. To do this, it is possible to read a topology and then modify parts of it. The new XML topology can be used with the `--tpr` option, as any other topology file.

For example, if information about a molecule is not present at all, one can create one from a `.pdb` file as follows

```
<topology base="snapshot.pdb">
  <molecules>
    <clear/>
    <define name="mCP" first="1" nbeads="52" nmols="216"/>
  </molecules>
</topology>
```

where <clear/> clears all information that was present before.

Old versions of GROMACS did not store molecule names. In order to use this feature, a recent `.tpr` file containing molecule names should always be provided. For old topologies, rerun GROMACS **grompp** to update the old topology file.

If molecule information is already present in the parent topology but molecules are not named properly (as it is the case with old GROMACS `.tpr` files), one can rename them using

```
<topology base="topol.tpr">
  <molecules>
    <rename name="PPY3" range="1:125"/>
    <rename name="Cl" range="126:250"/>
  </molecules>
</topology>
```

```
<cg_molecule>
  <name>ppn</name> <!-- molecule name in cg representation -->
  <ident>ppn</ident> <!-- molecule name in atomistic topology -->

  <topology> <!-- topology of one molecule -->
    <cg_beads>
      <cg_bead>  <!-- definition of a coarse-grained bead -->
        <name>A1</name>
        <type>A</type>
        <mapping>A</mapping> <!-- reference to a map -->
        <!-- atoms belonging to this bead -->
        <beads>1:ppn:C1 1:ppn:H4 1:ppn:H5 1:ppn:H6</beads>
      </cg_bead>
      <!-- more bead definitions -->
    </cg_beads>

    <cg_bonded> <!-- bonded interactions -->
      <bond>
        <name>bond</name>
        <beads>
          A1 B1
          B1 A2
        </beads>
      </bond>

      <angle>
        <name>angle</name>
        <beads>
          A1 B1 A2
        </beads>
      </angle>
    </cg_bonded>
  </topology>

  <maps>
    <map> <!-- mapping A -->
      <name>A</name>
      <weights> 12 1 1 1 </weights>
    </map>
    <!-- more mapping definitions -->
  </maps>
</cg_molecule> <!-- end of the molecule -->
```

Figure 3.2: An extract from the mapping file `propane.xml` of a propane molecule. The complete file can be found in the `propane/single_molecule` tutorial.

Here, the file `topol.tpr` is loaded first and all molecules are renamed afterwards.

If you do not have a .pdb/.gro file and you want to read trajectory from LAMMPS .dump file or H5MD then it is also possible to directly define topology in XML file. Here is an example of such file where the trajectory is read from H5MD file:

```
<topology>
    <!-- particle group name in H5MD file -->
    <h5md_particle_group name="atoms" />
    <molecules>
        <!-- define molecule, number of beads, number of mols -->
        <molecule name="BUT" nmols="4000" nbeads="4">
            <!-- composition of molecule, bead definition -->
            <bead name="C1" type="C" mass="15.035" q="0.0" />
            <bead name="C2" type="C" mass="14.028" q="0.0" />
            <bead name="C3" type="C" mass="14.028" q="0.0" />
            <bead name="C4" type="C" mass="15.035" q="0.0" />
        </molecule>
    </molecules>
    <!-- bonded terms -->
    <bonded>
        <bond>
            <name>bond1</name>
            <beads>
                BUT:C1 BUT:C2
            </beads>
        </bond>
        <bond>
            <name>bond2</name>
            <beads>
                BUT:C2 BUT:C3
            </beads>
        </bond>
        <angle>
            <name>angle1</name>
            <beads>
                BUT:C1 BUT:C2 BUT:C3
                BUT:C2 BUT:C3 BUT:C4
            </beads>
        </angle>
        <dihedral>
            <name>dihedral1</name>
            <beads>
                BUT:C1 BUT:C2 BUT:C3 BUT:C4
            </beads>
        </dihedral>
    </bonded>
</topology>
```

The list of molecules is defined in section `molecules` where every `molecule` is replicated `nmols` times. Inside `molecule` the list of `bead` has to be defined with the name, type, mass and charge.

The box size can be set by the tag `box`:

```
<box xx="6.0" yy="6.0" zz="6.0" />
```

where `xx, yy, zz` are the dimensions of the box.

A complete reference for XML topology file can be found in sec. 10.3.

3.4 Trajectories

A trajectory is a set of frames containing coordinates (velocities and forces) for the beads defined in the topology. VOTCA currently supports `.trr`, `.xtc`, `.pdb`, `.gro` and H5MD `.h5` trajectory formats.

Once the mapping file is created, it is easy to convert an atomistic to a coarse-grained trajectory using **csg_map**

```
csg_map --top topol.tpr --trj traj.trr --cg propane.xml --out cg.gro
```

The program **csg_map** also provides the option `--no-map`. In this case, no mapping is done and **csg_map** works as a trajectory converter. In general, mapping can be enabled and disabled in most analysis tools, e.g. in **csg_stat** or **csg_fmatch**.

Note that the topology files can have a different contents as bonded interactions are not provided in all formats. In this case, mapping files can be used to define and relabel bonds.

Also note that the default setting concerning mapping varies individually between the programs. Some have a default setting that does mapping (such as **csg_map**, use `--no-map` to disable mapping) and some have mapping disabled by default (e.g. **csg_stat**, use `--cg` to enable mapping).

3.5 Setting files

```
<cg>
  <non-bonded> <!-- non-bonded interactions -->
    <name>A-A</name> <!-- name of the interaction -->
    <type1>A</type1> <!-- types involved in this interaction -->
    <type2>A</type2>
    <min>0</min>  <!-- dimension + grid spacing of tables-->
    <max>1.36</max>
    <step>0.01</step>
    <inverse>
      ... specific commands
    </inverse>

    ... specific section for inverse boltzmann, force matching etc.
  </non-bonded>
</cg>
```

Figure 3.3: Abstract of a `settings.xml` file. See sec. 7.1.1 for a full version.

A setting file is written in the format `.xml`. It consists of a general section displayed above, and a specific section depending on the program used for simulations. The setting displayed above is later extended in the sections on iterative boltzmann inversion (**csg_inverse**), force matching (**csg_fmatch**) or statistical analysis (**csg_stat**).

Generally, **csg_stat** is an analysis tool which can be used for computing radial distribution functions and analysing them. As an example, the command

```
csg_stat --top topol.tpr --trj traj.xtc --options settings.xml
```

computes the distributions of all interactions specified in `settings.xml` and writes all tabulated distributions as files "`interaction name`".dist.new.

3.6 Table formats

In the iterative framework distribution functions, potentials and forces are returned as tables and saved in a file. Those tables generally have the format

```
x y [error] flag
```

where `x` is input quantity (e.g. radius r, angles θ or ϕ), `y` is the computed quantity (e.g. a potential) and `[error]` is an optional error for `y`. The token `flag` can take the values `i`, `o` or `u`. In the first case, `i` (`in range`) describes a value that lies within the data range, `o` (`out of range`) symbolises a value out of the data range and `u` stands for an `undefined` value.

The token `flag` will be important when extrapolating the table as described in sec. 4.2.

For historical reasons, **csg_boltzmann** uses a slightly different table format, it has no `flag` column and uses the third column as a force column when outputting a potential.

Chapter 4

Preparing coarse-grained runs

Preliminary note

The coarse-grained run requires the molecule topology on the one hand and suitable potentials on the other. In this chapter, the generation of coarse-grained runs is decribed next, followed by a post-processing of the potential.

If the potential is of such a form that it allows direct fitting of a functional form, the section on post-processing can be skipped. Instead, a program of choice should be used to fit a functional form to the potential. Nevertheless, special attention should be paid to units (angles, bondlengths). The resulting curve can then be specified in the MD package used for simulation. However, most potentials don't allow an easy processing of this kind and tabulated potentials have to be used.

4.1 Generating a topology file for a coarse-grained run

WARNING: This section describes experimental features. The exact names and options of the program might change in the near future. The section is specific to GROMACS support though a generalization for other MD packages is planned.

The mapping definition is close to a topology needed for a coarse grained run. To avoid redundant work, **csg_gmxtopol** can be used to automatically generate a gromacs topology based on an atomistic reference system and a mapping file.

At the current state, **csg_gmxtopol** can only generate the topology for the first molecule in the system. If more molecule types are present, a special tpr file has to be prepared. The program can be executed by

```
csg_gmxtopol --top topol.tpr --cg map.xml --out cgtop
```

which will create a file `cgtop.top`. This file includes the topology for the first molecule including definitions for atoms, bonds, angles and dihedrals. It can directly be used as a topology in GROMACS, however the force field definitions (atom types, bond types, etc.) still have to be added manually.

4.2 Post-processing of the potential

The VOTCA package provides a collection of scripts to handle potentials. They can be modified, refined, integrated or inter- and extrapolated. These scripts are the same ones as those used for iterative methods in chapter 7. Scripts are called by **csg_call**. A complete list of available scripts can be found in sec. 10.5.

The post-processing roughly consists of the following steps (see further explanations below):

- (manually) clipping poorly sampled (border) regions

- resampling the potential in order to change the grid to the proper format (**csg_resample**)
- extrapolation of the potential at the borders (**csg_call** table extrapolate)
- exporting the table to xvg (**csg_call** convert_potential gromacs)

4.2.1 Clipping of poorly sampled regions

Regions with an irregular distribution of samples should be deleted first. This is simply done by editing the `.pot` file and by deleting those values.

Alternatively, manually check the range where the potential still looks good and is not to noisy and set the flags in the potential file of the bad parts by hand to `o` (for `out of range`). Those values will later be extrapolated and overwritten.

4.2.2 Resampling

Use the command

```
csg_resample --in table.pot --out table_resample.pot \
             --grid min:step:max
```

to resample the potential given in file –`table.pot` from `min` to `max` with a grid spacing of `step` steps. The result is written to the file specified by `out`. Additionally, **csg_resample** allows the specification of spline interpolation (`spfit`), the calculation of derivatives (`derivative`) and comments (`comment`). Check the help (`help`) for further information.

It is important to note that the values `min` and `max` *don't* correspond to the minimum and maximum value in the input file, but to the range of values the potential is desired to cover after extrapolation. Therefore, values in [min, max] that are not covered in the file are automatically marked by a flag `o` (for `out of range`) for extrapolation in the next step.

The potential *don't* have to start at 0, this is done by the export script (to xvg) automatically.

4.2.3 Extrapolation

The following line

```
csg_call table extrapolate [options] table_resample.pot \
         table_extrapolate.pot
```

calls the extrapolation procedure, which processes the range of values marked by **csg_resample**. The input file is `table_resample.pot` created in the last step.

After resampling, all values in the potential file that should be used as a basis for extrapolation are marked with an `i`, while all values that need extrapolation are marked by `o`. The command above now extrapolates all `o` values from the `i` values in the file. Available options include averaging over a certain number of points (`avgpoints`), changing the functional form (`function`, default is quadratic), extrapolating just the left or right region of the file (`region`) and setting the curvature (`curvature`).

The output `table_extrapolate.pot` of the extrapolation step can now be used for the coarse-grained run. If GROMACS is used as a molecule dynamics package, the potential has to be converted and exported to a suitable GROMACS format as described in the final step.

4.2.4 Exporting the table

Finally, the table is exported to `xvg`. The conversion procedure requires a small xml file `table.xml` as shown below:

```
<cg>
  <non-bonded>
    <name>XXX</name>
    <step>0.01</step>
  </non-bonded>
  <inverse>
    <gromacs>
      <pot_max>1e8</pot_max>
      <table_end>8.0</table_end>
      <table_bins>0.002</table_bins>
    </gromacs>
  </inverse>
</cg>
```

where <table_end> is the GROMACS rvdw+table_extension and <pot_max> is just a number slightly smaller than the upper value of single/ double precision. The value given in <table_bins> corresponds to the step value of csg_resample -grid min:step:max.

Using the xml file above, call

```
csg_call --options table.xml --ia-type non-bonded --ia-name XXX \
  convert_potential gromacs table_extrapolate.pot table.xvg
```

to convert the extrapolated values in table_extrapolate.pot to table.xvg (The file will contain the GROMACS C12 parts only which are stored in the sixth und seventh column, this can be changed by adding the -ia-type C6 option (for the fourth and fiveth column) or -ia-type CB option (for the second and third column) after **csg_call**. Ensure compatibility with the GROMACS topology. See the GROMACS manual for further information).

To obtain a bond table, run

```
csg_call --ia-type bond --ia-name XXX --options table.xml \
convert_potential gromacs table_extrapolate.pot table.xvg
```

It is also possible to use angle and dihedral as type as well, but make to sure to have a bonded section similar to the non-bonded section above with the corresponding interaction name.

Internally convert_potential gromacs will do the following steps:

- Resampling of the potential from 0 (or -180 for dihedrals) to table_end (or 180 for angles and dihedrals) with step size table_bins. This is needed for gromacs the table must start with 0 or -180.
- Extrapolate the left side (to 0 or -180) exponentially
- Extrapolate the right side (to table_end or 180) exponentially (or constant for non-bonded interactions)
- Shift it so that the potential is zero at table_end for non-bonded interactions or zero at the minimum for bonded interaction
- Calculate the force (assume periodicity for dihedral potentials)
- Write to the format needed by gromacs

4.2.5 An example on non-bonded interactions

```
csg_call pot shift_nonbonded table.pot table.pot.refined
csg_resample --grid 0.3:0.05:2 --in table.pot.refined \
         --out table.pot.refined
csg_call table extrapolate --function quadratic --region left \
```

```
          table.pot.refined table.pot.refined
csg_call table extrapolate --function constant --region right \
          table.pot.refined table.pot.refined
```

4.3 Alternatives

Additionally to the two methods described above, namely (a) providing the MD package directly with a functional form fitted with a program of choice or (b) using `csg_resample`, `csg_call table extrapolate` and `csg_call convert_potential`, another method would be suitable. This is integrating the force table as follows

```
-Integrate the table
$csg_call table integrate force.d minus_pot.d
-multiply by -1
$csg_call table linearop minus_pot.d pot.d -1 0
```

Chapter 5

Boltzmann Inversion

Boltzmann inversion provides a potential of mean force for a given degree of freedom. It is mostly used for deriving *bonded* interactions from canonical sampling of a single molecule in vacuum, e. g. for polymer coarse-graining, where it is difficult to separate bonded and non-bonded degrees of freedom [7]. The non-bonded potentials can then be obtained by using iterative methods or force matching.

The main tool which can be used to calculate histograms, cross-correlate coarse-grained variables, create exclusion lists, as well as prepare tabulated potentials for coarse-grained simulations is **csg_boltzmann**. It parses the whole trajectory and stores all information on bonded interactions in memory, which is useful for interactive analysis. For big systems, however, one can run out of memory. In this case **csg_stat** can be used which, however, has a limited number of tasks it can perform (see sec. 3.5 for an example on its usage).

Another useful tool is **csg_map**. It can be used to convert an atomistic trajectory to a coarse-grained one, as it is discussed in sec. 3.4.

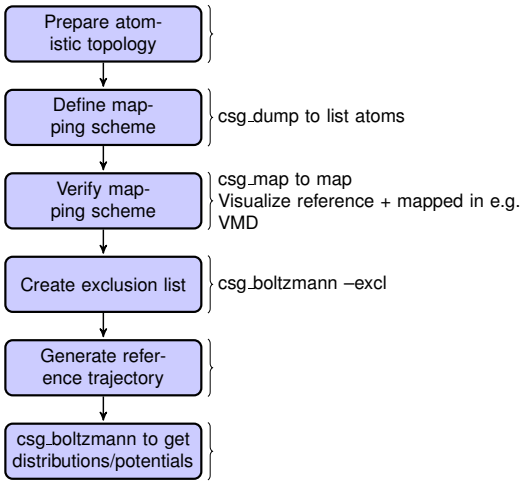


Figure 5.1: Flowchart demonstrating useful options of the tool.

To use **csg_boltzmann** one has to first define a mapping scheme. This is outlined in sec. 3.1. Once the mapping scheme is specified, it is possible to generate an exclusion list for the proper sampling of the atomistic resolution system.

5.1 Generating exclusion lists

Exclusion lists are useful when sampling from a special reference system is needed, for example for polymer coarse-graining with a separation of bonded and non-bonded degrees of freedom.

To generate an exclusion list, an atomistic topology without exclusions and a mapping scheme have to be prepared first. Once the .tpr topology and .xml mapping files are ready, simply run

```
csg_boltzmann --top topol.tpr --cg mapping.xml --excl exclusions.txt
```

This will create a list of exclusions for all interactions that are not within a bonded interaction of the coarse-grained sub-bead. As an example, consider coarse-graining of a linear chain of three beads which are only connected by bonds. In this case, **csg_boltzmann** will create exclusions

for all non-bonded interactions of atoms in the first bead with atoms of the 3rd bead as these would contribute only to the non-bonded interaction potential. Note that **csg_boltzmann** will only create the exclusion list for the fist molecule in the topology.

To add the exclusions to the GROMACS topology of the molecule, either include the file specified by the –excl option into the .top file as follows

```
[ exclusions ]
#include "exclusions.txt"
```

or copy and paste the content of that file to the exclusions section of the gromacs topology file.

5.2 Statistical analysis

For statistical analysis **csg_boltzmann** provides an interactive mode. To enter the interactive mode, use the `–trj` option followed by the file name of the reference trajectory

```
csg_boltzmann --top topol.tpr --trj traj.trr --cg mapping.xml
```

To get help on a specific command of the interactive mode, type

```
help <command>
```

for example

```
help hist
help hist set periodic
```

Additionally, use the

```
list
```

command for a list of available interactions. Note again that **csg_boltzmann** loads the whole trajectory and all information on bonded interactions into the memory. Hence, its main application should be single molecules. See the introduction of this chapter for the **csg_stat** command.

If a specific interaction shall be used, it can be referred to by

```
molecule:interaction-group:index
```

Here, `molecule` is the molecule number in the whole topology, `interaction-group` is the name specified in the `<bond>` section of the mapping file, and `index` is the entry in the list of interactions. For example, `1:AA-bond:10` refers to the 10th bond named `AA-bond` in molecule 1. To specify a couple of interactions during analysis, either give the interactions separated by a space or use wildcards (e.g. `*:AA-bond*`).

To exit the interactive mode, use the command `q`.

If analysis commands are to be read from a file, use the pipe or stdin redirects from the shell.

```
cat commands | csg_boltzmann topol.top --trj traj.trr --cg mapping.xml
```

5.2.1 Distribution functions and tabulated potentials

Distribution functions (tabulated potentials) can be created with the `hist` (`tab`) command. For instance, to write out the distribution function for all interactions of group AA-bond (where AA-bond is the name specified in the mapping scheme) to the file AA.txt, type

```
hist AA.txt *:AA-bond:*
```

The command

```
hist set
```

prints a list of all parameters that can be changed for the histogram: the number `n` of bins for the table, bounds `min` and `max` for table values, scaling and normalizing, a flag `periodic` to ensure periodic values in the table and an `auto` flag. If `auto` is set to 1, bounds are calculated automatically, otherwise they can be specified by `min` and `max`. Larger values in the table might extend those bounds, specified by parameter `extend`.

To directly write the Boltzmann-inverted potential, the `tab` command can be used. Its usage and options are very similar to the `hist` command. If tabulated potentials are written, special care should be taken to the parameters `T` (temperature) and the `scale`. The `scale` enables volume normalization as given in eq. 2.13. Possible values are `no` (no scaling), `bond` (normalize bonds) and `angle` (normalize angles). To write out the tabulated potential for an angle potential at a temperature of 300K, for instance, type:

```
tab set T 300
tab set scale angle
tab angle.pot *:angle:*
```

The table is then written into the file `angle.pot` in the format described in sec. 3.6. An optional correlation analysis is described in the next section. After the file has been created by command `tab`, the potential is prepared for the coarse-grained run in chapter 4.

5.2.2 Correlation analysis

The factorization of P in eq. 2.14 assumed uncorrelated quantities. **csg_boltzmann** offers two ways to evaluate correlations of interactions. One option is to use the linear correlation coefficient (command `cor`).

However, this is not a good measure since `cor` calculates the linear correlation only which might often lead to misleading results [3]. An example for such a case are the two correlated random variables $X \sim U[-1, 1]$ with uniform distribution, and $Y := X^2$. A simple calculation shows $cov(X, Y) = 0$ and therefore

$$cor = \frac{cov(X, Y)}{\sqrt{var(X)var(Y)}} = 0.$$

A better way is to create 2D histograms. This can be done by specifying all values (e.g. bond length, angle, dihedral value) using the command *vals*, e.g.:

```
vals vals.txt 1:AA-bond:1 1:AAA-angle:A
```

This will create a file which contains 3 columns, the first being the time, and the second and third being bond and angle, respectively. Columns 2 and 3 can either be used to generate the 2D histogram, or a simpler plot of column 3 over 2, whose density of points reflect the probability.

Two examples for 2D histograms are shown below: one for the propane molecule and one for hexane.

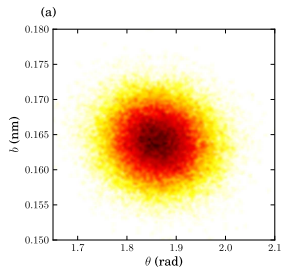


Figure 5.2: propane histogram

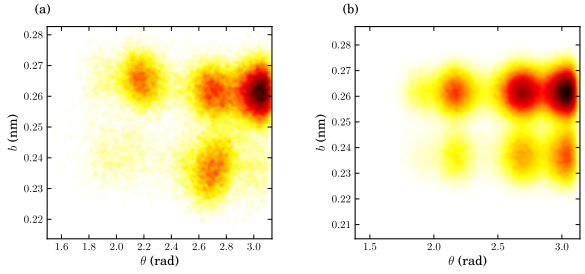


Figure 5.3: hexane histograms: before and after the coarse-grained run

The two plots show the correlations between angle and bondlength for both molecules. In the case of propane, the two quantities are not correlated as shown by the centered distribution, while correlations exist in the case of hexane. Moreover, it is visible from the hexane plot that the partition of the correlations has changed slightly during coarse-graining.

The tabulated potentials created in this section can be further modified and prepared for the coarse-grained run: This includes fitting of a smooth functional form, extrapolation and clipping of poorly sampled regions. Further processing of the potential is decribed in chapter 4.

Chapter 6

Force matching

The force matching algorithm with cubic spline basis is implemented in the **csg_fmatch** utility. A list of available options can be found in the reference section of **csg_fmatch** (command `-h`).

6.1 Program input

csg_fmatch needs an atomistic reference run to perform coarse-graining. Therefore, the trajectory file *must contain forces* (note that there is a suitable option in the GROMACS `.mdp` file), otherwise **csg_fmatch** will not be able to run.

In addition, a mapping scheme has to be created, which defines the coarse-grained model (see sec. 3). At last, a control file has to be created, which contains all the information for coarse-graining the interactions and parameters for the force-matching run. This file is specified by the tag `-options` in the XML format. An example might look like the following

```
<cg>
  <!--fmatch section -->
  <fmatch>
    <!--Number of frames for block averaging -->
    <frames_per_block>6</frames_per_block>
    <!--Constrained least squares?-->
    <constrainedLS>false</constrainedLS>
  </fmatch>
  <!-- example for a non-bonded interaction entry -->
  <non-bonded>
    <!-- name of the interaction -->
    <name>CG-CG</name>
    <type1>A</type1>
    <type2>A</type2>
    <!-- fmatch specific stuff -->
    <fmatch>
      <min>0.27</min>
      <max>1.2</max>
      <step>0.02</step>
      <out_step>0.005</out_step>
    </fmatch>
  </non-bonded>
</cg>
```

Similarly to the case of spline fitting (see sec. 10.1 on **csg_resample**), the parameters `min` and `max` have to be chosen in such a way as to avoid empty bins within the grid. Determining `min` and

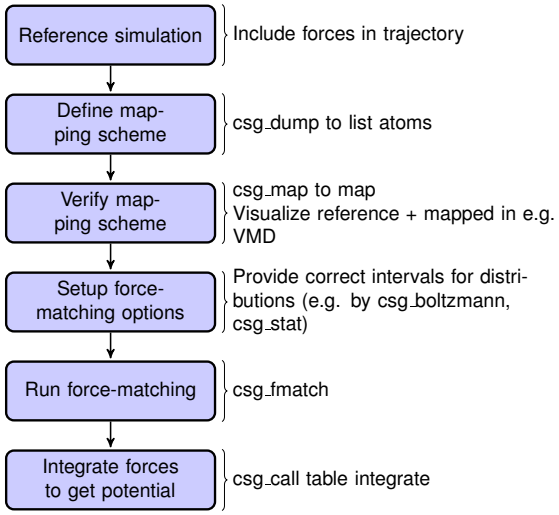


Figure 6.1: Flowchart to perform force matching.

`max` by using **csg_stat** is recommended (see sec. 3.5). A full description of all available options can be found in sec. 10.4.

6.2 Program output

csg_fmatch produces a separate `.force` file for each interaction, specified in the CG-options file (option `options`). These files have 4 columns containing distance, corresponding force, a table flag and the force error, which is estimated via a block-averaging procedure. If you are working with an angle, then the first column will contain the corresponding angle in radians.

To get table-files for GROMACS, integrate the forces in order to get potentials and do extrapolation and potentially smoothing afterwards.

Output files are not only produced at the end of the program execution, but also after every successful processing of each block. The user is free to have a look at the output files and decide to stop **csg_fmatch**, provided the force error is small enough.

6.3 Integration and extrapolation of .force files

To convert forces (`.force`) to potentials (`.pot`), tables have to be integrated. To use the built-in integration command from the scripting framework, execute

```
$csg_call table integrate CG-CG.force minus_CG-CG.pot
$csg_call table linearop minus_CG-CG.d CG-CG.d -1 0
```

This command calls the **table_integrate.pl** script, which integrates the force and writes the potential to the `.pot` file.

In general, each potential contains regions which are not sampled. In this case or in the case of further post-processing, the potential can be refined by employing resampling or extrapolating methods. See sec. 4.2 for further details.

Chapter 7

Iterative methods

The following sections deal with the methods of Iterative Boltzmann Inversion (IBI), Inverse Monte Carlo (IMC), and Relative Entropy (RE).

In general, IBI, IMC, and RE are implemented within the same framework. Therefore, most settings and parameters of those methods are similar and thus described in a general section (see sec. 7.3). Further information on iterative methods follows in the next chapters, in particular on the IBI, IMC, and RE methods.

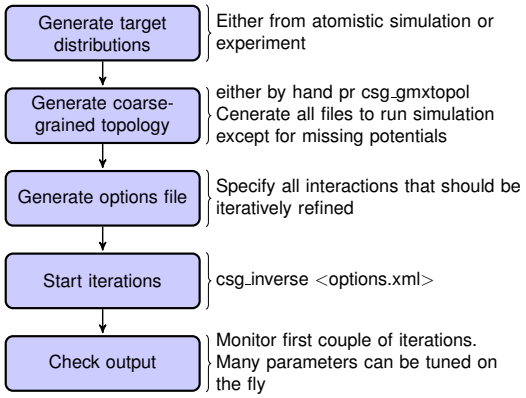


Figure 7.1: Flowchart to perform iterative Boltzmann inversion.

7.1 Iterative workflow control

Iterative workflow control is essential for the IBI, IMC, and RE methods.

The general idea of iterative workflow is sketched in fig. 7.2. During the global initialization the initial guess for the coarse-grained potential is calculated from the reference function or converted from a given potential guess into the internal format. The actual iterative step starts with an iteration initialization. It searches for possible checkpoints and copies and converts files from the previous step and the base directory. Then, the simulation run is prepared by converting potentials into the format required by the external sampling program and the actual sampling is performed.

After sampling the phasespace, the potential update is calculated. Often, the update requires postprocessing, such as smoothing, interpolation, extrapolation or fitting to an analytical form.

Finally, the new potential is determined and postprocessed. If the iterative process continues, the next iterative step will start to initialize.

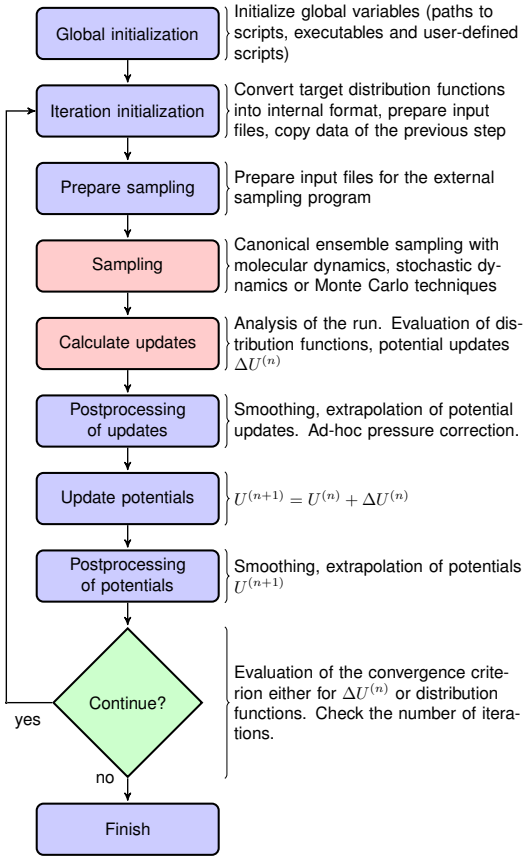


Figure 7.2: Block-scheme of the workflow control for the iterative methods. The most time-consuming parts are marked in red.

How to start:

The first thing to do is generate reference distribution functions. These might come from experiments or from atomistic simulations. To get reasonable results out of the iterative process, the reference distributions should be of good quality (little noise, etc).

VOTCA can create initial guesses for the coarse-grained potentials by boltzmann inverting the distribution function. If a custom initial guess for an interaction shall be used instead, the table can be provided in *<interaction>.pot.in*. As already mentioned, VOTCA automatically creates potential tables to run a simulation. However, it does not know how to run a coarse-grained simulation. Therefore, all files needed to run a coarse-grained simulation, except for the potentials that are iteratively refined, must be provided and added to the filelist in the settings XML-file. If an atomistic topology and a mapping definition are present, VOTCA offers tools to assist the setup of a coarse-grained topology (see chapter 4).

To get an overview of how input files look like, it is suggested to take a look at one of the tutorials provided on WWW.VOTCA.ORG.

In what follows we describe how to set up the iterative coarse-graining, run the main script, continue the run, and add customized scripts.

7.1.1 Preparing the run

To start the first iteration, one has to prepare the input for the sampling program. This means that all files for running a coarse-grained simulation must be present and described in a separate

XML file, in our case `settings.xml` (see sec. 3.5 for details). An extract from this file is given below. The only exception are tabulated potentials, which will be created and updated by the script in the course of the iterative process.

The input files include: target distributions, initial guess (optional) and a list of interactions to be iteratively refined. As a target distribution, any table file can be given (e.g. GROMACS output from `g_rdf`). The program automatically takes care to resample the table to the correct grid spacing according to the options provided in `settings.xml`.

The initial guess is normally taken as a potential of mean force and is generated by Boltzmann-inversion of the corresponding distribution function. It is written in `step_000/<name>.pot.new`. If you want to manually specify the initial guess for a specific interaction, write the potential table to a file called `<name>.pot.in` in the folder where you plan to run the iterative procedure.

A list of interactions to be iteratively refined has to be given in the options file. As an example, the `setting.xml` file for a propane is shown in listing 7.3. For more details, see the full description of all options in ref. 10.4.

7.1.2 Starting the iterative process

After all input files have been set up, the run can be started by

```
csg_inverse --options settings.xml
```

Each iteration is stored in a separate directory, named `step_<iteration>`. `step_000` is a special folder which contains the initial setup. For each new iteration, the files required to run the CG simulation (as specified in the config file) are copied to the current working directory. The updated potentials are copied from the last step, `step_<n-1>/<interaction>.pot.new`, and used as the new working potentials `step_<n>/<interaction>.pot.cur`.

After the run preparation, all potentials are converted into the format of the sampling program and the simulation starts. Once the sampling has finished, analysis programs generate new distributions, which are stored in `<interaction>.dist.new`, and new potential updates, stored in `<interaction>.dpot.new`.

Before adding the update to the old potential, it can be processed in the `post_update` step. For each script that is specified in the postupdate, `<interaction>.dpot.new` is renamed to `<interaction>.dpot.old` and stored in `<interaction>.dpot.<a-number>` before the processing script is called. Each processing script uses the current potential update `<interaction>.dpot.cur` and writes the processed update to `<interaction>.dpot.new`. As an example, a pressure correction is implemented as a postupdate script within this framework.

After all postupdate scripts have been called, the update is added to the potential and the new potential `<interaction>.pot.new` is written. Additional post-processing of the potential can be performed in the `post_add` step which is analogous to the `post_update` step except for a potential instead of an update.

To summarize, we list all standard output files for each iterative step:

`*.dist.new`	distribution functions of the current step
`*.dpot.new`	the final potential update, created by `calc_update`
`*.dpot.<number>`	for each postupdate script, the `.dpot.new` is saved and a new one is created
`*.pot.cur`	the current potential used for the actual run
`*.pot.new`	the new potential after the add step
`*.pot.<number>`	same as `dpot.<number>` but for `post_add`

If a sub-step fails during the iteration, additional information can be found in the log file. The name of the log file is specified in the steering XML file.

7.1.3 Restarting and continuing

The interrupted or finished iterative process can be restarted either by extending a finished run or by restarting the interrupted run. When the script **csg_inverse** is called, it automatically checks

```
<cg>
  <non-bonded> <!-- non-bonded interactions -->
    <name>A-A</name> <!-- name of the interaction -->
    <type1>A</type1> <!-- types involved in this interaction -->
    <type2>A</type2>
    <min>0</min>  <!-- dimension + grid spacing of tables-->
    <max>1.36</max>
    <step>0.01</step>
    <inverse>
      <target>A-A.dist.tgt</target> <!-- target distribution -->
      <do_potential>1 0 0</do_potential>  <!-- update cycles -->
      <gromacs>
        <table>table_A_A.xvg</table>
      </gromacs>
    </inverse>
  </non-bonded>
  <!-- ... more non-bonded interactions -->

  <!-- general options for the inverse script -->
  <inverse>
    <kBT>1.6629</kBT> <!-- 300*0.00831451 gromacs units -->
    <program>gromacs</program> <!-- use gromacs to sample -->
    <gromacs> <!-- gromacs specific options -->
      <equi_time>10</equi_time> <!-- ignore so many frames -->
      <table_bins>0.002</table_bins> <!-- grid for table*.xvg -->
      <pot_max>1000000</pot_max> <!-- cut the potential at value -->
      <table_end>2.0</table_end> <!-- extend the tables to value -->
      <topol>topol.tpr</topol> <!-- topology + trajectory files -->
      <traj>traj.xtc</traj>
    </gromacs>
    <!-- these files are copied for each new run -->
    <filelist>grompp.mdp topol.top table.xvg
        table_a1.xvg table_b1.xvg index.ndx
    </filelist>
    <iterations_max>300</iterations_max> <!-- number of iterations -->
    <method>ibi</method> <!-- inverse Boltzmann or inverse MC -->
    <log_file>inverse.log</log_file> <!-- log file -->
    <restart_file>restart_points.log</restart_file> <!-- restart -->
   </inverse>
</cg>
```

Figure 7.3: `settings.xml` file specifies interactions to be refined, grid spacings, sampling engine, and the iterative method. The complete file can be found in the `propane/ibm` tutorial.

for a file called `done` in the current directory. If this file is found, the program assumes that the run is finished. To extend the run, simply increase *inverse.iterations_max* in the settings file and remove the file called `done`. After that, **csg_inverse** can be restarted, which will automatically recognize existing steps and continue after the last one.

If the iteration was interrupted, the script **csg_inverse** might not be able to restart on its own. In this case, the easiest solution is to delete the last step and start again. The script will then repeat the last step and continue. However, this method is not always practical since sampling and analysis might be time-consuming and the run might have only crashed due to some inadequate post processing option. To avoid repeating the entire run, the script **csg_inverse** creates a file with restart points and labels already completed steps such as simulation, analysis, etc. The file name is specified in the option *inverse.restart_file*. If specific actions should be redone, one can simply remove the corresponding lines from this file. Note that a file `done` is also created in each folder for those steps which have been successfully finished.

7.2 Iterative Boltzmann Inversion

7.2.1 Input preparation

This section describes the usage of IBI, implemented within the scripting framework described in the previous section 7.1. It is suggested to get a basic understanding of this framework before proceeding.

An outline of the workflow for performing IBI is given in fig. 7.1.

To specify Iterative Boltzmann Inversion as algorithm in the script, add `ibi` in the `method` section of the XML setting file as shown below.

```
<cg>
  ...
  <inverse>
    <method>ibi</method>
  </inverse>
</cg>
```

7.3 Inverse Monte Carlo

In this section, additional options are described to run IMC coarse graining. The usage of IMC is similar to the one of IBI and understanding the use of the scripting framework described in chapter 7.1 is necessary.

WARNING: multicomponent IMC is still experimental!

7.3.1 General considerations

In comparison to IBI, IMC needs significantly more statistics to calculate the potential update[3]. It is advisable to perform smoothing on the potential update. Smoothing can be performed as described in sec. 7.7. In addition, IMC can lead to problems related to finite size: for methanol, an undersized system proved to lead to a linear shift in the potential[3]. It is therefore always necessary to check that the system size is sufficiently large and that runlength csg smoothing iterations are well balanced.

7.3.2 Correlation groups

Unlike IBI, IMC also takes cross-correlations of interactions into account in order to calculate the update. However, it might not always be beneficial to evaluate cross-correlations of all pairs of interactions. By specifying *inverse.imc.group*, VOTCA allows to define groups of interactions,

amongst which cross-correlations are taken into account, where *inverse.imc.group* can be any name.

```
<non-bonded>
  <name>CG-CG</name>
  <type1>CG</type1>
  <type2>CG</type2>
  ...
  <imc>
    <group>solvent</group>
 </imc>
</non-bonded>
<non-bonded>
```

7.3.3 Regularization

To use the regularized version of IMC a λ value > 0 has to be specified by setting *inverse.imc.reg*. If set to 0 (default value) the unregularized version of IMC is applied.

```
<non-bonded>
  <name>CG-CG</name>
  <type1>CG</type1>
  <type2>CG</type2>
   ...
  <inverse>
    <imc>
      <reg>300</reg>
    </imc>
  </inverse>
</non-bonded>
```

7.4 Relative Entropy

In this section, additional options are described to run RE coarse graining. The usage of RE is similar to the one of IBI and IMC and understanding the use of the scripting framework described in chapter 7.1 is necessary.

Currently, RE implementation supports optimization of two-body non-bonded pair interactions. Support for bonded and N-body interactions is possible by further extension of RE implementation.

7.4.1 Potential function and parameters

In RE, CG potentials are modeled using analytical functional forms. Therefore, for each CG interaction, an analytical functional must be specified in the XML setting file as

```
<non-bonded>
  <name>CG-CG</name>
  <type1>CG</type1>
  <type2>CG</type2>
  ...
  <re>
    <function>cbspl or lj126</function>
      <cbspl>
        <nknots>48</nknots>
```

```
        </cbspl>
    </re>
    ...
</non-bonded>
```

Currently, standard Lennard-Jones 12-6 (lj126) and uniform cubic B-splines-based piecewise polynomial (cbspl) functional forms are supported. For lj126, the parameters to optimize are the usual C_{12} and C_6. The cbspl form is defined as

$$u_{\text{cbspl}}(r) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} c_k \\ c_{k+1} \\ c_{k+2} \\ c_{k+3} \end{bmatrix}, \tag{7.1}$$

where $\{c_0, c_1, c_2, ..., c_m\}$ are the spline knot values tabulated for m evenly spaced intervals of size $\Delta r = r_{\text{cut}}/(m-2)$ along the separation distance $r_i = i \times \Delta r$ with the cut-off r_{cut}, and t is given by

$$t = \frac{r - r_k}{\Delta r}, \tag{7.2}$$

where index k is determined such that $r_k \leq r < r_{k+1}$. For cbspl, the knot values, $\{c_0, c_1, c_2, ..., c_m\}$, are optimized. The number of knot values to use must be specified in the XML setting file as shown in the above snippet. $u_{\text{cbspl}}(r)$ exhibits remarkable flexibility, and it can represent various complex functional characteristics of pair potentials for sufficiently large number of knots.

7.4.2 Update scaling parameter

Depending on the quality of the initial guess and sensitivity of the CG system to the CG parameters, scaling of the parameter update size may be required to ensure the stability and convergence of the RE minimization. The scaling parameter, $\chi \in (0...1)$, value can be specified in the XML settings file.

7.4.3 Statistical averaging of parameters

Due to stochastic nature of the CG simulations, near convergence, the CG potential paramters may fluctuate around the mean converged values. Therefore, the optimal CG parameters can be estimated by averaging over the last few iterations. To specify averaging, the `average`, keyword should be specified in the `post_update` options in the XML settings file.

7.4.4 General considerations

To ensure the stability of the relative entropy minimization, some precautionary measures are taken. For the Newton-Raphson update to converge towards a minimum, the Hessian, $\mathbf{H}$, must be positive definite at each step. With a good initial guess for the CG parameters and by adjusting the value of the relaxation parameter, χ, stability of the Newton-Raphson method can be ensured. One approach to initialize the CG parameters can be to fit them to PMF obtained by inverting the pair distributions of the CG sites obtained from the reference AA ensemble. For the lj126 and cbspl forms, which are linear in its parameters, the second derivative of S_{rel} is never negative, hence the minimization converges to a single global minimum. However, due to locality property of the cbspl form, i.e., update to c_i affects only the value of the potential near r_i, and the poor sampling of the very small separation distances in the high repulsive core, the rows of $\mathbf{H}$ corresponding to the first few spline knots in the repulsive core may become zero causing $\mathbf{H}$ to be a singular matrix. To avoid this singularity issue, we specify a minimum separation distance, r_{min}, for each CG pair interaction and remove the spline knots corresponding to the $r \leq r_{\text{min}}$ region from the Newton-Raphson update. Once the remaining knot values are updated, the knot values in the poorly sampled region, i.e., $r \leq r_{\text{min}}$, are linearly extrapolated. The value of r_{min} at each iteration

is estimated from the minimum distance at which the CG RDF from the CG-MD simulation is nonzero. Also, to ensure that the CG pair potentials and forces go smoothly to zero near r_{cut}, 2 knot values before and after r_{cut}, i.e., total 4, are fixed to zero.

7.5 Pressure correction

The pressure of the coarse-grained system usually does not match the pressure of the full atomistic system. This is because iterative Boltzmann inversion only targets structural properties but not thermodynamic properties. In order correct the pressure in such a way that it matches the target pressure (*inverse.p_ target*)., different strategies have been used based on small modifications of the potential. The correction can be enable by adding pressure to the list of *inverse.post_ update* scripts. The type of pressure correction is selected by setting *inverse.post_ update_ options.pressure.type*.

7.5.1 Simple pressure correction

In ref.[8] a simple linear attractive potential was added to the coarse-grained potential

$$\Delta V(r) = A\left(1 - \frac{r}{r_{cutoff}}\right), \tag{7.3}$$

with prefactor A

$$A = -\operatorname{sgn}(\Delta P) 0.1 k_B T \min(1, |f\Delta P)\,, \tag{7.4}$$

$\Delta p = P_i - P_{\mathrm{target}}$, and scaling factor f and P_{target} can be specified in the settings file as *inverse.post_ update_ options.pressure.simple.scale* and *inverse.p_ target*.

As an example for a block doing simple pressure correction, every third interaction is

```
<post_update>pressure</post_update>
<post_update_options>
  <pressure>
    <type>simple</type>
    <do>0 0 1</do>
    <simple>
      <scale>0.0003</scale>
    </simple>
  </pressure
</post_update_options>
```

Here, *inverse.post_ update_ options.pressure.simple.scale* is the scaling factor f. In order to get the correct pressure it can become necessary to tune the scaling factor f during the iterative process.

7.5.2 Advanced pressure correction

In [21] a pressure correction based on the virial expression of the pressure was introduced. The potential term remains as in the simple form while a different sturcture of the A factor is used:

$$A = \left[\frac{-2\pi\rho^2}{3r_{cut}} \int_0^{r_{cut}} r^3 g_i(r) dr\right] A_i = \Delta P. \tag{7.5}$$

This factor requires the particle density ρ as additional input parameter, which is added as *inverse.particle_ dens* in the input file.

7.6 Kirkwood-Buff correction

In order to reproduce the exact Kirkwood-Buff ingetrals (KBIs), an correction term can be added into the coarse-grained potential [22],

$$\Delta U_{ij}^{(n)}(r) = \frac{k_B T}{} A \left(G_{ij}^{(n)} - G_{ij}^{\mathrm{ref}}\right) \left(1 - \frac{r}{r_{\mathrm{ramp}}}\right), \tag{7.6}$$

where $G_{ij}^{(ref)}$ is the KBI calculated from the reference all-atom simulation and $G_{ij}^{(n)}$ is the KBI after the n^{th} iteration.

The Kirkwood-Buff integrals are calculated from the radial distribution functions as follows:

$$G_{ij} = 4\pi \int_0^\infty \left[g_{ij}(r) - 1\right] r^2 dr \ . \tag{7.7}$$

For simulations of finite box size we calculate the running integral up to distance R

$$G_{ij}(R) = 4\pi \int_0^R \left[g_{ij}(r) - 1\right] r^2 dr \ . \tag{7.8}$$

The average of those running integrals in the interval, where $G_{ij}(R)$ gets flat, gives a good estimate for G_{ij}:

$$G_{ij} \approx < G_{ij}(R) > |_{R=r_1}^{R=r_2} \tag{7.9}$$

As an example for a block doing Kirkwood-Buff correction, every iteraction without doing potential update

```
<do_potential>0</do_potential>
<post_update>kbibi</post_update>
<post_update_options>
  <kbibi>
    <do>1</do>
    <start>1.0</start>
    <stop>1.4</stop>
    <factor>0.05</factor>
    <r_ramp>1.4</r_ramp>
  </kbibi>
</post_update_options>
```

Here, *inverse.post_ update_ options.kbibi.factor* is the scaling factor A. *inverse.post_ update_ options.kbibi.start* is r_1 and *inverse.post_ update_ options.kbibi.stop* is r_2 used to calculate the average of $G_{ij}(R)$.

7.7 Runtime optimization

Most time per iteration is spent on running the coarse-grained system and on calculating the statistics. To get a feeling on how much statistics is needed, it is recommended to plot the distribution functions and check whether they are sufficiently smooth. Bad statistics lead to rough potential updates which might cause the iterative refinement to fail. All runs should be long enough to produce distributions/rdfs of reasonable quality.

Often, runtime can be improved by smoothing the potential updates. Our experience has shown that it is better to smooth the potential update instead of the rdf or potential itself. If the potential or rdf is smoothed, sharp features like the first peak in SPC/E water might get lost. Smoothing on the delta potential works quite well, since the sharp features are already present from the initial guess. By applying iterations of a simple triangular smoothing ($\Delta U_i = 0.25\Delta U_{i-1} + 0.5\Delta U_i + 0.25\Delta U_{i+1}$), a reasonable coarse-grained potential for SPC/E water could be produced in less than 10 minutes. Smoothing is implemented as a post_update script and can be enabled by adding

```
<post_update>smooth</post_update>
<post_update_options>
  <smooth>
      <iterations>2</iterations>
  </smooth>
</post_update_options>
```

to the inverse section of an interaction in the settings XML file.

7.8 Coordination Iterative Boltzmann Inversion

The method $\mathcal{C}-$IBI (Coordination Iterative Boltzmann Inversion) uses pair-wise cumulative coordination as a target function within an iterative Boltzmann inversion. This method reproduces solvation thermodynamics of binary and ternary mixtures [23].

The estimation of coordination is given by:

$$\mathcal{C}_{ij}(r) = 4\pi \int_0^r \mathrm{g}_{ij}(r')r'^2 dr' \tag{7.10}$$

with the indices i and j standing for every set of pairs, uses a volume integral of $\mathrm{g}(r)$.

The Kirkwood and Buff theory (KB) [24] connects the pair-wise coordinations with particule fluctuations and, thus, with the solution thermodynamics [25, 26]. This theory make use of the Kirkwood-Buff integrals (KBI) G_{ij} defined as,

$$\mathrm{G}_{ij} = 4\pi \int_0^\infty \left[\mathrm{g}_{ij}(r) - 1\right] r^2 dr. \tag{7.11}$$

For big system sizes the G_{ij} can be approximated:

$$\mathrm{G}_{ij} = \mathcal{C}_{ij}(r) - \frac{4}{3}\pi r^3, \tag{7.12}$$

were the second therm is a volume correction to $\mathcal{C}_{ij}(r)$.

Thus the initial guess for the potential of the CG model is obtained from the all atom simulations,

$$\mathrm{V}_0(r) = -k_B T \ln\left[\mathrm{g}_{ij}(r)\right], \tag{7.13}$$

however, the iterative protocol is modified to target $\mathcal{C}_{ij}(r)$ given by,

$$\mathrm{V}_n^{\mathcal{C}-\mathrm{IBI}}(r) = \mathrm{V}_{n-1}^{\mathcal{C}-\mathrm{IBI}}(r) + k_B T \ln\left[\frac{\mathcal{C}_{ij}^{n-1}(r)}{\mathcal{C}_{ij}^{target}(r)}\right]. \tag{7.14}$$

To perform the $\mathcal{C}-$IBI is necessary include some lines inside of the .xml file:

```
<cg>
 <non-bonded>
  <name>A-A</name>
  ...
  <inverse>
   <post_update>cibi</post_update>
   <post_update_options>
     <cibi>
       <do>1</do>
     </cibi>
   </post_update_options>
   ...
</cg>
```

Chapter 8

DL_POLY interface

WARNING: The DL_POLY interface is still experimental (in development) but it does support the Iterative Boltzmann Inversion and Inverse Monte Carlo schemes. The Force Matching might work as well, although it has not been tested thoroughly.

8.1 General remarks on using VOTCA with DL_POLY

The DL_POLY interface fully supports coarse-grain mapping of a full-atom system previuosly simulated with any version of DL_POLY, including DL_POLY-Classic. However, the full optimization of the effective potentials with the aid of iterative methods will only become possible when the new release of DL_POLY-4 (4.06) is made public; the reason being the incapability of earlier DL_POLY versions of using user-specified tabulated force-fields for intramolecular, aka "bonded", interactions: bonds, angles, dihedral angles (torsions). Below the coarse-graining and CG force-field optimization with the aid of the latest DL_POLY-4 version (4.06+) are outlined.

Running VOTCA with DL_POLY-4 as MD simulation engine is very similar to doing so with GROMACS. The three types of required input files in the case of DL_POLY are: CONTROL – containing the simulation directives and parameters (instead of `.mdp` file for GROMACS), FIELD – the topology and force-field specifications (instead of `.top` and `.tpr` files), and CONFIG (instead of `.gro` file) – the initial configuration file, containing the MD cell matrix and particle coordinates (it can also include initial velocities and/or forces); for details see DL_POLY-4 manual. Most of the VOTCA tools and scripts described above in the case of using GROMACS will work in the same manner, with the following conventional substitutions for the (default) file names used in options for VOTCA scripts, as necessary:

```
.dlpf = the topology read from FIELD or written to FIELD_CGV
.dlpc = the configuration read from CONFIG or written to CONFIG_CGV
.dlph = the trajectory read from HISTORY or written to HISTORY_CGV
```

It is also possible to specify file names different from the standard DL_POLY convention, in which case the user has to use the corresponding dot-preceded extension(s); for example: FA-FIELD.dlpf instead of FIELD or CG-HISTORY.dlph instead of HISTORY_CGV (see section 10.1, as well as the man pages or output of VOTCA commands, with option `--help`).

VOTCA follows the DL_POLY conventions for file names and formats. Thus, `csg_dlptopol` and `csg_map` produce the CG topology (FIELD_CGV by default), configuration (CONFIG_CGV), and/or trajectory (HISTORY_CGV) files fully compatible with and usable by DL_POLY. **Note that the ability of these tools to read and write a plethora of different file formats provides means to convert input and output files between the simulation packages supported by VOTCA, e.g. GROMACS– DL_POLY or vice versa. The user is, however, strongly advised to check the resulting files for consistency before using them).**

Similarly, the distribution analysis and potential/force generation utilities, such as `csg_stat` and VOTCA scripts, will read and write DL_POLY-formatted files; in particular, the tabulated force-field files containing the potential and force/virial data: TABLE – for short-range (VdW) "non-bonded" interactions, TABBND, TABANG and TABDIH – for "bonded" interations: bonds, bending angles and dihedrals, correspondingly (for the format details see DL_POLY-4 manual). Note, however, that the latter three files can only be used by DL_POLY-4 (4.06+).

The user is advised to search for "dlpoly" through the `csg_defaults.xml`, `csg_table` files and in scripts located in `share/votca/scripts/inverse/` in order to find out about the xml-tags and options specific for DL_POLY; see also sections 10.4 and 10.5.

Chapter 9

Advanced topics

9.1 Customization

Each sub-step of an iteration and all direct calls can be adjusted to the user needs. The internal part of the iterative framework is organized as follows: all scripts are called using two keywords

```
csg_call key1 key2
```

For example, `csg_call update imc` calls the `update` script for the inverse Monte Carlo procedure. The corresponding keywords are listed in sec. 10.5 or can be output directly by calling

```
csg_call --list
```

It is advised not to change already implemented scripts. To customize a script or add a new one, copy the script to your own directory (set by *inverse.scriptpath*) and redirect its call by creating your own `csg_table` file in this directory which looks like this

```
key1 key2 script1 options
key3 key4 script2
```

If the local keys are already in use, the existing call will be overloaded.

As an example, we will illustrate how to overload the script which calls the sampling package. The **csg_inverse** script runs **mdrun** from the GROMACS package only on one cpu. Our task will be to change the script so that GROMACS uses 8 cpus, which is basically the same as adding mpirun options in *inverse.gromacs.mdrun.command*.

First we find out which script calls **mdrun**:

```
csg_call --list | grep gromacs
```

The output should look as follows

```
init gromacs initalize_gromacs.sh
prepare gromacs prepare_gromacs.sh
run gromacs run_gromacs.sh
pressure gromacs calc_pressure_gromacs.sh
rdf gromacs calc_rdf_gromacs.sh
imc_stat gromacs imc_stat_generic.sh
convert_potential gromacs potential_to_gromacs.sh
```

the third line indicates the script we need. If the output of **csg_call** is not clear, one can try to find the right script in sec. 10.5. Alternatively, check the folder

```
<csg-installation>/share/scripts/inverse
```

for all available scripts.

Analyzing the output of

```
csg_call --cat run gromacs
```

we can conclude that this is indeed the script we need as the content (in shorted form is):

```
critical mdrun
```

Now we can create our own `SCRIPTDIR`, add a new script there, make it executable and overload the call of the script:

```
mkdir -p SCRIPTDIR
cp `csg_call --quiet --show run gromacs` SCRIPTDIR/my_run_gromacs.sh
chmod 755 SCRIPTDIR/my_run_gromacs.sh
echo "run gromacs my_run_gromacs.sh" >> SCRIPTDIR/csg_table
```

Please note that `my_run_gromacs.sh` is the name of the script and `SCRIPTDIR` is the custom script directory, which can be a global or a local path. Now we change the last line of `my_run_gromacs.sh` to:

```
critical mpirun -np 8 mdrun
```

This completes the customization. Do not forget to add `SCRIPTDIR` to *inverse.scriptpath* in the setting XML file (see sec. 10.4).

You can check the new script by running:

```
csg_call --scriptdir SCRIPTDIR --list
csg_call --scriptdir SCRIPTDIR --run run gromacs
```

Finally, do not forget to remove the license infomation and change the version number of the script.

9.2 Used external packages

9.2.1 GroMaCS

Get it from www.gromacs.org

- **mdrun**
- **grompp**

9.2.2 ESPResSo

Get it from www.espressomd.org

9.2.3 DL_POLY

Get it from www.ccp5.ac.uk/DL_POLY

9.2.4 Gnuplot

Get it from www.gnuplot.info

9.2.5 GNU Octave

Get it from www.gnu.org

9.2.6 LAMMPS

Get it from lammps.sandia.gov

9.2.7 Matlab

Get it from www.mathworks.com

9.2.8 NumPy

Get it from http://numpy.scipy.org

Chapter 10

Reference

10.1 Programs

10.1.1 csg_boltzmann

Performs tasks that are needed for simple boltzmann inversion in an interactive environment.
Allowed options:

`-h [ --help ]` display this help and exit
`-v [ --verbose ]` be loud and noisy
`--top arg` atomistic topology file

Mapping options:

`--cg arg` coarse graining mapping and bond definitions (xml-file)
`--map-ignore arg` list of molecules to ignore separated by ;
`--no-map` disable mapping and act on original trajectory
Special options:
`--excl arg` write atomistic exclusion list to file

Trajectory options:

`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time (only works for Gromacs files)
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.2 csg_call

This script calls scripts and functions for the iterative framework. Function can be executed or shows if key1='function'.
Usage: `csg_call [OPTIONS] key1 key2 [SCRIPT OPTIONS]`
Allowed options:

`-l, --list` Show list of all script
`--cat` Show the content of the script
`--show` Show the path to the script
`--show-share` Shows the used VOTCASHARE dir and exits
`--scriptdir DIR` Set the user script dir (Used if no options xml file is given) Default: empty
`--options FILE` Specify the options xml file to use
`--log FILE` Specify the log file to use Default: stdout
`--ia-type type` Specify the interaction type to use
`--ia-name name` Specify the interaction name to use
`--nocolor` Disable colors

`--sloppy-tables` Allow tables without flags
`--debug` Enable debug mode with a lot of information
`-h, --help` Show this help
Examples:
`csg_call table smooth [ARGUMENTS]`
`csg_call --show run gromacs`

10.1.3 csg_density

Calculates the mass density distribution along a box axis or radial density profile from reference point
Allowed options:
`-h [ --help ]` display this help and exit
`-v [ --verbose ]` be loud and noisy
`--top arg` atomistic topology file
Mapping options:
`--cg arg` [OPTIONAL] coarse graining mapping and bond definitions (xml-file). If no file is given, program acts on original trajectory
`--map-ignore arg` list of molecules to ignore if mapping is done separated by ;
Specific options::
`--type arg (=mass)` density type: mass or number
`--axis arg (=r)` [x|y|z|r] density axis (r=spherical)
`--step arg (=0.01)` spacing of density
`--block-length arg` write blocks of this length, the averages are cleared after every write
`--out arg` Output file
`--rmax arg` rmax (default for [r] =min of all box vectors/2, else l)
`--scale arg (=1)` scale factor for the density
`--molname arg (=*)` molname
`--filter arg (=*)` filter bead names
`--ref arg` reference zero point
Trajectory options:
`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time (only works for Gromacs files)
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.4 csg_dlptopol

Create a dlpoly topology template based on an existing (atomistic) topology and a mapping xml-file. The created template file needs to be inspected and amended by the user!
Examples:
`csg_dlptopol --top .dlpf --out .dlpf --cg cg-map.xml` convert FIELD to FIELD_CGV using cg-map.xml
`csg_dlptopol --top FA-dlpoly.dlpf --out CG-dlpoly.dlpf --cg cg-map.xml`
`csg_dlptopol --top FA-gromacs.tpr --out FA-dlpoly.dlpf --no-map`
Allowed options:
`-h [ --help ]` display this help and exit
`-v [ --verbose ]` be loud and noisy
`--top arg` atomistic topology file
`--out arg` output topology in dlpoly format
Mapping options:

--cg arg coarse graining mapping and bond definitions (xml-file)
--map-ignore arg list of molecules to ignore separated by ;
--no-map disable mapping and act on original trajectory

10.1.5 csg_dump

Print atoms that are read from topology file to help debugging atom naming.
Allowed options:
-h [--help] display this help and exit
-v [--verbose] be loud and noisy
--top arg atomistic topology file
Mapping options:
--cg arg [OPTIONAL] coarse graining mapping and bond definitions (xml-file). If no file is given, program acts on original trajectory
--map-ignore arg list of molecules to ignore if mapping is done separated by ;
Specific options:
--excl display exclusion list instead of molecule list

10.1.6 csg_fmatch

Perform force matching (also called multiscale coarse-graining)
Allowed options:
-h [--help] display this help and exit
-v [--verbose] be loud and noisy
--top arg atomistic topology file
--options arg options file for coarse graining
--trj-force arg coarse-grained trajectory containing forces of already known interactions
Mapping options:
--cg arg coarse graining mapping and bond definitions (xml-file)
--map-ignore arg list of molecules to ignore separated by ;
--no-map disable mapping and act on original trajectory
Trajectory options:
--trj arg atomistic trajectory file
--begin arg (=0) skip frames before this time (only works for Gromacs files)
--first-frame arg (=0) start with this frame
--nframes arg process the given number of frames

10.1.7 csg_gmxtopol

Create skeleton for gromacs topology based on atomistic topology and a mapping file. File still needs to be modified by the user.
Allowed options:
-h [--help] display this help and exit
-v [--verbose] be loud and noisy
--top arg atomistic topology file
--out arg output topology (will create .top and in future also .itp)
Mapping options:
--cg arg coarse graining mapping and bond definitions (xml-file)
--map-ignore arg list of molecules to ignore separated by ;
--no-map disable mapping and act on original trajectory

10.1.8 csg_imcrepack

This program is internally called by inversion scripts to kick out zero entries in matrix for inverse Monte Carlo. It also extracts the single potential updates out of the full solution.
Allowed options:

`--in arg` files to read
`--out arg` files to write
`--unpack arg` extract all tables from this file
`--help` display help message

10.1.9 csg_inverse

Start the script to run ibi, imc, etc. or clean out current dir
Usage: `csg_inverse [OPTIONS] --options settings.xml [clean]`
Allowed options:

`-h, --help` show this help
`-N, --do-iterations N` only do N iterations (ignoring settings.xml)
`--wall-time SEK` Set wall clock time
`--options FILE` Specify the options xml file to use
`--debug` enable debug mode with a lot of information
`--nocolor` disable colors

Examples:

```
csg_inverse --options cg.xml
csg_inverse -6 --options cg.xml
```

10.1.10 csg_map

Convert a reference atomistic trajectory or configuration into a coarse-grained one based on a mapping xml-file. The mapping can be applied to either an entire trajectory or a selected set of frames only (see options).
Examples:

```
csg_map --top FA-topol.tpr --trj FA-traj.trr --out CG-traj.xtc --cg
cg-map.xml
csg_map --top FA-topol.tpr --trj FA-conf.gro --out CG-conf.gro --cg
cg-map.xml
csg_map --top FA-topol.tpr --trj FA-traj.xtc --out FA-history.dlph --no-map
csg_map --top FA-field.dlpf --trj FA-history.dlph --out CG-history.dlph
--cg cg-map.xml
csg_map --top .dlpf --trj .dlph --out .dlph --cg cg-map.xml convert HIS-
TORY to HISTORY_CGV
```

Allowed options:

`-h [ --help ]` display this help and exit
`-v [ --verbose ]` be loud and noisy
`--top arg` atomistic topology file
`--out arg` output file for coarse-grained trajectory
`--vel` Write mapped velocities (if available)
`--force` Write mapped forces (if available)
`--hybrid` Create hybrid trajectory containing both atomistic and coarse-grained

Mapping options:

`--cg arg` coarse graining mapping and bond definitions (xml-file)

--map-ignore arg list of molecules to ignore separated by ;
--no-map disable mapping and act on original trajectory
Trajectory options:
--trj arg atomistic trajectory file
--begin arg (=0) skip frames before this time (only works for Gromacs files)
--first-frame arg (=0) start with this frame
--nframes arg process the given number of frames

10.1.11 csg_property

Helper program called by inverse scripts to parse xml file.
Allowed options:
--help produce this help message
--path arg list option values that match given criteria
--filter arg list option values that match given criteria
--print arg (=.) list option values that match given criteria
--file arg xml file to parse
--short short version of output
--with-path include path of node in output

10.1.12 csg_resample

Change grid and interval of any sort of table files. Mainly called internally by inverse script, can also be used to manually prepare input files for coarse-grained simulations.
Allowed options:
--help produce this help message
--in arg table to read
--out arg table to write
--derivative arg table to write
--grid arg new grid spacing (min:step:max). If 'grid' is specified only, interpolation is performed.
--type arg (=akima) [cubic|akima|linear]. If option is not specified, the default type 'akima' is assumed.
--fitgrid arg specify fit grid (min:step:max). If 'grid' and 'fitgrid' are specified, a fit is performed.
--nocut Option for fitgrid: Normally, values out of fitgrid boundaries are cut off. If they shouldn't, choose --nocut.
--comment arg store a comment in the output table
--boundaries arg (natural|periodic|derivativezero) sets boundary conditions

10.1.13 csg_reupdate

computes relative entropy update.
Allowed options:
-h [--help] display this help and exit
-v [--verbose] be loud and noisy
--top arg atomistic topology file (only needed for RE update)
RE Specific options:
--options arg options file for coarse graining
--gentable arg (=0) only generate potential tables from given parameters, NO RE update!

`--interaction arg` [OPTIONAL] generate potential tables only for the specified interactions, only valid when 'gentable' is true
`--param-in-ext arg (=param.cur)` Extension of the input parameter tables
`--param-out-ext arg (=param.new)` Extension of the output parameter tables
`--pot-out-ext arg (=pot.new)` Extension of the output potential tables
`--hessian-check arg (=1)` Extension of the output potential tables
Threading options:
`--nt arg (=1)` number of threads
Trajectory options:
`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time (only works for Gromacs files)
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.1.14 csg_stat

Calculate all distributions (bonded and non-bonded) specified in options file. Optionally calculates update Eigen::Matrix3d for invere Monte Carlo. This program is called inside the inverse scripts. Unlike csg_boltzmann, big systems can be treated as well as non-bonded interactions can be evaluated.
Allowed options:
`-h [ --help ]` display this help and exit
`-v [ --verbose ]` be loud and noisy
`--top arg` atomistic topology file
Mapping options:
`--cg arg` [OPTIONAL] coarse graining mapping and bond definitions (xml-file). If no file is given, program acts on original trajectory
`--map-ignore arg` list of molecules to ignore if mapping is done separated by ;
Specific options:
`--options arg` options file for coarse graining
`--do-imc` write out additional Inverse Monte Carlo data
`--block-length arg` write blocks of this length, the averages are cleared after every write
`--ext arg (=dist.new)` Extension of the output
Threading options:
`--nt arg (=1)` number of threads
Trajectory options:
`--trj arg` atomistic trajectory file
`--begin arg (=0)` skip frames before this time (only works for Gromacs files)
`--first-frame arg (=0)` start with this frame
`--nframes arg` process the given number of frames

10.2 Mapping file

The root node always has to be cg_molecule. It can contain the following keywords:
Please mind that dots in xml tags have to replaced by subtags, e.g. x.y has to be converted to x with subtag y.

cg_molecule
ident Molecule name in reference topology.
maps Section containing definitions of mapping schemes.
map Section for a mapping for 1 bead.

name Name of the mapping
weights Weights of the mapping matrix. Entries are normalized to 1, number of entries must match the number of reference beads in a coarse-grained bead.
name Name of molecule in coarse-grained representation.
topology Section defining coarse grained beads of molecule.
cg_beads Section defining coarse grained beads of molecule.
cg_bead Definition of a coarse grained bead.
cg_bead.beads The beads section lists all atoms of the reference system that are mapped to this particular coarse grained bead. The syntax is RESID:RESNAME:ATOMNAME the beads are separated by spaces.
cg_bead.mapping Mapping scheme to be used for this bead (specified in section mapping) to map from reference system.
cg_bead.name Name of coarse grained bead.
cg_bead.type Type of coarse grained bead.
cg_bonded The cg_bonded section contains all bonded interaction of the molecule. Those can be bond, angle or dihedral. An entry for each group of bonded interaction can be specified, e.g. several groups (types) of bonds can be specified. A specific bonded interaction can be later on addressed by MOLECULE:NAME:NUMBER, where MOLECULE is the molecule ID in the whole topology, NAME the name of the interaction group and NUMBER addresses the interaction in the group.
angle Definition of a group of angles.
angle.beads List of triples of beads that define a bond. Names specified in cg_beads
angle.name Name of the angle
bond Definition of a group of bonds.
bond.beads List of pair of beads that define a bond. Names specified in cg_beads
bond.name Name of the bond.
dihedral Definition of a group of dihedrals. Since the exact functional form does not matter, this combines proper as well as improper dihedrals.
dihedral.beads List of quadruples of beads that define a bond. Names specified in cg_beads
dihedral.name Name of the dihedral

10.3 Topology file

The XML topology file

Please mind that dots in xml tags have to replaced by subtags, e.g. x.y has to be converted to x with subtag y.

topology The XML topology root element, the base for the topology can be defined by the "name" attribute
beadtypes Allows defining bead types
mass Define the mass of the bead type; attributes: "name" - the bead type name, "value" - the new mass
rename Rename the bead type; attributes: "name" - the old name, "newname" - the new name
bonded This section defines the topology of the molecules, it is used to generate proper exclusions for calculating rdfs
angle Describes the angle
beads The triplet of the beads in the format MOLECULE_NAME:BEAD_NAME
name The name of the angle
bond Describes the bond

beads The pair of the beads in the format MOLECULE_NAME:BEAD_NAME
name The name of the bond
dihedral Describes the dihedrals
beads The quadruplet of the beads in the format MOLECULE_NAME:BEAD_NAME
name The name of the dihedral
h5md_particle_group Attribute name holds the name of particles group in H5MD file
molecules The the molecules in the trajectory or other operation on the molecules.
clear Clear the information about the molecules
define Define the molecules; attributes: "name" - the name of molecule, "first" - the id of first molecule, "nbeads" - the number of beads in the molecule, "nmols" - the number of molecules
molecule Definition of the molecule, with attributes: name, nmols and nbeads. The name defines residue name, nmols tells how many times this molecule has to be replicated to match with trajectory file and nbeads defines number of beads in every molecule.
bead Define the bead in the molecule. Attributes are: name - the name of bead, type - the type of bead, mass - the mass of bead, q - the value of charge and resid - the id of the residue the bead belongs to (>=1).
rename Rename the molecules; attributes: "name" - the new name, "range" - the range where the new name will be set in the format start_range:end_range

10.4 Settings file

All options for the iterative script are stored in an xml file. **Please mind that dots in xml tags have to replaced by subtags, e.g. x.y has to be converted to x with subtag y.**

cg Section containing the all coarse-graining options
bonded Interaction specific option for bonded interactions, see the cg.non-bonded section for all options
dlpoly
header Header of the interaction in dlpoly TABBND or TABANG file. The header should be a unique set of the interaction-site names, and these should match the corresponding names specified in the mapping file.
name Name of the bonded interaction. The name can be arbitrary but should be unique. For bonded interactions, this should match the name specified in the mapping file.
periodic set to 1 when calculating bond dihedral potentials with csg_fmatch -> enforces periodicity of potential. (default is 0) (default 0)
fmatch Force matching options
constrainedLS boolean variable: false - simple least squares, true - constrained least squares. For details see the VOTCA paper. Practically, both algorithms give the same results, but simple least squares is faster. If you are a mathematician and you think that a spline can only then be called a spline if it has continuous first and second derivatives, use constrained least squares.
dist Accuracy for evaluating the difference in bead positions. Default is 1e-5 (default 1e-5)
frames_per_block number of frames, being used for block averaging. Atomistic trajectory, specified with --trj option, is divided into blocks and the force matching equations are solved separately for each block. Coarse-grained force-field, which one gets on the output is averaged over those blocks.
inverse general options for inverse script
average

steps number of steps to be used for average computation. For relative entropy method, these many last iteration steps are used to compute average CG potentials or parameters or both. (default 1)

cleanlist these files are removed after each iteration

convergence_check

limit lower convergency limit to stop (default 0)

type type of convergence check to do (default none)

dist_min minimal value for the rdf to consider for initial guess of the potential) (default 1e-10)

dlpoly general dlpoly specific options

angles dlpoly specs for tabulated bonded potentials (applied to all angles)

angles.table_grid dlpoly internal grid number for tabulated potentials

bonds dlpoly specs for tabulated bonded potentials (applied to all bonds)

bonds.table_end dlpoly internal grid end point for tabulated potentials

bonds.table_grid dlpoly internal grid number for tabulated potentials

checkpoint Names of the dlpoly checkpoint files (default REVIVE REVCON)

command command to run dlpoly (name or absolute path or 'mpirun dlpoly' or such) (default DLPOLY.Z)

dihedrals dlpoly specs for tabulated bonded potentials (applied to all dihedrals)

dihedrals.table_grid dlpoly internal grid number for tabulated potentials

table_end dlpoly internal grid end point for tabulated non-bonded potentials (applied to all non-bonded)

table_grid dlpoly internal grid number for tabulated non-bonded potentials (applied to all non-bonded)

topol Name of dlpoly topology file (default .dlpf)

traj Name of the output dlpoly trajectory file (default .dlph)

espresso

command Command to run espresso (name or absolute path or mpirun espresso..) (default python3)

first_frame trash the given number of frames at the beginning of trajectory (default 0)

opts option to be given to espresso program, use ${script} in there (default ${script})

table_bins espresso internal grid for tabulated potentials

traj Name of the output Espresso trajectory file

espressopp

command Command to run espresso (name or absolute path or mpirun espresso..) (default python2)

first_frame trash the given number of frames at the beginning of trajectory (default 0)

opts option to be given to espresso program, use ${script} in there (default ${script})

filelist these files are copied to each iteration step

gnuplot

bin gnuplot binary to use (default gnuplot)

gromacs gromacs specific options

conf Name of the coordinate file read by grompp (default conf.gro)

conf_out Name of the original outcome coordinate written by mdrun (default confout.gro)

density

density.block_length Length of the block for the error analysis

density.with_errors calculate error on the density: yes/no (default no)

equi_time begin analysis after this time when using gromacs (max of this

and first_frame is used) (default 0)
first_frame trash the given number of frames at the beginning of trajectory (max of this and first_frame is used) (default 0)
g_energy
g_energy.bin Name (or absolute path) of the g_energy binary (default /usr/bin/gmx_d energy)
g_energy.opts Additional options to Gromacs g_energy (e.g. -P 1)
g_energy.pressure options for pressure calculation using g_energy
g_energy.pressure.allow_nan is nan an allowed result: yes/no (default no)
g_energy.topol Gromacs binary topol (tpr) file to use by g_energy
gmxrc GMXRC to source at the startup
grompp
grompp.bin Name (or absolute path) of the grompp binary (default /usr/bin/gmx_d grompp)
grompp.opts Additional options to Gromacs grompp (e.g. -maxwarn 1)
index Gromacs grompp index file to used by grompp (default index.ndx)
log Separate log file for gromacs programs (useful with mdrun -v)
mdp Gromacs mdp file to be used by grompp (default grompp.mdp)
mdrun
mdrun.checkpoint Name of the checkpint to use in case of restarted simulation (default state.cpt)
mdrun.command Command to run mdrun (name or absolute path or mpirun mdrun..) (default /usr/bin/gmx_d mdrun)
mdrun.multidir List of directories for multidir simulations
mdrun.opts Additional options to Gromacs mdrun (e.g. -nosum)
pot_max cut the potential at this value (gromacs bug) (default 1000000)
pre_simulation A pre simulation (e.g. minimization / equilibration) is a simulation with a different mdp/topol/index (default no)
pre_simulation.index Gromacs grompp index file to used by grompp in the pre simulation
pre_simulation.mdp Gromacs mdp file to be used by grompp in the pre simulation
pre_simulation.topol_in Gromacs text topol (top) file to use by grompp in the pre simulation
rdf
rdf.block_length Length of the block for the error analysis
rdf.map Space separated list of special mapping file(s) for rdf calculations needed for bonded interactions
rdf.with_errors calculate error on the rdf: yes/no (default no)
ref Options for the case that calculation of reference system is needed
ref.equi_time begin analysis after this time when using gromacs (max of this and first_frame is used) (default 0)
ref.first_frame trash the given number of frames at the beginning of trajectory (max of this and first_frame is used) (default 0)
ref.mapping Mapping to apply on the coarse-grained topology, use autogenerated ones (cg.inverse.optimizer.mapping.output) and given ones (map other components)
ref.rdf Contains options for Reference rdf calculation
ref.rdf.opts Extra options to give to csg_stat (e.g. --nframes 100)
ref.topol Reference binary topology(global or local path)
ref.traj Reference trajectory(global or local path)
table_bins grid for gromacs xvg table (default 0.002)
table_end extend the gromacs xvg tables to this value
temp_check check kBT against t_ref in mdp file: yes/no (default yes)

topol Gromacs binary topology (tpr) file to be written by grompp and used for the simlation (default topol.tpr)
topol_in Gromacs text topology (top) file read by grompp (default topol.top)
traj Gromacs trajectory file to use (default traj.xtc)
trjcat
trjcat.bin Name (or absolute path) of the trjcat binary (default /usr/bin/gmx_d trjcat)

hoomd-blue
command Command to run hoomd-blue (name or absolute path or mpirun ..) (default hoomd)
opts option to be given to hoomd-blue program, use ${script} in there (default ${script})

imc general imc specific options
matlab
matlab.bin Name (or absolute path) of the matlab binary (default matlab)
octave
octave.bin Name (or absolute path) of the octave binary (default octave)
solver solver for solving a linear equation system: octave/numpy/matlab

initial_configuration what initial configuration to use in every step: maindir/laststep/nowhere. (default maindir)
iterations_max do the given number of iterations (0=inf)
kBT kBT in simulation progam units (XXX K *0.00831451 for gromacs)
lammps general lammps specific options
command command to run lammps (name or absolute path or mpirun lammps..) (default /usr/bin/lmp)
opts option to be given to lammps program, use ${script} in there (default -in ${script})
pressure_file pressure file generated by lammps, use "fix print" in lammps input (e.g., "fix pressure all print 50 "${mypress}" file lammps.pressure screen no title "LAMMPS_PRESSURE" " ; pressure_file would be lammps.pressure in this example). The title can be anything as VOTCA skips over this line as a header when parsing
script lammps script to run
traj trajectory file to be created by lammps, use a format like xyz, which can be read by csg_stat

log_file name of the log file (default inverse.log)
map Special mapping file(s) for rdf calculations needed for bonded interactions
method method to be performed: ibi/imc/ft/optimizer
optimizer
cma general options for the cma optimizer
cma.eps standard epsilon, in which the best solution is searched
type Type of optimizer to be used

program simulation package to be used (gromacs/espresso/lammps) (default gromacs)
re general options for realtive entropy method
csg_reupdate
csg_reupdate.opts options for the csg_reupdate command

restart_file Name of the restart file in case a step has to be resumed (default restart_points.log)
scriptpath list of directories for user scripts (e.g. $PWD) separated by a colon (like PATH)
sim_prog options, which apply to all simulation programs
command Command to run for the simulation (name or absolute path or mpirun XXX ..)

conf Name of the coordinate file read by the simulation program (if needed)
conf_out Name of the original outcome coordinate written by simulation program (if any)
density
density.block_length Length of the block for the error analysis
density.with_errors calculate error on the density: yes/no (default no)
equi_time begin analysis after this time (max of this and first_frame is used) (default 0)
first_frame trash the given number of frames at the beginning of trajectory (max of this and first_frame is used) (default 0)
imc
imc.topol Special topology file to be used for csg_stat used in imc
imc.traj Special trajectory file to be used for csg_stat used in imc
opts option to be given to simulation program, use ${script} in there
rdf
rdf.block_length Length of the block for the error analysis
rdf.map Space separated list of special mapping file(s) for rdf calculations needed for bonded interactions
rdf.topol Special topology file to be used for csg_stat
rdf.traj Special trajectory file to be used for csg_stat
rdf.with_errors calculate error on the rdf: yes/no (default n)
re
re.topol Special topology file to be used for csg_reupdate
re.traj Special trajectory file to be used for csg_reupdate
script simulation script to run (if any)
topol General topology file to be use if no special one is specified
traj trajectory file to be created by the simulation program

simulation simulation options
background tells csg_inverse that simulation was send to the backgroud (default no)
tasks number of threads to use for csg_stat (default auto)

nbsearch Grid search algorithm, simple (N square search) or grid (default grid)
non-bonded Interaction specific option for non-bonded interactions
bondtype Internal alias for "non-bonded" or "bonded", set automatically
dlpoly
header Header of the interaction in dlpoly TABLE file. The header should be a unique pair of the interaction-site names, and these should match the corresponding names specified in the mapping file.

fmatch Force matching options
max Maximum value of interval for distribution sampled in atomistic MD simulation. One can get this number by looking at the distribution function for this interaction. For non-bonded interactions it's the cut-off of the interaction.
min Minimum value of interval for distribution sampled in atomistic MD simulation. One can get this number by looking at the distribution function for this interaction. For non-bonded interactions it's the distance to the rdf start. For CG bonds and angles the variable has the similar meaning (note, that for angles it is specified in radians).
out_step Grid spacing for the output grid. Normally, one wants to have this parameter smaller than fmatch.step, to have a smooth curve, without additional spline interpolation. As a rule of thumb we normally use fmatch.out_step which is approximately 5 times smaller than fmatch.step.
step grid spacing for the spline, which represents the interaction. This parameter should not be too big, otherwise you might lose some features of the interaction potential, and not too small either, otherwise you will have un-

sampled bins which result in an ill-defined equation system and NaNs in the output.

inverse Contains all information relevant to iterative process

do_potential Update cycle for the potential update. 1 means update, 0 don't update. 1 1 0 means update 2 iterations, then don't one iteration update, then repeat. (default 1)

espresso Espresso specific options for this interations

espresso.table Name of file for tabulated potential of this interaction. This fill will be created from the internal tabulated potential format in every step. Note, though, that the original espresso script needs to contain the name of that table as the tabulated interaction (see tutorial methanol ibi_espresso for details).

gromacs Gromacs specific options for this interations

gromacs.table Name of file for tabulated potential of this interaction. This fill will be created from the internal tabulated potential format in every step.

imc ection containing inverse monte carlo specific options.

imc.group Group of interaction. Cross-correlations of all members of a group are taken into account for calculating the update. If no cross correlations should be calculated, interactions have to be put into different groups.

imc.reg magnitude for regularization parameter, default =0 (default 0)

lammps lammps specific options for this interations

lammps.scale scaling factor for the potential output, can be used to convert VOTCA units, nm, to other units, e.g. angstroms (default 1)

lammps.table Name of file for tabulated potential of this interaction. This fill will be created from the internal tabulated potential format in every step. Note, though, that the lammps script needs to contain the name of that table as the tabulated interaction and the interaction is stored in the VOTCA section of the file..

optimizer

optimizer.density Contains all options for the density calculation of the optimizer

optimizer.density.axis Axis along which the density is calculated (default x)

optimizer.density.max Upper bound of interval in which density calculation is performed.

optimizer.density.min Lower bound of interval in which density calculation is performed.

optimizer.density.molname The molname of this interaction (default *)

optimizer.density.scale Scaling factor for density (default 1.0)

optimizer.density.step Step size of interval in which density calculation is performed.

optimizer.density.target Filename of the target denstiy distribution in the maindir

optimizer.function Functional form of the interaction, using parameters in here

optimizer.functionfile If the function is very complicated it can be defined in this files, which is used as an header

optimizer.mapping option related to mapping changes

optimizer.mapping.change Does the mapping change in optimization: yes/no (default no)

optimizer.mapping.output Output file name for mapping (default no)

optimizer.mapping.template template for the mapping optimization

optimizer.parameters Parameters to be fitted by the optimizer for this interaction. Note that the parameter names are global

optimizer.pressure Contains all options for the pressure calculation of the

optimizer
optimizer.pressure.undef Pressure to use if pressure from the simulation was nan (use a big number)
optimizer.rdf Contains all options for the rdf calculation of the optimizer
optimizer.rdf.target Filename of the target rdf in the maindir
optimizer.rdf.weight Weighting function for calculating the convergency of the rdf
optimizer.rdf.weightfile File with the weighting function definition calculating the rdf
optimizer.target_weights Weight of the targets, amount has to be the same as of targets (default 1)
optimizer.targets Targets to be fitted by the optimizer (default rdf)
p_target pressure contribution of this interaction
particle_dens particle density of this species (for wjk pressure correction)
post_add Additional post processing of U after dU added to potential. This is a list of scripts separated by spaces which are called. See section on iterative framework for details.
post_add_options Contains all options of post add scripts
post_add_options.average
post_add_options.average.what list for which averages of last few steps are to computed: param, pot, ... For relative entropy method, specify param before pot.
post_add_options.compress Contains all options of the postadd compress sripts
post_add_options.compress.filelist Files to be compressed
post_add_options.compress.program Compression command to run (default gzip)
post_add_options.compress.program_opts Option to give to the compression command (default -9)
post_add_options.convergence
post_add_options.convergence.base what base values to be used to compute convergene error: tgt, cur, .. (default tgt)
post_add_options.convergence.norm which norm to use to compute error: 1 first norm, 2 second norm (default 1)
post_add_options.convergence.weight weight factors for the convergence of this interaction, should be a list of same length as inverse.post_add_options.convergence.what (default 1)
post_add_options.convergence.what list from what to calc the convergence: dist pot, .. (default dist)
post_add_options.copyback Contains all options of the postadd copyback sripts
post_add_options.copyback.filelist File to be copied to back to maindir
post_add_options.overwrite Contains all options of the overwrite postadd scripts
post_add_options.overwrite.do Cycle for overwrite postadd script (1 do, 0 do not) like do_potential. (default 1)
post_add_options.plot Contains all options of the plot postadd scripts
post_add_options.plot.fd file descriptor to use, make it unique if you want to plot multiple things (default 8)
post_add_options.plot.gnuplot_opts extra options to give to gnuplot_bin like -persist or -geometry
post_add_options.plot.kill kill all processes with that name before ploting (e.g. gnuplot_x11), this is more reliable than using named pipes
post_add_options.plot.script plot script to give to gnuplot

post_update Additional post-processing of dU before added to potential. This is a list of scripts separated by spaces which are called. See section on iterative framework for details.
post_update_options Contains all options of post update scripts
post_update_options.cibi Contains all options of the Kirkwood-Buff integral corrections scripts (default no)
post_update_options.cibi.do Update cycle for the Kirkwood-Buff integral correction (1 do, 0 do not). To do the correction every third step specify "0 0 1", similar to do_potential (default 1)
post_update_options.cibi.kbint_with_errors calculate errors on the Kirkwood-Buff integral: yes/no (default no)
post_update_options.extrapolate
post_update_options.extrapolate.points Number of point to calculate the average from for the extrapolation (default 5)
post_update_options.kbibi Contains all options of the Kirkwood-Buff ramp corrections scripts (default no)
post_update_options.kbibi.do Update cycle for the Kirkwood-Buff ramp correction (1 do, 0 do not). To do the correction every third step specify "0 0 1", similar to do_potential (default 1)
post_update_options.kbibi.factor scaling factor for the ramp correction
post_update_options.kbibi.kbint_with_errors calculate errors on the Kirkwood-Buff integral: yes/no (default no)
post_update_options.kbibi.r_ramp cutoff of the ramp
post_update_options.kbibi.start Where to start averaging the Kirkwood-Buff integral for the ramp
post_update_options.kbibi.stop Where to stop averaging the Kirkwood-Buff integral for the ramp
post_update_options.lj Contains all options of the Lennard-Jones potential update
post_update_options.lj.c12 The c12 value for the extra LJ potential
post_update_options.lj.c6 The c6 value for the extra LJ potential
post_update_options.pressure Contains all options of the pressure correction scripts
post_update_options.pressure.do Update cycle for the pressure correction (1 do, 0 do not). To do pressure correction every third step specify "0 0 1", similar to do_potential (default 1)
post_update_options.pressure.ptype Generic Pressure correction options
post_update_options.pressure.ptype.scale slope of the pressure correction
post_update_options.pressure.simple Contains all options of the simple pressure correction
post_update_options.pressure.simple.scale slope of the simple pressure correction
post_update_options.pressure.type Pressure correction type, can be simple or wjk (default simple)
post_update_options.pressure.wjk Contains all options of the wjk pressure correction
post_update_options.pressure.wjk.scale extra scaling factor of pressure wjk correction (default 1.0)
post_update_options.scale scale factor for the update (default 1.0)
post_update_options.smooth Contains all options of the post_update smooth script
post_update_options.smooth.iterations number of triangular smooth to be performed (default 1)

post_update_options.splinesmooth Contains all options of the post_update spline smooth script
post_update_options.splinesmooth.step grid spacing for spline fit when doing spline smoothing
sim_prog interaction specific options, which apply to all simulation programs
sim_prog.table Name of file for tabulated potential of this interaction. This fill will be created from the internal tabulated potential format in every step. Note, though, that the original simulation script needs to contain the name of that table as the tabulated interaction (see tutorial methanol ibi_espresso for details).
sim_prog.table_begin Start of the tabulated potential of this interaction. (Automatic for gromacs)
sim_prog.table_bins Binszie of the tabulated potential of this interaction. (gromacs uses a non interaction specific option)
sim_prog.table_end End of the tabulated potential of this interaction. (Automatic for gromacs)
sim_prog.table_left_extrapolation Extrapolation function to use on the left. Default: exponential(non-bonded), linear (bonded), Options: constant linear quadratic exponential sasha
sim_prog.table_right_extrapolation Extrapolation function to use on the right. Default: constant(non-bonded), linear (bonded), Options: constant linear quadratic exponential sasha
target target distribution (e.g. rdf) which is tried to match during iterations to match
max Upper bound of interval for potential table in which calculations are performed. Should be set based on reference distributions.
min Lower bound of interval for potential table in which calculations are performed. Should be set based on reference distributions.
name Name of the interaction. The name can be arbitrary but should be unique. For bonded interactions, this should match the name specified in the mapping file.
re Relative entropy options
cbspl options specific to cbspl function form
cbspl.nknots Number of knot values to be used for the cbspl functional form. Uniform grid size of the CBSPL depends on this parameter; for fixed potential range more the nknots smaller the grid spacing. Make sure grid spacing is sufficiently large and enough CG simulation steps are performed such that the bins at distance greater than the minimum distance are sampled sufficiently otherwise ill-defined system of equation would give NaNs in the output.
function Functional form for the potential. Available functional forms: lj126 (Lennard-Jones 12-6), ljg (Lennard-Jones 12-6 plus Gaussian), and cbspl (uniform cubic B-splines).
step Step size of interval for potential table in which calculations are performed. If step site is too small, lots of statistics is needed (long runs). If it's too big, features in the distribtuion/potentials might get lost.
type1 Bead type 1 of non-bonded interaction.
type2 Bead type 2 of non-bonded interaction.

10.5 Scripts

Scripts are used by **csg_call** and **csg_inverse**. The script table commonly used (compare `csg_call -list`):

Key1	Key2	Scriptname
tag	file	tag_file.sh
dummy	dummy	dummy.sh
functions	common	functions_common.sh
csg	master	inverse.sh
prepare	ibi	prepare_generic.sh
prepare	imc	prepare_imc.sh
prepare	generic	prepare_generic.sh
prepare	optimizer	prepare_optimizer.sh
prepare	re	prepare_re.sh
prepare_single	ibi	prepare_generic_single.sh
prepare_single	imc	prepare_generic_single.sh
prepare_single	optimizer	prepare_optimizer_single.sh
initstep	ibi	initialize_step_generic.sh
initstep	imc	initialize_step_generic.sh
initstep	optimizer	initialize_step_optimizer.sh
initstep	re	initialize_step_re.sh
prepare	ibm	prepare_ibm.sh
update	ibm	update_ibm.sh
add_pot	ibi	add_pot_generic.sh
add_pot	imc	add_pot_generic.sh
add_pot	optimizer	dummy.sh
add_pot	re	dummy.sh
pre_update	ibi	dummy.sh
pre_update	imc	dummy.sh
pre_update	optimizer	dummy.sh
pre_update	re	pre_update_re.sh
post_update	ibi	post_update_generic.sh
post_update	imc	post_update_generic.sh
post_update	optimizer	dummy.sh
post_update	re	post_update_generic.sh
post_update_single	ibi	post_update_generic_single.sh
post_update_single	imc	post_update_generic_single.sh
post_update_single	re	post_update_re_single.sh
postupd	scale	postupd_scale.sh
postupd	pressure	postupd_pressure.sh
postupd	lj	postupd_addlj.sh
postupd	splinesmooth	postupd_splinesmooth.sh
postupd	smooth	postupd_smooth.sh
postupd	shift	postadd_shift.sh
postupd	dummy	postadd_dummy.sh
postupd	tag	tag_file.sh
postupd	extrapolate	postupd_extrapolate.sh
postupd	kbibi	postupd_kbibi_correction.sh
postupd	cibi	postupd_cibi_correction.sh
post	add	post_add.sh
post	add_single	post_add_single.sh
postadd	tag	tag_file.sh
postadd	dummy	postadd_dummy.sh
postadd	copyback	postadd_copyback.sh
postadd	compress	postadd_compress.sh
postadd	convergence	postadd_convergence.sh
postadd	acc_convergence	postadd_acc_convergence.sh
postadd	shift	postadd_shift.sh

postadd	overwrite	postadd_overwrite.sh
postadd	plot	postadd_plot.sh
postadd	average	postadd_average.sh
convergence_check	default	convergence_check_default.sh
resample	target	resample_target.sh
dpot	crop	dpot_crop.pl
update	ibi	update_ibi.sh
update	ibi_single	update_ibi_single.sh
update	ibi_pot	update_ibi_pot.pl
update	imc	update_imc.sh
imcsolver	matlab	solve_matlab.sh
solve	matlab	linsolve.m
imcsolver	octave	solve_octave.sh
solve	octave	linsolve.octave
imcsolver	numpy	solve_numpy.sh
solve	numpy	linsolve.py
imc	purify	imc_purify.sh
optimizer	prepare_state	optimizer_prepare_state.sh
optimizer	parameters_to_potential	optimizer_parameters_to_potential.sh
optimizer	state_to_potentials	optimizer_state_to_potentials.sh
optimizer	state_to_mapping	optimizer_state_to_mapping.sh
update	optimizer	update_optimizer.sh
update	optimizer_single	update_optimizer_single.sh
optimizer_target	rdf	optimizer_target_rdf.sh
optimizer_target	density	optimizer_target_density.sh
optimizer_target	pressure	optimizer_target_pressure.sh
simplex	precede_state	simplex_downhill_processor.pl
cma	precede_state	cma_processor.py
update	re	update_re.sh
calc	target_rdf	calc_target_rdf_generic.sh
pressure_cor	simple	pressure_cor_simple.pl
pressure_cor	wjk	pressure_cor_wjk.pl
compute_lj	12_6	lj_126.pl
kbibi	ramp_correction	kbibi_ramp_correction.pl
calc	kbint	calc_kbint.sh
table	add	add_POT.pl
table	integrate	table_integrate.pl
table	extrapolate	table_extrapolate.pl
table	merge	merge_tables.pl
table	smooth	table_smooth.pl
table	linearop	table_linearop.pl
table	dummy	table_dummy.sh
table	get_value	table_get_value.pl
table	switch_border	table_switch_border.pl
table	compare	table_combine.pl
table	combine	table_combine.pl
table	average	table_average.sh
table	scale	table_scale.pl
table	change_flag	table_change_flag.sh
table	functional	table_functional.sh
potential	extrapolate	potential_extrapolate.sh
potential	shift	potential_shift.pl
convert_potential	tab	table_to_tab.pl
dist	adjust	dist_adjust.pl

dist	invert	dist_boltzmann_invert.pl
configuration	compare	configuration_compare.py
tables	jackknife	tables_jackknife.pl
initstep	gromacs	initialize_step_genericsim.sh
run	gromacs	run_gromacs.sh
clean	gromacs	clean_generic.sh
presimulation	gromacs	run_gromacs.sh
pressure	gromacs	calc_pressure_gromacs.sh
pressure	lammps	calc_pressure_lammps.sh
rdf	gromacs	calc_rdf_generic.sh
imc_stat	gromacs	imc_stat_generic.sh
density	gromacs	calc_density_generic.sh
convert_potential	gromacs	potential_to_gromacs.sh
convert_potentials	gromacs	potentials_to_generic.sh
convert_potential	xvg	table_to_xvg.pl
functions	gromacs	functions_gromacs.sh
initstep	espresso	initialize_step_genericsim.sh
run	espresso	run_genericsim.sh
clean	espresso	clean_generic.sh
rdf	espresso	calc_rdf_generic.sh
imc_stat	espresso	imc_stat_generic.sh
density	espresso	calc_density_generic.sh
convert_potential	espresso	potential_to_generic.sh
convert_potentials	espresso	potentials_to_generic.sh
functions	espresso	functions_genericsim.sh
convert_potential	lammps	potential_to_lammps.sh
convert_potentials	lammps	potentials_to_generic.sh
initstep	lammps	initialize_step_genericsim.sh
run	lammps	run_genericsim.sh
clean	lammps	clean_generic.sh
rdf	lammps	calc_rdf_generic.sh
imc_stat	lammps	imc_stat_generic.sh
density	lammps	calc_density_generic.sh
functions	lammps	functions_genericsim.sh
convert_potential	espressopp	potential_to_generic.sh
convert_potentials	espressopp	potentials_to_generic.sh
initstep	espressopp	initialize_step_genericsim.sh
run	espressopp	run_genericsim.sh
clean	espressopp	clean_generic.sh
rdf	espressopp	calc_rdf_generic.sh
imc_stat	espressopp	imc_stat_generic.sh
density	espressopp	calc_density_generic.sh
functions	espressopp	functions_genericsim.sh
initstep	dlpoly	initialize_step_genericsim.sh
run	dlpoly	run_genericsim.sh
clean	dlpoly	clean_generic.sh
rdf	dlpoly	calc_rdf_generic.sh
imc_stat	dlpoly	imc_stat_generic.sh
density	dlpoly	calc_density_generic.sh
functions	dlpoly	functions_dlpoly.sh
convert_potential	dlpoly	potential_to_dlpoly.sh
convert_potentials	dlpoly	potentials_to_dlpoly.sh
convert_potential	hoomd-blue	potential_to_generic.sh
convert_potentials	hoomd-blue	potentials_to_generic.sh

initstep	hoomd-blue	initialize_step_genericsim.sh
run	hoomd-blue	run_genericsim.sh
clean	hoomd-blue	clean_generic.sh
rdf	hoomd-blue	calc_rdf_generic.sh
imc_stat	hoomd-blue	imc_stat_generic.sh
density	hoomd-blue	calc_density_generic.sh
functions	hoomd-blue	functions_genericsim.sh

Script calls can be overwritten by adding a line with the 3rd column changed to `csg_table` in *inverse.scriptpath* directory.

10.5.1 add_POT.pl

This script adds up two potentials In addition, it does some magic tricks:

order of infiles MATTERS !!!!

if infile2 contains an undefined value, it uses the value from infile1

if value for infile1 and infile2 are both invalid, the result is also invalid

Usage: `csg_call [OPTIONS] table add infile1 infile2 outfile`

10.5.2 add_pot_generic.sh

This script adds up the tables
Usage: `csg_call [OPTIONS] add_pot ibi`
Used xml options:
cg.{non-}bonded.name

10.5.3 calc_density_generic.sh

This script calcs the density using csg_density
Usage: `csg_call [OPTIONS] density gromacs outputfile csg_density_options`
Used xml options:
cg.inverse.$sim_prog.density.block_length
cg.inverse.$sim_prog.density.with_errors
cg.inverse.$sim_prog.equi_time
cg.inverse.$sim_prog.first_frame
cg.inverse.$sim_prog.topol
cg.inverse.$sim_prog.traj
cg.inverse.program
cg.{non-}bonded.name

10.5.4 calc_kbint.sh

This script calculates the Kirkwood-Buff integral out of the rdf
Usage: `csg_call [OPTIONS] calc kbint [options] infile outfile`
Allowed options:
`--help` show this help
`--clean` remove all intermediate temp files

10.5.5 calc_pressure_gromacs.sh

This script calcs the pressure for gromacs and writes it to outfile
Usage: `csg_call [OPTIONS] pressure gromacs outfile`
Used external packages: gromacs
Used xml options:
- cg.inverse.gromacs.g_energy.bin
- cg.inverse.gromacs.g_energy.opts (optional)
- cg.inverse.gromacs.g_energy.pressure.allow_nan
- cg.inverse.gromacs.g_energy.topol (optional)
- cg.inverse.gromacs.topol

10.5.6 calc_pressure_lammps.sh

This script calcs the pressure for lammps and writes it to outfile
Usage: `csg_call [OPTIONS] pressure lammps outfile`
Used external packages: lammps
Used xml options:
- cg.inverse.lammps.pressure_file

10.5.7 calc_rdf_generic.sh

This script implemtents statistical analysis for the iterative Boltzmann inversion using generic csg tools (csg_stat)
Usage: `csg_call [OPTIONS] rdf gromacs`
Used xml options:
- cg.bonded.name (optional)
- cg.inverse.$sim_prog.equi_time
- cg.inverse.$sim_prog.first_frame
- cg.inverse.$sim_prog.rdf.block_length
- cg.inverse.$sim_prog.rdf.map (optional)
- cg.inverse.$sim_prog.rdf.topol (optional)
- cg.inverse.$sim_prog.rdf.traj (optional)
- cg.inverse.$sim_prog.rdf.with_errors
- cg.inverse.$sim_prog.topol
- cg.inverse.$sim_prog.traj
- cg.inverse.map (optional)
- cg.inverse.program
- cg.{non-}bonded.name

10.5.8 calc_target_rdf_generic.sh

This script calculated reference rdf using generic csg_stat
Usage: `csg_call [OPTIONS] calc target_rdf`
Used xml options:
- cg.inverse.gromacs.ref.equi_time
- cg.inverse.gromacs.ref.first_frame
- cg.inverse.gromacs.ref.mapping
- cg.inverse.gromacs.ref.rdf.opts (optional)
- cg.inverse.gromacs.ref.topol
- cg.inverse.gromacs.ref.traj

cg.inverse.program

10.5.9 clean_generic.sh

This script cleans up after a simulation step
Usage: `csg_call [OPTIONS] clean gromacs`
Used xml options:
cg.inverse.cleanlist (optional)

10.5.10 cma_processor.py

Usage: `csg_call [OPTIONS] cma precede_state [options] statefile-in statefile-out`
Options:
`-h, --help` show this help message and exit
`--eps=EPS` tolerance for initialization

10.5.11 configuration_compare.py

Usage: `csg_call [OPTIONS] configuration compare [options] conf1 conf2`
Options:
`-h, --help` show this help message and exit
`--eps=EPS` tolerance for mismatch

10.5.12 convergence_check_default.sh

Calculated the sum of all convergence files and create a file 'stop' if the sum is bigger than a given limit
Usage: `csg_call [OPTIONS] convergence_check default`
Used xml options:
cg.inverse.convergence_check.limit
cg.{non-}bonded.name

10.5.13 dist_adjust.pl

This script adjusts a distribution in such a way that value smaller 0 will be replaces with 0.
Usage: `csg_call [OPTIONS] dist adjust [OPTIONS] <in> <out>`
Allowed options:
`-h, --help` Show this help message
Examples:
`dist_adjust.pl CG-CG.dist.tmp CG-CG.dist.new`

10.5.14 dist_boltzmann_invert.pl

Boltzmann inverts a distribution ("$F(x)=-k_B T\ln g(x)$")
In addtion, it does some magic tricks:
do not crash when calc log(0)
choose the right normalization depending on the type of interaction
input dist should be unnormalized (like csg_stat calcs it)

Usage: `csg_call [OPTIONS] dist invert [OPTIONS] <in> <out>`
Allowed options:
`--kbT NUMBER` use NUMBER as "k_B*T" for the entropic part
`--type XXX` change the type of interaction Default: non-bonded
`--min XXX` minimum value to consider Default: 1e-10
`-h, --help` Show this help message
Possible types: non-bonded, bond, angle, dihedral
Examples:
`dist_boltzmann_invert.pl --kbT 2.49435 --min 0.001 tmp.dist tmp.pot`

10.5.15 dpot_crop.pl

crop the potential update at poorly sampled ends
Usage: `csg_call [OPTIONS] dpot crop [OPTIONS] <file> <a> <b>`
Allowed options:
`-h, --help` Show this help message
Examples:
`dpot_crop.pl tmp.dpot.cur tmp.dpot.new`

10.5.16 dummy.sh

dummy script (does nothing), useful to overwrite default by nothing
Usage: `csg_call [OPTIONS] dummy dummy`

10.5.17 functions_common.sh

This file defines some commonly used functions:
`msg` -- echos a msg on the screen and send it to the logfile if logging is enabled
`show_callstack` -- show the current callstack
`die` -- make the iterative frame work stopp
`cat_external` -- takes a two tags and shows content of the according script
`do_external` -- takes two tags, find the according script and excute it
`critical` -- executes arguments as command and calls die if not succesful
`csg_get_interaction_property` -- gets an interaction property from the xml file, should only be called from inside a for_all loop or with --all option
`csg_get_property` -- get an property from the xml file
`trim_all` -- make multiple lines into one and strip white space from beginning and the end, reads from stdin
`mark_done` -- mark a task (1st argument) as done in the restart file
`is_done` -- checks if something is already do in the restart file
`is_int` -- checks if all arguments are integers
`to_int` -- convert all given numbers to int using awk's int function
`is_part` -- checks if 1st argument is part of the set given by other arguments
`has_duplicate` -- check if one of the arguments is double
`remove_duplicate` -- remove duplicates list of arguments
`is_num` -- checks if all arguments are numbers
`get_stepname` -- get the dir name of a certain step number (1st argument)
`get_current_step_dir` -- print the directory of the current step
`get_last_step_dir` -- print the directory of the last step
`get_main_dir` -- print the main directory
`get_current_step_nr` -- print the main directory

`get_step_nr` -- print the number of a certain step directory (1st argument)
`cp_from_main_dir` -- copy something from the main directory
`cp_from_last_step` -- copy something from the last step
`get_time` -- gives back current time in sec from 1970
`get_number_tasks` -- get the number of possible tasks from the xml file or determine it automatically under some systems
`get_table_comment` -- get comment lines from a table and add common information, which include the git id and other information
`csg_inverse_clean` -- clean out the main directory
`check_path_variable` -- check if a variable contains only valid paths
`add_to_csgshare` -- added an directory to the csg internal search directories
`globalize_dir` -- convert a local directory to a global one
`globalize_file` -- convert a local file name to a global one
`source_function` -- source an extra function file
`csg_banner` -- print a big banner
`csg_calc` -- simple calculator, a + b, ...
`show_csg_tables` -- show all concatinated csg tables
`get_command_from_csg_tables` -- print the name of script belonging to certain tags (1st, 2nd argument)
`source_wrapper` -- print the full name of a script belonging to two tags (1st, 2nd argument)
`find_in_csgshare` -- find a script in csg script search path
`enable_logging` -- enables the logging to a certain file (1st argument) or the logfile taken from the xml file
`get_restart_file` -- print the name of the restart file to use
`check_for_obsolete_xml_options` -- check xml file for obsolete options
`command_not_found_handle` -- print and error message if a command or a function was not found

Used xml options:

cg.inverse.log_file (default: 2>/dev/null)
cg.inverse.map (optional)
cg.inverse.program
cg.inverse.restart_file
cg.inverse.simulation.tasks
cg.{non-}bonded.bondtype
cg.{non-}bonded.min
cg.{non-}bonded.name

10.5.18 functions_dlpoly.sh

Useful functions for the generic simulation program:

`simulation_finish` -- checks if simulation is finished
`checkpoint_exist` -- check if a checkpoint exists (REVIVE _and_ REVCON - both are needed!)
`get_simulation_setting` -- gets parameter a parameter from the settings file (1st argument) from simulation setting file (not implemented)

Used xml options:

cg.inverse.dlpoly.checkpoint
cg.inverse.dlpoly.topol
cg.inverse.dlpoly.traj
cg.inverse.program

10.5.19 functions_genericsim.sh

Useful functions for the generic simulation program:

`simulation_finish` -- checks if simulation is finished
`checkpoint_exist` -- check if a checkpoint exists (not implemented)
`get_simulation_setting` -- gets parameter a parameter from the settings file (1st argument) from simulation setting file (not implemented)

Used xml options:

cg.inverse.$sim_prog.traj
cg.inverse.program

10.5.20 functions_gromacs.sh

Useful functions for gromacs:

`get_simulation_setting` -- gets a parameter (1st argument) from gromacs mdp file (default 2nd parameter)
`check_temp` -- compares k_B T in xml with temp in mpd file
`simulation_finish` -- checks if simulation is finished
`checkpoint_exist` -- check if a checkpoint exists
`calc_begin_time` -- return the max of dt*frames and eqtime
`calc_end_time` -- return dt * nsteps
`gromacs_log` -- redirect stdin to a separate gromacs log file, 1st argument can be the name of the command to echo if redirection takes place

Used external packages: gromacs
Used xml options:

cg.inverse.gromacs.conf_out
cg.inverse.gromacs.equi_time
cg.inverse.gromacs.first_frame
cg.inverse.gromacs.gmxrc (optional)
cg.inverse.gromacs.log (optional)
cg.inverse.gromacs.mdp
cg.inverse.gromacs.mdrun.checkpoint
cg.inverse.gromacs.pre_simulation
cg.inverse.gromacs.temp_check
cg.inverse.gromacs.traj
cg.inverse.kBT
cg.inverse.log_file

10.5.21 imc_purify.sh

This scripts cleans up the dpot tables for each interaction when using IMC
Usage: `csg_call [OPTIONS] imc purify`
Used xml options:

cg.inverse.kBT
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.do_potential
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.22 imc_stat_generic.sh

This script implemtents statistical analysis for the Inverse Monte Carlo Method using generic csg tools (csg_stat)
Usage: `csg_call [OPTIONS] imc_stat gromacs`
Used xml options:

- cg.inverse.$sim_prog.equi_time
- cg.inverse.$sim_prog.first_frame
- cg.inverse.$sim_prog.imc.topol (optional)
- cg.inverse.$sim_prog.imc.traj (optional)
- cg.inverse.$sim_prog.topol
- cg.inverse.$sim_prog.traj
- cg.inverse.program
- cg.{non-}bonded.inverse.target
- cg.{non-}bonded.name

10.5.23 initialize_step_generic.sh

This script implements the initialization for every step in a generic way
Usage: `csg_call [OPTIONS] initstep ibi`
Used xml options:

- cg.inverse.program
- cg.{non-}bonded.name

10.5.24 initialize_step_genericsim.sh

This script initializes an iteration for the generic simulation program
Usage: `csg_call [OPTIONS] initstep gromacs`
Used xml options:

- cg.inverse.$sim_prog.conf (optional)
- cg.inverse.$sim_prog.conf_out (optional)
- cg.inverse.initial_configuration
- cg.inverse.program

10.5.25 initialize_step_optimizer.sh

This script implements the initialization for every step in a generic way
Usage: `csg_call [OPTIONS] initstep optimizer`
Used xml options:

- cg.inverse.optimizer.type
- cg.inverse.program
- cg.{non-}bonded.name

10.5.26 initialize_step_re.sh

This script implements the initialization for every step of relative entropy method by csg_reupdate program
Usage: `csg_call [OPTIONS] initstep re`
Used xml options:

- cg.inverse.program
- cg.{non-}bonded.name

10.5.27 inverse.sh

Start the script to run ibi, imc, etc. or clean out current dir
Usage: `csg_call [OPTIONS] csg master [OPTIONS] --options settings.xml [clean]`
Allowed options:

`-h, --help` show this help
`-N, --do-iterations N` only do N iterations (ignoring settings.xml)
`--wall-time SEK` Set wall clock time
`--options FILE` Specify the options xml file to use
`--debug` enable debug mode with a lot of information
`--nocolor` disable colors

Examples:

```
inverse.sh --options cg.xml
inverse.sh -6 --options cg.xml
```

Used xml options:

cg.inverse.convergence_check.type
cg.inverse.filelist (optional)
cg.inverse.iterations_max
cg.inverse.method
cg.inverse.program
cg.inverse.scriptpath (optional)
cg.inverse.simulation.background

10.5.28 kbibi_ramp_correction.pl

This script calculates Kirkwood-Buff correction as described in: P. Ganguly, D. Mukherji, C. Junghans, N. F. A. van der Vegt, Kirkwood-Buff coarse-grained force fields for aqueous solutions, J. Chem. Theo. Comp., 8, 1802 (2012), doi:10.1021/ct3000958
Usage: `csg_call [OPTIONS] kbibi ramp_correction [OPTIONS] kbint target_kbint outfile kBT min:step:max int_start:int_end ramp_factor`
Allowed options:

`-h, --help` Show this help message

10.5.29 linsolve.m

This script has no help

10.5.30 linsolve.octave

This script has no help

10.5.31 linsolve.py

Usage: `csg_call [OPTIONS] solve numpy [options] group output`
Options:

`-h, --help` show this help message and exit
`--reg=REG` regularization factor

10.5.32 lj_126.pl

This script has no help

10.5.33 merge_tables.pl

Merge two tables
Usage: `csg_call [OPTIONS] table merge [OPTIONS] <source> <dest> <out>`
Allowed options:

`-v, --version` Print version
`-h, --help` Show this help message
`--withflag` only change entries with specific flag in src
`--noflags` don't copy flags
`--novalues` don't copy values

Examples:

```
merge_tables.pl intable intable2 outtable
```

10.5.34 optimizer_parameters_to_potential.sh

This script generates a single potential (.pot.new) out a parameter value string (1st argument)
Usage: `csg_call [OPTIONS] optimizer parameters_to_potential parametervalues`
Used xml options:

cg.{non-}bonded.inverse.optimizer.function
cg.{non-}bonded.inverse.optimizer.functionfile (optional)
cg.{non-}bonded.inverse.optimizer.parameters
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.35 optimizer_prepare_state.sh

This script generates the initial state file and puts all in-file together
Usage: `csg_call [OPTIONS] optimizer prepare_state outputfile`
Used xml options:

cg.inverse.optimizer.cma.eps
cg.inverse.optimizer.type
cg.{non-}bonded.inverse.optimizer.parameters
cg.{non-}bonded.name

10.5.36 optimizer_state_to_mapping.sh

This script generates a mapping for the reference mapping from the parameters of the active in input state using the mapping template
Usage: `csg_call [OPTIONS] optimizer state_to_mapping input`
Used xml options:

cg.{non-}bonded.inverse.optimizer.mapping.change
cg.{non-}bonded.inverse.optimizer.mapping.output
cg.{non-}bonded.inverse.optimizer.mapping.template
cg.{non-}bonded.inverse.optimizer.parameters
cg.{non-}bonded.name

10.5.37 optimizer_state_to_potentials.sh

This script generates potential (.pot.new) for all interactions out the first pending line in the input state file and flags this line active in output state
Usage: `csg_call [OPTIONS] optimizer state_to_potentials input output`

10.5.38 optimizer_target_density.sh

Calculated the difference between rdf
Usage: `csg_call [OPTIONS] optimizer_target density`
Used xml options:

cg.inverse.program
cg.{non-}bonded.inverse.optimizer.density.axis
cg.{non-}bonded.inverse.optimizer.density.max
cg.{non-}bonded.inverse.optimizer.density.min
cg.{non-}bonded.inverse.optimizer.density.molname
cg.{non-}bonded.inverse.optimizer.density.scale
cg.{non-}bonded.inverse.optimizer.density.step
cg.{non-}bonded.inverse.optimizer.density.target
cg.{non-}bonded.name

10.5.39 optimizer_target_pressure.sh

Calculates the difference current and target pressure
Usage: `csg_call [OPTIONS] optimizer_target pressure`
Used xml options:

cg.inverse.program
cg.{non-}bonded.inverse.optimizer.pressure.undef (optional)
cg.{non-}bonded.inverse.p_target
cg.{non-}bonded.name

10.5.40 optimizer_target_rdf.sh

Calculated the difference between rdf
Usage: `csg_call [OPTIONS] optimizer_target rdf`
Used xml options:

cg.inverse.optimizer.type
cg.inverse.program
cg.{non-}bonded.inverse.optimizer.mapping.change
cg.{non-}bonded.inverse.optimizer.rdf.target
cg.{non-}bonded.inverse.optimizer.rdf.weight (optional)
cg.{non-}bonded.inverse.optimizer.rdf.weightfile (optional)
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.41 post_add.sh

This script makes all the post update
Usage: `csg_call [OPTIONS] post add`

10.5.42 post_add_single.sh

This script makes all the post update with backup for single pairs
Usage: `csg_call [OPTIONS] post add_single`
Used xml options:

cg.{non-}bonded.inverse.post_add (optional)
cg.{non-}bonded.name

10.5.43 post_update_generic.sh

This script makes all the post update
Usage: `csg_call [OPTIONS] post_update ibi`
Used xml options:

cg.inverse.method

10.5.44 post_update_generic_single.sh

This script makes all the post update with backup for single pairs incl. backups
Usage: `csg_call [OPTIONS] post_update_single ibi`
Used xml options:

cg.{non-}bonded.inverse.post_update (optional)
cg.{non-}bonded.name

10.5.45 post_update_re_single.sh

This script makes all the post update with backup for single pairs incl. backups
Usage: `csg_call [OPTIONS] post_update_single re`
Used xml options:

cg.{non-}bonded.inverse.post_update (optional)
cg.{non-}bonded.name

10.5.46 postadd_acc_convergence.sh

postadd accumulate convergence script: accumulate ${name}.conv of all steps
Usage: `csg_call [OPTIONS] postadd acc_convergence infile outfile`
Used xml options:

cg.{non-}bonded.name

10.5.47 postadd_average.sh

postadd average script, calcs averages of (${name}.DIST.cur) for the past few steps and saves it to ${name}.DIST.avg DIST can be specified by average.what option
usage: postadd_average.sh
Used xml options:

- cg.inverse.average.steps (default: 2)
- cg.inverse.method
- cg.{non-}bonded.inverse.post_add_options.average.what
- cg.{non-}bonded.name

10.5.48 postadd_compress.sh

postadd compress script, compresses files
Usage: `csg_call [OPTIONS] postadd compress`
Used xml options:

- cg.{non-}bonded.inverse.post_add_options.compress.filelist
- cg.{non-}bonded.inverse.post_add_options.compress.program
- cg.{non-}bonded.inverse.post_add_options.compress.program_opts (optional)

10.5.49 postadd_convergence.sh

postadd convergence script, calcs norm of error (${name}.DIST.BASE-${name}.DIST.new) and saves it to ${name}.conv. DIST stands for 'dist', but can be changed by onvergence.what option
usage: postadd_convergence.sh
Used xml options:

- cg.inverse.method
- cg.{non-}bonded.inverse.post_add_options.convergence.base
- cg.{non-}bonded.inverse.post_add_options.convergence.norm
- cg.{non-}bonded.inverse.post_add_options.convergence.weight
- cg.{non-}bonded.inverse.post_add_options.convergence.what
- cg.{non-}bonded.inverse.target
- cg.{non-}bonded.max
- cg.{non-}bonded.min
- cg.{non-}bonded.name
- cg.{non-}bonded.step

10.5.50 postadd_copyback.sh

postadd copyback script, copies files back to the maindir
Usage: `csg_call [OPTIONS] postadd copyback`
Used xml options:

- cg.{non-}bonded.inverse.post_add_options.copyback.filelist

10.5.51 postadd_dummy.sh

postadd dummy script (cp infile to outfile), useful to overwrite default by nothing
Usage: `csg_call [OPTIONS] postupd dummy infile outfile`

10.5.52 postadd_overwrite.sh

postadd overwrite script, overwrites potential of all other interactions with this one
Usage: `csg_call [OPTIONS] postadd overwrite infile outfile`
Used xml options:
- cg.{non-}bonded.inverse.post_add
- cg.{non-}bonded.inverse.post_add_options.overwrite.do
- cg.{non-}bonded.name

10.5.53 postadd_plot.sh

postadd plot script, send a certain plot script to gnuplot
Usage: `csg_call [OPTIONS] postadd plot`
Used external packages: gnuplot
Used xml options:
- cg.inverse.gnuplot.bin
- cg.{non-}bonded.inverse.post_add_options.plot.gnuplot_opts (optional)
- cg.{non-}bonded.inverse.post_add_options.plot.kill (optional)
- cg.{non-}bonded.inverse.post_add_options.plot.script

10.5.54 postadd_shift.sh

postadd shift script, shift pot and dpot
Usage: `csg_call [OPTIONS] postupd shift infile outfile`
Used xml options:
- cg.{non-}bonded.bondtype

10.5.55 postupd_addlj.sh

This script adds LJ 12-6 component to the CG potential
Usage: `csg_call [OPTIONS] postupd lj infile outfile`
Used xml options:
- cg.{non-}bonded.inverse.post_update_options.lj.c12
- cg.{non-}bonded.inverse.post_update_options.lj.c6
- cg.{non-}bonded.max
- cg.{non-}bonded.min
- cg.{non-}bonded.name
- cg.{non-}bonded.step

10.5.56 postupd_cibi_correction.sh

This script implemtents the post update routine for the integral Kirkwood-Buff corrections described in: T. E. de Oliveira, P. A. Netz, K. Kremer, C. Junghans, and D. Mukherji, C-IBI: Targeting cumulative coordination within an iterative protocol to derive coarse-grained models of (multi-component) complex fluids, J. Chem. Phys. (in press).
Usage: `csg_call [OPTIONS] postupd cibi`
Used xml options:
- cg.inverse.$sim_prog.rdf.with_errors
- cg.inverse.kBT
- cg.inverse.program
- cg.{non-}bonded.bondtype

cg.{non-}bonded.inverse.post_update_options.cibi.do
cg.{non-}bonded.inverse.post_update_options.cibi.kbint_with_errors
cg.{non-}bonded.inverse.target
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.57 postupd_extrapolate.sh

This script implements extrapolation undefined region of the potential update (.dpot)
Usage: `csg_call [OPTIONS] postupd extrapolate infile outfile`
Used xml options:

cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.post_update_options.extrapolate.points
cg.{non-}bonded.name

10.5.58 postupd_kbibi_correction.sh

This script implemtents the post update routine for the ramp Kirkwood-Buff corrections as described in: P. Ganguly, D. Mukherji, C. Junghans, N. F. A. van der Vegt, Kirkwood-Buff coarse-grained force fields for aqueous solutions, J. Chem. Theo. Comp., 8, 1802 (2012), doi:10.1021/ct3000958
Usage: `csg_call [OPTIONS] postupd kbibi`
Used xml options:

cg.inverse.$sim_prog.rdf.with_errors
cg.inverse.kBT
cg.inverse.program
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.post_update_options.kbibi.do
cg.{non-}bonded.inverse.post_update_options.kbibi.factor
cg.{non-}bonded.inverse.post_update_options.kbibi.kbint_with_errors
cg.{non-}bonded.inverse.post_update_options.kbibi.r_ramp (optional)
cg.{non-}bonded.inverse.post_update_options.kbibi.start
cg.{non-}bonded.inverse.post_update_options.kbibi.stop
cg.{non-}bonded.inverse.target
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.59 postupd_pressure.sh

This script implements the pressure update
Usage: `csg_call [OPTIONS] postupd pressure infile outfile`
Used xml options:

cg.inverse.kBT
cg.inverse.program
cg.{non-}bonded.inverse.p_target
cg.{non-}bonded.inverse.particle_dens
cg.{non-}bonded.inverse.post_update_options.pressure.$ptype.scale
cg.{non-}bonded.inverse.post_update_options.pressure.do

cg.{non-}bonded.inverse.post_update_options.pressure.type
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.60 postupd_scale.sh

This script implements scaling of the potential update (.dpot)
Usage: `csg_call [OPTIONS] postupd scale infile outfile`
Used xml options:

cg.{non-}bonded.inverse.post_update_options.scale
cg.{non-}bonded.name

10.5.61 postupd_smooth.sh

This script implements smoothing of the potential update (.dpot)
Usage: `csg_call [OPTIONS] postupd smooth infile outfile`
Used xml options:

cg.{non-}bonded.inverse.post_update_options.smooth.iterations
cg.{non-}bonded.name

10.5.62 postupd_splinesmooth.sh

This script implements smoothing of the potential update (.dpot)
Usage: `csg_call [OPTIONS] postupd splinesmooth infile outfile`
Used xml options:

cg.{non-}bonded.inverse.post_update_options.splinesmooth.step
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.63 potential_extrapolate.sh

This script extrapolates a potential in the correct way depending on its type.
Usage: `csg_call [OPTIONS] potential extrapolate [options] input output`
Allowed options:

`--help` show this help
`--clean` remove all intermediate temp files
`--type TYPE` type of the potential possible: non-bonded bond angle dihedral
`--lfct FCT` type of the left extrapolation function possible: default: exponential(non-bonded), linear (bonded)
`--rfct FCT` type of the right extrapolation function possible: constant linear quadratic exponential sasha default: constant(non-bonded), linear (bonded)
`--avg-point INT` number of average points default: 3

10.5.64 potential_shift.pl

This script shifts the whole potential by minimum (bonded potentials) or last value (non-bonded potentials).
Usage: `csg_call [OPTIONS] potential shift [OPTIONS] <in> <out>`
Allowed options:

`-h, --help` show this help message
`--type XXX` change the type of potential Default: non-bonded
Possible types: non-bonded, bond, angle, dihedral, bonded

Examples:

```
potential_shift.pl --type bond table.in table.out
```

10.5.65 potential_to_dlpoly.sh

This script is a high class wrapper to convert a potential to the dlpoly format
Usage: `csg_call [OPTIONS] convert_potential dlpoly`
Used xml options:

cg.inverse.dlpoly.angles.table_grid
cg.inverse.dlpoly.bonds.table_end
cg.inverse.dlpoly.bonds.table_grid
cg.inverse.dlpoly.dihedrals.table_grid
cg.inverse.dlpoly.table_end
cg.inverse.dlpoly.table_grid
cg.{non-}bonded.bondtype
cg.{non-}bonded.dlpoly.header
cg.{non-}bonded.dlpoly.header (optional)
cg.{non-}bonded.step
cg.{non-}bonded.type1
cg.{non-}bonded.type2

10.5.66 potential_to_generic.sh

This script is a high class wrapper to convert a potential to the generic 3 column tab format used by espresso and espressopp
Usage: `csg_call [OPTIONS] convert_potential espresso`
Used xml options:

cg.inverse.program
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.$sim_prog.table_begin (optional)
cg.{non-}bonded.inverse.$sim_prog.table_bins (optional)
cg.{non-}bonded.inverse.$sim_prog.table_end (optional)
cg.{non-}bonded.inverse.$sim_prog.table_left_extrapolation (optional)
cg.{non-}bonded.inverse.$sim_prog.table_right_extrapolation (optional)
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.step

10.5.67 potential_to_gromacs.sh

This script is a wrapper to convert a potential to gromacs
Usage: `csg_call [OPTIONS] convert_potential gromacs [options] input output`

Allowed options:
--help show this help
--clean remove all intermediate temp files
--no-r2d do not converts rad to degree (scale x axis with 180/3.1415) for angle and dihedral
Note: VOTCA calcs in rad, but gromacs in degree
--no-shift do not shift the potential
--step XXX use XXX as step for the interaction
Used xml options:
cg.inverse.gromacs.mdp
cg.inverse.gromacs.pot_max (optional)
cg.inverse.gromacs.table_bins
cg.inverse.gromacs.table_end
cg.inverse.gromacs.table_end (optional)
cg.{non-}bonded.bondtype
cg.{non-}bonded.max
cg.{non-}bonded.step

10.5.68 potential_to_lammps.sh

This script is a high class wrapper to convert a potential to the lammps format
Usage: `csg_call [OPTIONS] convert_potential lammps [options] input output`
Allowed options:
--help show this help
--clean remove all intermediate temp files
--no-r2d do not converts rad to degree (scale x axis with 180/3.1415) for angle interactions
Note: VOTCA calcs in rad, but lammps uses degrees for angle
--no-shift do not shift the potential
Used xml options:
cg.inverse.program
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.$sim_prog.table_begin (optional)
cg.{non-}bonded.inverse.$sim_prog.table_bins (optional)
cg.{non-}bonded.inverse.$sim_prog.table_end (optional)
cg.{non-}bonded.inverse.$sim_prog.table_left_extrapolation (optional)
cg.{non-}bonded.inverse.$sim_prog.table_right_extrapolation (optional)
cg.{non-}bonded.inverse.lammps.scale
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.step

10.5.69 potentials_to_dlpoly.sh

This script converts all potentials to the format needed by dlpoly
Usage: `csg_call [OPTIONS] convert_potentials dlpoly`
Used xml options:
cg.{non-}bonded.name

10.5.70 potentials_to_generic.sh

This script converts all potentials to the format needed by the simulation program
Usage: `csg_call [OPTIONS] convert_potentials gromacs`

Used xml options:
cg.inverse.program
cg.{non-}bonded.inverse.$sim_prog.table
cg.{non-}bonded.name

10.5.71 pre_update_re.sh

This script implements the pre update tasks for the Relative Entropy Method
Usage: `csg_call [OPTIONS] pre_update re`
Used xml options:
cg.inverse.program

10.5.72 prepare_generic.sh

This script prepares potentials in a generic way
Usage: `csg_call [OPTIONS] prepare ibi`
Used xml options:
cg.inverse.method
cg.inverse.program

10.5.73 prepare_generic_single.sh

This script implements the prepares the potential in step 0, using pot.in or by resampling the target distribution
Usage: `csg_call [OPTIONS] prepare_single ibi`
Used xml options:
cg.inverse.dist_min
cg.inverse.kBT
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.target
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.74 prepare_ibm.sh

Informs users that ibm was renamed to ibi.
Usage: `csg_call [OPTIONS] prepare ibm`

10.5.75 prepare_imc.sh

This script initializes potentials for imc
Usage: `csg_call [OPTIONS] prepare imc`
Used xml options:
cg.bonded.name (optional)
cg.{non-}bonded.name

10.5.76 prepare_optimizer.sh

This script initizalizes potentials for optimizer methods
Usage: `csg_call [OPTIONS] prepare optimizer`
Used xml options:
cg.inverse.optimizer.type
cg.inverse.program
cg.{non-}bonded.inverse.optimizer.parameters

10.5.77 prepare_optimizer_single.sh

This script
reads sinple interaction optimizer infile
checks if the number of values are enough
Usage: `csg_call [OPTIONS] prepare_single optimizer N`
where N is the total number of parameters
Used xml options:
cg.inverse.optimizer.type
cg.{non-}bonded.inverse.optimizer.parameters
cg.{non-}bonded.name

10.5.78 prepare_re.sh

This script implements the preparation of the relative entropy method iteration
Usage: `csg_call [OPTIONS] prepare re`
Used xml options:
cg.inverse.program
cg.{non-}bonded.inverse.target
cg.{non-}bonded.name

10.5.79 pressure_cor_simple.pl

This script calls the pressure corrections "$dU=A*(1-r/r_c)$", where "$A=-0.1k_B T * \max(1,|p_cur-p_target|*scale) * \sign(p_cur-p_target)$"
Usage: `csg_call [OPTIONS] pressure_cor simple p_cur outfile kBT min:step:max scale p_target`

10.5.80 pressure_cor_wjk.pl

This script calls the pressure corrections like in Wan, Junghans & Kremer, Euro. Phys. J. E 28, 221 (2009) Basically dU=A*(1-r/r_c) with A= -max(0.1k_B T, Int) * sign(p_cur-p_target) and Int is the integral from Eq. 7 in the paper.
Usage: `csg_call [OPTIONS] pressure_cor wjk p_cur outfile kBT min:step:max scale p_target particle_dens rdf_file`

10.5.81 resample_target.sh

This script resamples distribution to grid spacing of the setting xml file and extrapolates if needed
Usage: `csg_call [OPTIONS] resample target input output`
Used xml options:

cg.{non-}bonded.bondtype
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.82 run_genericsim.sh

This script runs a generic simulation program
Usage: `csg_call [OPTIONS] run espresso`
Used xml options:
cg.inverse.$sim_prog.command
cg.inverse.$sim_prog.opts (optional)
cg.inverse.$sim_prog.script (optional)
cg.inverse.method
cg.inverse.program

10.5.83 run_gromacs.sh

This script runs a gromacs simulation or pre-simulation
Usage: `csg_call [OPTIONS] run gromacs [--pre]`
Used external packages: gromacs
Used xml options:
cg.inverse.gromacs.conf
cg.inverse.gromacs.conf_out
cg.inverse.gromacs.grompp.bin
cg.inverse.gromacs.grompp.opts (optional)
cg.inverse.gromacs.index
cg.inverse.gromacs.mdp
cg.inverse.gromacs.mdrun.checkpoint
cg.inverse.gromacs.mdrun.command
cg.inverse.gromacs.mdrun.multidir (optional)
cg.inverse.gromacs.mdrun.opts (optional)
cg.inverse.gromacs.pre_simulation
cg.inverse.gromacs.topol
cg.inverse.gromacs.topol_in
cg.inverse.gromacs.traj
cg.inverse.gromacs.trjcat.bin

10.5.84 simplex_downhill_processor.pl

Changes a simplex state according to the current state using the Nelder–Mead method or downhill simplex algorithm.
Usage: `csg_call [OPTIONS] simplex precede_state current_state new_state`

10.5.85 solve_matlab.sh

This script solves a linear equation system from imc using matlab
Usage: `csg_call [OPTIONS] imcsolver matlab <group> <outfile>`
Used external packages: matlab

Used xml options:
cg.inverse.imc.matlab.bin

10.5.86 solve_numpy.sh

This script solves a linear equation system from imc using numpy
Usage: `csg_call [OPTIONS] imcsolver numpy <group> <outfile> <reg>`
Uses external packages: numpy

10.5.87 solve_octave.sh

This script solves a linear equation system from imc using octave
Usage: `csg_call [OPTIONS] imcsolver octave <group> <outfile>`
Used external packages: octave
Used xml options:
cg.inverse.imc.octave.bin

10.5.88 table_average.sh

This script creates averages tables and also calculates the error.
Usage: `csg_call [OPTIONS] table average [options] table1 table2 table3 ....`
Allowed options:
`-h, --help` show this help
`-o, --output NANE` output file name
`--cols NUM` Number of columns per file Default: 3
`--col-y NUM` y-data column Default: 2
`--col-x NUM` x-data column Default: 1
"--clean " Clean intermediate files
Examples:
`table_average.sh --output CG-CG.dist.new CG-CG*.dist.new`

10.5.89 table_change_flag.sh

This script changes the flags (col 3) of a table
Usage: `csg_call [OPTIONS] table change_flag input outfile`

10.5.90 table_combine.pl

This script combines two tables with a certain operation
Usage: `table_combine.pl [OPTIONS] <in> <in2> <out>`
Allowed options:
`--error` ERR Relative error Default: 1e-05
`--op OP` Operation to perform Possible: =,+,-,*,/,d,d2,x d = |y1-y2|, d2 = (y1-y2)^2, x=* (to avoid shell trouble)
`--sum` Output the sum instead of a new table
`--die` Die if op '=' fails
`--no-flags` Do not check for the flags
`--scale XXX` Scale output/sum with this number Default 1
`--withflag` FL only operate on entries with specific flag in src

`-h, --help` Show this help message

10.5.91 table_dummy.sh

This script creates a zero table with grid min:step:max using linear interpolation
Usage: `csg_call [OPTIONS] table dummy [options] min:step:max outfile`
Allowed options:

`--y1` X.X using X.X instead of 0 for the 1st y-value this creates a linear instead of a constant table

`--y2` X.X using X.X instead of 0 for the 2nd y-value this creates a linear instead of a constant table

`--help` show this help

`--clean` remove all intermediate temp files

10.5.92 table_extrapolate.pl

This script extrapolates a table
Usage: `csg_call [OPTIONS] table extrapolate [OPTIONS] <in> <out>`
Allowed options:

`--avgpoints A` average over the given number of points to extrapolate: default is 3

`--function` constant, linear, quadratic or exponential, sasha: default is quadratic

`--no-flagupdate` do not update the flag of the extrapolated values

`--region` left, right, or leftright: default is leftright

`--curvature C` curvature of the quadratic function: default is 10000, makes sense only for quadratic extrapolation, ignored for other cases

`-h, --help` Show this help message

Extrapolation methods: always "$m = dy/dx= (y[i+A]-y[i])/(x[i+A]-x[i])$"

constant: "$y = y0$"
linear: "$y = ax + b\;\;b = - m*x_0 + y_0;;a = m$"
sasha: "$y = a*(x-b)^2\;\;b = (x0 - 2y_0/m)\;\; a = m^2/(4*y_0)$"
exponential: "$y = a*\exp(b*x)\;\;a = y0*\exp(-m*x0/y0)\;\;b = m/y_0$"
quadratic: "$y = C*(x+a)^2 + b\;\;a = m/(2*C) - x0\;\; b = y_0 - m^2/(4*C)$"

10.5.93 table_functional.sh

This script creates a table with grid min:step:max for the a functional form
Usage: `csg_call [OPTIONS] table functional [options] output`
Allowed options:

`-h, --help` show this help

`--grid` XX:XX:XX Output grid of the table

`--var X=Y` Set a variable used in the function

`--fct FCT` functional form of the table

`--headerfile XXX` Extra headerfile for the plot script (useful for complicated functions)

`--gnuplot CMD` Gnuplot command to use Default: gnuplot

"--clean " Clean intermediate files

Used external packages: gnuplot
Examples:

```
table_functional.sh --grid 0:0.1:1 --fct x**2 CG-CG.tab.new
```

10.5.94 table_get_value.pl

This script print the y value of x, which is closest to X.
Usage: `csg_call [OPTIONS] table get_value [OPTIONS] X infile`
Allowed options:

`-h, --help` Show this help message

10.5.95 table_integrate.pl

This script calculates the integral of a table. Please note the force is the NEGATIVE integral of the potential (use 'table linearop' and multiply the table with -1)
Usage: `csg_call [OPTIONS] table integrate [OPTIONS] <in> <out>`
Allowed options:

`--with-errors` calculate error
`--with-S` Add entropic contribution to force "$2k_B T/r$"
`--kbT NUMBER` use NUMBER as "k_B*T" for the entropic part
`--from` Integrate from left or right (to define the zero point) Default: right
`--sphere` Add spherical volume term ("r^2")
`-h, --help` Show this help message

Examples:

```
table_integrate.pl --with-S --kbT 2.49435 tmp.force tmp.dpot
```

10.5.96 table_linearop.pl

This script performs a linear operation on the y values: "$y_{new} = a*y_{old} + b$"
Usage: `csg_call [OPTIONS] table linearop [OPTIONS] <in> <out> <a> <b>`
Allowed options:

`-h, --help` Show this help message
`--withflag` FL only change entries with specific flag in src
`--with-errors` also read and calculate errors
`--on-x` work on x values instead of y values

Examples:

```
table_linearop.pl tmp.dpot.cur tmp.dpot.new 1.0 0.0
```

10.5.97 table_scale.pl

This script applies a prefactor to infile. The prefactor is is interpolated lines between the prefactor1 and prefactor2.
Usage: `csg_call [OPTIONS] table scale [OPTIONS] infile outfile prefactor1 prefactor2`
Allowed options:

`-h, --help` Show this help message

10.5.98 table_smooth.pl

This script smoothes a table
Usage: `csg_call [OPTIONS] table smooth infile outfile`

10.5.99 table_switch_border.pl

This script applies a switching function to the end of the table to switch it smoothly to zero by y = y*cos(pi*(x-x_switch)/(2*(x_end-x_switch)))
Usage: `csg_call [OPTIONS] table switch_border infile outfile <x_switch>`

10.5.100 table_to_tab.pl

This script converts csg potential files to the tab format (as read by espresso or lammps or dlpoly).
In addition, it does some magic tricks:

shift the potential, so that it is zero at the cutoff

Usage: `csg_call [OPTIONS] convert_potential tab [OPTIONS] <in> <derivatives_in> <out>`
Allowed options:

`-h, --help` show this help message
`--type XXX` change the type of xvg table Default: non-bonded
`--header XXX` Write a special simulation programm header

Examples:

```
table_to_tab.pl --type non-bonded table.in table_b0.xvg
```

10.5.101 table_to_xvg.pl

This script converts csg potential files to the xvg format.
Usage: `csg_call [OPTIONS] convert_potential xvg [OPTIONS] <in> <out>`
Allowed options:

`-h, --help` show this help message
`--type XXX` change the type of xvg table Default: non-bonded
`--max MAX` Replace all pot value bigger MAX by MAX

Possible types: non-bonded (=C12), bond, C12, C6, CB, angle, dihedral
Examples:

```
table_to_xvg.pl --type bond table.in table_b0.xvg
```

10.5.102 tables_jackknife.pl

This script has no help

10.5.103 tag_file.sh

Add table_comment to the head of a file
Usage: `csg_call [OPTIONS] tag file input output`

10.5.104 update_ibi.sh

This script implements the function update for the Inverse Boltzmann Method
Usage: `csg_call [OPTIONS] update ibi`
Used xml options:

cg.inverse.program

10.5.105 update_ibi_pot.pl

This script calcs dU out of two rdfs with the rules of inverse boltzmann
In addition, it does some magic tricks:

do not update if one of the two rdf is undefined

Usage: `csg_call [OPTIONS] update ibi_pot target_rdf new_rdf cur_pot outfile kBT`

10.5.106 update_ibi_single.sh

This script implemtents the function update for a single pair for the Inverse Boltzmann Method
Usage: `csg_call [OPTIONS] update ibi_single`
Used xml options:

cg.inverse.kBT
cg.{non-}bonded.bondtype
cg.{non-}bonded.inverse.do_potential
cg.{non-}bonded.inverse.target
cg.{non-}bonded.max
cg.{non-}bonded.min
cg.{non-}bonded.name
cg.{non-}bonded.step

10.5.107 update_ibm.sh

Informs users that ibm was renamed to ibi.
Usage: `csg_call [OPTIONS] update ibm`

10.5.108 update_imc.sh

This script implements the function update for the Inverse Monte Carlo Method
Usage: `csg_call [OPTIONS] update imc`
Used xml options:

cg.inverse.imc.solver
cg.inverse.program
cg.{non-}bonded.inverse.imc.group
cg.{non-}bonded.inverse.imc.reg

10.5.109 update_optimizer.sh

This script:

implements the update function for each non-bonded interaction
performs optimizer algorithm if no pending parameter sets present
continues with next parameter set in table if otherwise

Usage: `csg_call [OPTIONS] update optimizer`
Used xml options:

cg.inverse.optimizer.type
cg.{non-}bonded.name

10.5.110 update_optimizer_single.sh

This script:
calculates the new property
compares it to the target property and calculates the target function accordingly
Usage: `csg_call [OPTIONS] update optimizer_single`
Used xml options:
cg.inverse.program
cg.{non-}bonded.inverse.optimizer.target_weights
cg.{non-}bonded.inverse.optimizer.targets
cg.{non-}bonded.inverse.post_update (optional)
cg.{non-}bonded.name

10.5.111 update_re.sh

This script implements update step of relative entropy method by csg_reupdate program
Usage: `csg_call [OPTIONS] update re`
Used xml options:
cg.inverse.$sim_prog.equi_time
cg.inverse.$sim_prog.first_frame
cg.inverse.$sim_prog.re.topol (optional)
cg.inverse.$sim_prog.re.traj (optional)
cg.inverse.$sim_prog.topol
cg.inverse.$sim_prog.traj
cg.inverse.program
cg.inverse.re.csg_reupdate.opts (optional)
cg.{non-}bonded.inverse.target
cg.{non-}bonded.name

Bibliography

[1] SY Mashayak, Mara N Jochum, Konstantin Koschke, NR Aluru, Victor Rühle, and Christoph Junghans. Relative entropy and optimization-driven coarse-graining methods in votca. *Plos One*, 10:e131754, 2015.

[2] Victor Rühle and Christoph Junghans. Hybrid approaches to coarse-graining using the votca package: Liquid hexane. *Macromolecular Theory and Simulations*, 20(7):472–477, 2011.

[3] V. Rühle, C. Junghans, A. Lukyanov, K. Kremer, and D. Andrienko. Versatile object-oriented toolkit for coarse-graining applications. *J. Chem. Theor. Comp.*, 5:3211, 2009.

[4] Berk Hess, Carsten Kutzner, David van der Spoel, and Erik Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theo. Comp.*, 4(3):435–447, 2008.

[5] W G Noid, J Chu, G S Ayton, V Krishna, S Izvekov, G Voth, A Das, and H C Andersen. The multiscale coarse graining method. 1. a rigorous bridge between atomistic and coarse-grained models. *J. Chem. Phys.*, 128:244114, JUN 2008.

[6] Dominik Fritz, Vagelis A. Harmandaris, Kurt Kremer, and Nico F. A. van der Vegt. Coarse-grained polymer melts based on isolated atomistic chains: Simulation of polystyrene of different tacticities. *Macromolecules*, 42(19):7579–7588, 2009.

[7] W Tschöp, K Kremer, J Batoulis, T Burger, and O Hahn. Simulation of polymer melts. i. coarse-graining procedure for polycarbonates. *Acta Polymerica*, 49:61–74, 1998.

[8] D Reith, M Pütz, and F Müller-Plathe. Deriving effective mesoscale potentials from atomistic simulations. *J. Comp. Chem.*, 24(13):1624–1636, 2003.

[9] Ap Lyubartsev and A Laaksonen. Calculation of effective interaction potentials from radial-distribution functions - a reverse monte-carlo approach. *Phys. Rev. E*, 52(4):3730–3737, 1995.

[10] T Murtola, E Falck, M Karttunen, and I Vattulainen. Coarse-grained model for phospholipid/cholesterol bilayer employing inverse monte carlo with thermodynamic constraints. *J. Chem. Phys.*, 126(7):075101, 2007.

[11] David Rosenberger, Martin Hanke, and Nico F. A. van der Vegt. Comparison of iterative inverse coarse-graining methods. *European Physical Journal Special Topics*, 225:1323–1345, 2016.

[12] F. Ercolessi and J. B. Adams. Interatomic potentials from 1st-principles calculations - the force-matching method. *Europhys. Lett.*, 26(8):583–588, 1994.

[13] S Izvekov and GA Voth. Multiscale coarse graining of liquid-state systems. *J. Chem. Phys.*, 123(13):134105, OCT 1 2005.

[14] WG Noid, JW Chu, GS Ayton, and GA Voth. Multiscale coarse-graining and structural correlations: Connections to liquid-state theory. *J. Phys. Chem. B*, 111(16):4116–4127, APR 2007.

[15] Di Wu and David A. Kofke. Phase-space overlap measures. I. Fail-safe bias detection in free energies calculated by molecular simulation. *The Journal of chemical physics*, 123:054103, 2005.

[16] M Scott Shell. The relative entropy is fundamental to multiscale and inverse thermodynamic problems. *The Journal of chemical physics*, 129:144108, 2008.

[17] Aviel Chaimovich and M. Scott Shell. Coarse-graining errors and numerical optimization using a relative entropy framework. *The Journal of Chemical Physics*, 134:094112, 2011.

[18] Joseph F. Rudzinski and W. G. Noid. Coarse-graining entropy, forces, and structures. *The Journal of Chemical Physics*, 135:214101, 2011.

[19] Alexander Lyubartsev, Alexander Mirzoev, LiJun Chen, and Aatto Laaksonen. Systematic coarse-graining of molecular models by the newton inversion method. *Faraday discussions*, 144:43–56, 2010.

[20] Jibao Lu, Yuqing Qiu, Riccardo Baron, and Valeria Molinero. Coarse-graining of TIP4p/2005, TIP4p-ew, SPC/e, and TIP3p to monatomic anisotropic water models using relative entropy minimization. *Journal of Chemical Theory and Computation*, 10:4104–4120, 2014.

[21] H Wang, C Junghans, and K Kremer. Comparative atomistic and coarse-grained study of water: What do we lose by coarse-graining? *Eur. Phys. J. E*, 28(2):221–229, 2009.

[22] P. Ganguly, D. Mukherji, C. Junghans, and Nico F. A. van der Vegt. Kirkwood-buff coarse-grained force fields for aqueous solutions. *J. Chem. Theor. Comp.*, page accepted, 2012.

[23] Tiago E. de Oliveira, Paulo A. Netz, Kurt Kremer, Christoph Junghans, and Debashish Mukherji. C–ibi: Targeting cumulative coordination within an iterative protocol to derive coarse-grained models of (multi-component) complex fluids. *The Journal of Chemical Physics*, 144(17), 2016.

[24] John G. Kirkwood and Frank P. Buff. The statistical mechanical theory of solutions. i. *The Journal of Chemical Physics*, 19(6):774–777, 1951.

[25] D. Mukherji and K. Kremer. Coil–globule–coil transition of pnipam in aqueous methanol: Coupling all-atom simulations to semi-grand canonical coarse-grained reservoir. *Macromolecules*, 46(22):9158–9163, 2013.

[26] A. Ben-Naim. *Molecular Theory of Solutions*. Oxford University Press, New York, 2006.